1. Understanding and Plotting Vectors.

❖ Cartesian coordinate

Source code:

```
x1 = 8; %input('Enter the x-coordinate of the starting point: ');
y1 = 6;   %input('Enter the y-coordinate of the starting point: ');
z1 = 7;   %input('Enter the z-coordinate of the starting point: ');
x2 = 9;  %input('Enter the x-coordinate of the ending point: ');
y2 = 5;   %  input('Enter the y-coordinate of the ending point: ');
z2 = 6; %input('Enter the z-coordinate of the ending point: ');
u = x2 - x1; % x-component
v = y2 - y1; % y-component
w = z2 - z1; % z-component
figure;
quiver3(x1, y1, z1, u, v, w);
axis equal;
grid on;
xlabel('X');
ylabel('Y');
zlabel('Z');
title('3D Cartesian Vector');
```

❖ spherical coordinates

Source code:

```
% Define the spherical coordinates
rho = 5; % Radial distance
theta = pi/4; % Azimuthal angle (45 degrees)
phi = pi/6; % Polar angle (30 degrees)
% Convert spherical coordinates to Cartesian coordinates
x = rho * sin(phi) * cos(theta);
y = rho * sin(phi) * sin(theta);
z = rho * cos(phi);
% Create a sphere for visualization
[phi_s, theta_s] = meshgrid(linspace(0, pi, 50), linspace(0, 2*pi, 100));
x_s = rho * sin(phi_s) .* cos(theta_s);
y_s = rho * sin(phi_s) .* sin(theta_s);
z_s = rho * cos(phi_s);
% Plot the spherical coordinate system
figure;
hold on;
% Plot the sphere
surf(x_s, y_s, z_s, 'FaceAlpha', 0.3, 'EdgeColor', 'none');
```

```
colormap('cool');
% Plot the radial line
plot3([0, x], [0, y], [0, z], 'r', 'LineWidth', 2);
% Plot the projection of the point in the xy-plane
plot3([0, x], [0, y], [0, 0], 'g--', 'LineWidth', 1.5);
% Plot the projection of the point on the z-axis
plot3([x, x], [y, y], [0, z], 'b--', 'LineWidth', 1.5);
% Labels and title
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Spherical Coordinate System');
% Adjust the view angle
view(3);
axis equal;
grid on;
% Add annotations
text(x, y, z, ' Point (x, y, z)', 'FontSize', 12, 'Color', 'red');

view([33.06 33.15])
```

