# PhysHW10

November 15, 2024

#### 0.0.1 Applied EMT

## 0.1 Dennies Bor

[1]:
```
#--------------------------------1.1-----------------------------------------
#       (Rs + iXs)
# |-----[Z_s]--------------|
# |                        |
# |                        |
# (~) Vs(t) = VoCos(wt)  [Z_l] (Rl + iXl)
# |                        |
# |                        |
# |                        |
# |------------------------|
```

The current in phasor form:

The voltage source in phasor is:

$$\bar{V}_s = V_o \angle 0^\circ$$

Total impedance:

$$\bar{Z}_t = (R_s + R_l) + j(X_s + X_l) = |Z_t| \angle \theta$$

where:

$$|Z_t| = \sqrt{(R_s + R_l)^2 + (X_s + X_l)^2}$$

$$\theta = \tan^{-1}\left(\frac{X_s + X_l}{R_s + R_l}\right)$$

Therefore, the current phasor is:

$$\bar{I} = \frac{\bar{V}_s}{\bar{Z}_t} = \frac{V_o \angle 0^\circ}{|Z_t| \angle \theta} = \frac{V_o}{|Z_t|} \angle (-\theta)$$

This gives us the instantaneous current:

$$i(t) = \frac{V_o}{|Z_t|} \cos(\omega t - \theta)$$

The load voltage phasor (voltage across $\bar{Z}_l$) is:

$$\bar{V}_l = \bar{I} \bar{Z}_l$$

Where load impedance is:

$$\bar{Z}_l = R_l + jX_l = |Z_l|\angle\phi$$

$$|Z_l| = \sqrt{R_l^2 + X_l^2}$$

$$\phi = \tan^{-1}\left(\frac{X_l}{R_l}\right)$$

Therefore:

$$\bar{V}_l = \frac{V_o}{|Z_t|}\angle(-\theta) \cdot |Z_l|\angle\phi = \frac{V_o|Z_l|}{|Z_t|}\angle(-\theta + \phi)$$

In time domain, load voltage is:

$$v_l(t) = \frac{V_o|Z_l|}{|Z_t|}\cos(\omega t - \theta + \phi)$$

The instantaneous power in the load is:

$$p_l(t) = v_l(t)i(t)$$

$$p_l(t) = \frac{V_o^2|Z_l|}{|Z_t|^2}\cos(\omega t - \theta + \phi)\cos(\omega t - \theta)$$

Using the cosine product identity:

$$\cos A \cos B = \frac{1}{2}[\cos(A - B) + \cos(A + B)]$$

We get:

$$p_l(t) = \frac{V_o^2|Z_l|}{2|Z_t|^2}[\cos(\phi) + \cos(2\omega t - 2\theta + \phi)]$$

[2]:
```
#---------------------------------1.2------------------------------------------
```

The average value of a periodic function:

$$P_l = \frac{1}{T}\int_0^T p_l(t)dt$$

Where $p_l(t)$ is:

$$p_l(t) = \frac{V_o^2|Z_l|}{2|Z_t|^2}[\cos(\phi) + \cos(2\omega t - 2\theta + \phi)]$$

Substituting:

$$P_l = \frac{1}{T}\int_0^T \frac{V_o^2|Z_l|}{2|Z_t|^2}[\cos(\phi) + \cos(2\omega t - 2\theta + \phi)]dt$$

Where $T = \frac{2\pi}{\omega}$:

$$P_l = \frac{\omega}{2\pi}\int_0^{2\pi/\omega} \frac{V_o^2|Z_l|}{2|Z_t|^2}[\cos(\phi) + \cos(2\omega t - 2\theta + \phi)]dt$$

$$P_l = \frac{\omega}{2\pi}\frac{V_o^2|Z_l|}{2|Z_t|^2}\int_0^{2\pi/\omega}[\cos(\phi) + \cos(2\omega t - 2\theta + \phi)]dt$$

Integrating the two terms separately:

$$P_l = \frac{\omega}{2\pi}\frac{V_o^2|Z_l|}{2|Z_t|^2}\left[t\cos(\phi) + \frac{1}{2\omega}\sin(2\omega t - 2\theta + \phi)\right]_0^{2\pi/\omega}$$

Evaluating at limits:

$$P_l = \frac{\omega}{2\pi}\frac{V_o^2|Z_l|}{2|Z_t|^2}\left[\left(\frac{2\pi}{\omega}\cos(\phi) + \frac{1}{2\omega}\sin(2\cdot 2\pi - 2\theta + \phi)\right) - \left(0\cdot\cos(\phi) + \frac{1}{2\omega}\sin(-2\theta + \phi)\right)\right]$$

