data_postprocessing

May 22, 2025

[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV files
file_4_2 = "Al_Cu_Ni_el_4.2.csv"
file_OK = "Al_Cu_Ni_el_OK.csv"

data_4_2 = pd.read_csv(file_4_2)
data_OK = pd.read_csv(file_OK)

# Plotting different etas with respect to time for Al_Cu_Ni_el_4.2.csv with␣
↪solid lines
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed

etas_4_2 = ['area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3',␣
↪'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']

for eta in etas_4_2:
    plt.plot(data_4_2['time'], data_4_2[eta], label=eta, linestyle='-')

# Plotting different etas with respect to time for Al_Cu_Ni_el_OK.csv with␣
↪dashed lines
etas_OK = ['area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3',␣
↪'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']

for eta in etas_OK:
    plt.plot(data_OK['time'], data_OK[eta], label=eta, linestyle='--')

plt.xlabel('Time')
plt.ylabel('Values')
plt.title('Different Etas with Respect to Time')
plt.legend()

plt.yscale('log')  # Set y-axis to logarithmic scale


plt.show()
```

[ ]:
```
pip install brokenaxes
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from brokenaxes import brokenaxes

# Your existing code to read data and prepare it remains unchanged

# Plotting the lines with axis breaks
bax = brokenaxes(ylims=((0, 60000), (260000, 320000)), d=0.01)

# Plot the lines using brokenaxes
bax.plot(data_4_2['time'], data_4_2['combined_line_1'], label='Combined Line 1␣
↪(4.2)', linestyle='solid')
bax.plot(data_4_2['time'], data_4_2['combined_line_2'], label='Combined Line 2␣
↪(4.2)', linestyle='solid')
bax.plot(data_4_2['time'], data_4_2['area_al_eta6'], label='Area Al Eta6 (4.
↪2)', linestyle='solid')

bax.plot(data_OK['time'], data_OK['combined_line_1'], label='Combined Line 1␣
↪(OK)', linestyle='dashed')
bax.plot(data_OK['time'], data_OK['combined_line_2'], label='Combined Line 2␣
↪(OK)', linestyle='dashed')
bax.plot(data_OK['time'], data_OK['area_al_eta6'], label='Area Al Eta6 (OK)',␣
↪linestyle='dashed')

bax.set_xlabel('Time')
bax.set_ylabel('Area')
bax.set_title('Area vs Time for Different Eta Values')
bax.legend()
plt.show()
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV files
file_4_2 = "Al_Cu_Ni_el_4.2.csv"
file_OK = "Al_Cu_Ni_el_OK.csv"

# Read CSV files into pandas dataframes
data_4_2 = pd.read_csv(file_4_2)
data_OK = pd.read_csv(file_OK)



# Extract the required columns
```

```python
cols_4_2 = ['time', 'area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3', 'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']
cols_OK = ['time', 'area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3', 'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']

data_4_2 = data_4_2[cols_4_2]
data_OK = data_OK[cols_OK]

data_4_2['time'] = data_4_2['time'] / 1e6
data_OK['time'] = data_OK['time'] / 1e6


# Create new columns for combined areas
data_4_2['combined_line_1'] = data_4_2['area_al2cu_eta1'] + data_4_2['area_al2cu_eta2']
data_4_2['combined_line_2'] = data_4_2['area_al3ni_eta3'] + data_4_2['area_al3ni_eta4'] + data_4_2['area_al3ni_eta5']

data_OK['combined_line_1'] = data_OK['area_al2cu_eta1'] + data_OK['area_al2cu_eta2']
data_OK['combined_line_2'] = data_OK['area_al3ni_eta3'] + data_OK['area_al3ni_eta4'] + data_OK['area_al3ni_eta5']

# Plotting the lines
plt.figure(figsize=(2.5, 5))

# Plot lines from Al_Cu_Ni_el_4.2.csv with solid lines
plt.plot(data_4_2['time'], data_4_2['combined_line_1'], label=r'$\mathbf{\phi_1}$', linestyle='solid', color='green', linewidth=2)
plt.plot(data_4_2['time'], data_4_2['combined_line_2'], label=r'$\mathbf{\phi_2}$', linestyle='solid', color='blue', linewidth=2)
plt.plot(data_4_2['time'], data_4_2['area_al_eta6'], label=r'$\mathbf{\phi_3}$', linestyle='solid', color='red', linewidth=2)

# Plot lines from Al_Cu_Ni_el_OK.csv with dashed lines
plt.plot(data_OK['time'], data_OK['combined_line_1'], label='', linestyle='dashed', color='green', linewidth=2)
plt.plot(data_OK['time'], data_OK['combined_line_2'], label='', linestyle='dashed', color='blue', linewidth=2)
plt.plot(data_OK['time'], data_OK['area_al_eta6'], label='', linestyle='dashed', color='red', linewidth=2)


# plt.grid(True)
plt.yscale('log')  # Set y-axis to logarithmic scale
```

```
# Adjusting axis tick labels font size and weight
plt.xlabel('Time (ms)', fontsize=18, fontweight='bold')
plt.ylabel(r'Area (nm$\mathbf{^2}$)', fontsize=18, fontweight='bold')
plt.title('Area vs Time', fontsize=14, fontweight='bold')
plt.legend(prop={'size': 14, 'weight': 'bold'}, bbox_to_anchor=(0.4, 0.9),␣
↪loc='upper left')


plt.tick_params(axis='both', which='major', labelsize=10, width=2, length=10,␣
↪direction='in', pad=5)
plt.tick_params(axis='both', which='minor', labelsize=10, width=1.2, length=6,␣
↪direction='in', pad=5)


# Adjust font size and weight for x and y axis ticks
plt.xticks(fontsize=16, fontweight='bold')
plt.yticks(fontsize=16, fontweight='bold')

plt.xlim(3)  # Start y-axis from 5,000,000
plt.ylim(1e4)  # Start y-axis from 5,000,000


plt.show()
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV files
file_4_2 = "Al_Cu_Ni_el_4.2.csv"
file_OK = "Al_Cu_Ni_el_OK.csv"

data_4_2 = pd.read_csv(file_4_2)
data_OK = pd.read_csv(file_OK)

data_4_2['time'] = data_4_2['time'] / 1e6
data_OK['time'] = data_OK['time'] / 1e6




# Plotting different etas with respect to time for Al_Cu_Ni_el_4.2.csv with␣
↪solid lines
plt.figure(figsize=(4, 8))  # Adjust the figure size as needed

# etas_4_2 = ['area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3',␣
↪'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']
etas_4_2 = ['area_al2cu_eta1', 'area_al3ni_eta3', 'area_al3ni_eta4',␣
↪'area_al3ni_eta5', 'area_al_eta6']
```

```python
for eta in etas_4_2:
    plt.plot(data_4_2['time'], data_4_2[eta], label=eta, linestyle='-')

# Plotting different etas with respect to time for Al_Cu_Ni_el_OK.csv with␣
↪dashed lines
etas_OK = ['area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3',␣
↪'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']

for eta in etas_OK:
    plt.plot(data_OK['time'], data_OK[eta], label=eta, linestyle='--')

# plt.grid(True)
plt.yscale('log')  # Set y-axis to logarithmic scale

# Adjusting axis tick labels font size and weight
plt.xlabel('Time (ms)', fontsize=24, fontweight='bold')
plt.ylabel(r'Area (nm$\mathbf{^2}$)', fontsize=24, fontweight='bold')
plt.title('Area vs Time', fontsize=24, fontweight='bold')
plt.legend(prop={'size': 24, 'weight': 'bold'}, bbox_to_anchor=(1.01, 0.9),␣
↪loc='upper left')


plt.tick_params(axis='both', which='major', labelsize=10, width=2, length=8,␣
↪direction='in', pad=5)
plt.tick_params(axis='y', which='major', labelsize=10, width=3, length=14,␣
↪direction='in', pad=5)

plt.tick_params(axis='both', which='minor', labelsize=10, width=1.6, length=6,␣
↪direction='in', pad=5)


# Adjust font size and weight for x and y axis ticks
plt.xticks(fontsize=20, fontweight='bold')
plt.yticks(fontsize=20, fontweight='bold')

plt.xlim(3)  # Start y-axis from 5,000,000
# plt.ylim(1e4)  # Start y-axis from 5,000,000


plt.show()
```

[ ]:

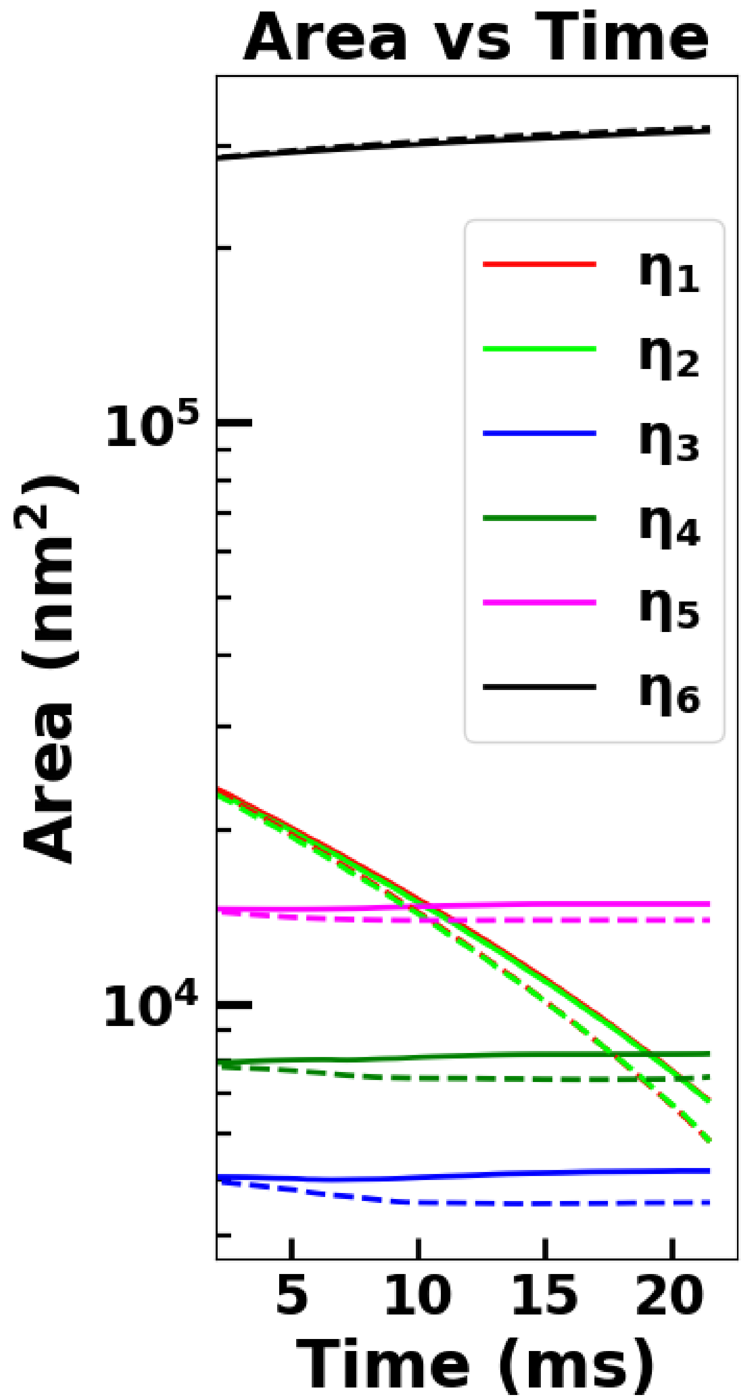
```python
import pandas as pd
import matplotlib.pyplot as plt
from brokenaxes import brokenaxes
```

```
# Read the CSV files (assumed you already have these steps)

# Your existing code to read data and prepare it remains unchanged

# Plotting the lines with axis break
fig = plt.figure(figsize=(3, 6))

bax = brokenaxes(ylims=((100, 3e4), (2.7e5, 3.3e5)))  # Set the limits for axis␣
↪break

# Plotting the lines using brokenaxes
etas_4_2 = ['area_al2cu_eta1', 'area_al3ni_eta3', 'area_al3ni_eta4',␣
↪'area_al3ni_eta5', 'area_al_eta6']
color=['r', 'b', 'g', 'k', 'magenta', 'lime']
for eta in etas_4_2:
    bax.plot(data_4_2['time'], data_4_2[eta], label=eta, linestyle='-',␣
↪color=color[1])

etas_OK = ['area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3',␣
↪'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']
for eta in etas_OK:
    bax.plot(data_OK['time'], data_OK[eta], label=eta, linestyle='--')

bax.set_xlabel('Time (ms)', fontsize=18, fontweight='bold')
bax.set_ylabel(r'Area (nm$\mathbf{^2}$)', fontsize=18, fontweight='bold')
bax.set_title('Area vs Time', fontsize=14, fontweight='bold')
bax.legend(prop={'size': 14, 'weight': 'bold'})

plt.tick_params(axis='both', which='major', labelsize=10, width=2, length=10,␣
↪direction='in', pad=5)
plt.tick_params(axis='both', which='minor', labelsize=10, width=1.6, length=6,␣
↪direction='in', pad=5)

# plt.yscale('log')  # Set y-axis to logarithmic scale


plt.show()
```

[13]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV files
file_4_2 = "Al_Cu_Ni_el_4.2.csv"
file_OK = "Al_Cu_Ni_el_OK.csv"
```

```
data_4_2 = pd.read_csv(file_4_2)
data_OK = pd.read_csv(file_OK)

data_4_2['time'] = data_4_2['time'] / 1e6
data_OK['time'] = data_OK['time'] / 1e6

# Define the colors
line_colors = ['r',   'lime', 'b',  'g', 'magenta', 'k']

# Plotting different etas with respect to time for Al_Cu_Ni_el_4.2.csv with␣
↪solid lines
plt.figure(figsize=(3.5, 8))  # Adjust the figure size as needed

etas_4_2 = ['area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3',␣
↪'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']

label_eta = ['$\mathbf{\u03b7_1}$',  '$\mathbf{\u03b7_2}$',␣
↪'$\mathbf{\u03b7_3}$', '$\mathbf{\u03b7_4}$',
             '$\mathbf{\u03b7_5}$', '$\mathbf{\u03b7_6}$']


for i, eta in enumerate(etas_4_2):
    plt.plot(data_4_2['time'], data_4_2[eta], label=label_eta[i],␣
↪linestyle='-', color=line_colors[i], linewidth=2)

# Plotting different etas with respect to time for Al_Cu_Ni_el_OK.csv with␣
↪dashed lines
etas_OK = ['area_al2cu_eta1', 'area_al2cu_eta2', 'area_al3ni_eta3',␣
↪'area_al3ni_eta4', 'area_al3ni_eta5', 'area_al_eta6']

for i, eta in enumerate(etas_OK):
    plt.plot(data_OK['time'], data_OK[eta], linestyle='--',␣
↪color=line_colors[i], linewidth=2)

# Additional plotting settings

# ... (Other settings such as scale, labels, etc.)

# plt.grid(True)
plt.yscale('log')  # Set y-axis to logarithmic scale

# Adjusting axis tick labels font size and weight
plt.xlabel('Time (ms)', fontsize=24, fontweight='bold')
plt.ylabel(r'Area (nm$\mathbf{^2}$)', fontsize=24, fontweight='bold')
plt.title('Area vs Time', fontsize=24, fontweight='bold')
```

```python
plt.legend(prop={'size': 20, 'weight': 'bold'}, bbox_to_anchor=(0.425, 0.9),␣
 ↪loc='upper left')


plt.tick_params(axis='both', which='major', labelsize=10, width=2, length=8,␣
 ↪direction='in', pad=5)
plt.tick_params(axis='y', which='major', labelsize=10, width=3, length=14,␣
 ↪direction='in', pad=5)

plt.tick_params(axis='both', which='minor', labelsize=10, width=1.6, length=6,␣
 ↪direction='in', pad=5)


# Adjust font size and weight for x and y axis ticks
plt.xticks(fontsize=20, fontweight='bold')
plt.yticks(fontsize=20, fontweight='bold')

plt.xlim(2)  # Start y-axis from 5,000,000
# plt.ylim(1e4)  # Start y-axis from 5,000,000

plt.savefig('eta_temp_comparison.png', bbox_inches='tight', dpi=1200)

plt.show()
```



Area vs Time
Area (nm²)
Time (ms)
10⁵
10⁴
5
10
15
20
η₁
η₂
η₃
η₄
η₅
η₆

1 Eigenstrain

[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Load and plot data for each file
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
↪your directory structure
    df = pd.read_csv(file_path)

    # Assuming 'time' is a column in the CSV files
    plt.figure(figsize=(8, 6))

    # Plotting area_al2cu_eta1 and area_al2cu_eta2 against time
    plt.plot(df['time'], df['area_al2cu_eta1'], label='area_al2cu_eta1')
    plt.plot(df['time'], df['area_al2cu_eta2'], label='area_al2cu_eta2')

    # Plotting area_al3ni_eta3, area_al3ni_eta4, and area_al3ni_eta5 against␣
↪time
    plt.plot(df['time'], df['area_al3ni_eta3'], label='area_al3ni_eta3')
    plt.plot(df['time'], df['area_al3ni_eta4'], label='area_al3ni_eta4')
    plt.plot(df['time'], df['area_al3ni_eta5'], label='area_al3ni_eta5')