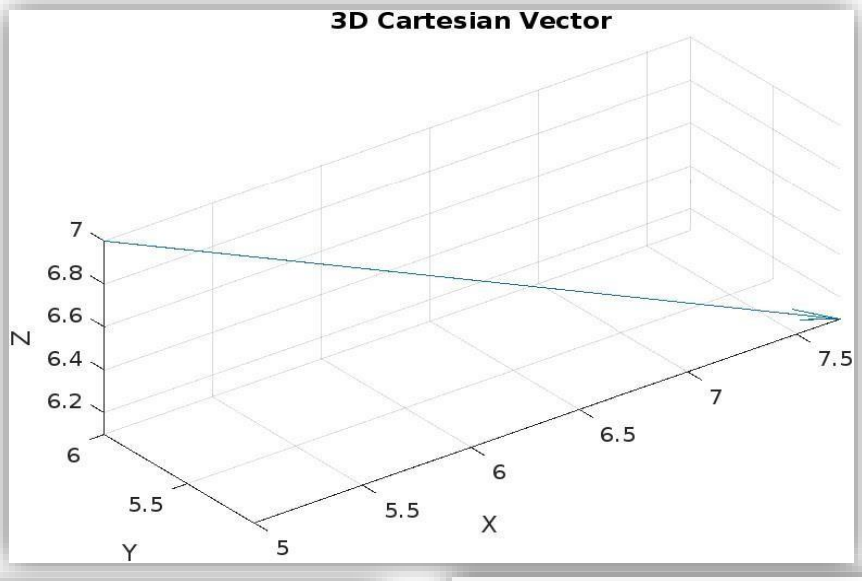
❖ cylindrical coordinates

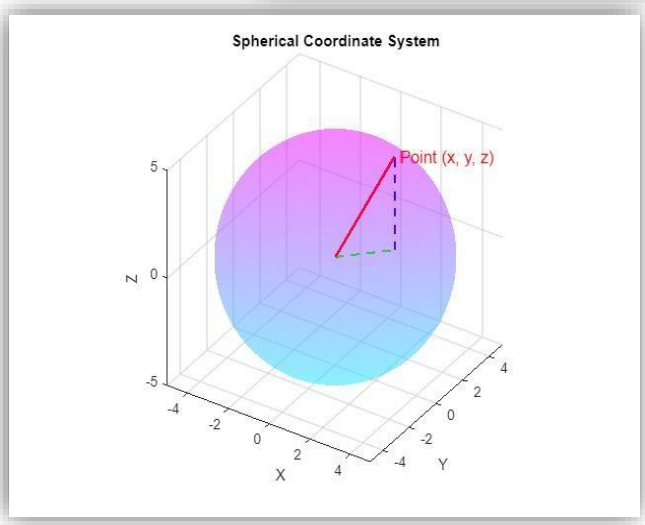
Source code:

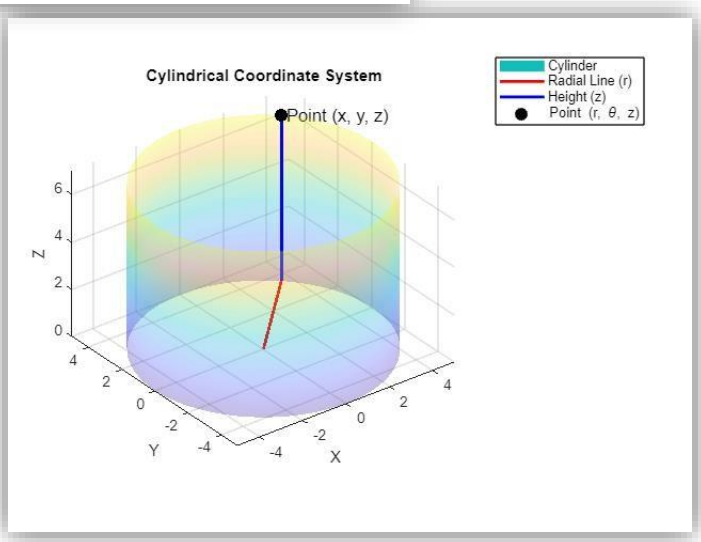
```
% Define the cylindrical coordinates
r = 5; % Radial distance
theta = pi/4; % Azimuthal angle (45 degrees)
z_cyl = 7; % Height
% Convert cylindrical coordinates to Cartesian coordinates
x = r * cos(theta);
y = r * sin(theta);
z = z_cyl;
% Create a cylinder for visualization
theta_c = linspace(0, 2*pi, 100);
z_c = linspace(0, z_cyl, 50);
[Theta_c, Z_c] = meshgrid(theta_c, z_c);
X_c = r * cos(Theta_c);
Y_c = r * sin(Theta_c);
% Plot the cylindrical coordinate system
figure;
hold on;
% Plot the cylinder
surf(X_c, Y_c, Z_c, 'FaceAlpha', 0.3, 'EdgeColor','texturemap');
%colormap('jet');
```

```
% Plot the radial line in the XY-plane
plot3([0, x], [0, y], [0, 0], 'r', 'LineWidth', 2);
% Plot the vertical line along the Z-axis
plot3([x, x], [y, y], [0, z], 'b', 'LineWidth', 2);
% Plot the point
plot3(x, y, z, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k');
% Labels and title
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Cylindrical Coordinate System');
% Adjust the view angle
view(3);
axis equal;
grid on;
% Add a legend
legend('Cylinder', 'Radial Line (r)', 'Height (z)', 'Point (r, \theta, z)');
% Add annotations
text(x, y, z, ' Point (x, y, z)', 'FontSize', 12, 'Color', 'k');
```