Since $\sin(2\cdot 2\pi + x) = \sin(x)$:

$$P_l = \frac{\omega}{2\pi}\frac{V_o^2|Z_l|}{2|Z_t|^2}\left[\frac{2\pi}{\omega}\cos(\phi) + \frac{1}{2\omega}(\sin(-2\theta + \phi) - \sin(-2\theta + \phi))\right]$$

The sine terms cancel out, leaving:

$$P_l = \frac{V_o^2|Z_l|}{2|Z_t|^2}\cos(\phi)$$

[3]:
```
#--------------------------------1.3-----------------------------------------
```

$$P_l = \frac{V_o^2|Z_l|}{2|Z_t|^2}\cos(\phi)$$

$$|Z_l| = \sqrt{R_l^2 + X_l^2}$$

$$|Z_t| = \sqrt{(R_s + R_l)^2 + (X_s + X_l)^2}$$

$$\cos(\phi) = \frac{R_l}{\sqrt{R_l^2 + X_l^2}} = \frac{R_l}{|Z_l|}$$

Substituting:

$$P_l = \frac{V_o^2\sqrt{R_l^2 + X_l^2}}{2[(R_s + R_l)^2 + (X_s + X_l)^2]}\cdot\frac{R_l}{\sqrt{R_l^2 + X_l^2}}$$

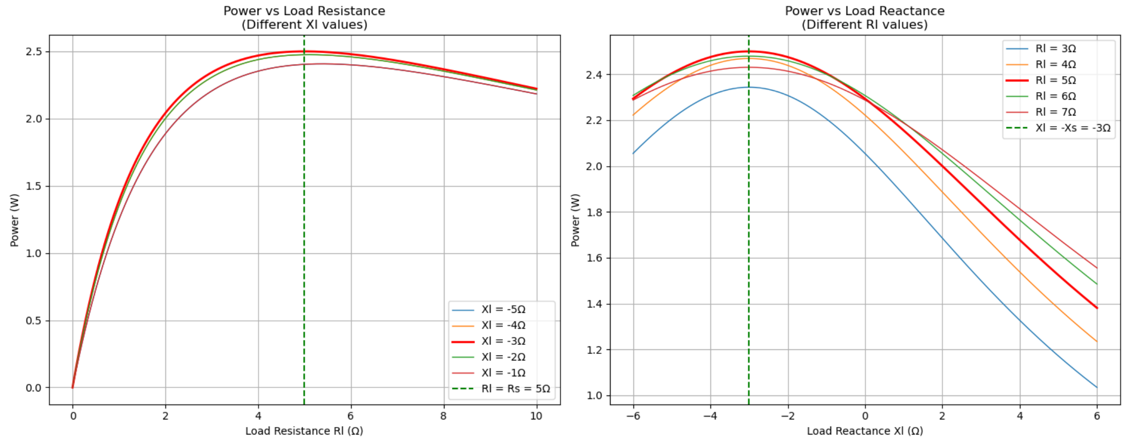
Simplifying:

$$P_l = \frac{V_o^2 R_l}{2[(R_s + R_l)^2 + (X_s + X_l)^2]}$$

[4]:
```
# Proof by brute force
import numpy as np
import matplotlib.pyplot as plt
```

```python
# Constants
Vo = 10  # Source voltage
Rs = 5   # Source resistance
Xs = 3   # Source reactance

# Create figure with subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Plot 1: P vs Rl for different Xl
Rl = np.linspace(0, 10, 1000)
Xl_values = [-Xs-2, -Xs-1, -Xs, -Xs+1, -Xs+2]

for Xl in Xl_values:
    P = (Vo**2 * Rl) / (2 * ((Rs + Rl)**2 + (Xs + Xl)**2))
    label = f'Xl = {Xl}Ω'
    linewidth = 2 if Xl == -Xs else 1
    style = '-r' if Xl == -Xs else '-'
    ax1.plot(Rl, P, style, label=label, linewidth=linewidth)

# Add vertical line where Rl = Rs
ax1.axvline(x=Rs, color='g', linestyle='--', label=f'Rl = Rs = {Rs}Ω')

ax1.set_xlabel('Load Resistance Rl (Ω)')
ax1.set_ylabel('Power (W)')
ax1.set_title('Power vs Load Resistance\n(Different Xl values)')
ax1.grid(True)
ax1.legend()

# Plot 2: P vs Xl for different Rl
Xl = np.linspace(-6, 6, 1000)
Rl_values = [Rs-2, Rs-1, Rs, Rs+1, Rs+2]

for Rl_val in Rl_values:
    P = (Vo**2 * Rl_val) / (2 * ((Rs + Rl_val)**2 + (Xs + Xl)**2))
    label = f'Rl = {Rl_val}Ω'
    linewidth = 2 if Rl_val == Rs else 1
    style = '-r' if Rl_val == Rs else '-'
    ax2.plot(Xl, P, style, label=label, linewidth=linewidth)

# Add vertical line where Xl = -Xs
ax2.axvline(x=-Xs, color='g', linestyle='--', label=f'Xl = -Xs = {-Xs}Ω')

ax2.set_xlabel('Load Reactance Xl (Ω)')
ax2.set_ylabel('Power (W)')
ax2.set_title('Power vs Load Reactance\n(Different Rl values)')
ax2.grid(True)
```

```
ax2.legend()

plt.tight_layout()
plt.show()
```




