Plotter.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def plot_phase_data(phase_data_file="phase_data.csv", output_image_file="phase_plot.png"):
    # Load the 2D phase data from the CSV file
    phase_deg_2d = pd.read_csv(phase_data_file, header=None).to_numpy()

    # Plot the phase data
    plt.figure(figsize=(8, 8))
    plt.imshow(phase_deg_2d, cmap='jet', origin='lower')
    plt.colorbar(label="Phase (degrees)")
    plt.title("Phase Distribution")
    plt.xlabel("X (mm)")
    plt.ylabel("Y (mm)")
    plt.savefig(output_image_file)
    plt.close()

if __name__ == "__main__":
    plot_phase_data("phase_data.csv", "phase_plot.png")
```

Generate_hfss_script.py

```
import ScriptEnv
import json


def main():
    ScriptEnv.Initialize("Ansoft.ElectronicsDesktop")
    oDesktop.RestoreWindow()
    oProject = oDesktop.SetActiveProject("meta-surface7")
    oDesign = oProject.SetActiveDesign("HFSSDesign1")
    oEditor = oDesign.SetActiveEditor("3D Modeler")


    # Load pre-calculated patch data
    with open("F:\Work files\University files\Ph.D\Toturials\HFSS_Meta-surface\patch_data.json", "r") as
f:
        patch_data = json.load(f)


    # User-defined fixed dimensions for ground and dielectric
    ground_height = 0.1  # mm
    dielectric_height = 1.0  # mm
    top_patch_height = 0.1  # mm
    dielectric_material = "Rogers RO3203 (tm)"
    uedim = 5  # Example cell spacing in mm


    for i, patch in enumerate(patch_data):
        xi = patch["xi"]
        yi = patch["yi"]
        zi = patch["zi"]
        patch_size = patch["patch_size"]
```

```
# Centering the top patches
xi_center = xi + uedim / 2
yi_center = yi + uedim / 2

# Create ground layer
oEditor.CreateBox(
   [
      "NAME:BoxParameters",
      "XPosition:=", "{}mm".format(xi),
      "YPosition:=", "{}mm".format(yi),
      "ZPosition:=", "{}mm".format(zi-0.0001),
      "XSize:=", "{}mm".format(uedim-0.0001),
      "YSize:=", "{}mm".format(uedim-0.0001),
      "ZSize:=", "{}mm".format(ground_height)
   ],
   [
      "NAME:Attributes",
      "Name:=", "Ground_{}".format(i),
      "MaterialValue:=", "\"pec\"",
      "SolveInside:=", False  # Disable SolveInside for perfect conductors like "pec"
   ]
)

# Create dielectric layer (PCB) on top of the ground
oEditor.CreateBox(
   [
      "NAME:BoxParameters",
      "XPosition:=", "{}mm".format(xi),
      "YPosition:=", "{}mm".format(yi),
```

```
        "ZPosition:=", "{}mm".format(zi + ground_height),
        "XSize:=", "{}mm".format(uedim-0.0001),
        "YSize:=", "{}mm".format(uedim-0.0001),
        "ZSize:=", "{}mm".format(dielectric_height)
    ],
    [
        "NAME:Attributes",
        "Name:=", "Dielectric_{}".format(i),
        "MaterialValue:=", "\"{}\"".format(dielectric_material),
        "SolveInside:=", True,  # Solve inside for dielectric materials
        "Color:=", "(128 128 128)"  # Set the color to gray (R: 128, G: 128, B: 128)
    ]
)

# Create top metallic patch on top of the dielectric layer, centered within the element
oEditor.CreateBox(
    [
        "NAME:BoxParameters",
        "XPosition:=", "{}mm".format(xi_center - patch_size / 2),
        "YPosition:=", "{}mm".format(yi_center - patch_size / 2),
        "ZPosition:=", "{}mm".format(zi + ground_height + dielectric_height),
        "XSize:=", "{}mm".format(patch_size),
        "YSize:=", "{}mm".format(patch_size),
        "ZSize:=", "{}mm".format(top_patch_height)
    ],
    [
        "NAME:Attributes",
        "Name:=", "TopPatch_{}".format(i),
        "MaterialValue:=", "\"pec\"",
```

```
            "SolveInside:=", False  # Disable SolveInside for perfect conductors like "pec"
        ]
    )


if __name__ == "__main__":
    main()
```

Jason_data_creator

```
import numpy as np
import pandas as pd
from scipy.constants import speed_of_light

def load_unit_cell_data(file_path):
    df = pd.read_csv(file_path)  # Assuming CSV file with columns: 'Phase', 'Size'
    phases = df['Phase'].to_numpy()
    sizes = df['Size'].to_numpy()
    return phases, sizes

def calculate_patch_size(target_phase, phases, sizes):
    idx = (np.abs(phases - target_phase)).argmin()  # Find the closest phase match
    return sizes[idx]

def pre_calculate_data(output_file="patch_data.json", phase_data_file="phase_data.csv"):
    # Load reflection data
    phases, sizes = load_unit_cell_data("phase_vs_size.csv")

    # User-defined parameters (these values should be adjusted as needed)
    freq = 30e9  # Frequency in Hz (example value)
    wavelength = (speed_of_light * 1000) / freq
    k = 2 * np.pi / wavelength

    # Example values, should be adjusted based on requirements
    the_dir = 0  # Theta direction in degrees
    phi_dir = 0  # Phi direction in degrees
    pha_zer = -180  # Reference phase
    x_cor, y_cor, z_cor = 0, 0, 55  # Coordinates of the feed (example values)
```

```
rad = 100  # Diameter in mm
uedim = 5  # Cell spacing in mm
OAM_m = 0  # Orbital Angular Momentum mode (example value)
FOD = 1000 #F/D ratio
f = FOD*rad  # Focal length (example value)

# Calculate height parameter
h = (rad ** 2) / (16 * f)

# Set Conformal mode (Convex or Concave)
sign_z = +1  # 1 for Convex, -1 for Concave (example value)

# Generate coordinate grid within the radius
xi_range = np.arange(-rad / 2 + rad % uedim, rad / 2 + uedim, uedim)
yi_range = np.arange(-rad / 2 + rad % uedim, rad / 2 + uedim, uedim)
XI, YI = np.meshgrid(xi_range, yi_range)

# Calculate the angular phase
angular_phase = OAM_m * np.arctan2(YI, XI)

# Initialize 2D array for storing the calculated phase (in degrees)
phase_deg_2d = np.full(XI.shape, np.nan)

patch_data = []

# Loop through the grid and generate patches
for ix, xi in enumerate(xi_range):
    for iy, yi in enumerate(yi_range):
        if np.sqrt(xi ** 2 + yi ** 2) < rad / 2:
```

```
            # Calculate the height
            zi = sign_z * (h - (xi ** 2 + yi ** 2) / (4 * f))

            # Calculate the distance from the feed
            R = np.sqrt((x_cor - xi) ** 2 + (y_cor - yi) ** 2 + (z_cor - zi) ** 2)

            # Calculate the phase shift
            phase = k * (R - zi * np.cos(np.deg2rad(the_dir)) - np.sin(np.deg2rad(the_dir)) * (xi *
np.cos(np.deg2rad(phi_dir)) + yi * np.sin(np.deg2rad(phi_dir))))

            # Calculate the modified phase in degrees
            m_phase = np.mod(angular_phase[iy, ix] + phase, 2 * np.pi)
            m_phase_deg = m_phase * 180 / np.pi + pha_zer

            # Store the calculated phase in the 2D array
            phase_deg_2d[iy, ix] = m_phase_deg

            # Determine the size of the patch based on the calculated phase
            patch_size = calculate_patch_size(m_phase_deg, phases, sizes)

            # Append calculated patch parameters to patch_data list
            patch_data.append({
                "xi": xi,
                "yi": yi,
                "zi": zi,
                "patch_size": patch_size,
                "m_phase_deg": m_phase_deg  # Include m_phase_deg for reference if needed
            })
```

```
    # Save the 2D phase data to a CSV file
    pd.DataFrame(phase_deg_2d).to_csv(phase_data_file, header=False, index=False)

    # Save the patch data to a JSON file
    pd.DataFrame(patch_data).to_json(output_file, orient="records")

if __name__ == "__main__":
    pre_calculate_data("patch_data.json", "phase_data.csv")
```