    # Plotting area_al_eta6 against time
    plt.plot(df['time'], df['area_al_eta6'], label='area_al_eta6')

    plt.title(f'Values with Time for {file_name}')
    plt.xlabel('Time')
    plt.legend()
    plt.show()
```

[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
```

```
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Load and plot data for each file - area_al2cu_eta1 and area_al2cu_eta2
plt.figure(figsize=(8, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
 ↪your directory structure
    df = pd.read_csv(file_path)
    plt.plot(df['area_al2cu_eta1']+df['area_al2cu_eta2'], label=f'{file_name} -␣
 ↪area_al2cu_eta1')
#     plt.plot(df['area_al2cu_eta2'], label=f'{file_name} - area_al2cu_eta2')

plt.title('Comparison of area_al2cu_eta1 and area_al2cu_eta2')
plt.xlabel('Index')
plt.ylabel('Values')
plt.legend()
plt.show()

# Load and plot data for each file - area_al3ni_eta3, area_al3ni_eta4, and␣
 ↪area_al3ni_eta5
plt.figure(figsize=(8, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
 ↪your directory structure
    df = pd.read_csv(file_path)
    plt.plot(df['area_al3ni_eta3'], label=f'{file_name} - area_al3ni_eta3')
    plt.plot(df['area_al3ni_eta4'], label=f'{file_name} - area_al3ni_eta4')
    plt.plot(df['area_al3ni_eta5'], label=f'{file_name} - area_al3ni_eta5')

plt.title('Comparison of area_al3ni_eta3, area_al3ni_eta4, and area_al3ni_eta5')
plt.xlabel('Index')
plt.ylabel('Values')
plt.legend()
plt.show()

# Load and plot data for each file - area_al_eta6
plt.figure(figsize=(8, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
 ↪your directory structure
    df = pd.read_csv(file_path)
    plt.plot(df['area_al_eta6'], label=f'{file_name} - area_al_eta6')

plt.title('Comparison of area_al_eta6')
plt.xlabel('Index')
```

```
plt.ylabel('Values')
plt.legend()
plt.show()
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Load and plot data for each file - area_al2cu_eta1 and area_al2cu_eta2
plt.figure(figsize=(8, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
 ↪your directory structure
    df = pd.read_csv(file_path)
    plt.plot(df['time'],df['area_al2cu_eta1']+df['area_al2cu_eta2'],␣
 ↪label=f'{file_name} - area_al2cu_eta1')
#     plt.plot(df['area_al2cu_eta2'], label=f'{file_name} - area_al2cu_eta2')

plt.title('Comparison of area_al2cu_eta1 and area_al2cu_eta2')
plt.xlabel('Index')
plt.ylabel('Values')
plt.legend()
plt.show()

# Load and plot data for each file - area_al3ni_eta3, area_al3ni_eta4, and␣
 ↪area_al3ni_eta5
plt.figure(figsize=(8, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
 ↪your directory structure
    df = pd.read_csv(file_path)
    plt.plot(df['area_al3ni_eta3']+df['area_al3ni_eta4']+df['area_al3ni_eta5'],␣
 ↪label=f'{file_name} - area_al3ni_eta3')
#     plt.plot(df['area_al3ni_eta4'], label=f'{file_name} - area_al3ni_eta4')
#     plt.plot(df['area_al3ni_eta5'], label=f'{file_name} - area_al3ni_eta5')

plt.title('Comparison of area_al3ni_eta3, area_al3ni_eta4, and area_al3ni_eta5')
plt.xlabel('Index')
plt.ylabel('Values')
plt.legend()
```

```
plt.show()

# Load and plot data for each file - area_al_eta6
plt.figure(figsize=(8, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
 ↪your directory structure
    df = pd.read_csv(file_path)
    plt.plot(df['area_al_eta6'], label=f'{file_name} - area_al_eta6')

plt.title('Comparison of area_al_eta6')
plt.xlabel('Index')
plt.ylabel('Values')
plt.legend()
plt.show()
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Load and plot data for each file - area_al2cu_eta1 and area_al2cu_eta2
plt.figure(figsize=(8, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
 ↪your directory structure
    df = pd.read_csv(file_path)
    df['time'] = df['time'] / 1e6  # Divide time by 1e6
    plt.plot(df['time'], df['area_al2cu_eta1'] + df['area_al2cu_eta2'],␣
 ↪label=f'{file_name} - area_al2cu_eta1 + area_al2cu_eta2')

plt.title('Comparison of area_al2cu_eta1 and area_al2cu_eta2')
plt.xlabel('Time (ms)')  # Adjust x-axis label
plt.ylabel('Values')
plt.legend()
plt.show()

# Load and plot data for each file - area_al3ni_eta3, area_al3ni_eta4, and␣
 ↪area_al3ni_eta5
plt.figure(figsize=(8, 6))
for file_name in file_names:
```

```
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
↪your directory structure
    df = pd.read_csv(file_path)
    df['time'] = df['time'] / 1e6  # Divide time by 1e6
    plt.plot(df['time'], df['area_al3ni_eta3'] + df['area_al3ni_eta4'] +␣
↪df['area_al3ni_eta5'], label=f'{file_name} - area_al3ni_eta3 +␣
↪area_al3ni_eta4 + area_al3ni_eta5')

plt.title('Comparison of area_al3ni_eta3, area_al3ni_eta4, and area_al3ni_eta5')
plt.xlabel('Time (ms)')  # Adjust x-axis label
plt.ylabel('Values')
plt.legend()
plt.show()

# Load and plot data for each file - area_al_eta6
plt.figure(figsize=(8, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
↪your directory structure
    df = pd.read_csv(file_path)
    df['time'] = df['time'] / 1e6  # Divide time by 1e6
    plt.plot(df['time'], df['area_al_eta6'], label=f'{file_name} -␣
↪area_al_eta6')

plt.title('Comparison of area_al_eta6')
plt.xlabel('Time (ms)')  # Adjust x-axis label
plt.ylabel('Values')
plt.legend()
plt.show()
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Apply cutoff and load data for each file - area_al2cu_eta1 and area_al2cu_eta2
plt.figure(figsize=(4, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to␣
↪your directory structure
    df = pd.read_csv(file_path)
```

```
    df['time'] = df['time'] / 1e6  # Divide time by 1e6
    df_cutoff = df[df['time'] <= 21.5]  # Apply cutoff for time
    plt.plot(df_cutoff['time'], df_cutoff['area_al2cu_eta1'] + df_cutoff['area_al2cu_eta2'], label=f'{file_name} - area_al2cu_eta1 + area_al2cu_eta2')

plt.title('Comparison of area_al2cu_eta1 and area_al2cu_eta2')
plt.xlabel('Time (ms)')  # Adjust x-axis label
plt.ylabel('Values')
plt.legend()
plt.xlim(left=2)  # Set cutoff for x-axis
plt.show()

# Apply cutoff and load data for each file - area_al3ni_eta3, area_al3ni_eta4, and area_al3ni_eta5
plt.figure(figsize=(4, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to your directory structure
    df = pd.read_csv(file_path)
    df['time'] = df['time'] / 1e6  # Divide time by 1e6
    df_cutoff = df[df['time'] <= 21.5]  # Apply cutoff for time
    plt.plot(df_cutoff['time'], df_cutoff['area_al3ni_eta3'] + df_cutoff['area_al3ni_eta4'] + df_cutoff['area_al3ni_eta5'], label=f'{file_name} - area_al3ni_eta3 + area_al3ni_eta4 + area_al3ni_eta5')

plt.title('Comparison of area_al3ni_eta3, area_al3ni_eta4, and area_al3ni_eta5')
plt.xlabel('Time (ms)')  # Adjust x-axis label
plt.ylabel('Values')
plt.legend()
plt.xlim(left=2)  # Set cutoff for x-axis
plt.show()

# Apply cutoff and load data for each file - area_al_eta6
plt.figure(figsize=(4, 6))
for file_name in file_names:
    file_path = f'eigenstrain/{file_name}'  # Update this path according to your directory structure
    df = pd.read_csv(file_path)
    df['time'] = df['time'] / 1e6  # Divide time by 1e6
    df_cutoff = df[df['time'] <= 21.5]  # Apply cutoff for time
    plt.plot(df_cutoff['time'], df_cutoff['area_al_eta6'], label=f'{file_name} - area_al_eta6')

plt.title('Comparison of area_al_eta6')
plt.xlabel('Time (ms)')  # Adjust x-axis label
```

```
plt.ylabel('Values')
plt.legend()

plt.xlim(left=2)  # Set cutoff for x-axis
plt.show()
```

[14]:

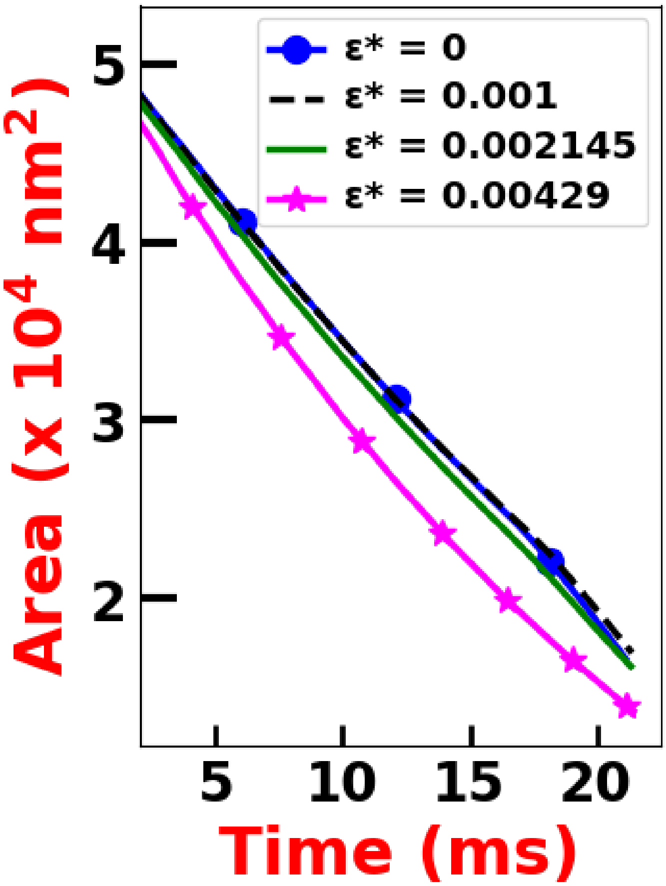
```
import pandas as pd
import matplotlib.pyplot as plt

# File names and their associated 'e' values
file_info = {
    'Al_Cu_Ni_el_0.0.csv': 0,
    'Al_Cu_Ni_el_1.0.csv': 0.001,
    'Al_Cu_Ni_el_2.145.csv': 0.002145,
    'Al_Cu_Ni_el_4.2.csv': 0.00429
}

# Plot for area_al2cu_eta1 and area_al2cu_eta2
plt.figure(figsize=(3.5, 5))
lines_area_al2cu = []
for idx, (file_name, e_value) in enumerate(file_info.items()):
    df = pd.read_csv(f'eigenstrain/{file_name}')
    df['time'] = df['time'] / 1e6
    df_cutoff = df[df['time'] <= 21.5]

    # Define marker and linestyle based on index
    if idx == 0:
        marker = 'o'
        linestyle = '-'
        color='b'
    elif idx == 1:
        marker = None
        linestyle = '--'
        color='k'
    elif idx == 2:
        marker = None
        linestyle = '-'
        color='g'
    else:
        marker = '*'
        linestyle = '-'
        color='magenta'

    line, = plt.plot(df_cutoff['time'], df_cutoff['area_al2cu_eta1']/1e4 +␣
↪df_cutoff['area_al2cu_eta2']/1e4,
                     linewidth=2.5, marker=marker,markevery=10,markersize=10,␣
↪linestyle=linestyle, color=color)
```

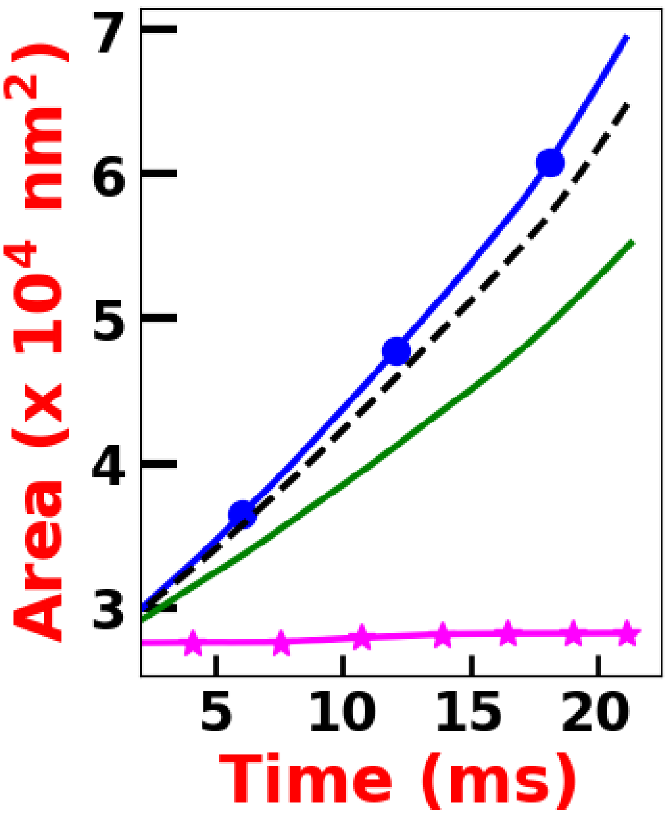
```
    lines_area_al2cu.append(line)

# Legend formatting with epsilon symbol as text
legend = plt.legend(lines_area_al2cu, [f"\u03B5* = {e_value}" for e_value in␣
 ↪file_info.values()])
for text in legend.get_texts():
    text.set_fontsize(14)  # Set font size
    text.set_fontweight('bold')  # Set font weight
# plt.title('1 2')
plt.xlabel('Time (ms)', fontsize=24, fontweight='bold', color='r')
plt.ylabel(r'Area (x $\mathbf{10^4}$ nm$\mathbf{^2}$)', fontsize=24,␣
 ↪fontweight='bold', color='r')
plt.xlim(left=2)  # Set cutoff for x-axis

# Adjusting y-tick labels to display only 2, 3, 4, 5
plt.yticks([2, 3, 4, 5], fontsize=20, fontweight='bold')

plt.tick_params(axis='both', which='major', labelsize=10, width=2, length=8,␣
 ↪direction='in', pad=5)
plt.tick_params(axis='y', which='major', labelsize=10, width=3, length=14,␣
 ↪direction='in', pad=5)
plt.tick_params(axis='both', which='minor', labelsize=10, width=1.6, length=6,␣
 ↪direction='in', pad=5)

# Adjust font size and weight for x and y axis ticks
plt.xticks(fontsize=20, fontweight='bold')
plt.yticks(fontsize=20, fontweight='bold')
plt.show()

###################################
# Plot for area_al3ni_eta3, area_al3ni_eta4, and area_al3ni_eta5
plt.figure(figsize=(3.5, 4.5))
lines_area_al3ni = []

for idx, (file_name, e_value) in enumerate(file_info.items()):
    df = pd.read_csv(f'eigenstrain/{file_name}')
    df['time'] = df['time'] / 1e6
    df_cutoff = df[df['time'] <= 21.5]

    # Define marker and linestyle based on index
    if idx == 0:
        marker = 'o'
        linestyle = '-'
        color='b'
    elif idx == 1:
        marker = None
        linestyle = '--'
```

```
        color='k'
    elif idx == 2:
        marker = None
        linestyle = '-'
        color='g'
    else:
        marker = '*'
        linestyle = '-'
        color='magenta'

    line, = plt.plot(df_cutoff['time'], df_cutoff['area_al3ni_eta3']/1e4 +
                     df_cutoff['area_al3ni_eta4']/1e4 +␣
↪df_cutoff['area_al3ni_eta5']/1e4,
                     linewidth=2.5, marker=marker,markevery=10,markersize=10,␣
↪linestyle=linestyle, color=color)
    lines_area_al3ni.append(line)

# legend = plt.legend(lines_area_al3ni, [f"$\epsilon$ = {e_value}" for e_value␣
↪in file_info.values()])
# for text in legend.get_texts():
#     text.set_fontsize(12)  # Set font size
#     text.set_fontweight('bold')  # Set font weight


# plt.title('3 4 5')
plt.xlim(left=2)  # Set cutoff for x-axis

# Adjusting axis tick labels font size and weight
plt.xlabel('Time (ms)', fontsize=24, fontweight='bold', color='r')
plt.ylabel(r'Area (x $\mathbf{10^4}$ nm$\mathbf{^2}$)', fontsize=24,␣
↪fontweight='bold', color='r')
plt.title(' ', fontsize=24, fontweight='bold')

plt.tick_params(axis='both', which='major', labelsize=10, width=2, length=8,␣
↪direction='in', pad=5)
plt.tick_params(axis='y', which='major', labelsize=10, width=3, length=14,␣
↪direction='in', pad=5)
plt.tick_params(axis='both', which='minor', labelsize=10, width=1.6, length=6,␣
↪direction='in', pad=5)

# Adjust font size and weight for x and y axis ticks
plt.xticks(fontsize=20, fontweight='bold')
plt.yticks(fontsize=20, fontweight='bold')
plt.show()
##########################################################
# Plot for area_al_eta6
```