### Comment

The power is maximum when $dP/d(X_i/R_i)$ is zero

[5]:
```
#--------------------------------1.2----------------------------------------
# Considering a 3 step ladder
#     0     L1     1     L2     2     L3     3
# o--o---oooo---o---oooo---o---oooo---o
# |   |           |          |          |
# Vo  C0          C1         C2         ZL
# |   |           |          |          |
# o--o----------o----------o----------o
# |
# ground
```

From node 2:

$$Z_{series3} = j\omega L3 + Z_L$$

$$Z_{eq2} = \frac{(j\omega L3 + Z_L)(\frac{1}{j\omega C2})}{j\omega L3 + Z_L + \frac{1}{j\omega C2}}$$

From node 1:

$$Z_{series2} = j\omega L2 + Z_{eq2}$$

$$Z_{eq1} = \frac{(j\omega L2 + Z_{eq2})(\frac{1}{j\omega C1})}{j\omega L2 + Z_{eq2} + \frac{1}{j\omega C1}}$$

$$Z_{series1} = j\omega L1 + Z_{eq1}$$

$$Z_{total} = \frac{(j\omega L1 + Z_{eq1})(\frac{1}{j\omega C0})}{j\omega L1 + Z_{eq1} + \frac{1}{j\omega C0}}$$

The pattern for n sections: - Start with load $Z_L$ - For each section i (from n to 1): - Add series: $Z_{series_i} = j\omega L_i + Z_{eq_{i-1}}$ - Add parallel: $Z_{eq_i} = \frac{Z_{series_i} \cdot \frac{1}{j\omega C_i}}{Z_{series_i} + \frac{1}{j\omega C_i}}$

### 0.1.1 Solving Vs

Starting with

$$\bar{Z}_{eqn} = \bar{Z}_L$$

at rightmost node, for i from n down to 1:

$$\bar{Z}_{eq(i-1)} = \frac{(j\omega L_i + \bar{Z}_{eq_i})(\frac{1}{j\omega C_{i-1}})}{j\omega L_i + \bar{Z}_{eq_i} + \frac{1}{j\omega C_{i-1}}}$$