Output:







2. Point to point and Vector Transformation from Cartesian to cylindrical co-ordinate system and vice versa.

Source code:

```
% Cartesian coordinates
x = 4;
y = 3;
z = 5;
% Transform to Cylindrical coordinates
r = sqrt(x^2 + y^2);
theta = atan2(y, x);
z_cyl = z;
disp(['Cylindrical coordinates: r = ', num2str(r), ', theta = ',
num2str(theta),', z = ', num2str(z_cyl)]);
```

Output:

```
Cylindrical coordinates: r = 5, theta = 0.6435, z = 5
```

Source code:

```
% Cylindrical coordinates
r = 5;
theta = pi/4;  % Example angle in radians
z_cyl = 7;
% Transform to Cartesian coordinates
x = r * cos(theta);
y = r * sin(theta);
z = z_cyl;
disp(['Cartesian coordinates: x = ', num2str(x), ', y = ', num2str(y),
',z=',num2str(z)]);
```

Output:

```
Cartesian coordinates: x = 3.5355, y = 3.5355, z = 7
```

3. Point to point and Vector Transformation from Cartesian to Spherical co-ordinate system and vice versa

Source code:

```
% Cartesian coordinates
x = 4;
y = 3;
z = 5;
% Transform to Spherical coordinates
r_sph = sqrt(x^2 + y^2 + z^2);
theta_sph = atan2(y, x);
phi = acos(z / r_sph);
disp(['Spherical coordinates: r = ', num2str(r_sph), ', theta =
',num2str(theta_sph), ', phi = ', num2str(phi)]);
```

Output:

```
Spherical coordinates: r = 7.0711, theta = 0.6435, phi = 0.7854
```

Source code:

```
% Spherical coordinates
r_sph = 7;
theta_sph = pi/4; % 45 degrees
phi = pi/6; % 30 degrees
% Transform to Cartesian coordinates
x = r_sph * sin(phi) * cos(theta_sph);
y = r_sph * sin(phi) * sin(theta_sph);
z = r_sph * cos(phi);
disp(['Cartesian coordinates: x = ', num2str(x), ', y = ', num2str(y), ', z =
',num2str(z)]);
```

Output:

```
Cartesian coordinates: x = 2.4749, y = 2.4749, z = 6.0622
```

4. Point to point and Vector Transformation from Cylindrical to Spherical co-ordinate system and vice versa

Source code:

```
% Spherical coordinates
rho = 8;
phi = pi/6; % 30 degrees
theta = pi/4; % 45 degrees
% Transform to Cylindrical coordinates
r = rho * sin(phi);
z_cyl = rho * cos(phi);
disp(['Cylindrical coordinates: r = ', num2str(r), ', theta = ',
num2str(theta), ', z = ', num2str(z_cyl)]);
```

Output:

```
Cylindrical coordinates: r = 4, theta = 0.7854, z = 6.9282
```

Source code:

```
% Cylindrical coordinates
r = 5;
theta = pi/4; % 45 degrees
z_cyl = 7;
% Transform to Spherical coordinates
rho = sqrt(r^2 + z_cyl^2);
phi = atan2(r, z_cyl);
disp(['Spherical coordinates: rho = ', num2str(rho), ', phi = ', num2str(phi),
', theta = ', num2str(theta)]);
```

Output:

```
Spherical coordinates: rho = 8.6023, phi = 0.62025, theta = 0.7854
```