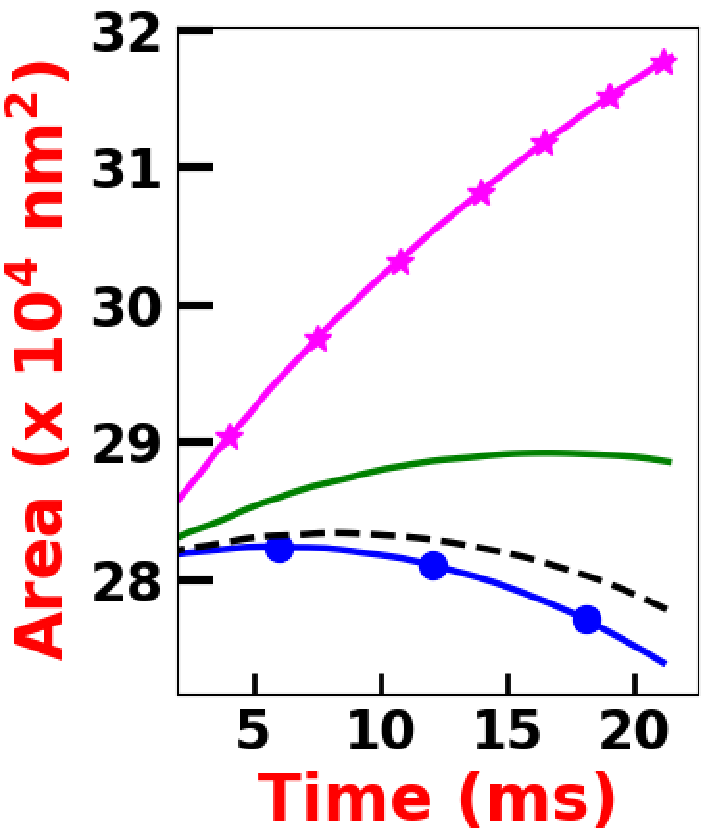
```
plt.figure(figsize=(3.5, 4.5))
lines_area_al_eta6 = []
for idx, (file_name, e_value) in enumerate(file_info.items()):
    df = pd.read_csv(f'eigenstrain/{file_name}')
    df['time'] = df['time'] / 1e6
    df_cutoff = df[df['time'] <= 21.5]

    # Define marker and linestyle based on index
    if idx == 0:
        marker = 'o'
        linestyle = '-'
        color='b'
    elif idx == 1:
        marker = None
        linestyle = '--'
        color='k'
    elif idx == 2:
        marker = None
        linestyle = '-'
        color='g'
    else:
        marker = '*'
        linestyle = '-'
        color='magenta'
    line, = plt.plot(df_cutoff['time'], df_cutoff['area_al_eta6']/1e4,␣
 ↪linewidth=2.5, marker=marker,markevery=10,markersize=10,␣
 ↪linestyle=linestyle, color=color)
    lines_area_al_eta6.append(line)


plt.xlim(left=2)  # Set cutoff for x-axis

# Adjusting axis tick labels font size and weight
plt.xlabel('Time (ms)', fontsize=24, fontweight='bold', color='r')
plt.ylabel(r'Area (x $\mathbf{10^4}$ nm$\mathbf{^2}$)', fontsize=24,␣
 ↪fontweight='bold', color='r')
# plt.title(' ', fontsize=24, fontweight='bold')
# plt.legend(prop={'size': 12, 'weight': 'bold'})

plt.tick_params(axis='both', which='major', labelsize=10, width=2, length=8,␣
 ↪direction='in', pad=5)
plt.tick_params(axis='y', which='major', labelsize=10, width=3, length=14,␣
 ↪direction='in', pad=5)

plt.tick_params(axis='both', which='minor', labelsize=10, width=1.6, length=6,␣
 ↪direction='in', pad=5)
```

```
# Adjust font size and weight for x and y axis ticks
plt.xticks(fontsize=20, fontweight='bold')
plt.yticks(fontsize=20, fontweight='bold')


plt.show()
```





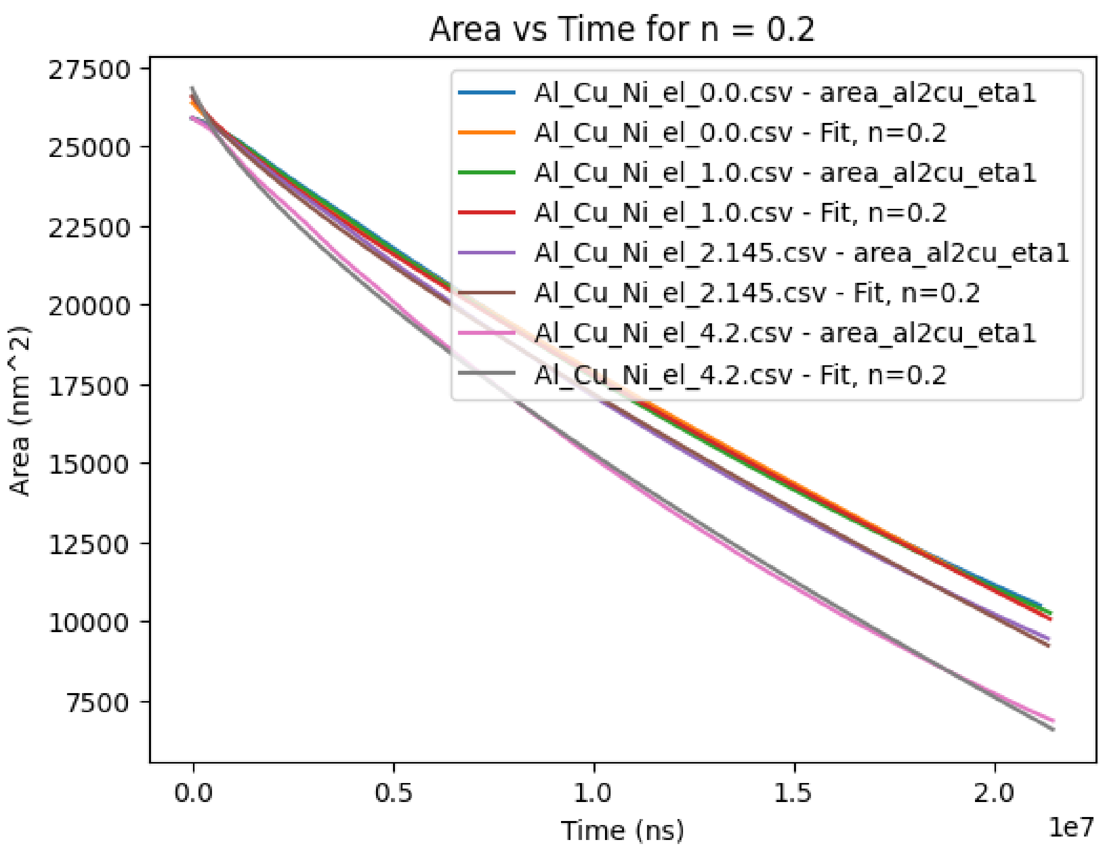
Area (x 10^4 nm^2)
7
6
5
4
3
5
10
15
20
Time (ms)



2 Curve fitting

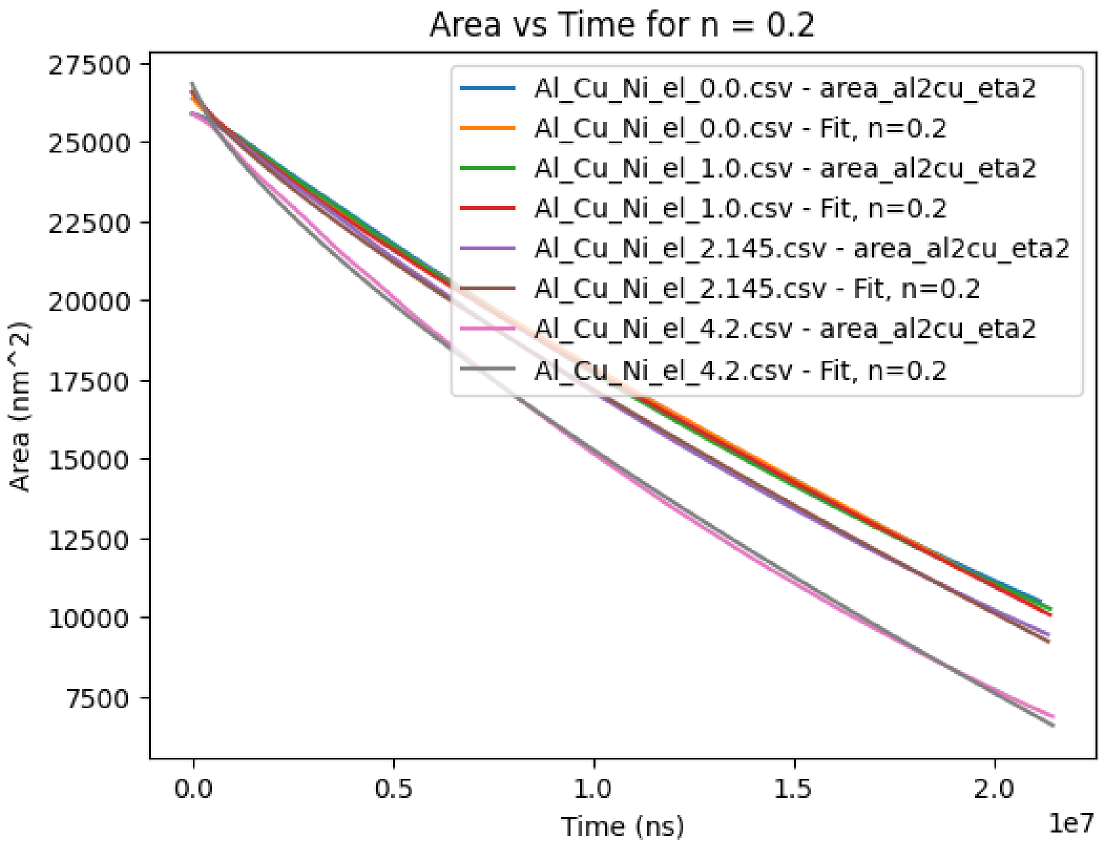
[59]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]
```

```
# Function to read area data from CSV files and plot area vs time for a␣
↪specific n value
def plot_area_vs_time_for_n(file_names, n_value):
    fig, ax = plt.subplots()

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and y values
        t_values = data['time']
        y_values = data['area_al2cu_eta1']  # Using 'area_al2cu_eta1' for␣
↪fitting

        # Curve fitting for n = n_value
#         popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
↪p0=(0, n_value, 1.0))
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
↪p0=(0, n_value, 1.0), maxfev=10000)

        y_fit = func(t_values, *popt)

        # Plotting area vs time
        ax.plot(t_values, y_values, label=f'{file_name} - area_al2cu_eta1')
        ax.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.1f}')

    ax.legend()
    ax.set_title(f'Area vs Time for n = {n_value}')
    ax.set_xlabel('Time (ns)')
    ax.set_ylabel('Area (nm^2)')
    plt.show()

# Perform plot for 'area_al2cu_eta1' against time for n = 0.8
plot_area_vs_time_for_n(file_names, 0.2)
```



[60]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Function to read area data from CSV files and plot area vs time for a
↪specific n value
def plot_area_vs_time_for_n(file_names, n_value):
```

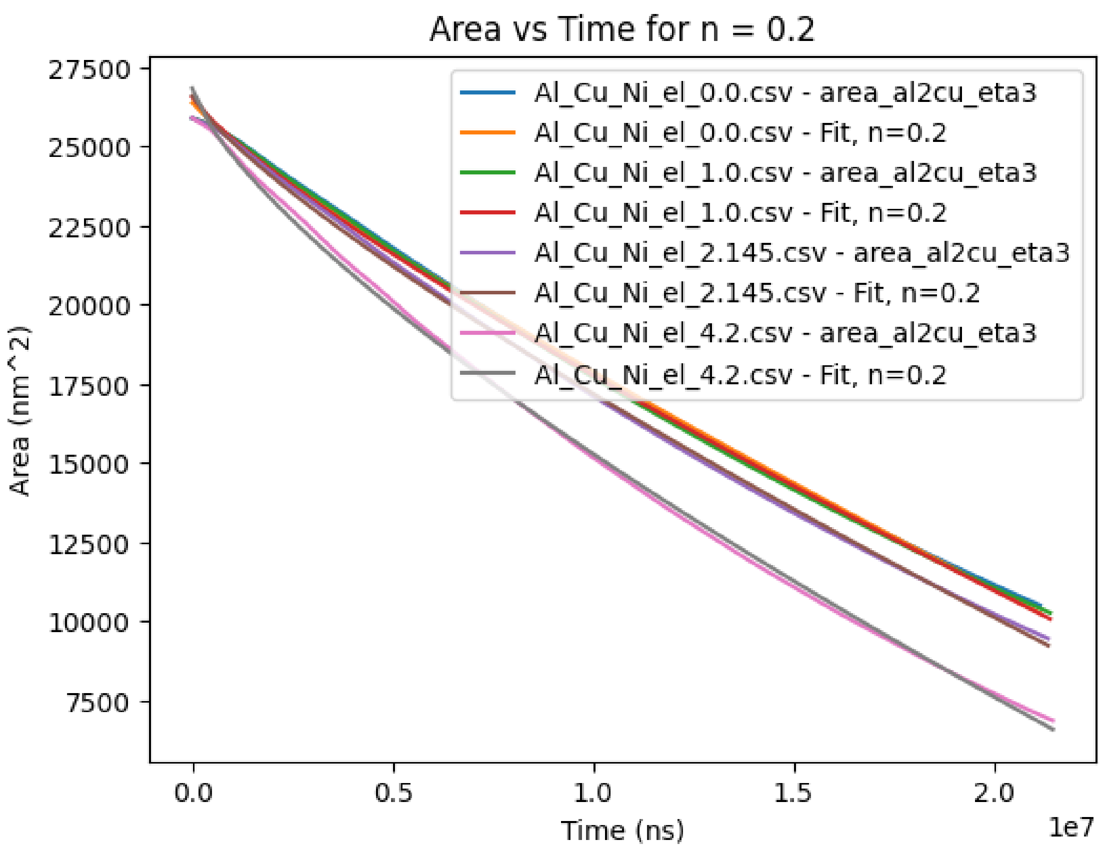
```
    fig, ax = plt.subplots()

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
 ↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and y values
        t_values = data['time']
        y_values = data['area_al2cu_eta1']  # Using 'area_al2cu_eta1' for␣
 ↪fitting

        # Curve fitting for n = n_value
#         popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0))
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0), maxfev=10000)

        y_fit = func(t_values, *popt)

        # Plotting area vs time
        ax.plot(t_values, y_values, label=f'{file_name} - area_al2cu_eta2')
        ax.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.1f}')

    ax.legend()
    ax.set_title(f'Area vs Time for n = {n_value}')
    ax.set_xlabel('Time (ns)')
    ax.set_ylabel('Area (nm^2)')
    plt.show()

# Perform plot for 'area_al2cu_eta1' against time for n = 0.8
plot_area_vs_time_for_n(file_names, 0.2)
```



[61]:

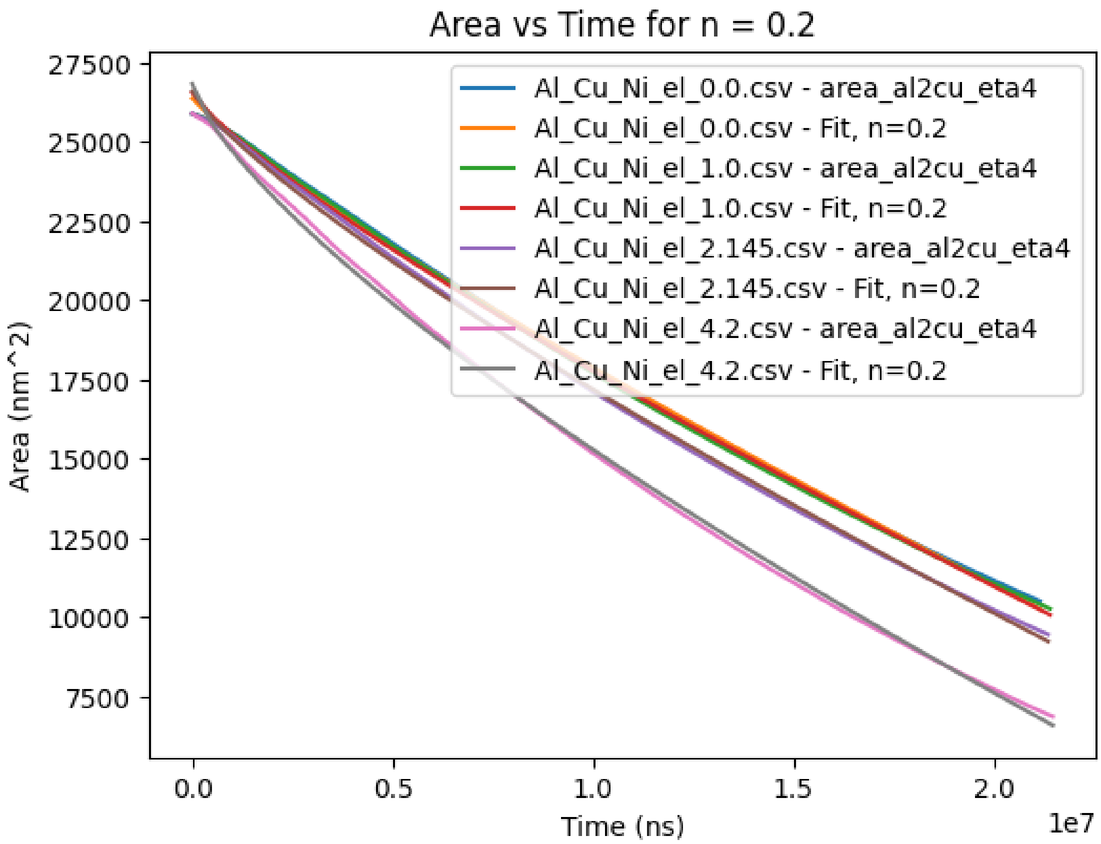
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Function to read area data from CSV files and plot area vs time for a␣
↪specific n value
def plot_area_vs_time_for_n(file_names, n_value):
```

```
    fig, ax = plt.subplots()

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
 ↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and y values
        t_values = data['time']
        y_values = data['area_al2cu_eta1']  # Using 'area_al2cu_eta1' for␣
 ↪fitting