Then find voltages left-to-right: Starting with

$$\bar{I}_0 = \frac{\bar{V}_0}{\bar{Z}_{eq0}}$$

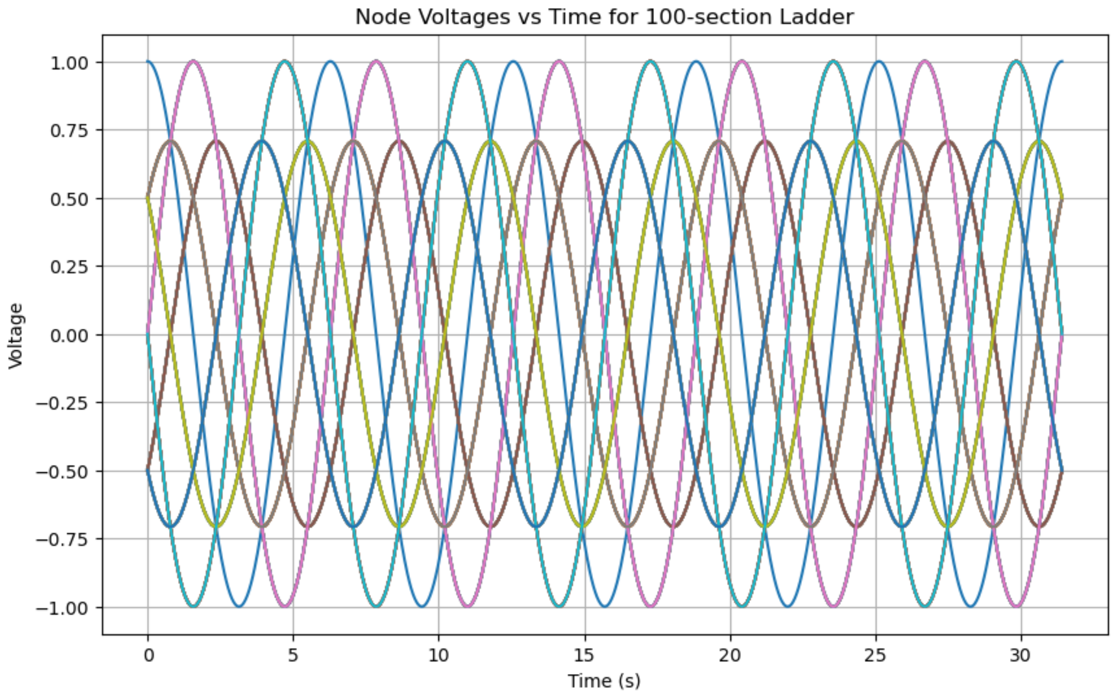
at leftmost node, for i from 1 to n:

$$\bar{V}_i = \bar{I}_{i-1} \cdot \bar{Z}_{eq_i}$$

$$\bar{I}_i = \bar{I}_{i-1} - \frac{\bar{V}_i}{\frac{1}{j\omega C_i}}$$

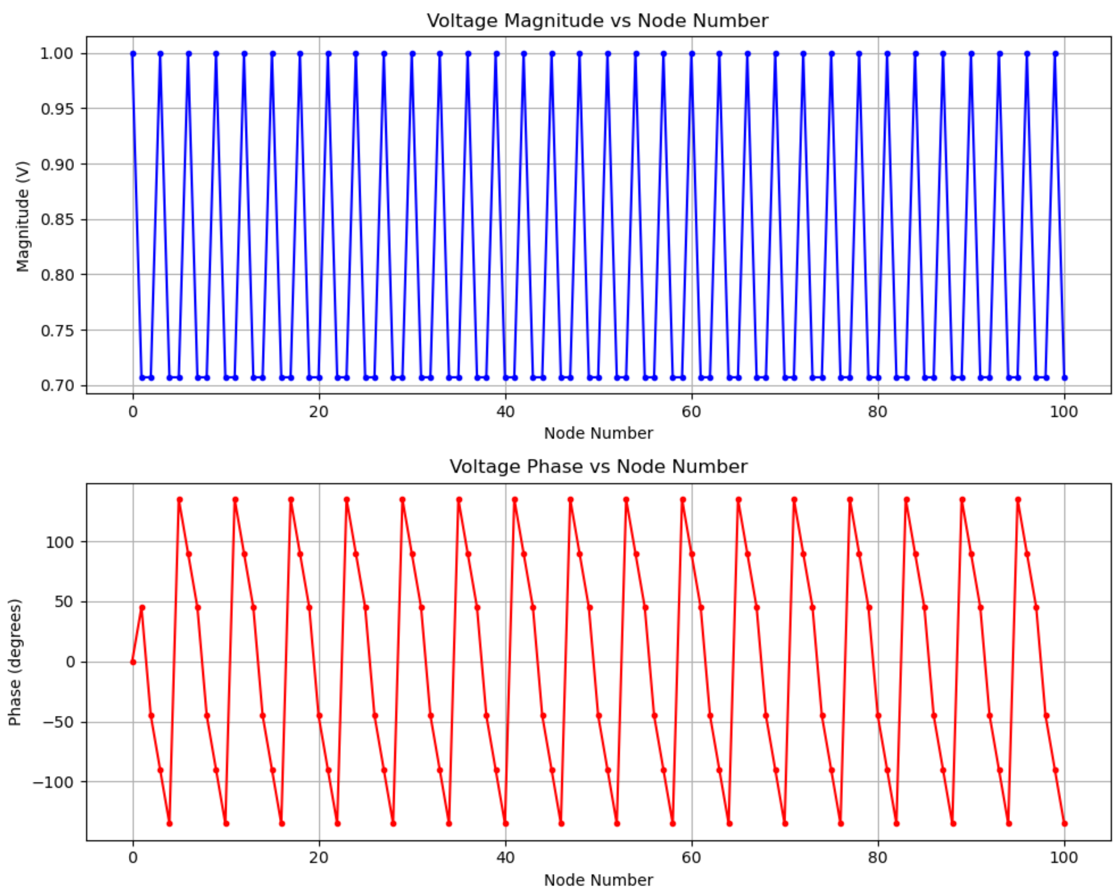
[6]:
```
def calculate_Zeq_array(n, w, L, C, ZL):
    """Calculate array of all Zeq values right to left"""
    Zeq = np.zeros(n+1, dtype=complex)
    Zeq[n] = ZL

    for i in range(n-1, -1, -1):
        Z_series = 1j*w*L + Zeq[i+1]
        Z_parallel = 1/(1j*w*C)
        Zeq[i] = (Z_series * Z_parallel)/(Z_series + Z_parallel)

    return Zeq

def calculate_node_voltages(n, w, L, C, ZL):
    """Calculate complex voltages using derived equations"""
    # Get all equivalent impedances
    Zeq = calculate_Zeq_array(n, w, L, C, ZL)

    # Initialize arrays
```

```
    V = np.zeros(n+1, dtype=complex)
    I = np.zeros(n+1, dtype=complex)

    # Initial conditions
    V[0] = 1