5. Representation of the Gradient of a scalar field, Divergence and Curl of Vector Fields.

Source code:

```
% MATLAB code to compute the Gradient, Divergence, and Curl

syms x y z  % Define symbolic variables

  % Define a scalar field
f = x^2 + y^2 + z^2;  % Example scalar field

  % Define a vector field
F = [x*y, y*z, z*x]; % Example vector field

  %% 1. Gradient of the Scalar Field
grad_f = gradient(f, [x, y, z]);
disp('Gradient of the Scalar Field:');
disp(grad_f);

  %% 2. Divergence of the Vector Field
div_F = divergence(F, [x, y, z]);
disp('Divergence of the Vector Field:');
disp(div_F);

%% 3. Curl of the Vector Field
curl_F = curl(F, [x, y, z]);
disp('Curl of the Vector Field:');
disp(curl_F);
```

Output:

Gradient of the Scalar Field:

$$\begin{pmatrix} 2x \\ 2y \\ 2z \end{pmatrix}$$

Divergence of the Vector Field: $x + y + z$

Curl of the Vector Field:

$$\begin{pmatrix} -y \\ -z \\ -x \end{pmatrix}$$

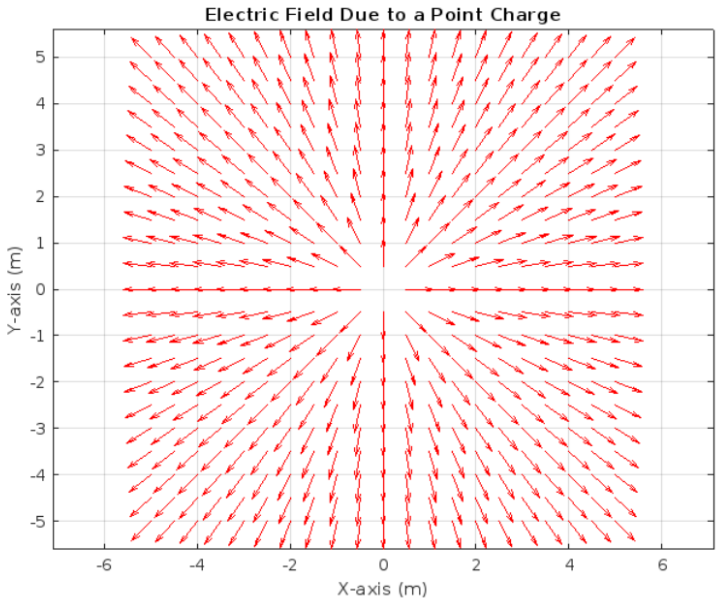
6. Plots of Electric field due to charge distributions.

- **Point Charge:** The field radiates out symmetrically, with vectors pointing outward (positive charge) or inward (negative charge).

Source code:

```
% Define grid
[x, y] = meshgrid(-5:0.5:5, -5:0.5:5); % Grid range for X and Y axes
z = 0; % Assume point charge lies in the XY-plane
% Charge properties
q = 1; % Charge magnitude
epsilon_0 = 8.854e-12; % Permittivity of free space
% Calculate distance r and field components
r = sqrt(x.^2 + y.^2 + z^2); % Distance from charge
Ex = (q / (4 * pi * epsilon_0)) .* (x ./ r.^3); % X-component of E
Ey = (q / (4 * pi * epsilon_0)) .* (y ./ r.^3); % Y-component of E
% Normalize for visualization
Ex_norm = Ex ./ sqrt(Ex.^2 + Ey.^2);
Ey_norm = Ey ./ sqrt(Ex.^2 + Ey.^2);
% Plot the field
figure;
quiver(x, y, Ex_norm, Ey_norm, 'r');
title('Electric Field Due to a Point Charge');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
axis equal;
grid on;
```

Output:



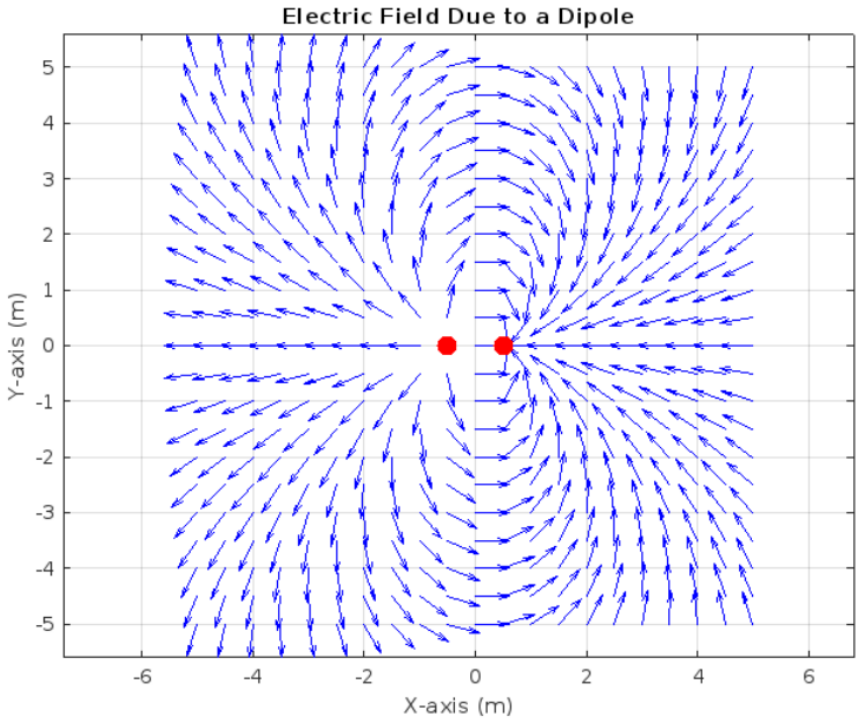
- **Dipole:** The field forms a characteristic pattern with loops. Vectors point from positive to negative charge.

Source code:

```
% Define grid
[x, y] = meshgrid(-5:0.5:5, -5:0.5:5); % Grid range for X and Y axes
z = 0; % Assume dipole lies in the XY-plane


% Dipole properties
q = 1; % Charge magnitude
epsilon_0 = 8.854e-12; % Permittivity of free space
d = 1; % Distance between charges


% Positions of charges
x_pos = -d/2; % Negative charge position
x_neg = d/2;  % Positive charge position


% Calculate distance from each charge
r_pos = sqrt((x - x_pos).^2 + y.^2 + z^2); % Distance to positive charge
r_neg = sqrt((x - x_neg).^2 + y.^2 + z^2); % Distance to negative charge


% Field components for each charge
Ex_pos = (q / (4 * pi * epsilon_0)) .* ((x - x_pos) ./ r_pos.^3); % X-component from
+q
Ey_pos = (q / (4 * pi * epsilon_0)) .* (y ./ r_pos.^3);           % Y-component from
+q


Ex_neg = (-q / (4 * pi * epsilon_0)) .* ((x - x_neg) ./ r_neg.^3); % X-component
from -q
Ey_neg = (-q / (4 * pi * epsilon_0)) .* (y ./ r_neg.^3);           % Y-component
from -q


% Total field components
Ex = Ex_pos + Ex_neg;
Ey = Ey_pos + Ey_neg;


% Normalize for visualization
Ex_norm = Ex ./ sqrt(Ex.^2 + Ey.^2);
Ey_norm = Ey ./ sqrt(Ex.^2 + Ey.^2);


% Plot the field
figure;
quiver(x, y, Ex_norm, Ey_norm, 'b');
hold on;
scatter([-d/2, d/2], [0, 0], 100, 'r', 'filled'); % Show charges
title('Electric Field Due to a Dipole');
```

```
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
axis equal;
grid on;
```

Output:



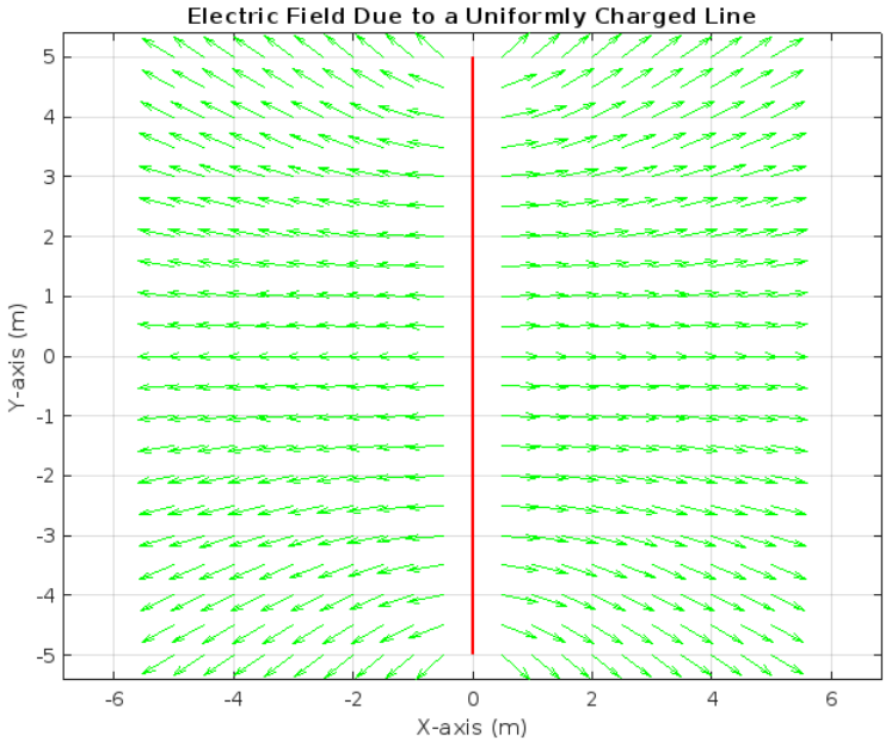
- **Line Charge:** The field has radial symmetry about the line, with vectors pointing outward.

Source code:

```
% Define grid
[x, y] = meshgrid(-5:0.5:5, -5:0.5:5);


% Line charge properties
lambda = 1e-6; % Charge density (C/m)
epsilon_0 = 8.854e-12; % Permittivity of free space
L = 10; % Length of the line charge


% Initialize field components
Ex = zeros(size(x));
Ey = zeros(size(y));


% Compute field using integration
for i = 1:numel(x)
    % Integrate along the line
    for yp = -L/2:0.1:L/2
```

```
        r = sqrt((x(i) - 0).^2 + (y(i) - yp).^2); % Distance to element
        dEx = (lambda / (2 * pi * epsilon_0)) * (x(i) / r^3); % dEx
        dEy = (lambda / (2 * pi * epsilon_0)) * ((y(i) - yp) / r^3); % dEy
        Ex(i) = Ex(i) + dEx;
        Ey(i) = Ey(i) + dEy;
    end
end


% Normalize for visualization
Ex_norm = Ex ./ sqrt(Ex.^2 + Ey.^2);
Ey_norm = Ey ./ sqrt(Ex.^2 + Ey.^2);


% Plot the field
figure;
quiver(x, y, Ex_norm, Ey_norm, 'g');
hold on;
plot([0, 0], [-L/2, L/2], 'r', 'LineWidth', 2); % Line charge
title('Electric Field Due to a Uniformly Charged Line');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
axis equal;
grid on;
```

Output:



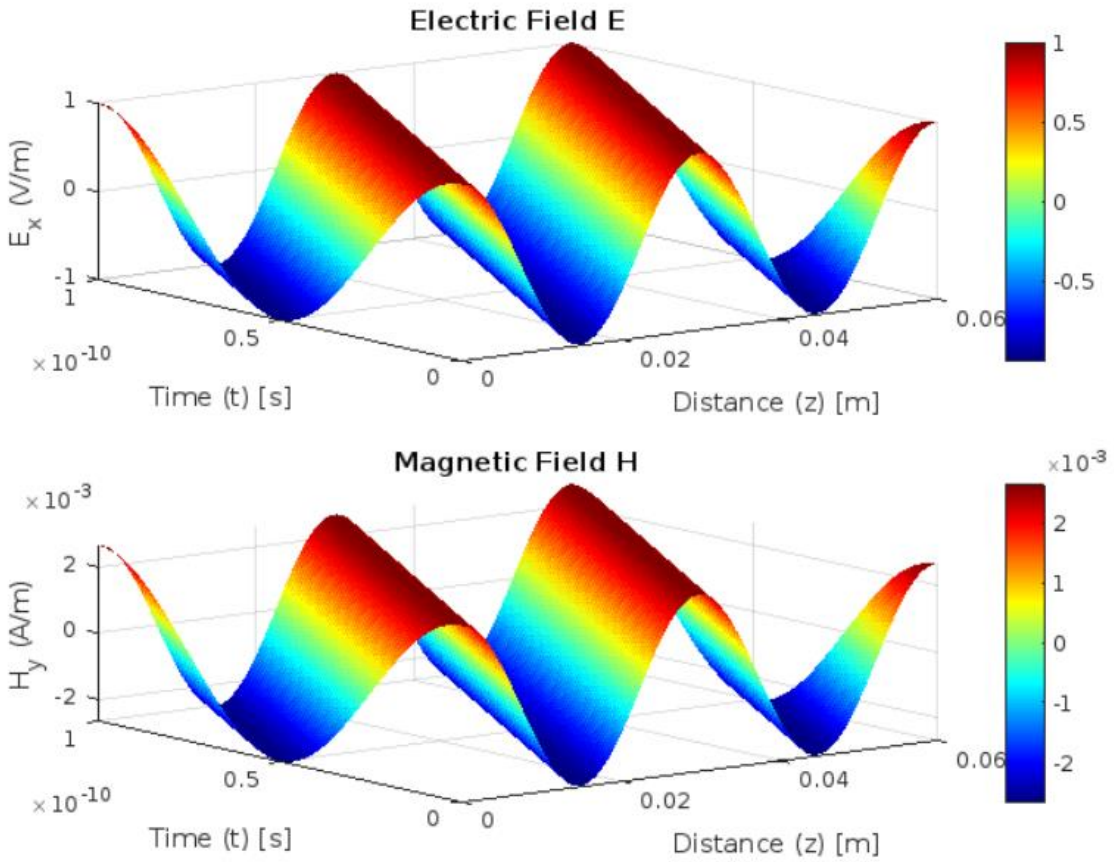
7. Find the Magnetic field from a given Electric field for a Uniform plane wave.