        # Curve fitting for n = n_value
#         popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0))
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0), maxfev=10000)

        y_fit = func(t_values, *popt)

        # Plotting area vs time
        ax.plot(t_values, y_values, label=f'{file_name} - area_al2cu_eta3')
        ax.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.1f}')

    ax.legend()
    ax.set_title(f'Area vs Time for n = {n_value}')
    ax.set_xlabel('Time (ns)')
    ax.set_ylabel('Area (nm^2)')
    plt.show()

# Perform plot for 'area_al2cu_eta1' against time for n = 0.8
plot_area_vs_time_for_n(file_names, 0.2)
```



[62]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Function to read area data from CSV files and plot area vs time for a␣
 ↪specific n value
def plot_area_vs_time_for_n(file_names, n_value):
```

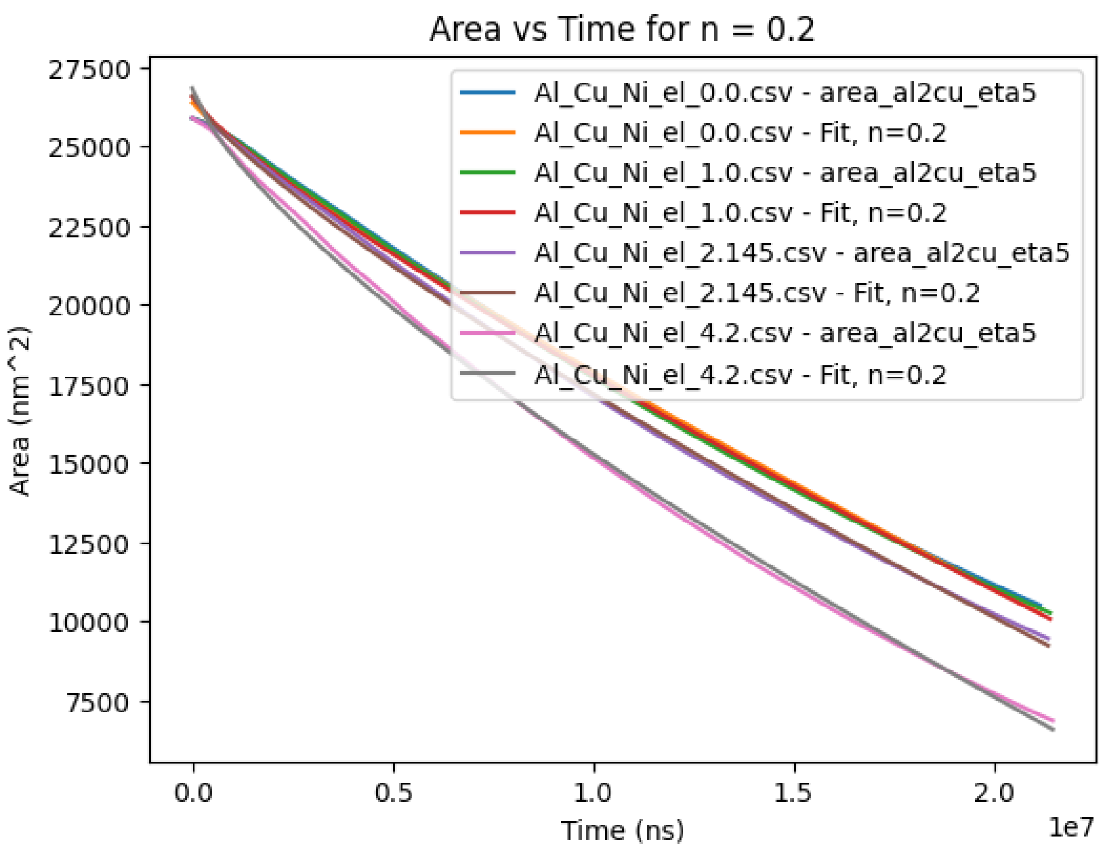
```python
    fig, ax = plt.subplots()

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
 ↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and y values
        t_values = data['time']
        y_values = data['area_al2cu_eta1']  # Using 'area_al2cu_eta1' for␣
 ↪fitting

        # Curve fitting for n = n_value
#         popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0))
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0), maxfev=10000)

        y_fit = func(t_values, *popt)

        # Plotting area vs time
        ax.plot(t_values, y_values, label=f'{file_name} - area_al2cu_eta4')
        ax.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.1f}')

    ax.legend()
    ax.set_title(f'Area vs Time for n = {n_value}')
    ax.set_xlabel('Time (ns)')
    ax.set_ylabel('Area (nm^2)')
    plt.show()

# Perform plot for 'area_al2cu_eta1' against time for n = 0.8
plot_area_vs_time_for_n(file_names, 0.2)
```



[63]:

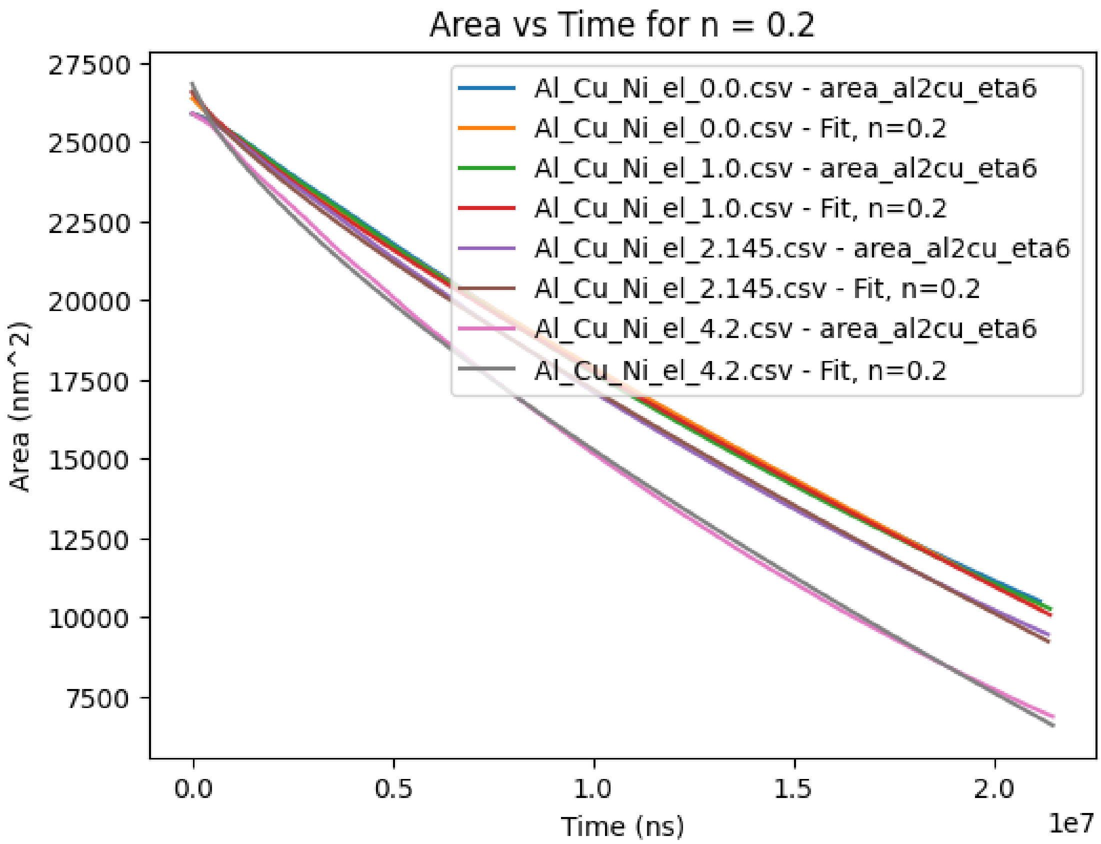
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Function to read area data from CSV files and plot area vs time for a␣
↪specific n value
def plot_area_vs_time_for_n(file_names, n_value):
```

```
    fig, ax = plt.subplots()

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
 ↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and y values
        t_values = data['time']
        y_values = data['area_al2cu_eta1']  # Using 'area_al2cu_eta1' for␣
 ↪fitting

        # Curve fitting for n = n_value
#         popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0))
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0), maxfev=10000)

        y_fit = func(t_values, *popt)

        # Plotting area vs time
        ax.plot(t_values, y_values, label=f'{file_name} - area_al2cu_eta5')
        ax.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.1f}')

    ax.legend()
    ax.set_title(f'Area vs Time for n = {n_value}')
    ax.set_xlabel('Time (ns)')
    ax.set_ylabel('Area (nm^2)')
    plt.show()

# Perform plot for 'area_al2cu_eta1' against time for n = 0.8
plot_area_vs_time_for_n(file_names, 0.2)
```



[64]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Function to read area data from CSV files and plot area vs time for a␣
 ↪specific n value
def plot_area_vs_time_for_n(file_names, n_value):
```

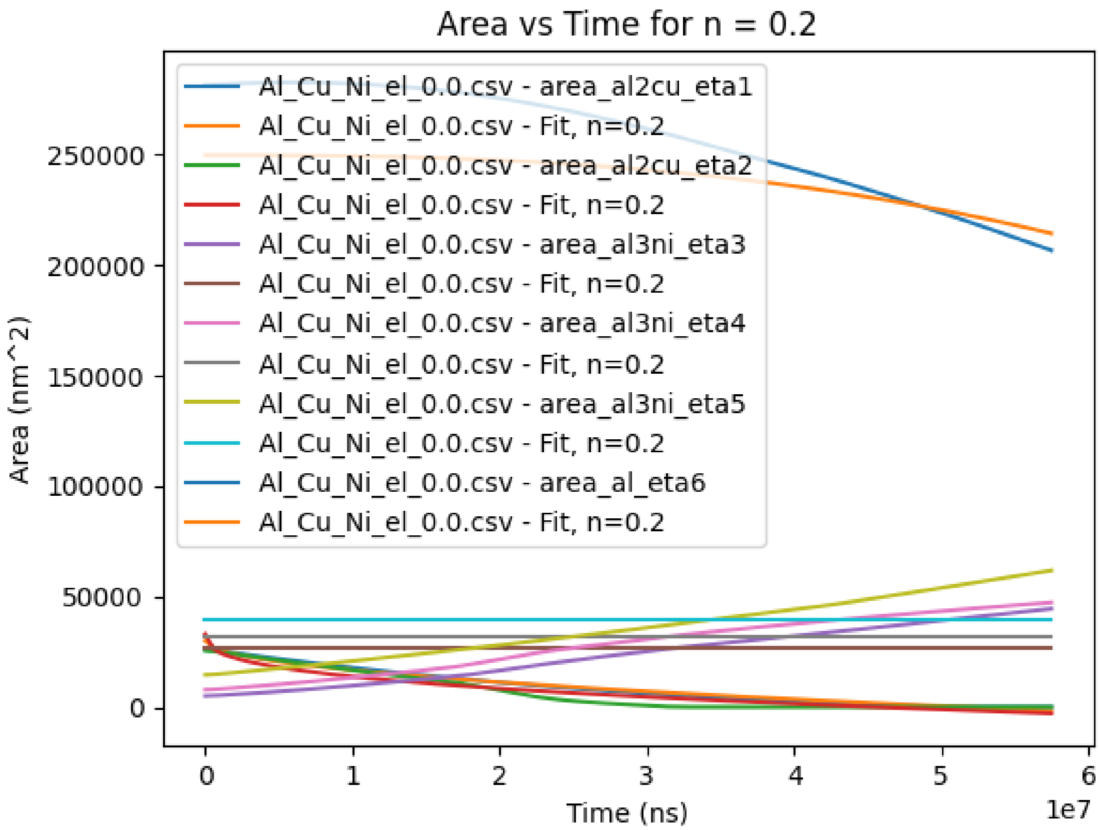
```
    fig, ax = plt.subplots()

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
 ↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and y values
        t_values = data['time']
        y_values = data['area_al2cu_eta1']  # Using 'area_al2cu_eta1' for␣
 ↪fitting

        # Curve fitting for n = n_value
#         popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0))
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0), maxfev=10000)

        y_fit = func(t_values, *popt)

        # Plotting area vs time
        ax.plot(t_values, y_values, label=f'{file_name} - area_al2cu_eta6')
        ax.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.1f}')

    ax.legend()
    ax.set_title(f'Area vs Time for n = {n_value}')
    ax.set_xlabel('Time (ns)')
    ax.set_ylabel('Area (nm^2)')
    plt.show()

# Perform plot for 'area_al2cu_eta1' against time for n = 0.8
plot_area_vs_time_for_n(file_names, 0.2)
```



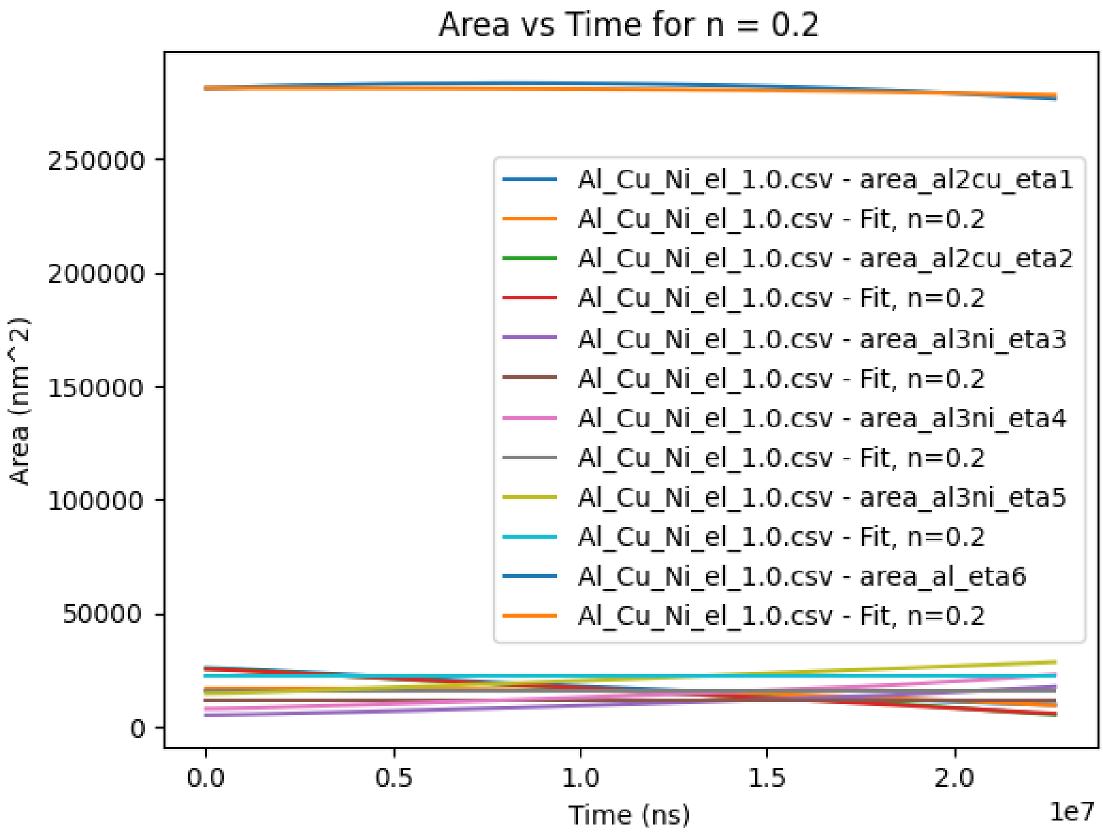
3 Value of slopes

[37]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]
```

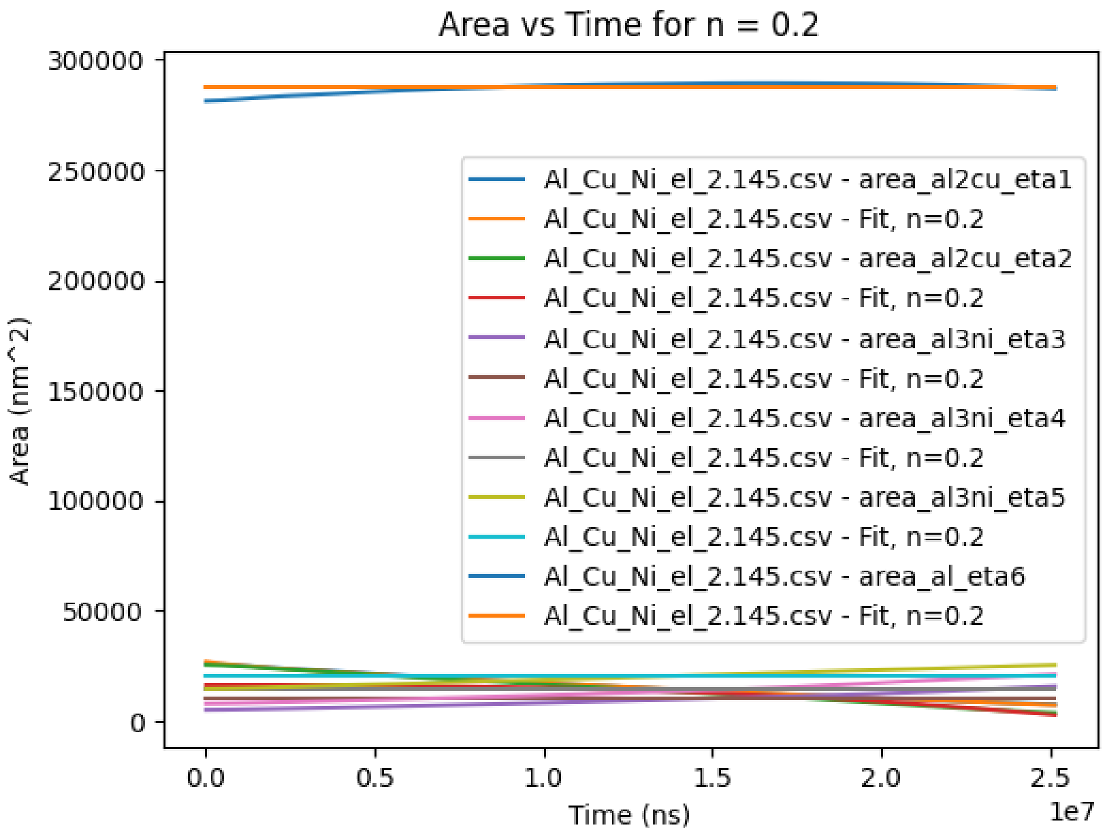
```
# Function to read area data from CSV files and fit curves for n = 0.2 to␣
↪obtain m for all etas and files
def fit_curves_for_n(file_names, n_value):
    m_values = {}

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
↪here

        fig, ax = plt.subplots()

        for col in data.columns:
            if col.startswith('area'):
                # Extracting t and y values
                t_values = data['time']
                y_values = data[col]  # Using columns starting with 'area' for␣
↪fitting

                # Curve fitting for n = n_value
                popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.
↪inf), p0=(1.0, n_value, 1.0), maxfev=10000)
                m_values[file_name + '_' + col] = popt[0]  # Store the value of␣
↪m for each file and eta

                y_fit = func(t_values, *popt)

                # Plotting area vs time
                ax.plot(t_values, y_values, label=f'{file_name} - {col}')
                ax.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.
↪1f}')

        ax.legend()
        ax.set_title(f'Area vs Time for n = {n_value}')
        ax.set_xlabel('Time (ns)')
        ax.set_ylabel('Area (nm^2)')
        plt.show()

    return m_values

# Perform curve fitting for n = 0.2 to obtain m for all etas and files
m_values_02 = fit_curves_for_n(file_names, 0.2)

# Print the obtained m values for each file and eta
for key, value in m_values_02.items():
    print(f'{key}: m = {value:.2f}')
```



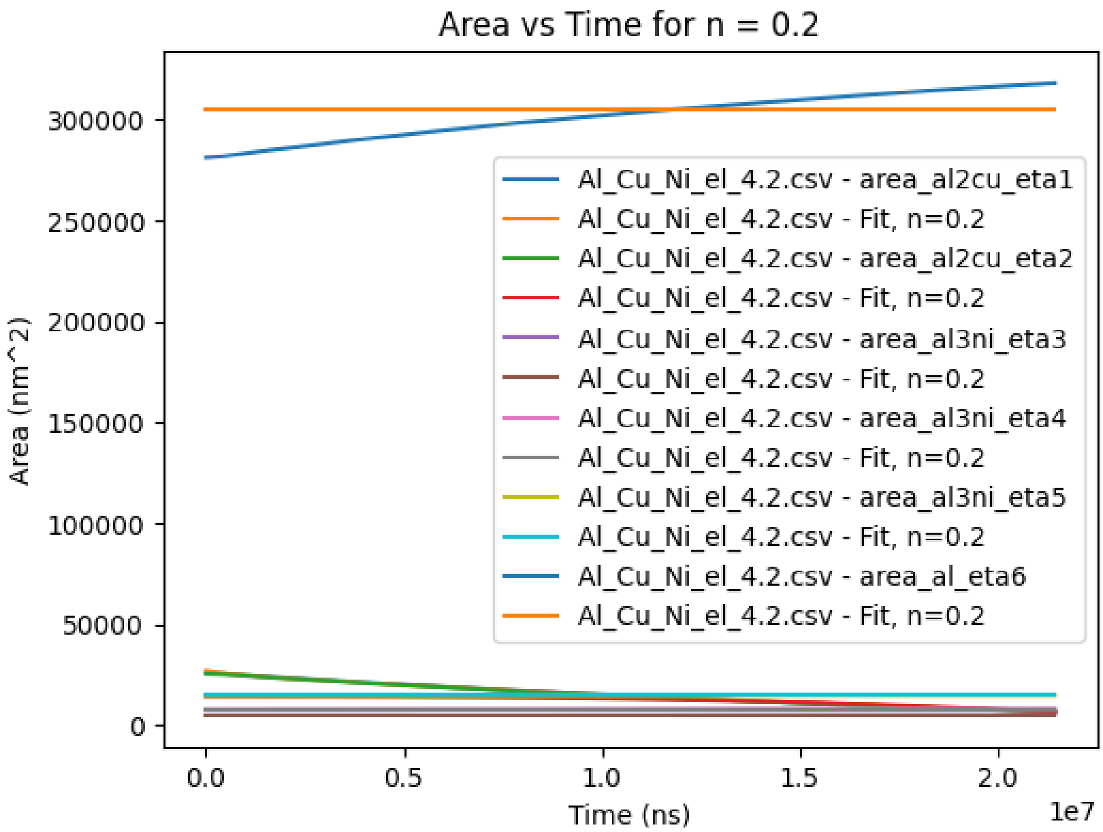
Area vs Time for n = 0.2
Al_Cu_Ni_el_0.0.csv - area_al2cu_eta1
Al_Cu_Ni_el_0.0.csv - Fit, n=0.2
Al_Cu_Ni_el_0.0.csv - area_al2cu_eta2
Al_Cu_Ni_el_0.0.csv - Fit, n=0.2
Al_Cu_Ni_el_0.0.csv - area_al3ni_eta3
Al_Cu_Ni_el_0.0.csv - Fit, n=0.2
Al_Cu_Ni_el_0.0.csv - area_al3ni_eta4
Al_Cu_Ni_el_0.0.csv - Fit, n=0.2
Al_Cu_Ni_el_0.0.csv - area_al3ni_eta5
Al_Cu_Ni_el_0.0.csv - Fit, n=0.2
Al_Cu_Ni_el_0.0.csv - area_al_eta6
Al_Cu_Ni_el_0.0.csv - Fit, n=0.2
Area (nm^2)
250000
200000
150000
100000
50000
0
Time (ns)
0
1
2
3
4
5
6
1e7



Area vs Time for n = 0.2
Area (nm^2)
0
50000
100000
150000
200000
250000
0.0
0.5
1.0
1.5
2.0
1e7
Time (ns)
Al_Cu_Ni_el_1.0.csv - area_al2cu_eta1
Al_Cu_Ni_el_1.0.csv - Fit, n=0.2
Al_Cu_Ni_el_1.0.csv - area_al2cu_eta2
Al_Cu_Ni_el_1.0.csv - Fit, n=0.2
Al_Cu_Ni_el_1.0.csv - area_al3ni_eta3
Al_Cu_Ni_el_1.0.csv - Fit, n=0.2
Al_Cu_Ni_el_1.0.csv - area_al3ni_eta4
Al_Cu_Ni_el_1.0.csv - Fit, n=0.2
Al_Cu_Ni_el_1.0.csv - area_al3ni_eta5
Al_Cu_Ni_el_1.0.csv - Fit, n=0.2
Al_Cu_Ni_el_1.0.csv - area_al_eta6
Al_Cu_Ni_el_1.0.csv - Fit, n=0.2



Area vs Time for n = 0.2
Area (nm^2)
Time (ns)
1e7
300000
250000
200000
150000
100000
50000
0
0.0
0.5
1.0
1.5
2.0
2.5
Al_Cu_Ni_el_2.145.csv - area_al2cu_eta1
Al_Cu_Ni_el_2.145.csv - Fit, n=0.2
Al_Cu_Ni_el_2.145.csv - area_al2cu_eta2
Al_Cu_Ni_el_2.145.csv - Fit, n=0.2
Al_Cu_Ni_el_2.145.csv - area_al3ni_eta3
Al_Cu_Ni_el_2.145.csv - Fit, n=0.2
Al_Cu_Ni_el_2.145.csv - area_al3ni_eta4
Al_Cu_Ni_el_2.145.csv - Fit, n=0.2
Al_Cu_Ni_el_2.145.csv - area_al3ni_eta5
Al_Cu_Ni_el_2.145.csv - Fit, n=0.2
Al_Cu_Ni_el_2.145.csv - area_al_eta6
Al_Cu_Ni_el_2.145.csv - Fit, n=0.2

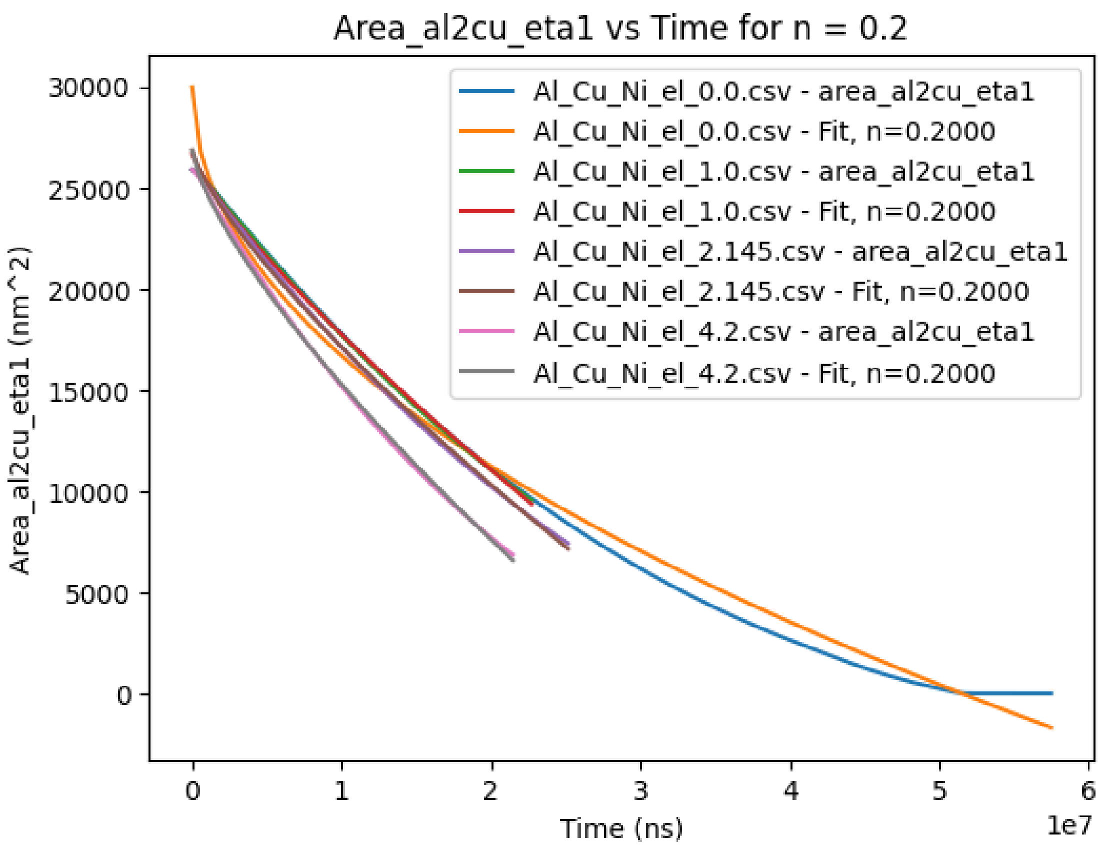



```
Al_Cu_Ni_el_0.0.csv_area_al2cu_eta1: m = 4.44
Al_Cu_Ni_el_0.0.csv_area_al2cu_eta2: m = 61.78
Al_Cu_Ni_el_0.0.csv_area_al3ni_eta3: m = 0.00
Al_Cu_Ni_el_0.0.csv_area_al3ni_eta4: m = 0.00
Al_Cu_Ni_el_0.0.csv_area_al3ni_eta5: m = 0.00
Al_Cu_Ni_el_0.0.csv_area_al_eta6: m = 0.00
Al_Cu_Ni_el_1.0.csv_area_al2cu_eta1: m = 0.00
Al_Cu_Ni_el_1.0.csv_area_al2cu_eta2: m = 0.00
Al_Cu_Ni_el_1.0.csv_area_al3ni_eta3: m = 0.00
Al_Cu_Ni_el_1.0.csv_area_al3ni_eta4: m = 0.00
Al_Cu_Ni_el_1.0.csv_area_al3ni_eta5: m = 0.00
Al_Cu_Ni_el_1.0.csv_area_al_eta6: m = 0.00
Al_Cu_Ni_el_2.145.csv_area_al2cu_eta1: m = 0.04
Al_Cu_Ni_el_2.145.csv_area_al2cu_eta2: m = 0.00
Al_Cu_Ni_el_2.145.csv_area_al3ni_eta3: m = 0.00
Al_Cu_Ni_el_2.145.csv_area_al3ni_eta4: m = 0.00
Al_Cu_Ni_el_2.145.csv_area_al3ni_eta5: m = 0.00
Al_Cu_Ni_el_2.145.csv_area_al_eta6: m = 0.00
Al_Cu_Ni_el_4.2.csv_area_al2cu_eta1: m = 0.08
Al_Cu_Ni_el_4.2.csv_area_al2cu_eta2: m = 0.00
Al_Cu_Ni_el_4.2.csv_area_al3ni_eta3: m = 0.00
```

```
Al_Cu_Ni_el_4.2.csv_area_al3ni_eta4: m = 0.00
Al_Cu_Ni_el_4.2.csv_area_al3ni_eta5: m = 0.00
Al_Cu_Ni_el_4.2.csv_area_al_eta6: m = 0.00
```

[39]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta1
area_value = 'area_al2cu_eta1'

# Function to read area data from CSV files and fit curve for a specific n value
def fit_curve_for_n(file_names, n_value):
    m_values = []  # To store fitted m values

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
↪here

        # Extracting t and area_value values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for n = n_value
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
↪p0=(0, n_value, 1.0), maxfev=10000)

        # Extracting fitted parameter m
        m_fit, _, _ = popt
        m_values.append(m_fit)

        # Plotting area_value vs time
        y_fit = func(t_values, *popt)
```

```
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.4f}')

    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = {n_value}')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    # Displaying fitted 'm' values
    print(f"Fitted 'm' values for n = {n_value}: {m_values}")

# Perform curve fitting for n = 0.2025 and plot the curve
fit_curve_for_n(file_names, 0.2)
```

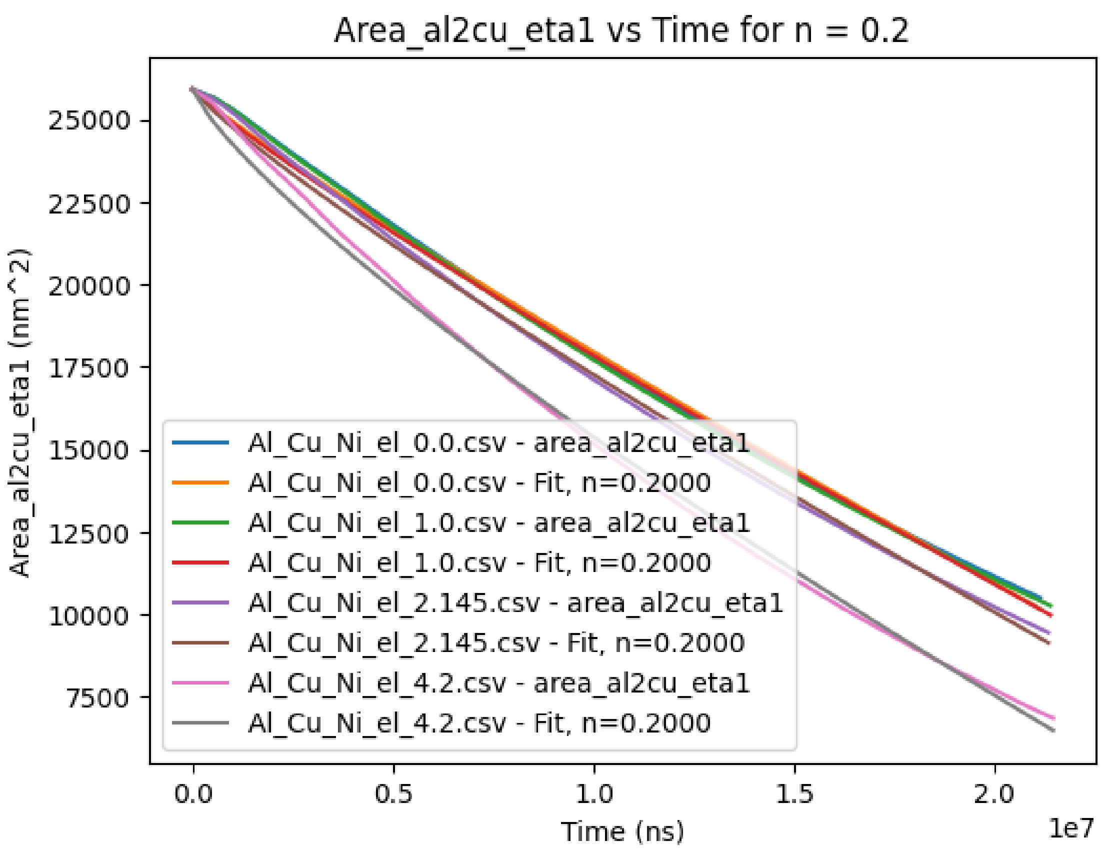



```
Fitted 'm' values for n = 0.2: [4.444309650408783, 0.019688044223607638,
0.04330290932552685, 0.08228087527653438]
```

[79]:
```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + 25902.432232656
 #29986.0587956299# c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta1
area_value = 'area_al2cu_eta1'

# Function to read area data from CSV files and fit curve for a specific n value
def fit_curve_for_n(file_names, n_value):
    m_values = []  # To store fitted m values
    c_values = []  # To store fitted c values

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
 ↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and area_value values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for n = n_value
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0), maxfev=10000)

        # Extracting fitted parameters m and c
        m_fit, _, c_fit = popt
        m_values.append(m_fit)
        c_values.append(c_fit)

        # Plotting area_value vs time
        y_fit = func(t_values, *popt)
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.4f}')
```

```
    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = {n_value}')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    # Displaying fitted 'm' and 'c' values
    print(f"Fitted 'm' values for n = {n_value}: {m_values}")
    print(f"Fitted 'c' values for n = {n_value}: {c_values}")

# Perform curve fitting for n = 0.2025 and plot the curve
fit_curve_for_n(file_names, 0.2)
```



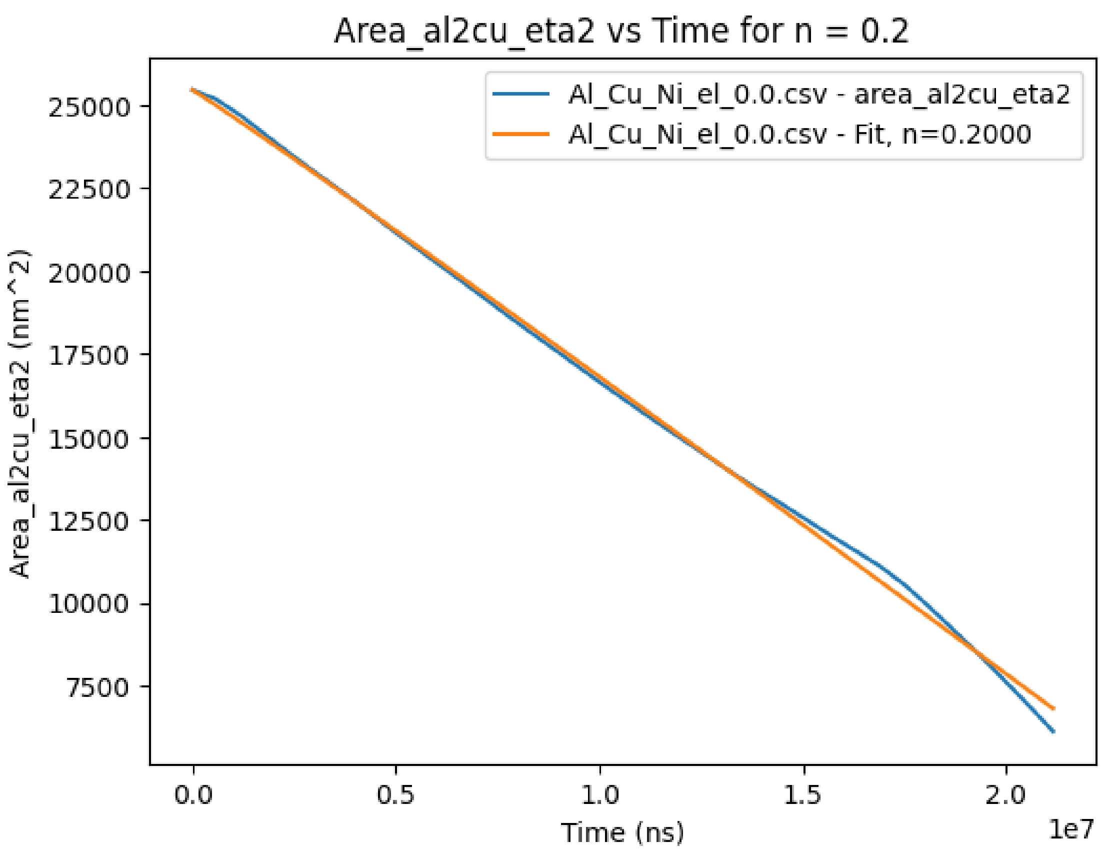

```
Fitted 'm' values for n = 0.2: [0.003268443633678075, 0.004391315940133426,
0.006565500005591682, 0.025772839468331872]
Fitted 'c' values for n = 0.2: [1.0, 1.0, 1.0, 1.0]
```

[83]:
```
import pandas as pd
import numpy as np
```

```python
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + 25457.51108317
 #29986.0587956299# c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
#     'Al_Cu_Ni_el_1.0.csv',
#     'Al_Cu_Ni_el_2.145.csv',
#     'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta1
area_value = 'area_al2cu_eta2'

# Function to read area data from CSV files and fit curve for a specific n value
def fit_curve_for_n(file_names, n_value):
    m_values = []  # To store fitted m values
    c_values = []  # To store fitted c values

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
 ↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and area_value values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for n = n_value
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=(0, n_value, 1.0), maxfev=10000)

        # Extracting fitted parameters m and c
        m_fit, _, c_fit = popt
        m_values.append(m_fit)
        c_values.append(c_fit)
        print(m_fit)

        # Plotting area_value vs time
        y_fit = func(t_values, *popt)
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
```

```
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.4f}')

    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = {n_value}')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    # Displaying fitted 'm' and 'c' values
    print(f"Fitted 'm' values for n = {n_value}: {m_values}")
    print(f"Fitted 'c' values for n = {n_value}: {c_values}")

# Perform curve fitting for n = 0.2025 and plot the curve
fit_curve_for_n(file_names, 0.2)
```

```
0.0005690105338097993
```



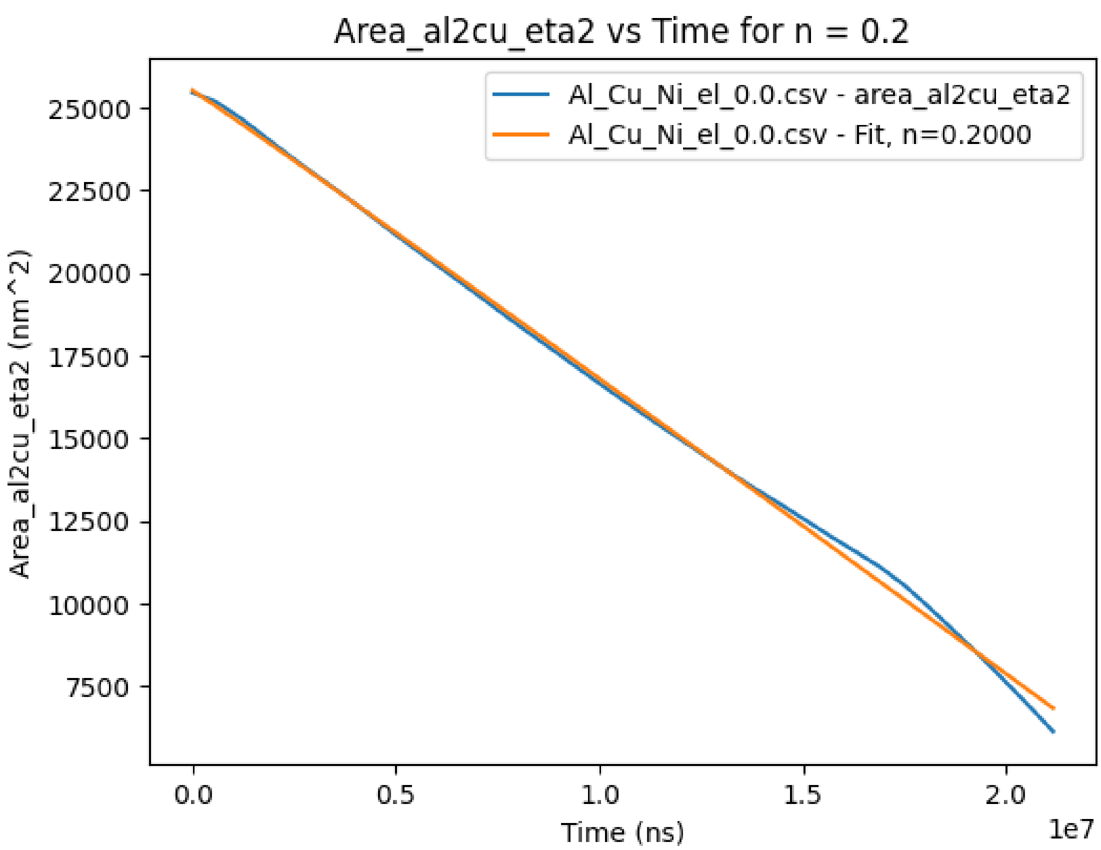

```
Fitted 'm' values for n = 0.2: [0.0005690105338097993]
Fitted 'c' values for n = 0.2: [1.0]
```

[86]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n, c):
    return -m * np.power(t, n) + c

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
]

# Define area_value as area_al2cu_eta2
area_value = 'area_al2cu_eta2'

# Function to read area data from CSV files and fit curve for a specific n value
def fit_curve_for_n(file_names, n_value):
    m_values = []  # To store fitted m values
    c_values = []  # To store fitted c values

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and area_value values
        t_values = data['time']
        y_values = data[area_value]

        # Getting the actual 'c' value from the CSV file at time close to 2e7
        actual_c_value = data.loc[data['time'].sub(2e7).abs().idxmin(),␣
↪area_value]
        print(f"Actual 'c' value from file '{file_name}' at time close to 2e7:␣
↪{actual_c_value}")

        # Curve fitting for n = n_value
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
↪p0=(0, n_value, actual_c_value), maxfev=10000)

        # Extracting fitted parameters m and c
        m_fit, _, c_fit = popt
        m_values.append(m_fit)
        c_values.append(c_fit)
```

```
        # Plotting area_value vs time
        y_fit = func(t_values, *popt)
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.4f}')

    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = {n_value}')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    # Displaying fitted 'm' and 'c' values
    print(f"Fitted 'm' values for n = {n_value}: {m_values}")
    print(f"Fitted 'c' values for n = {n_value}: {c_values}")

# Perform curve fitting for n = 0.2 and plot the curve
fit_curve_for_n(file_names, 0.2)
```

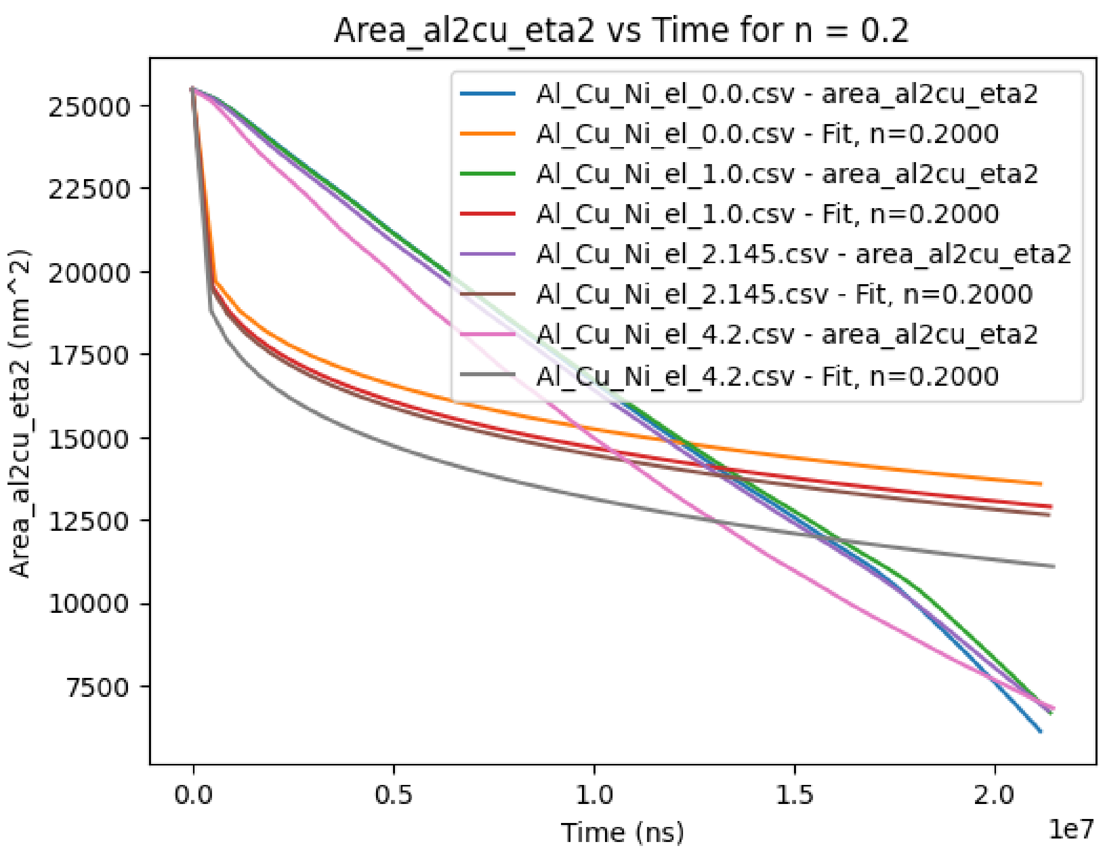
```
Actual 'c' value from file 'Al_Cu_Ni_el_0.0.csv' at time close to 2e7:
7694.0491130537
```



```
Fitted 'm' values for n = 0.2: [0.000657122264773203]
Fitted 'c' values for n = 0.2: [25525.768566390652]
```

[106]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Define the function y = mx^n + c
def func(t, m, n):
    return -m * np.power(t, n) + 25457.51108317  # 'c' provided in the equation

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta2
area_value = 'area_al2cu_eta2'

# Function to read area data from CSV files and fit curve for a specific n value
def fit_curve_for_n(file_names, n_value):
    m_values = []  # To store fitted m values

    for file_name in file_names:
        # Read CSV file
        data = pd.read_csv(f'eigenstrain/{file_name}')  # Adjusting file path␣
↪here
        data = data[data['time'] <= 21500000]

        # Extracting t and area_value values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for 'm'
        popt, pcov = curve_fit(lambda t, m: func(t, m, n_value), t_values,␣
↪y_values,
                               bounds=(0, np.inf), p0=(1000), maxfev=10000)

        # Extracting fitted parameter 'm'
        m_fit = popt[0]
        m_values.append(m_fit)

        # Plotting area_value vs time
```

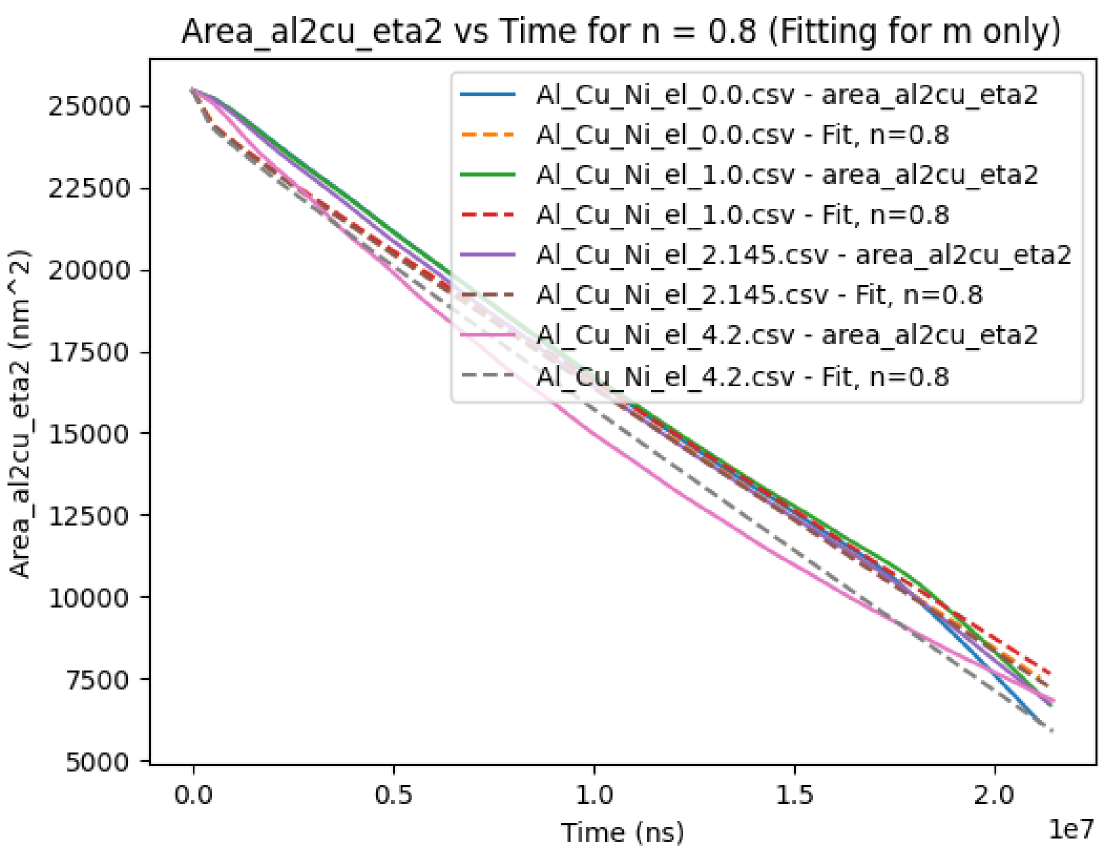
```
        y_fit = func(t_values, m_fit, n_value)
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n={n_value:.4f}')

    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = {n_value}')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    # Displaying fitted 'm' values
    print(f"Fitted 'm' values for n = {n_value}: {m_values}")

# Perform curve fitting for 'm' only for n = 0.2 and plot the curve
fit_curve_for_n(file_names, 0.2)
```




```
Fitted 'm' values for n = 0.2: [407.033437747191, 429.557233852665,
438.10694416099074, 490.6881895646265]
```

[191]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

c_value = 25457.51108317
# Define the function y = mx^n + c with fixed n and c
def func(t, m):
    n_value = 0.2  # Fixed value for 'n'
    return m * np.power(t/1e6, n_value)+(m/1) *np.power(t/1e6, 1.0) + c_value

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta2
area_value = 'area_al2cu_eta2'

# Function to read area data from CSV files and fit curve for 'm' only
def fit_curve_for_m_only(file_names):
    m_values = []  # To store fitted 'm' values

    for file_name in file_names:
        # Read CSV file and filter data
        data = pd.read_csv(f'eigenstrain/{file_name}')
        data = data[data['time'] <= 21500000]

        # Extracting time and area values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for 'm' only, with 'c' fixed
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(-np.inf, 0),␣
 ↪p0=-50, maxfev=10000)

        # Extracting the fitted parameter 'm'
        m_fit = popt[0]
        m_values.append(m_fit)

        # Plotting the original data and the fitted curve for 'm'
        y_fit = func(t_values, m_fit)
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
```

```
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n=0.8',␣
 ↪linestyle='--')

    # Labeling, displaying, and printing fitted 'm' values
    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = 0.8 (Fitting for m␣
 ↪only)')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    print(f"Fitted 'm' values for n = 0.8: {m_values}")

# Perform curve fitting for 'm' only
fit_curve_for_m_only(file_names)
```

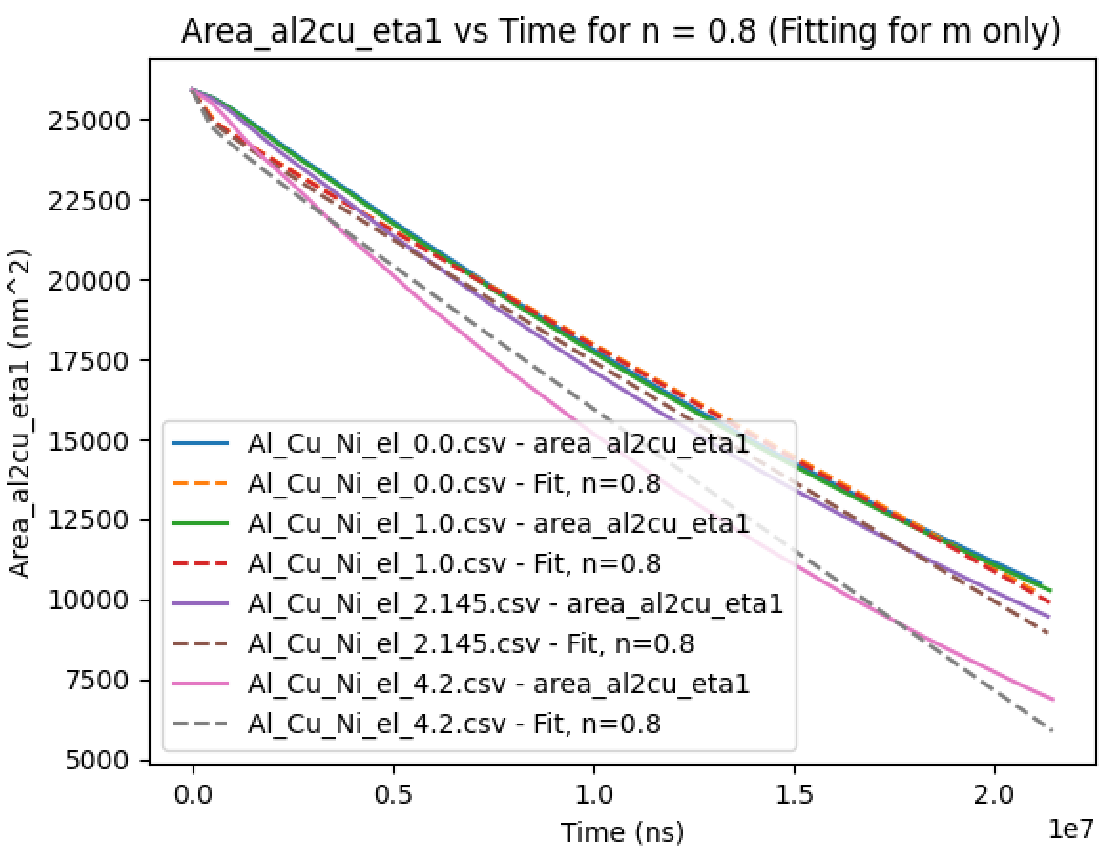



```
Fitted 'm' values for n = 0.8: [-780.5619870486865, -766.7804402971085,
-784.8090881542939, -839.9013321835404]
```

[190]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

c_value = 25902.432232656

# Define the function y = mx^n + c with fixed n and c
def func(t, m):
    n_value = 0.2  # Fixed value for 'n'
    return m * np.power(t/1e6, n_value)+(m/1) *np.power(t/1e6, 1.0) + c_value

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta2
area_value = 'area_al2cu_eta1'

# Function to read area data from CSV files and fit curve for 'm' only
def fit_curve_for_m_only(file_names):
    m_values = []  # To store fitted 'm' values

    for file_name in file_names:
        # Read CSV file and filter data
        data = pd.read_csv(f'eigenstrain/{file_name}')
        data = data[data['time'] <= 21500000]

        # Extracting time and area values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for 'm' only, with 'c' fixed
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(-np.inf, 0),␣
 ↪p0=-50, maxfev=10000)

        # Extracting the fitted parameter 'm'
        m_fit = popt[0]
        m_values.append(m_fit)

        # Plotting the original data and the fitted curve for 'm'
        y_fit = func(t_values, m_fit)
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
```

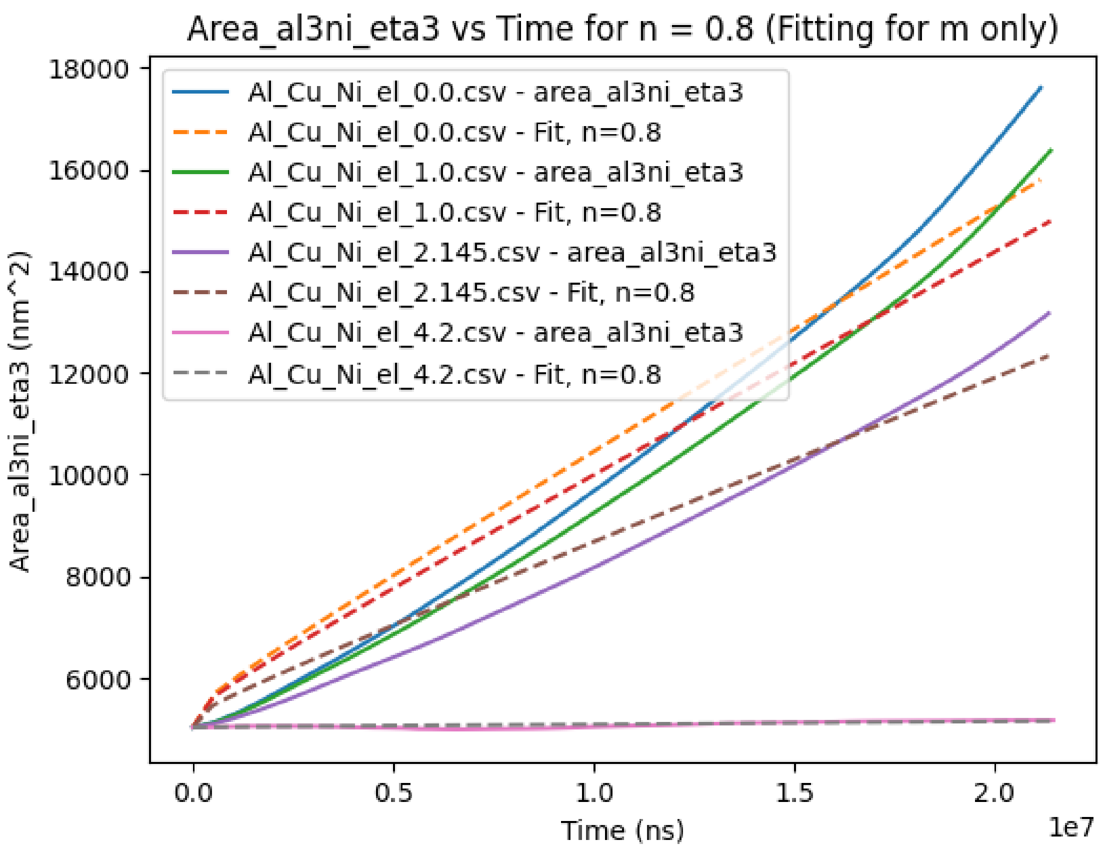
```
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n=0.8',␣
↪linestyle='--')

    # Labeling, displaying, and printing fitted 'm' values
    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = 0.8 (Fitting for m␣
↪only)')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    print(f"Fitted 'm' values for n = 0.8: {m_values}")

# Perform curve fitting for 'm' only
fit_curve_for_m_only(file_names)
```




```
Fitted 'm' values for n = 0.8: [-684.7557450457423, -689.2474950055083,
-731.9359330029571, -859.5218684176816]
```

[193]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

c_value = 5019.481642672


# Define the function y = mx^n + c with fixed n and c
def func(t, m):
    n_value = 0.2  # Fixed value for 'n'
    return m * np.power(t/1e6, n_value)+(m/1) *np.power(t/1e6, 1.0) + c_value

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta2
area_value = 'area_al3ni_eta3'

# Function to read area data from CSV files and fit curve for 'm' only
def fit_curve_for_m_only(file_names):
    m_values = []  # To store fitted 'm' values

    for file_name in file_names:
        # Read CSV file and filter data
        data = pd.read_csv(f'eigenstrain/{file_name}')
        data = data[data['time'] <= 21500000]

        # Extracting time and area values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for 'm' only, with 'c' fixed
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
 ↪p0=100, maxfev=10000)

        # Extracting the fitted parameter 'm'
        m_fit = popt[0]
        m_values.append(m_fit)

        # Plotting the original data and the fitted curve for 'm'
        y_fit = func(t_values, m_fit)
```

```
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n=0.8',␣
↪linestyle='--')

    # Labeling, displaying, and printing fitted 'm' values
    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = 0.8 (Fitting for m␣
↪only)')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    print(f"Fitted 'm' values for n = 0.8: {m_values}")

# Perform curve fitting for 'm' only
fit_curve_for_m_only(file_names)
```

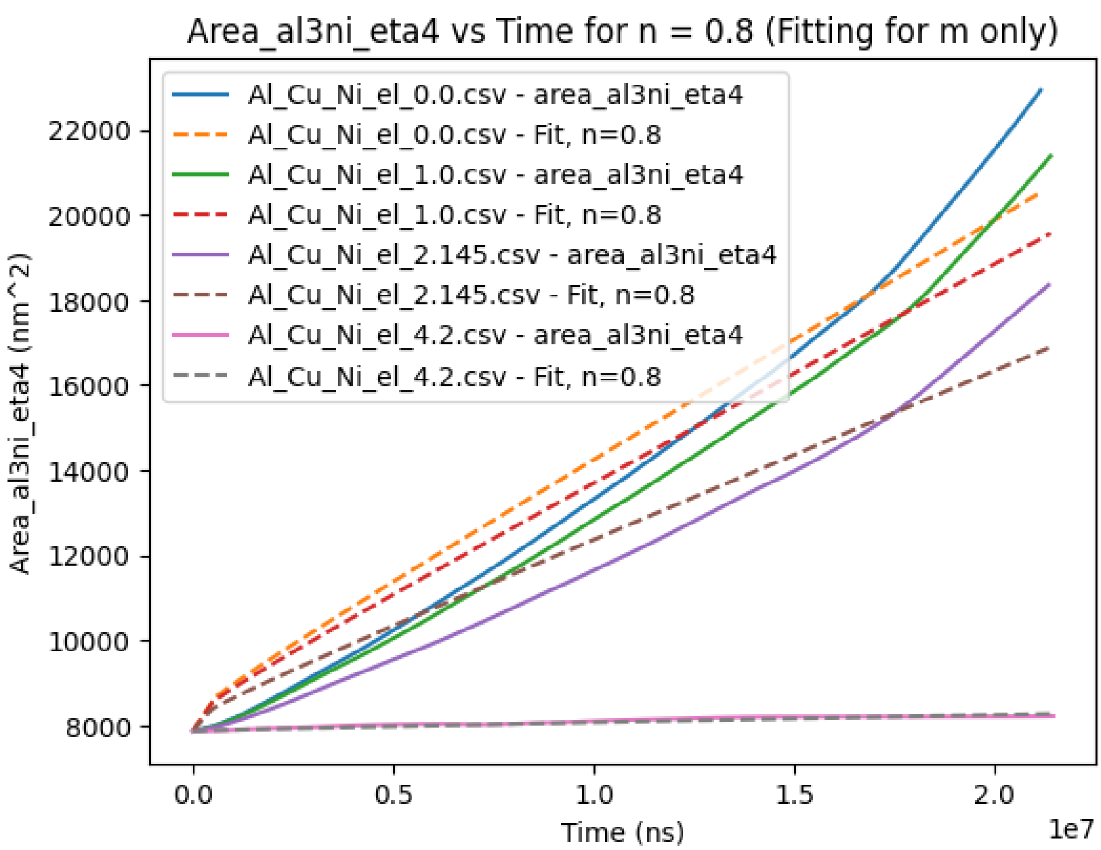



```
Fitted 'm' values for n = 0.8: [468.62377493279155, 428.6915261351951,
315.2658644731741, 5.419127856590954]
```

[197]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

c_value = 7860.9653404568



# Define the function y = mx^n + c with fixed n and c
def func(t, m):
    n_value = 0.2  # Fixed value for 'n'
    return m * np.power(t/1e6, n_value)+(m/1) *np.power(t/1e6, 1.0) + c_value

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta2
area_value = 'area_al3ni_eta4'

# Function to read area data from CSV files and fit curve for 'm' only
def fit_curve_for_m_only(file_names):
    m_values = []  # To store fitted 'm' values

    for file_name in file_names:
        # Read CSV file and filter data
        data = pd.read_csv(f'eigenstrain/{file_name}')
        data = data[data['time'] <= 21500000]

        # Extracting time and area values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for 'm' only, with 'c' fixed
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
↪p0=100, maxfev=10000)

        # Extracting the fitted parameter 'm'
        m_fit = popt[0]
        m_values.append(m_fit)

        # Plotting the original data and the fitted curve for 'm'
```

```
        y_fit = func(t_values, m_fit)
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n=0.8',␣
↪linestyle='--')

    # Labeling, displaying, and printing fitted 'm' values
    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = 0.8 (Fitting for m␣
↪only)')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    print(f"Fitted 'm' values for n = 0.8: {m_values}")

# Perform curve fitting for 'm' only
fit_curve_for_m_only(file_names)
```

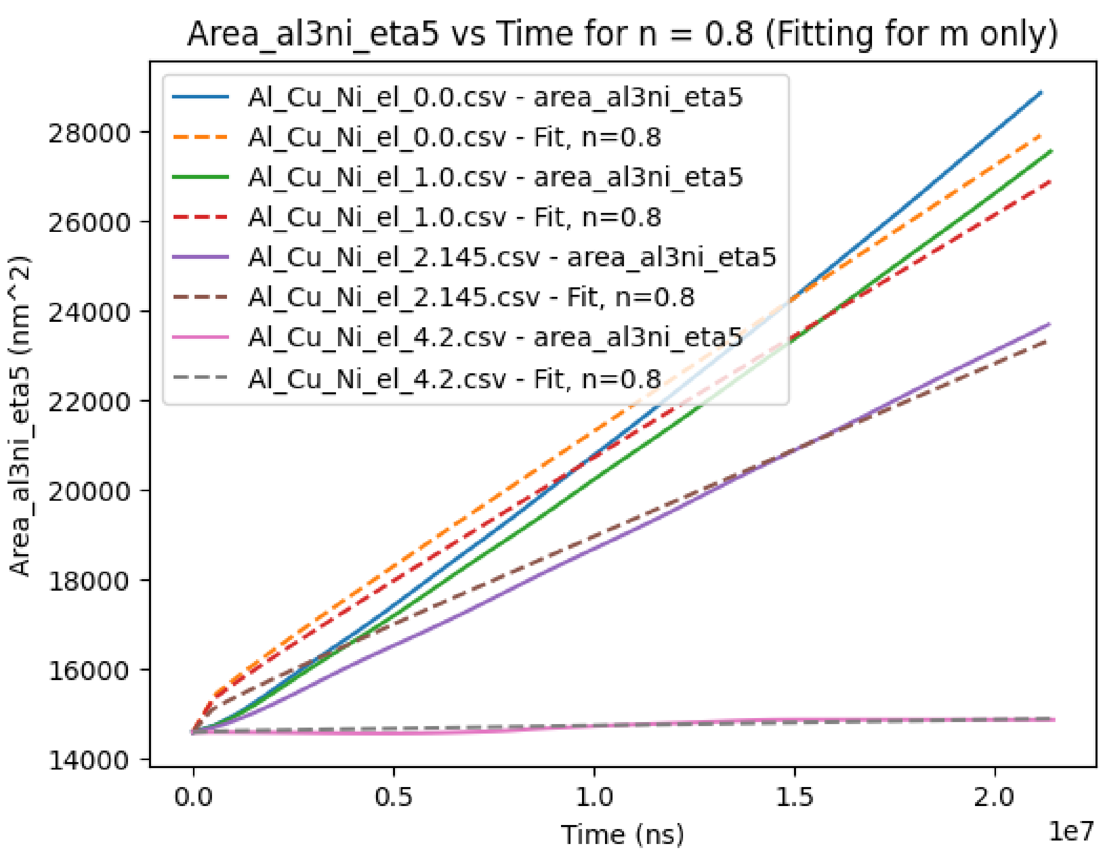



```
Fitted 'm' values for n = 0.8: [551.2164741039911, 503.76232143331106,
388.7530667552912, 17.770206306906417]
```

[198]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

c_value = 14594.041229196



# Define the function y = mx^n + c with fixed n and c
def func(t, m):
    n_value = 0.2  # Fixed value for 'n'
    return m * np.power(t/1e6, n_value)+(m/1) *np.power(t/1e6, 1.0) + c_value

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta2
area_value = 'area_al3ni_eta5'

# Function to read area data from CSV files and fit curve for 'm' only
def fit_curve_for_m_only(file_names):
    m_values = []  # To store fitted 'm' values

    for file_name in file_names:
        # Read CSV file and filter data
        data = pd.read_csv(f'eigenstrain/{file_name}')
        data = data[data['time'] <= 21500000]

        # Extracting time and area values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for 'm' only, with 'c' fixed
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(0, np.inf),␣
↪p0=100, maxfev=10000)

        # Extracting the fitted parameter 'm'
        m_fit = popt[0]
        m_values.append(m_fit)

        # Plotting the original data and the fitted curve for 'm'
        y_fit = func(t_values, m_fit)
```

```
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n=0.8',␣
↪linestyle='--')

    # Labeling, displaying, and printing fitted 'm' values
    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = 0.8 (Fitting for m␣
↪only)')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    print(f"Fitted 'm' values for n = 0.8: {m_values}")

# Perform curve fitting for 'm' only
fit_curve_for_m_only(file_names)
```

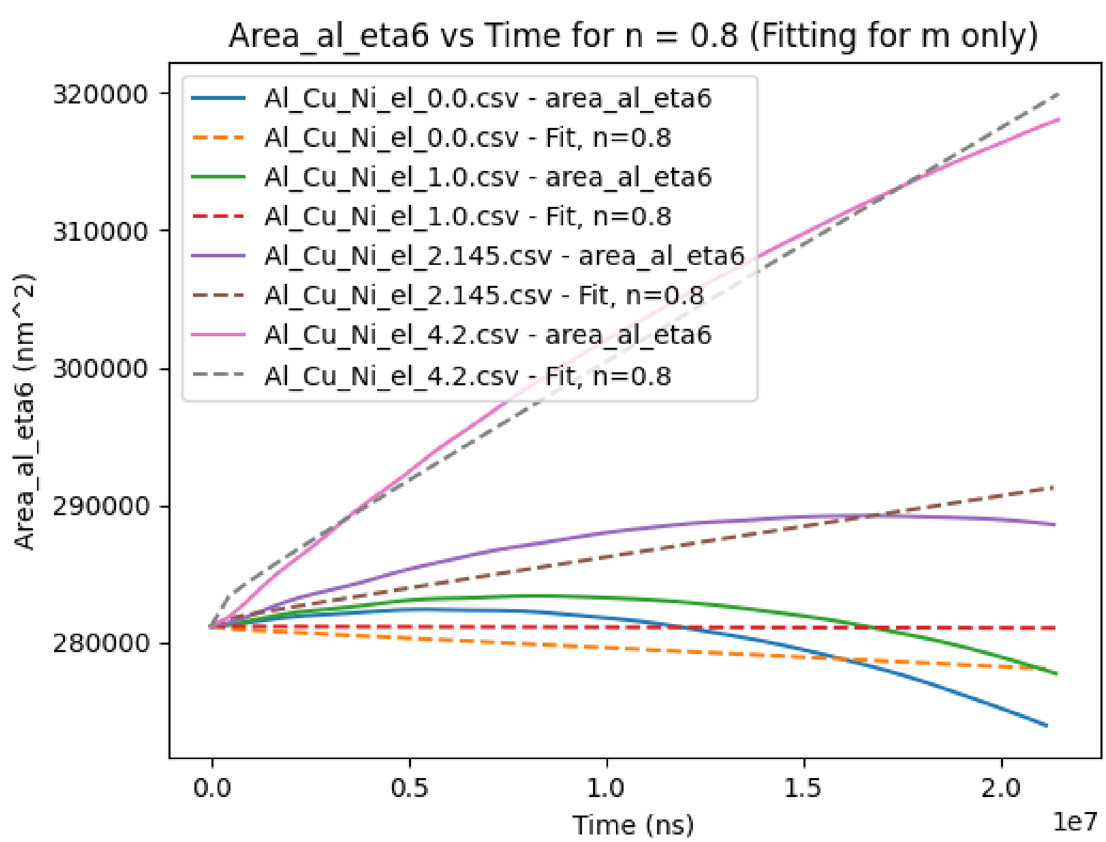



```
Fitted 'm' values for n = 0.8: [579.6328875273215, 529.1662277895351,
377.0973362656922, 13.032913143827928]
```

[258]:

```python
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

c_value = 281165.56847187


# Define the function y = mx^n + c with fixed n and c
def func(t, m):
    n_value = 0.2  # Fixed value for 'n'
    return m * np.power(t/1e6, n_value)+(m/1) *np.power(t/1e6, 1.0) + c_value

# File names
file_names = [
    'Al_Cu_Ni_el_0.0.csv',
    'Al_Cu_Ni_el_1.0.csv',
    'Al_Cu_Ni_el_2.145.csv',
    'Al_Cu_Ni_el_4.2.csv'
]

# Define area_value as area_al2cu_eta2
area_value = 'area_al_eta6'

# Function to read area data from CSV files and fit curve for 'm' only
def fit_curve_for_m_only(file_names):
    m_values = []  # To store fitted 'm' values

    for file_name in file_names:
        # Read CSV file and filter data
        data = pd.read_csv(f'eigenstrain/{file_name}')
        data = data[data['time'] <= 21500000]

        # Extracting time and area values
        t_values = data['time']
        y_values = data[area_value]

        # Curve fitting for 'm' only, with 'c' fixed
        popt, pcov = curve_fit(func, t_values, y_values, bounds=(-np.inf, np.
↪inf), p0=-20, maxfev=10000)

        # Extracting the fitted parameter 'm'
        m_fit = popt[0]
        m_values.append(m_fit)

        # Plotting the original data and the fitted curve for 'm'
```

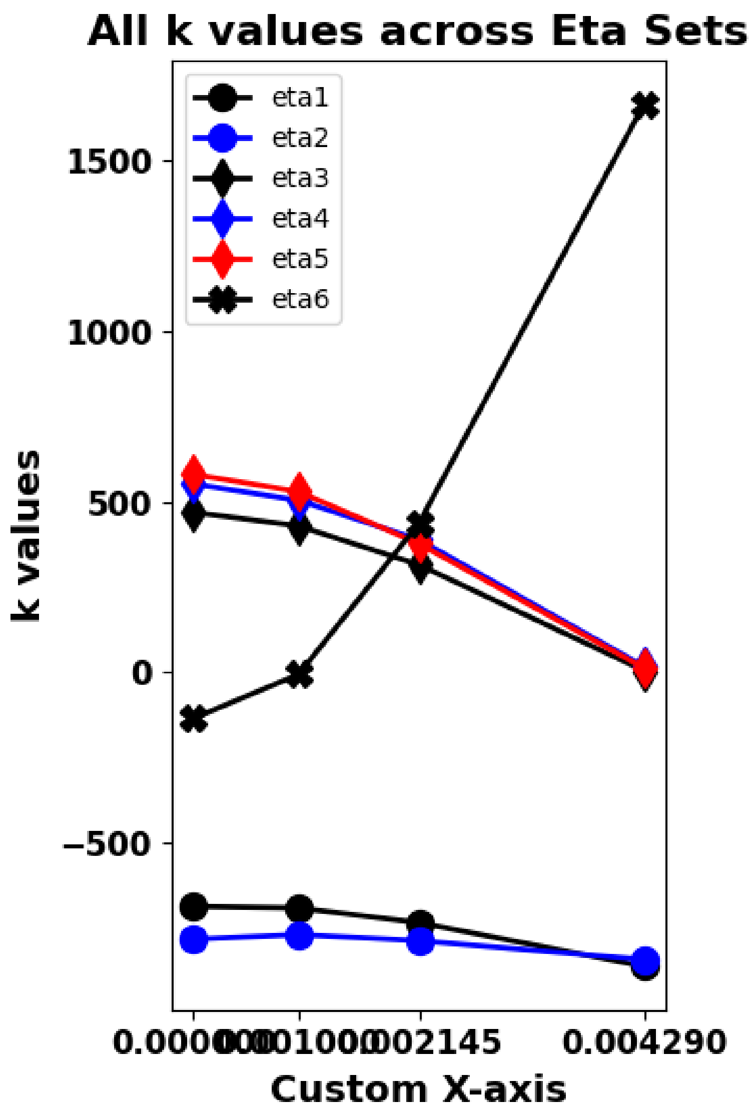
```
        y_fit = func(t_values, m_fit)
        plt.plot(t_values, y_values, label=f'{file_name} - {area_value}')
        plt.plot(t_values, y_fit, label=f'{file_name} - Fit, n=0.8', linestyle='--')

    # Labeling, displaying, and printing fitted 'm' values
    plt.legend()
    plt.title(f'{area_value.capitalize()} vs Time for n = 0.8 (Fitting for m only)')
    plt.xlabel('Time (ns)')
    plt.ylabel(f'{area_value.capitalize()} (nm^2)')
    plt.show()

    print(f"Fitted 'm' values for n = 0.8: {m_values}")

# Perform curve fitting for 'm' only
fit_curve_for_m_only(file_names)
```




```
Fitted 'm' values for n = 0.8: [-134.15542599891478, -5.592165541682647,
435.6287539644629, 1663.2009542054957]
```

[300]:

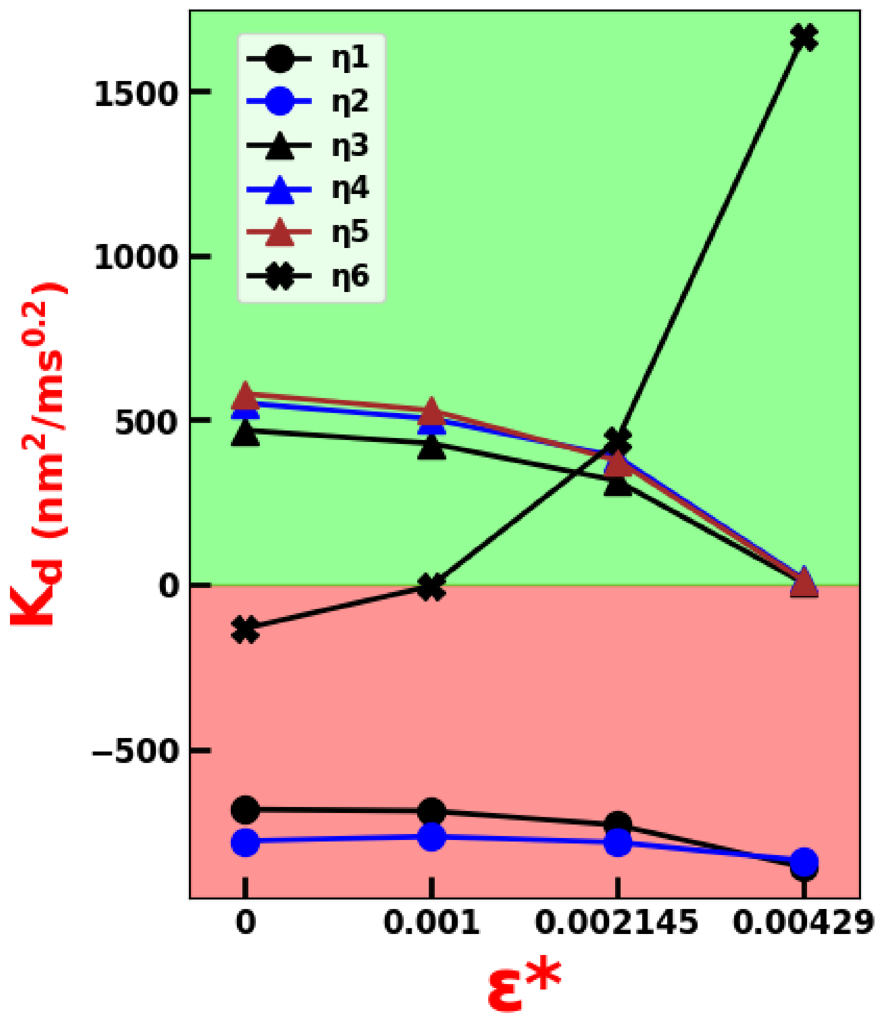
```
import matplotlib.pyplot as plt

# Given 'k' values for each eta set
k_values = [
    [-684.7557450457423, -689.2474950055083, -731.9359330029571, -859.
 ↪5218684176816],
    [-780.5619870486865, -766.7804402971085, -784.8090881542939, -839.
 ↪9013321835404],
    [468.62377493279155, 428.6915261351951, 315.2658644731741, 5.
 ↪419127856590954],
    [551.2164741039911, 503.76232143331106, 388.7530667552912, 17.
 ↪770206306906417],
    [579.6328875273215, 529.1662277895351, 377.0973362656922, 13.
 ↪032913143827928],
    [-134.15542599891478, -5.592165541682647, 435.6287539644629, 1663.
 ↪2009542054957]
]

# Custom x-axis tick labels
custom_ticks = [0, 0.001, 0.002145, 0.00429]

# Plotting customized connected scatter plot
plt.figure(figsize=(4, 6))

markers = ['o', 'o', 'd', 'd', 'd', 'X']  # Define markers for each dataset
colors = ['black', 'blue', 'black', 'blue', 'red', 'black']
linestyles = ['-', '-', '-', '-', '-', '-']

for i, k in enumerate(k_values):
    plt.plot(custom_ticks, k, marker=markers[i], color=colors[i], linewidth=2,␣
 ↪markersize=10, linestyle=linestyles[i], label=f'eta{i+1}')

plt.xticks(custom_ticks, fontsize=12, fontweight='bold')  # Set custom x-axis␣
 ↪tick labels with increased font size and bold
plt.yticks(fontsize=12, fontweight='bold')  # Set y-axis tick labels with␣
 ↪increased font size and bold
plt.xlabel('Custom X-axis', fontsize=14, fontweight='bold')  # Set x-axis label␣
 ↪with increased font size and bold
plt.ylabel('k values', fontsize=14, fontweight='bold')  # Set y-axis label with␣
 ↪increased font size and bold
plt.title('All k values across Eta Sets', fontsize=16, fontweight='bold')  #␣
 ↪Set title with increased font size and bold
plt.legend()
# plt.grid(True)
plt.tight_layout()
plt.show()
```



[389]:
```
import matplotlib.pyplot as plt

# Given 'k' values for each eta set
k_values = [
    [-684.7557450457423, -689.2474950055083, -731.9359330029571, -859.
 ↪5218684176816],
    [-780.5619870486865, -766.7804402971085, -784.8090881542939, -839.
 ↪9013321835404],
    [468.62377493279155, 428.6915261351951, 315.2658644731741, 5.
 ↪419127856590954],
    [551.2164741039911, 503.76232143331106, 388.7530667552912, 17.
 ↪770206306906417],
```

```
    [579.6328875273215, 529.1662277895351, 377.0973362656922, 13.032913143827928],
    [-134.15542599891478, -5.592165541682647, 435.6287539644629, 1663.2009542054957]
]

# # Custom x-axis tick labels
# custom_ticks = [0, 0.001, 0.002145, 0.00429]

# # Formatting x-axis tick labels without trailing zero
# custom_tick_labels = [f'{tick:g}' for tick in custom_ticks]

# Custom x-axis tick positions and labels
custom_ticks = [0, 1, 2, 3]  # Positions of ticks
custom_tick_labels = ['0', '0.001', '0.002145', '0.00429']  # Labels for the ticks

# Plotting customized connected scatter plot
plt.figure(figsize=(4.5, 6))

markers = ['o', 'o', '^', '^', '^', 'X']  # Define markers for each dataset
colors = ['black', 'blue', 'black', 'blue', 'brown', 'black']
linestyles = ['-', '-', '-', '-', '-', '-']
# plt.xlim(left=-0.001, right=0.005)
for i, k in enumerate(k_values):
    plt.plot(custom_ticks, k, marker=markers[i], color=colors[i], linewidth=2, markersize=10,
             linestyle=linestyles[i], label=f'\u03B7{i+1}')

plt.xticks(custom_ticks, labels=custom_tick_labels, fontsize=12, fontweight='bold')  # Set custom x-axis tick labels without trailing zero, increased font size, and bold
plt.yticks(fontsize=12, fontweight='bold')  # Set y-axis tick labels with increased font size and bold
plt.xlabel('\u03B5*', fontsize=28, fontweight='bold', color='r')  # Set x-axis label with increased font size and bold
plt.ylabel(r'K$_\mathbf{d\ (nm^2 / ms^{0.2})}$', fontsize=22, fontweight='bold', color='r')  # Set y-axis label with increased font size and bold
plt.legend(prop={'size': 11, 'weight': 'bold'}, bbox_to_anchor=(0.05, 0.99), loc='upper left')

# Create custom ticks using np.arange() from -0.3 to 3.3
custom_ticks = np.arange(-0.3, 4, 1)
# Fill between x=-0.3 and x=3.3
```

```
plt.fill_between(custom_ticks, global_min - 100, 0, where=(custom_ticks >= -0.
↪3) & (custom_ticks <= 4), color='red', alpha=0.42)
plt.fill_between(custom_ticks, global_max + 100, 0, where=(custom_ticks >= -0.
↪3) & (custom_ticks <= 4), color='lime', alpha=0.42)


# Fill between x values less than 0 and up to 0.005


# Adjusting tick parameters - making tick lines thicker and longer
plt.tick_params(axis='both', which='major', direction='in', width=2, length=8)
plt.tick_params(axis='both', which='minor',direction='in', width=1.5, length=6)
plt.ylim(top=1750, bottom=-950)

plt.xlim(-0.3,3.3)
# plt.grid(True)
plt.savefig('K_rate.png', dpi=1200, bbox_inches='tight')
plt.show()
```


η1
η2
η3
η4
η5
η6
1500
1000
500
0
−500
K_d (nm^2/ms$^{0.2}$)
0
0.001
0.002145
0.00429
ε*

```
[ ]:
```