    I[0] = V[0]/Zeq[0]  # Initial current

    # Calculate voltages and currents going forward
    for i in range(1, n+1):
        V[i] = I[i-1] * Zeq[i]
        I[i] = I[i-1] - V[i]/(1/(1j*w*C))

    return V

# Parameters
n = 100           # number of sections
w = L = C = ZL = 1.0  # unity values
t = np.linspace(0, 10*np.pi, 1000)  # Time range

# Get complex voltages
V_complex = calculate_node_voltages(n, w, L, C, ZL)

# Calculate time-domain voltages
V_time = np.zeros((len(t), n+1))
for i in range(n+1):
    magnitude = np.abs(V_complex[i])
    phase = np.angle(V_complex[i])
    V_time[:, i] = magnitude * np.cos(w*t - phase)

# Plotting
plt.figure(figsize=(10, 6))
for i in range(n+1):
    plt.plot(t, V_time[:, i], label=f'V{i}')

plt.xlabel('Time (s)')
plt.ylabel('Voltage')
plt.title(f'Node Voltages vs Time for {n}-section Ladder')
plt.grid(True)
plt.show()

# print("\nComplex voltages:")
# for i, v in enumerate(V_complex):
#     print(f"V{i} = {v:.4f}, Magnitude = {abs(v):.4f}, Phase = {np.angle(v,␣
↪deg=True):.2f}°")
```



