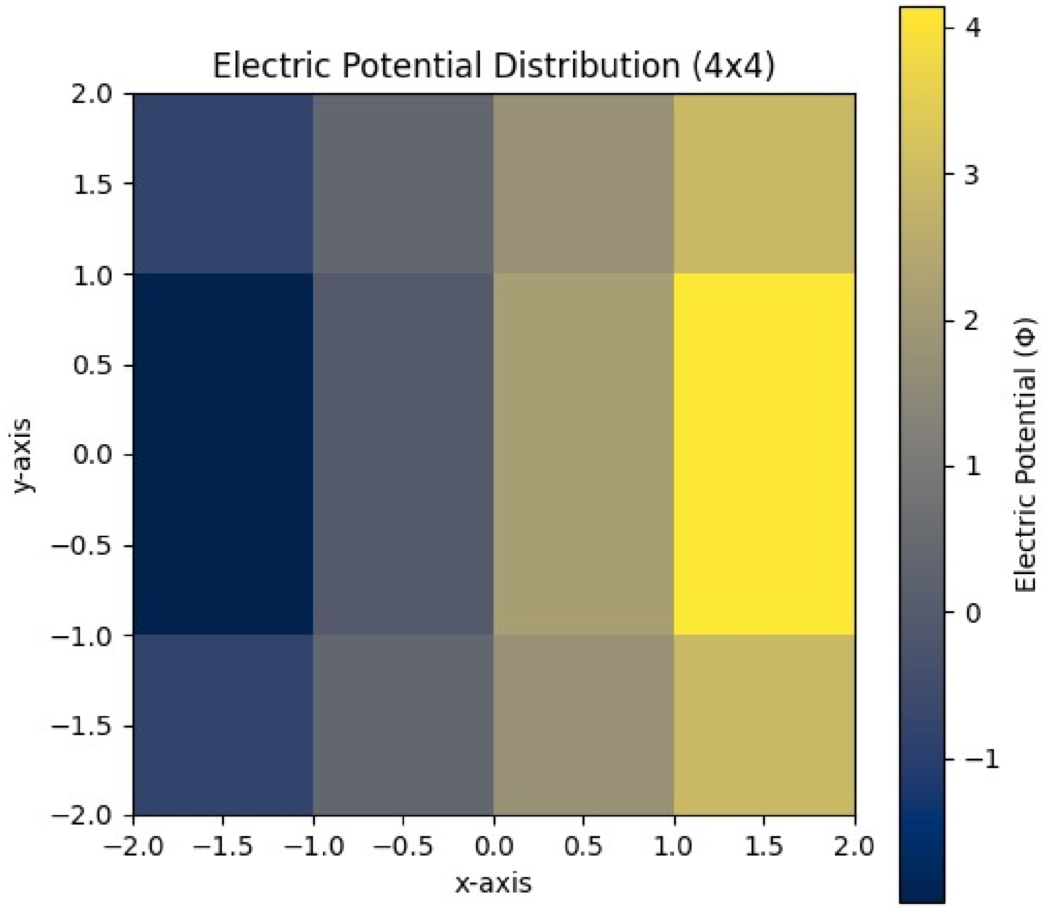
```python
import os
import numpy as np
import matplotlib.pyplot as plt
import time
import pandas as pd

# Convert 2D grid indices to a 1D index for the linear system
def map_indices(row, col, grid_rows):
    return row * grid_rows + col

# Calculate the harmonic mean for conductivity values
def compute_harmonic_mean(cond_a, cond_b):
    return 2 * cond_a * cond_b / (cond_a + cond_b)

# Create the coefficient matrix and source vector
def build_system_matrix_and_source(grid_rows, grid_cols,
conductivity_high, conductivity_low, current_value, step_x, step_y):
    total_cells = grid_rows * grid_cols
    system_matrix = np.zeros((total_cells, total_cells))
    source_vector = np.zeros(total_cells)

    # Define the conductivity distribution across the grid
    conductivity_field = np.ones((grid_rows, grid_cols)) *
conductivity_low
    conductivity_field[grid_rows // 4:3 * grid_rows // 4, grid_cols //
4:3 * grid_cols // 4] = conductivity_high

    # Precompute the transitional conductivity
    transitional_conductivity =
compute_harmonic_mean(conductivity_high, conductivity_low)

    # Helper function to determine the effective conductivity between
neighboring cells
    def calculate_effective_conductivity(cond_1, cond_2):
        return cond_1 if cond_1 == cond_2 else
transitional_conductivity

    # Populate the coefficient matrix and source vector
    for row in range(grid_rows):
        for col in range(grid_cols):
            current_idx = map_indices(row, col, grid_rows)
            upward =
calculate_effective_conductivity(conductivity_field[row, col],
conductivity_field[row - 1, col]) * step_x / step_y if row - 1 >= 0
else 0
            rightward =
calculate_effective_conductivity(conductivity_field[row, col],
conductivity_field[row, col + 1]) * step_y / step_x if col + 1 <
grid_cols else 0
            downward =
```

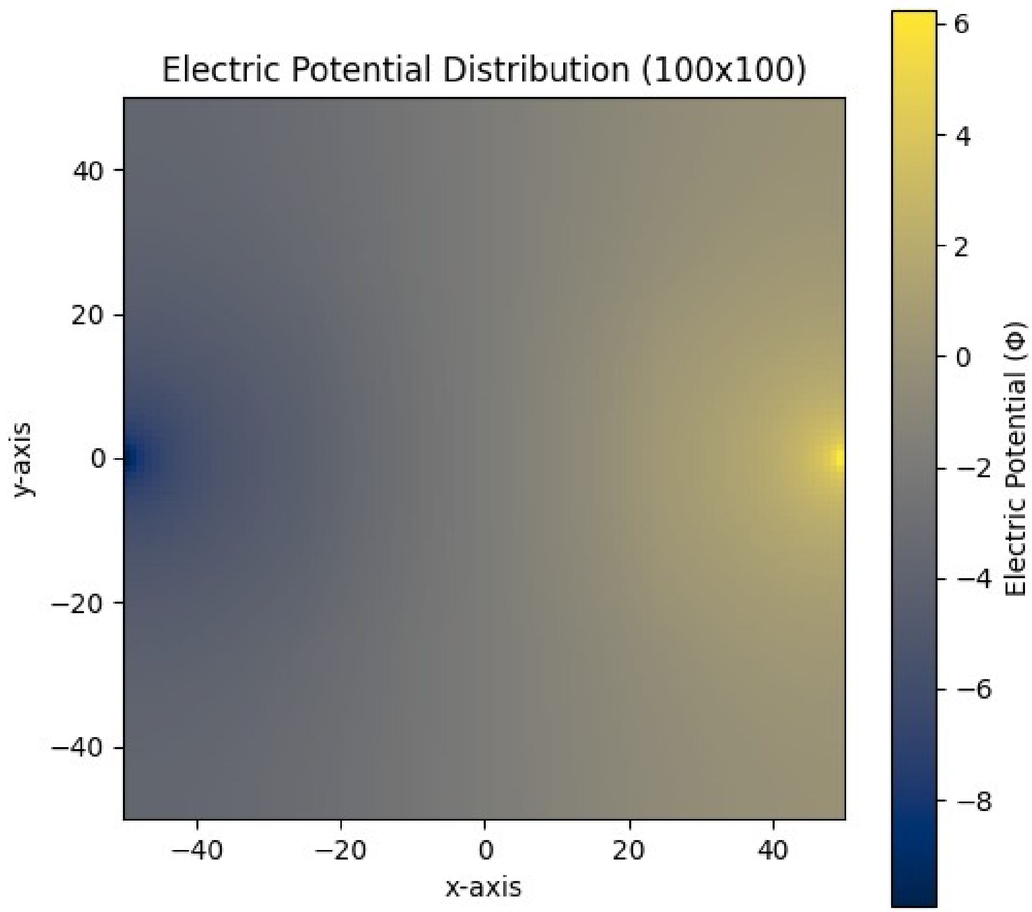
```
calculate_effective_conductivity(conductivity_field[row, col],
conductivity_field[row + 1, col]) * step_x / step_y if row + 1 <
grid_rows else 0
            leftward =
calculate_effective_conductivity(conductivity_field[row, col],
conductivity_field[row, col - 1]) * step_y / step_x if col - 1 >= 0
else 0
            center = -(upward + rightward + downward + leftward)

            system_matrix[current_idx, current_idx] = center
            if row - 1 >= 0:
                system_matrix[current_idx, map_indices(row - 1, col,
grid_rows)] = upward
            if col + 1 < grid_cols:
                system_matrix[current_idx, map_indices(row, col + 1,
grid_rows)] = rightward
            if row + 1 < grid_rows:
                system_matrix[current_idx, map_indices(row + 1, col,
grid_rows)] = downward
            if col - 1 >= 0:
                system_matrix[current_idx, map_indices(row, col - 1,
grid_rows)] = leftward

    # Apply boundary conditions for the source current
    mid_row_upper = grid_cols // 2 - 1
    mid_row_lower = grid_cols // 2

    source_vector[map_indices(mid_row_upper, 0, grid_rows)] = -
current_value / 2
    source_vector[map_indices(mid_row_lower, 0, grid_rows)] = -
current_value / 2
    source_vector[map_indices(mid_row_upper, grid_rows - 1,
grid_rows)] = current_value / 2
    source_vector[map_indices(mid_row_lower, grid_rows - 1,
grid_rows)] = current_value / 2

    return system_matrix, source_vector, conductivity_field

# Solve the finite volume problem
def solve_potential_distribution(grid_rows, grid_cols,
conductivity_high, conductivity_low, current_value, step_x, step_y,
reference_point):
    start_time = time.time()

    # Build the system matrix and source vector
    matrix, vector, conductivity_map =
build_system_matrix_and_source(grid_rows, grid_cols,
conductivity_high, conductivity_low, current_value, step_x, step_y)
    reference_idx = int(map_indices(*reference_point, grid_rows))
```

```
    # Eliminate the reference node to apply the ground potential
condition
    matrix = np.delete(matrix, reference_idx, axis=0)
    matrix = np.delete(matrix, reference_idx, axis=1)
    vector = np.delete(vector, reference_idx)
    vector = -vector  # Reverse polarity

    # Solve the system of equations
    potentials = np.linalg.solve(matrix, vector)

    # Reinsert the ground potential at the reference point
    potentials = np.insert(potentials, reference_idx, 0)
    potential_field = potentials.reshape((grid_rows, grid_cols))
    runtime = time.time() - start_time
    return potential_field, conductivity_map, runtime

# Visualize the potential distribution
def visualize_potential(potential_field, grid_rows, grid_cols, step_x,
step_y, axis_min, axis_max, title="Electric Potential Distribution"):
    title += f" ({grid_rows}x{grid_cols})"
    plt.figure(figsize=(6, 6))
    extent = [axis_min, axis_min + grid_rows * step_x, axis_max -
grid_cols * step_y, axis_max]
    plt.imshow(potential_field, cmap="cividis", origin="lower",
extent=extent)  # Changed colormap to 'cividis'
    plt.colorbar(label="Electric Potential (Φ)")
    plt.title(title)
    plt.xlabel("x-axis")
    plt.ylabel("y-axis")
    plt.show()

# Run the simulation for a given grid resolution
def execute_simulation(grid_rows, grid_cols, output_file,
reference_node, axis_limits=(-2, 2)):
    step_x, step_y = 1, 1
    conductivity_core, conductivity_outer = 0.1, 0.2
    current_magnitude = 1
    axis_min, axis_max = axis_limits

    potential_field, conductivity_map, runtime =
solve_potential_distribution(grid_rows, grid_cols, conductivity_core,
conductivity_outer, current_magnitude, step_x, step_y, reference_node)
    visualize_potential(potential_field, grid_rows, grid_cols, step_x,
step_y, axis_min, axis_max)
    print(f"Execution time for {grid_rows}x{grid_cols}: {runtime:.4f}
seconds")

# Entry point for the program
if __name__ == "__main__":
    execute_simulation(4, 4, "results_small_grid.xlsx", (2, 1), (-2,
```

```
2))
    execute_simulation(100, 100, "results_large_grid.xlsx", (0, 99),
(-50, 50))
```




```
Execution time for 4x4: 0.0005 seconds
```




```
Execution time for 100x100: 26.6773 seconds
```