Source code:

```
% Constants
epsilon_0 = 8.854e-12; % Permittivity of free space (F/m)
mu_0 = 4 * pi * 1e-7;  % Permeability of free space (H/m)
c = 1 / sqrt(epsilon_0 * mu_0); % Speed of light in vacuum (m/s)
eta_0 = sqrt(mu_0 / epsilon_0); % Intrinsic impedance of free space (ohms)


% Electric Field Parameters
E0 = [1; 0; 0]; % Amplitude of Electric Field (V/m) in x-direction
frequency = 10e9; % Frequency of the wave (Hz)
omega = 2 * pi * frequency; % Angular frequency (rad/s)
wavelength = c / frequency; % Wavelength (m)
k = 2 * pi / wavelength; % Wave number (rad/m)


% Propagation Direction
k_vec = [0; 0; 1]; % Wave vector (z-direction propagation)
k_hat = k_vec / norm(k_vec); % Unit vector in direction of propagation


% Magnetic Field Calculation
H0 = (1 / eta_0) * cross(k_hat, E0); % Magnetic Field amplitude


% Display results
fprintf('Electric Field Amplitude: [%f, %f, %f] V/m\n', E0);
fprintf('Magnetic Field Amplitude: [%f, %f, %f] A/m\n', H0);


% Time and Space Parameters for Visualization
z = linspace(0, 2*wavelength, 100); % Distance along propagation (z-axis)
t = linspace(0, 1/frequency, 50); % Time for one cycle
[Z, T] = meshgrid(z, t); % Create space-time grid


% Field Components over Space-Time
E_x = E0(1) * cos(k * Z - omega * T); % Electric field (x-direction)
H_y = H0(2) * cos(k * Z - omega * T); % Magnetic field (y-direction)


% Plotting
figure;


% Electric Field Plot
subplot(2, 1, 1);
surf(Z, T, E_x, 'EdgeColor', 'none');
```

```
xlabel('Distance (z) [m]');
ylabel('Time (t) [s]');
zlabel('E_x (V/m)');
title('Electric Field \bf{E}');
colormap jet;
colorbar;


% Magnetic Field Plot
subplot(2, 1, 2);
surf(Z, T, H_y, 'EdgeColor', 'none');
xlabel('Distance (z) [m]');
ylabel('Time (t) [s]');
zlabel('H_y (A/m)');
title('Magnetic Field \bf{H}');
colormap jet;
colorbar;
```

Output:



8. Find a Poynting Vector for a given electromagnetic field at a given point.

Source code:

```
% Clear workspace and set up parameters
clear; clc;


% Given Electric Field and Magnetic Field
E = [3; 0; 0] % Electric field in V/m (x-direction)
H = [0; 2; 0] % Magnetic field in A/m (y-direction)


% Calculate Poynting Vector
S = cross(E, H) % Poynting vector (W/m^2)


% Normalize vectors for visualization
E_norm = E / norm(E); % Unit vector for E
H_norm = H / norm(H); % Unit vector for H
S_norm = S / norm(S); % Unit vector for S


% Origin for vectors
origin = [0, 0, 0];


% Define scaled vectors for better visualization
scale_factor = 1; % Adjust for desired arrow size
E_scaled = scale_factor * E; % Scaled E vector
H_scaled = scale_factor * H; % Scaled H vector
S_scaled = scale_factor * S; % Scaled S vector


% Plotting
figure;
hold on;
grid on;
axis equal;


% Plot Electric Field (E)
quiver3(origin(1), origin(2), origin(3), E_scaled(1), E_scaled(2), E_scaled(3), 0,
...
    'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);
text(E_scaled(1), E_scaled(2), E_scaled(3), '\bf{E}', 'FontSize', 12, 'Color', 'r');


% Plot Magnetic Field (H)
quiver3(origin(1), origin(2), origin(3), H_scaled(1), H_scaled(2), H_scaled(3), 0,
...
    'b', 'LineWidth', 2, 'MaxHeadSize', 0.5);
text(H_scaled(1), H_scaled(2), H_scaled(3), '\bf{H}', 'FontSize', 12, 'Color', 'b');
```

```
% Plot Poynting Vector (S)
quiver3(origin(1), origin(2), origin(3), S_scaled(1), S_scaled(2), S_scaled(3), 0,
...
    'g', 'LineWidth', 2, 'MaxHeadSize', 0.5);
text(S_scaled(1), S_scaled(2), S_scaled(3), '\bf{S}', 'FontSize', 12, 'Color', 'g');


% Customize plot
xlabel('X-axis');
ylabel('Y-axis');
zlabel('Z-axis');
title('Electric Field (\bf{E}), Magnetic Field (\bf{H}), and Poynting Vector
(\bf{S})');
legend({'\bf{E} (Electric Field)', '\bf{H} (Magnetic Field)', '\bf{S} (Poynting
Vector)'}, 'Location', 'Best');
view(3); % 3D view for better visualization
hold off;
```

Output: