kriging3D

May 13, 2024

1 Imports

[23]:
```
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import pandas as pd
import pickle
```

2 Kriging 3D Interpolation

2.0.1 Define Functions

[24]:
```
def plot_3D_mesh(t, grid_x, grid_y, grid_z, title: str, color_scale='Viridis'):
    GZ, GY, GX = np.meshgrid(grid_z, grid_y, grid_x, indexing='ij')

    fig = go.Figure(
        data=[go.Volume(
            x=GX.flatten(),
            y=GY.flatten(),
            z=GZ.flatten(),
            value=t.flatten(),
            isomax=t.max(),
            isomin=t.min(),
            opacity=0.5,
            surface_count=25,  # Adjust the number of isosurfaces
            colorscale=color_scale
        )])

    # Update the layout of the plot
    fig.update_layout(
        height=600,
        width=700,
        title=title,
        scene=dict(
            xaxis_title='X',
            yaxis_title='Y',
            zaxis=dict(
```

```
                title='Depth',
                autorange='reversed'  # Automatically reverse the z-axis
            )
        )
    )

    # Show the plot
    fig.show()
```

[25]:
```
def load_data(name: str) -> pd.DataFrame:
    with open(f'data/t_{name}.pkl', 'rb') as f:
        interpolated_data = pickle.load(f)
    with open(f'data/ss3d_{name}.pkl', 'rb') as f:
        varience_data = pickle.load(f)
    return interpolated_data, varience_data
```

[26]:
```
def load_grid() -> np.ndarray:
    with open(f'data/grid.pkl', 'rb') as f:
        grid = pickle.load(f)
    return grid
```

2.0.2 Load Data

[27]:
```
grid = load_grid()
t_poro, ss3d_poro = load_data('poro')
t_hydr, ss3d_hydr = load_data('hydr')
```

2.0.3 Kriging Interpolation of Porosity

[ ]:
```
plot_3D_mesh(t_poro, *grid, "Kriging Interpolation of Porosity", "Plasma")
```

[ ]:
```
plot_3D_mesh(ss3d_poro, *grid, "Kriging Varience of Porosity")
```

2.0.4 Kriging Interpolation of Hydrate Saturation

[ ]:
```
plot_3D_mesh(t_hydr, *grid, "Kriging Interpolation of Hydrate Saturation",␣
 ↪"Plasma")
```

[ ]:
```
plot_3D_mesh(ss3d_hydr, *grid, "Kriging Varience of Hydrate Saturation")
```

3 Resource Distribution

[28]:
```
from loadAndPreprocess import load_and_preprocess
well_info, sensor_data_list = load_and_preprocess()

# Remove the data point with NaN value
well_info = well_info.dropna()
```

```python
for sensor_data in sensor_data_list:
    sensor_data.dropna(inplace=True)

X = well_info['X'].values
Y = well_info['Y'].values
Z = [sensor_data['Depth'] for sensor_data in sensor_data_list]
Z = np.concatenate(Z)

l = (max(X) - min(X)) / len(grid[0])
w = (max(Y) - min(Y)) / len(grid[1])
h = (max(Z) - min(Z)) / len(grid[2])

valid_volume = l * w * h
```

[29]:
```python
def estimate_resource(sensor_data: pd.Series) -> float:
    """Estimate the resource at a given location based on sensor data"""
    # Get the Porosity and the Hydrate saturation
    porosity = sensor_data['Porosity']
    hydrate_saturation = sensor_data['Hydrate Saturation']
    global valid_volume
    factor = 155 # Assume the factor is 155

    # Calculate the resource estimate
    return valid_volume * porosity * hydrate_saturation * factor
```

[30]:
```python
# Calculate the resource estimate for each point in the grid
resource_estimate = np.zeros_like(t_poro)
for i in range(t_poro.shape[0]):
    for j in range(t_poro.shape[1]):
        for k in range(t_poro.shape[2]):
            sensor_data = pd.Series({
                'Porosity': t_poro[i, j, k],
                'Hydrate Saturation': t_hydr[i, j, k]
            })
            resource_estimate[i, j, k] = estimate_resource(sensor_data)
```

[ ]:
```python
# Plot the resource estimate
plot_3D_mesh(resource_estimate, *grid, "Resource Estimate", "Plasma")
```

4 Total Resources

[31]:
```python
# Calculate the sum of the total resource estimate
total_resource_estimate = resource_estimate.sum()
print(f"Total Resource Estimate: {total_resource_estimate:.2f}")
```

```
Total Resource Estimate: 44417499906.32
```