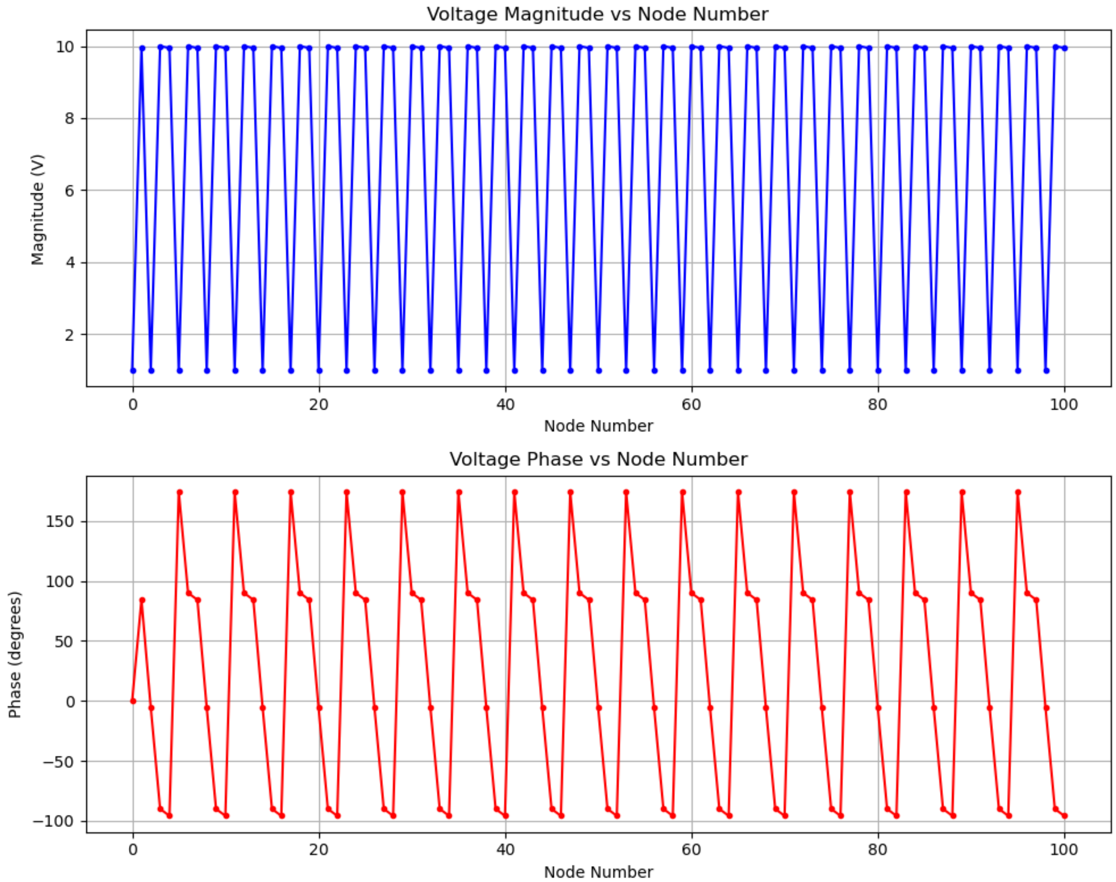

[7]:
```
# Phases vs magnitudes
# Parameters
n = 100
w = L = C = ZL = 1.0

# Get complex voltages
V_complex = calculate_node_voltages(n, w, L, C, ZL)

# Calculate magnitudes and phases
magnitudes = np.abs(V_complex)
phases = np.angle(V_complex, deg=True)  # in degrees
node_numbers = np.arange(n+1)

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

# Plot magnitude
ax1.plot(node_numbers, magnitudes, 'b.-')
ax1.set_xlabel('Node Number')
ax1.set_ylabel('Magnitude (V)')
ax1.set_title('Voltage Magnitude vs Node Number')
ax1.grid(True)

# Plot phase
```

```python
ax2.plot(node_numbers, phases, 'r.-')
ax2.set_xlabel('Node Number')
ax2.set_ylabel('Phase (degrees)')
ax2.set_title('Voltage Phase vs Node Number')
ax2.grid(True)

plt.tight_layout()
plt.show()
```




[8]:
```python
# PPhases vs mmagnitudes (10\root(L/C))
# Parameters
n = 100
w = L = C = 1
ZL = 10*np.sqrt(L/C)

# Get complex voltages
V_complex = calculate_node_voltages(n, w, L, C, ZL)

# Calculate magnitudes and phases
```

```python
magnitudes = np.abs(V_complex)
phases = np.angle(V_complex, deg=True)  # in degrees
node_numbers = np.arange(n+1)

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

# Plot magnitude
ax1.plot(node_numbers, magnitudes, 'b.-')
ax1.set_xlabel('Node Number')
ax1.set_ylabel('Magnitude (V)')
ax1.set_title('Voltage Magnitude vs Node Number')
ax1.grid(True)

# Plot phase
ax2.plot(node_numbers, phases, 'r.-')
ax2.set_xlabel('Node Number')
ax2.set_ylabel('Phase (degrees)')
ax2.set_title('Voltage Phase vs Node Number')
ax2.grid(True)

plt.tight_layout()
plt.show()
```




Using a small number due to divide by zero error

[11]:
```
# PPhases vs mnagnitude, Zl =0
# Parameters
n = 100
w = L = C = 1
ZL = 1e-100 # A very small number

# Get complex voltages
V_complex = calculate_node_voltages(n, w, L, C, ZL)

# Calculate magnitudes and phases
magnitudes = np.abs(V_complex)
phases = np.angle(V_complex, deg=True)  # in degrees
node_numbers = np.arange(n+1)

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
```

```python
# Plot magnitude
ax1.plot(node_numbers, magnitudes, 'b.-')
ax1.set_xlabel('Node Number')
ax1.set_ylabel('Magnitude (V)')
ax1.set_title('Voltage Magnitude vs Node Number')
ax1.grid(True)

# Plot phase
ax2.plot(node_numbers, phases, 'r.-')
ax2.set_xlabel('Node Number')
ax2.set_ylabel('Phase (degrees)')
ax2.set_title('Voltage Phase vs Node Number')
ax2.grid(True)

plt.tight_layout()
plt.show()
```