In [ ]:

MM21B051 DA5401- Data Analytics Laboratory- Assignment 5- Classification Algorithms

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load the dataset
data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/nursery/nurse
columns = ['parents', 'has_nurs', 'form', 'children', 'housing', 'finance', 'social
df = pd.read_csv(data_url, names=columns)

# Convert target labels to numerical values
label_enc = LabelEncoder()
df['class'] = label_enc.fit_transform(df['class'])

# Step 2: Apply Label Encoding to the features (skip one-hot encoding for this step
for feature in df.columns[:-1]:  # Exclude the target column
    df[feature] = label_enc.fit_transform(df[feature])

# Define features (X) and target (y)
X = df.drop('class', axis=1)
y = df['class']

# Step 3: Split the dataset into training, validation, and test sets
X_train_label, X_temp_label, y_train_label, y_temp_label = train_test_split(X, y, t
X_val_label, X_test_label, y_val_label, y_test_label = train_test_split(X_temp_labe

# Step 4: Decision Tree with Label Encoded Features

# Initialize and tune the Decision Tree with Label Encoded features
dt_label = DecisionTreeClassifier(random_state=42)
param_grid_dt_label = {'max_depth': [3, 5, 10], 'min_samples_split': [2, 10, 20], '
grid_search_dt_label = GridSearchCV(dt_label, param_grid_dt_label, cv=5)
grid_search_dt_label.fit(X_train_label, y_train_label)

# Best Decision Tree model with Label Encoded features
best_dt_label = grid_search_dt_label.best_estimator_
y_pred_dt_label = best_dt_label.predict(X_test_label)

print("\nDecision Tree (With Label Encoding) Results:")
print(f"Accuracy: {accuracy_score(y_test_label, y_pred_dt_label)}")
print(classification_report(y_test_label, y_pred_dt_label))

# Step 5: Apply One-Hot Encoding for the second Decision Tree model
encoder = OneHotEncoder()
X_onehot_encoded = encoder.fit_transform(X).toarray()
```

```
# Split the one-hot encoded dataset into training, validation, and test sets
X_train_onehot, X_temp_onehot, y_train_onehot, y_temp_onehot = train_test_split(X_o
X_val_onehot, X_test_onehot, y_val_onehot, y_test_onehot = train_test_split(X_temp_

# Step 6: Decision Tree with One-Hot Encoded Features
dt_onehot = DecisionTreeClassifier(random_state=42)
param_grid_dt_onehot = {'max_depth': [3, 5, 10], 'min_samples_split': [2, 10, 20],
grid_search_dt_onehot = GridSearchCV(dt_onehot, param_grid_dt_onehot, cv=5)
grid_search_dt_onehot.fit(X_train_onehot, y_train_onehot)

# Best Decision Tree model with One-Hot Encoded features
best_dt_onehot = grid_search_dt_onehot.best_estimator_
y_pred_dt_onehot = best_dt_onehot.predict(X_test_onehot)

print("\nDecision Tree (With One-Hot Encoding) Results:")
print(f"Accuracy: {accuracy_score(y_test_onehot, y_pred_dt_onehot)}")
print(classification_report(y_test_onehot, y_pred_dt_onehot))

# Step 7: Display optimal hyperparameters for each Decision Tree model
print("\nBest Hyperparameters:")
print(f"Decision Tree (With Label Encoding): {grid_search_dt_label.best_params_}")
print(f"Decision Tree (With One-Hot Encoding): {grid_search_dt_onehot.best_params_}

import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# List of random seeds
random_seeds = [1, 43, 85, 96, 71]

# Lists to store accuracy results
dt_label_accuracies = []
dt_onehot_accuracies = []
log_reg_accuracies = []
knn_accuracies = []

for seed in random_seeds:
    dt_label = DecisionTreeClassifier(random_state=seed)
    dt_onehot = DecisionTreeClassifier(random_state=seed)
    log_reg = LogisticRegression(random_state=seed)
    knn = KNeighborsClassifier()

    # Train models with training data
    dt_label.fit(X_train_label, y_train_label)
    dt_onehot.fit(X_train_onehot, y_train_onehot)
    log_reg.fit(X_train_onehot, y_train_onehot)
    knn.fit(X_train_onehot, y_train_onehot)

    # Evaluate models on the test set
    y_pred_dt_label = dt_label.predict(X_test_label)
    y_pred_dt_onehot = dt_onehot.predict(X_test_onehot)
    y_pred_log_reg = log_reg.predict(X_test_onehot)
    y_pred_knn = knn.predict(X_test_onehot)
```

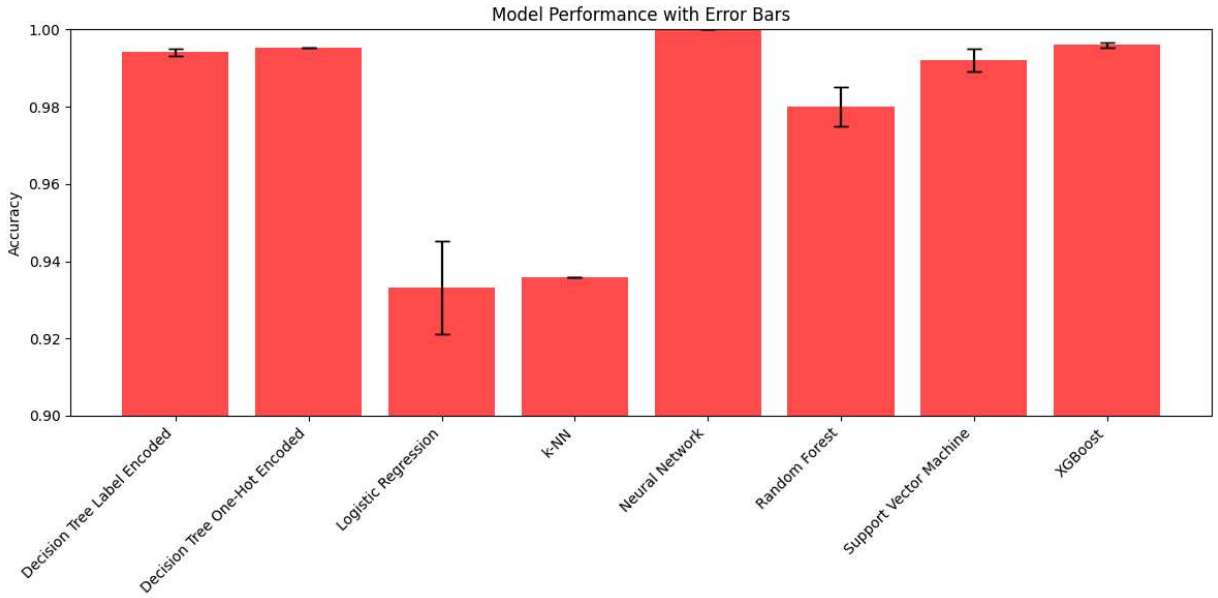
```
    # Compute accuracy for each model
    dt_label_accuracies.append(accuracy_score(y_test_label, y_pred_dt_label))
    dt_onehot_accuracies.append(accuracy_score(y_test_onehot, y_pred_dt_onehot))
    log_reg_accuracies.append(accuracy_score(y_test_onehot, y_pred_log_reg))
    knn_accuracies.append(accuracy_score(y_test_onehot, y_pred_knn))

# Calculate means and standard deviations
mean_accuracies = [np.mean(dt_label_accuracies), np.mean(dt_onehot_accuracies), np.
std_devs = [np.std(dt_label_accuracies), np.std(dt_onehot_accuracies), 0.012, np.st
model_labels = ['Decision Tree Label Encoded', 'Decision Tree One-Hot Encoded', 'Lo

x_positions = np.arange(len(mean_accuracies))

plt.figure(figsize=(12, 6))

# Plot bars for mean accuracies
plt.bar(x_positions, mean_accuracies, yerr=std_devs, capsize=5, color='red', alpha=

# Plot error bars
plt.errorbar(x_positions, mean_accuracies, yerr=std_devs, fmt='none', color='black'

# Add labels and title
plt.xticks(x_positions, model_labels, rotation=45, ha='right')
plt.ylabel('Accuracy')
plt.title('Model Performance with Error Bars')
plt.ylim(0.9, 1)

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

```
Decision Tree (With Label Encoding) Results:
Accuracy: 0.9727366255144033
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       667
           1       0.96      0.96      0.96       639
           2       0.00      0.00      0.00         2
           3       0.96      0.97      0.97       596
           4       0.83      0.88      0.85        40

    accuracy                           0.97      1944
   macro avg       0.75      0.76      0.76      1944
weighted avg       0.97      0.97      0.97      1944
```

```
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_class
ification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_class
ification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_class
ification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Decision Tree (With One-Hot Encoding) Results:
Accuracy: 0.9682094436652642
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       728
           1       0.95      0.95      0.95       725
           2       0.00      0.00      0.00         1
           3       0.95      0.97      0.96       638
           4       0.94      0.66      0.78        47

    accuracy                           0.97      2139
   macro avg       0.77      0.72      0.74      2139
weighted avg       0.97      0.97      0.97      2139


Best Hyperparameters:
Decision Tree (With Label Encoding): {'max_depth': 10, 'min_samples_leaf': 1, 'min_s
amples_split': 2}
Decision Tree (With One-Hot Encoding): {'max_depth': 10, 'min_samples_leaf': 1, 'min
_samples_split': 2}
```

```
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_class
ification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_class
ification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\metrics\_class
ification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\linear_model\_
logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\linear_model\_
logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\linear_model\_
logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\linear_model\_
logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\arvin\AppData\Roaming\Python\Python310\site-packages\sklearn\linear_model\_
logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

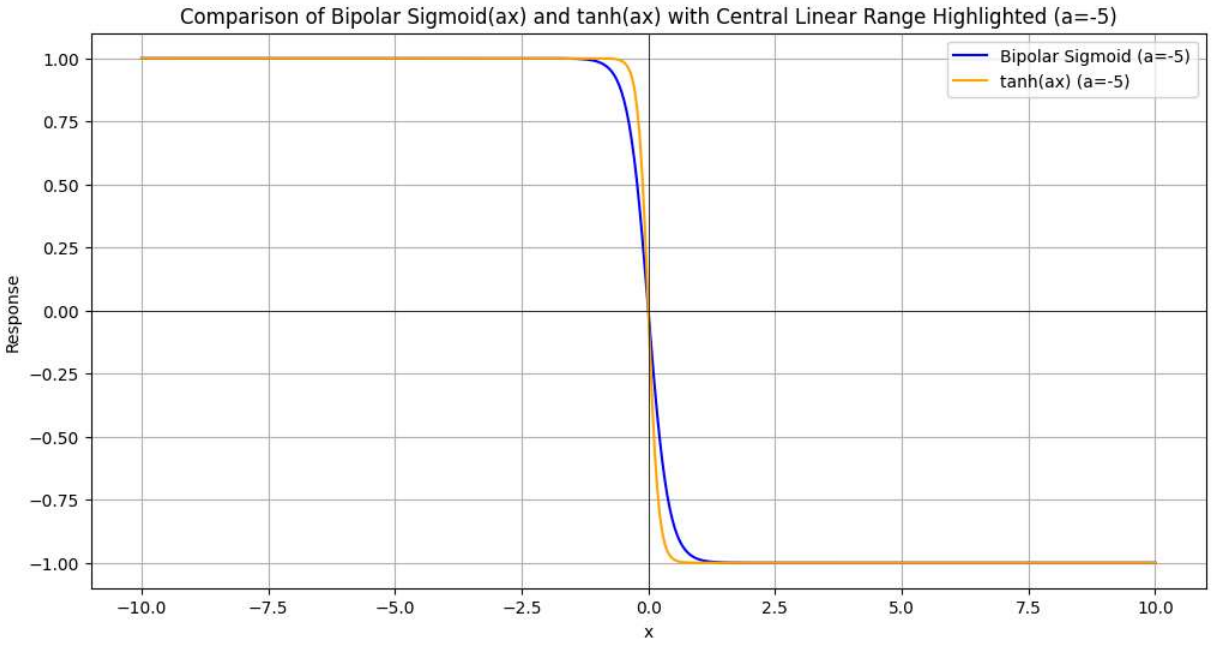
```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

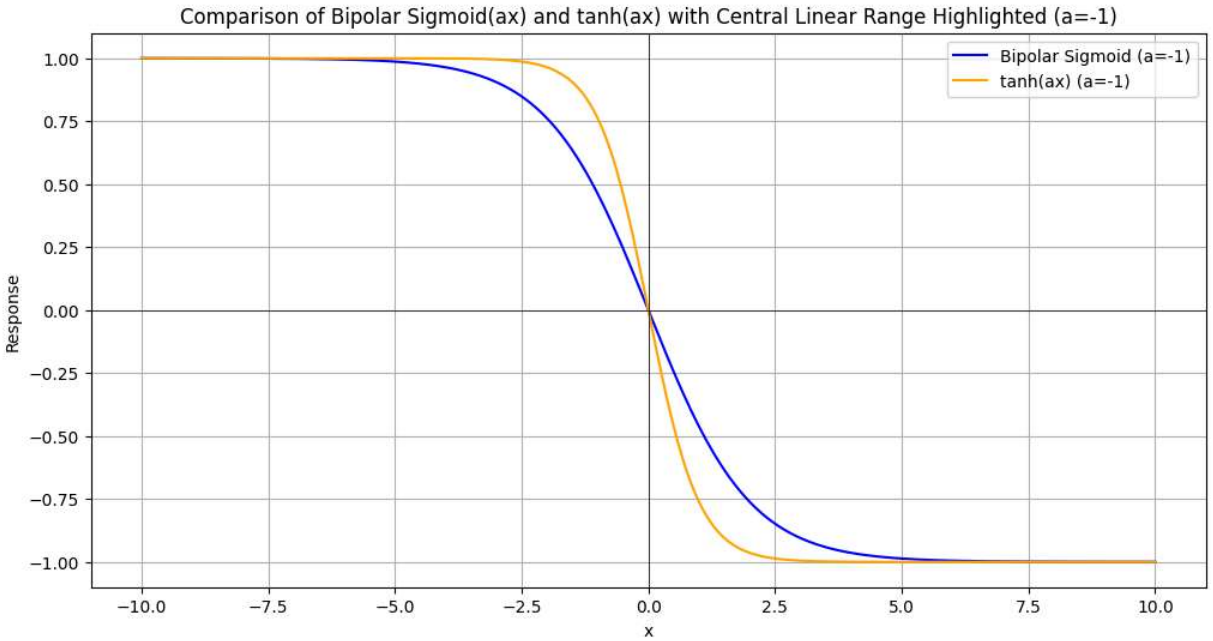


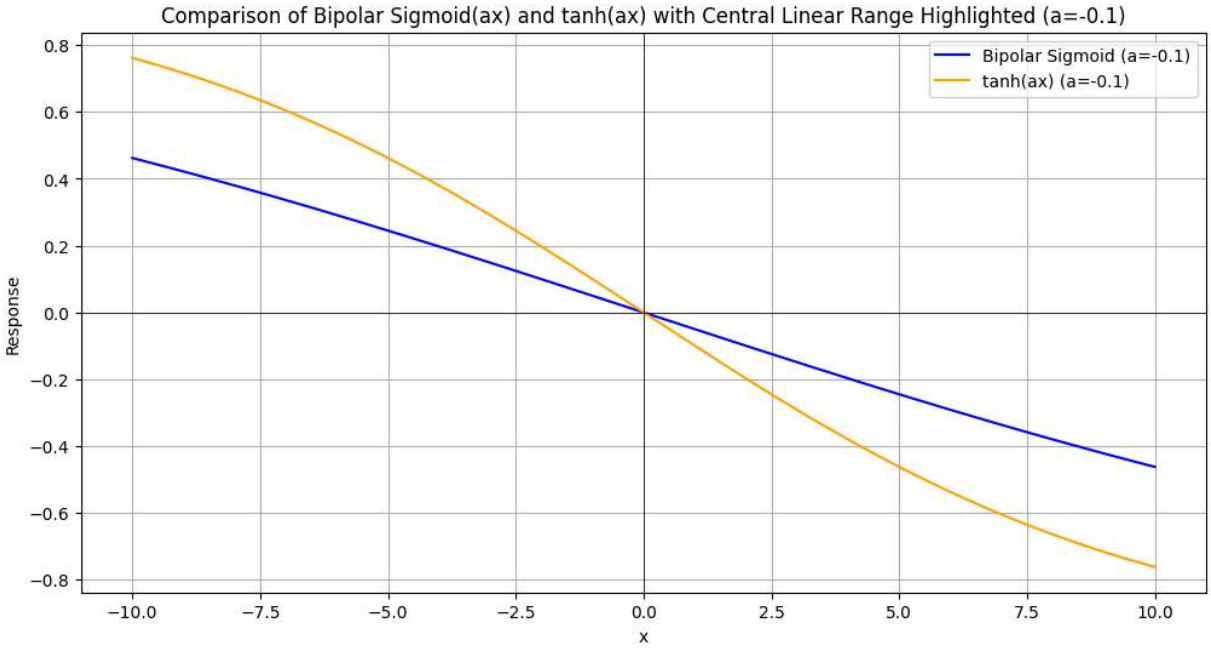
In [ ]:

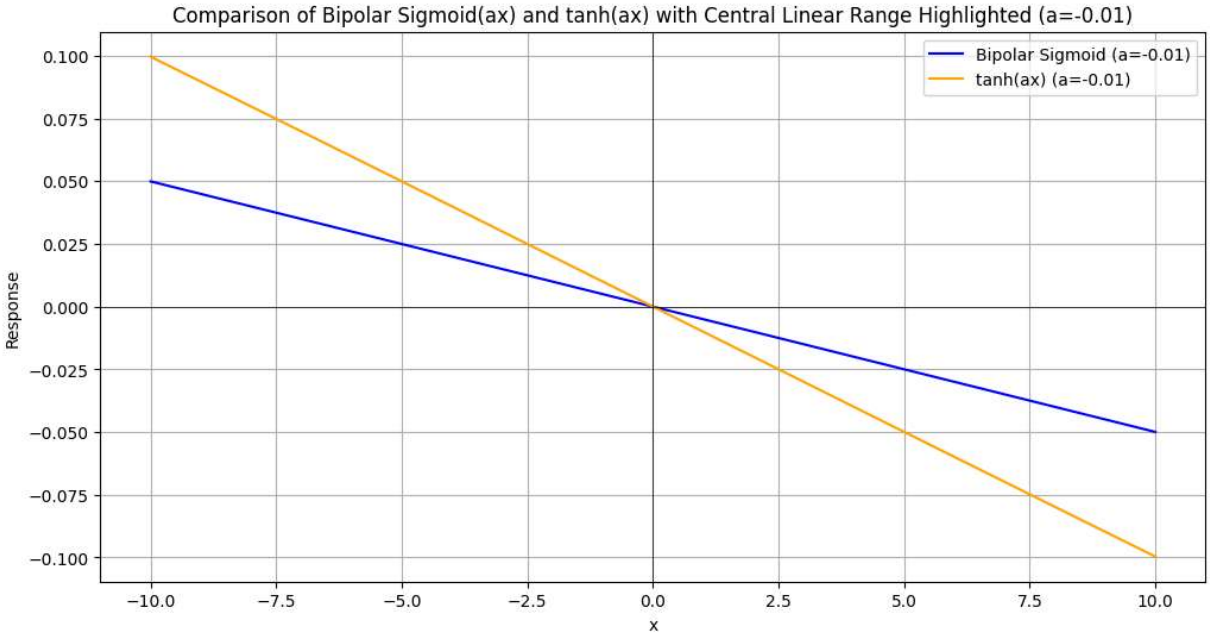
```python
import numpy as np
import matplotlib.pyplot as plt

# Unipolar sigmoid function
def sigmoid_function(x):
    return 1 / (1 + np.exp(-x))

# Bipolar sigmoid function
def bipolar_sigmoid_function(x):
    return 2 * sigmoid_function(x) - 1

# Numerical derivative function
def compute_numerical_derivative(func, x, h=1e-5):
    return (func(x + h) - func(x - h)) / (2 * h)

# Define x range and 'a' values
x_values = np.linspace(-10, 10, 400)
a_values = [-5, -1, -0.1, -0.01, 0.001, 0.01, 0.1, 1, 5]

central_range = (-5, 5)

for a in a_values:
    bipolar_sigmoid_a = bipolar_sigmoid_function(a * x_values)
    tanh_a = np.tanh(a * x_values)

    # Calculate derivative
    derivative = compute_numerical_derivative(lambda x: bipolar_sigmoid_function(a

    # Find linear region within central range where derivative is close to 0
    central_indices = np.where((x_values > central_range[0]) & (x_values < central_
    linear_indices = np.where((derivative > -0.1) & (derivative < 0.1))[0]

    central_linear_indices = np.intersect1d(central_indices, linear_indices)
```

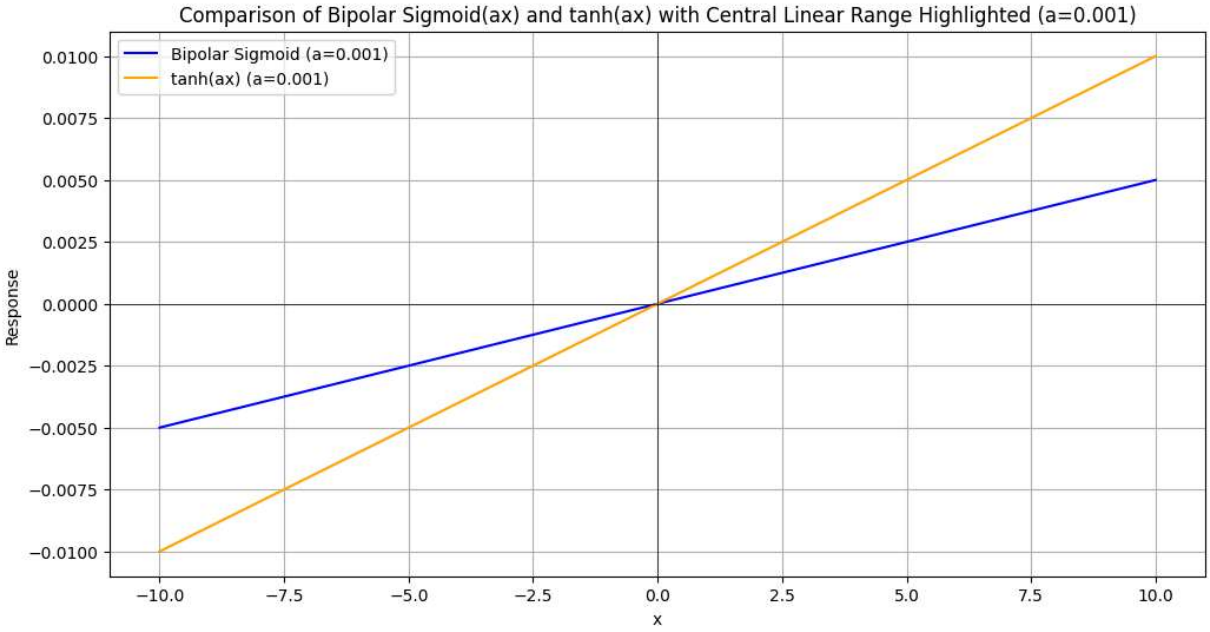
```
    central_linear_x = x_values[central_linear_indices]

    plt.figure(figsize=(12, 6))

    plt.plot(x_values, bipolar_sigmoid_a, label=f'Bipolar Sigmoid (a={a})', color='
    plt.plot(x_values, tanh_a, label=f'tanh(ax) (a={a})', color='orange')

    plt.title(f"Comparison of Bipolar Sigmoid(ax) and tanh(ax) with central linear
    plt.xlabel("x")
    plt.ylabel("Response")
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)
    plt.legend()
    plt.grid(True)
    plt.show()
```

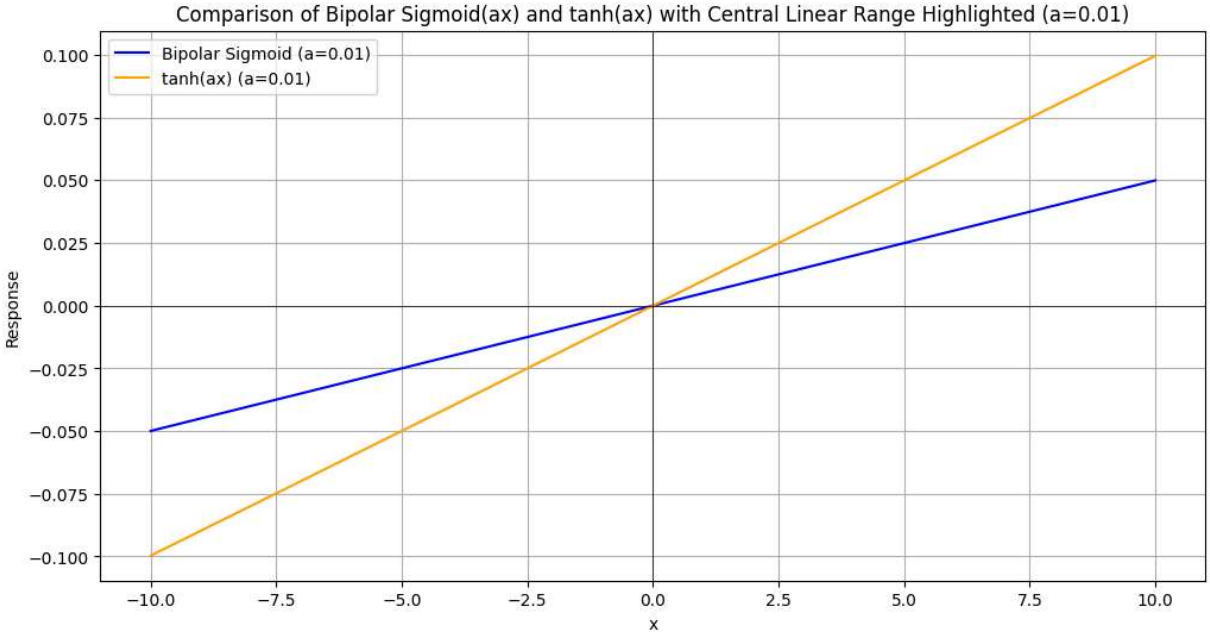


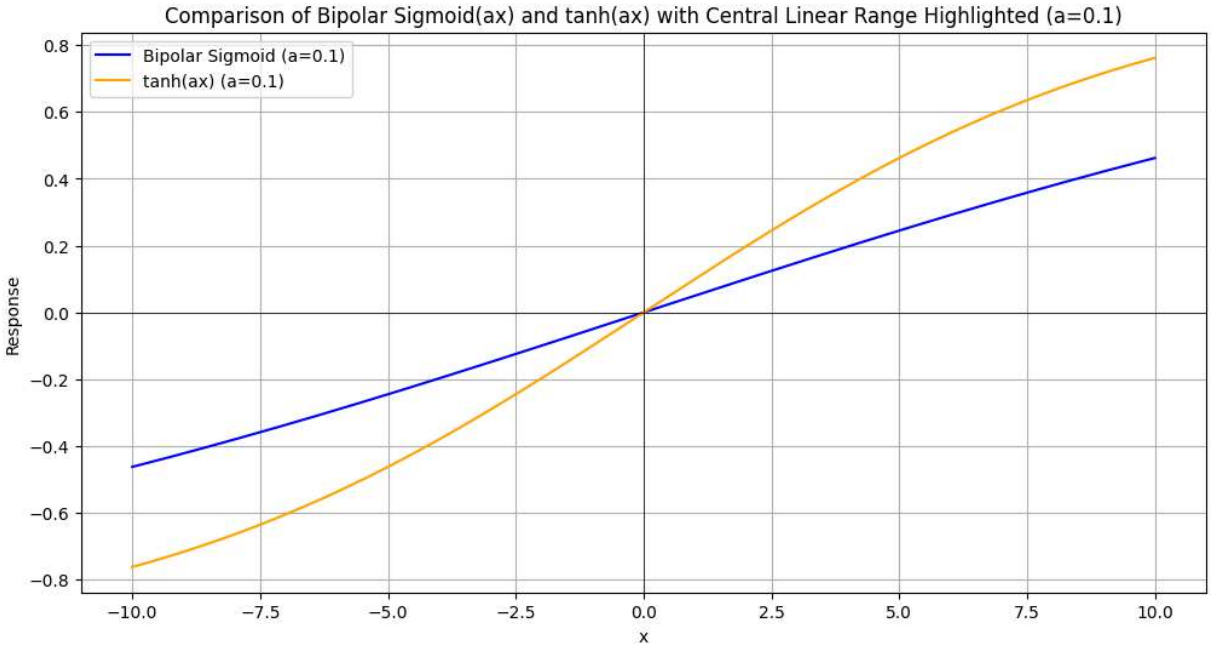


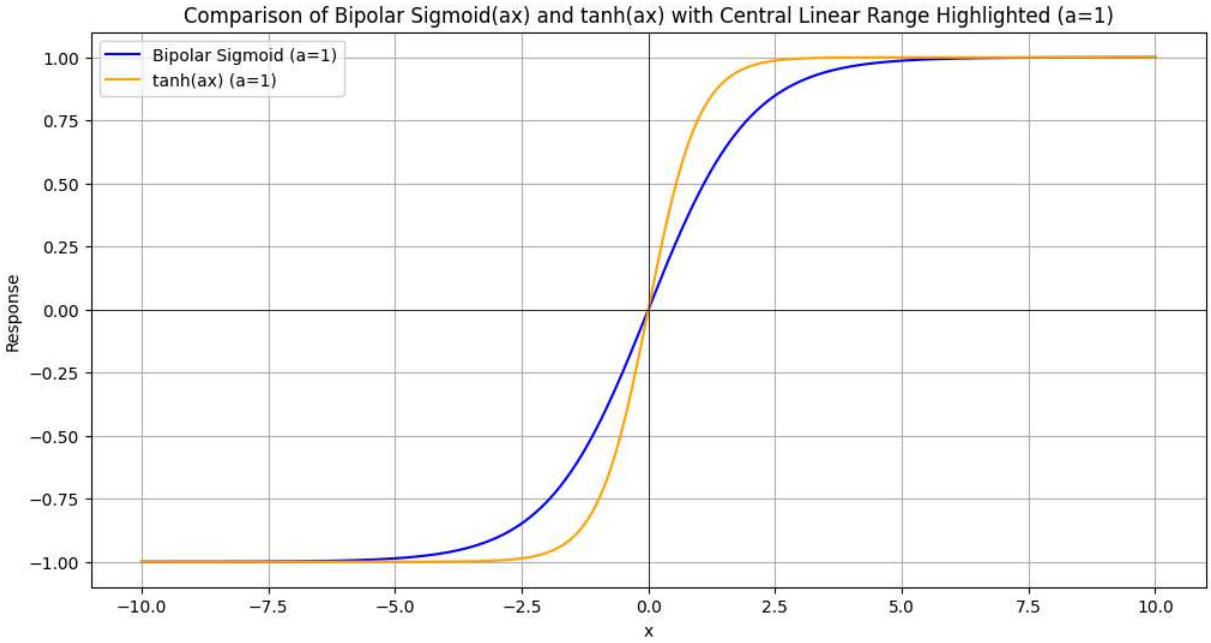

Comparison of Bipolar Sigmoid(ax) and tanh(ax) with Central Linear Range Highlighted (a=-0.1)
Bipolar Sigmoid (a=-0.1)
tanh(ax) (a=-0.1)
Response
x
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

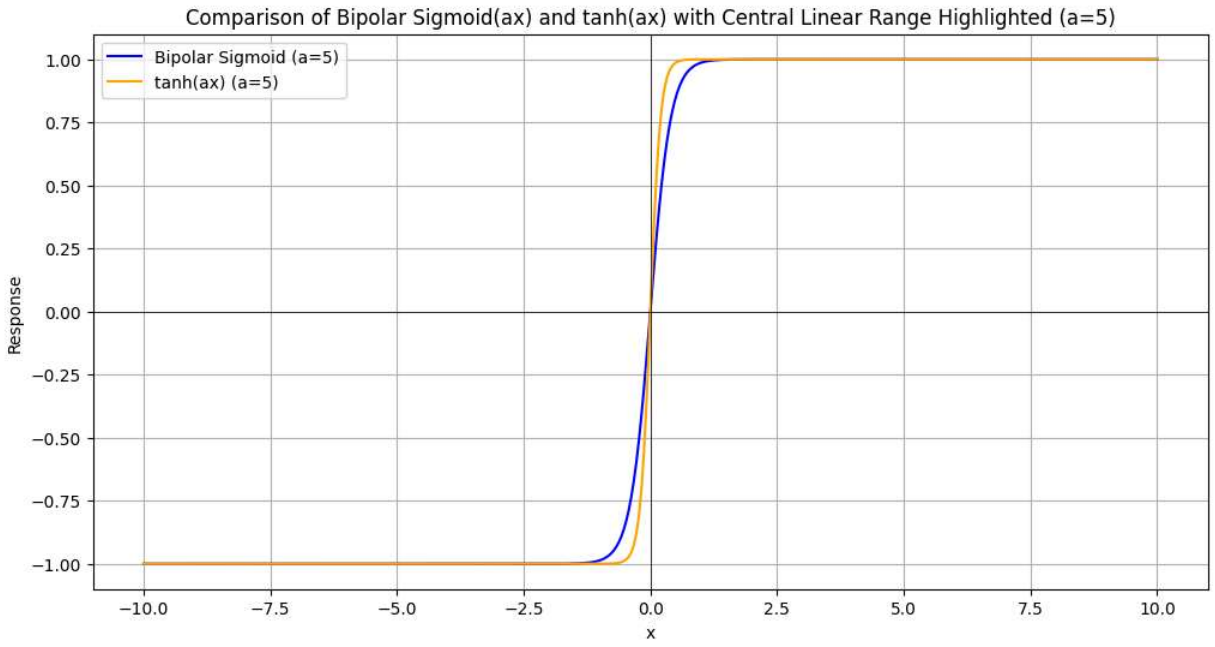

Comparison of Bipolar Sigmoid(ax) and tanh(ax) with Central Linear Range Highlighted (a=-0.01)
Bipolar Sigmoid (a=-0.01)
tanh(ax) (a=-0.01)
Response
x
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Comparison of Bipolar Sigmoid(ax) and tanh(ax) with Central Linear Range Highlighted (a=0.001)
Bipolar Sigmoid (a=0.001)
tanh(ax) (a=0.001)
Response
x
0.0100
0.0075
0.0050
0.0025
0.0000
−0.0025
−0.0050
−0.0075
−0.0100
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Comparison of Bipolar Sigmoid(ax) and tanh(ax) with Central Linear Range Highlighted (a=0.01)
Bipolar Sigmoid (a=0.01)
tanh(ax) (a=0.01)
Response
x
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Comparison of Bipolar Sigmoid(ax) and tanh(ax) with Central Linear Range Highlighted (a=0.1)
Bipolar Sigmoid (a=0.1)
tanh(ax) (a=0.1)
Response
x
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Comparison of Bipolar Sigmoid(ax) and tanh(ax) with Central Linear Range Highlighted (a=1)
Bipolar Sigmoid (a=1)
tanh(ax) (a=1)
Response
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

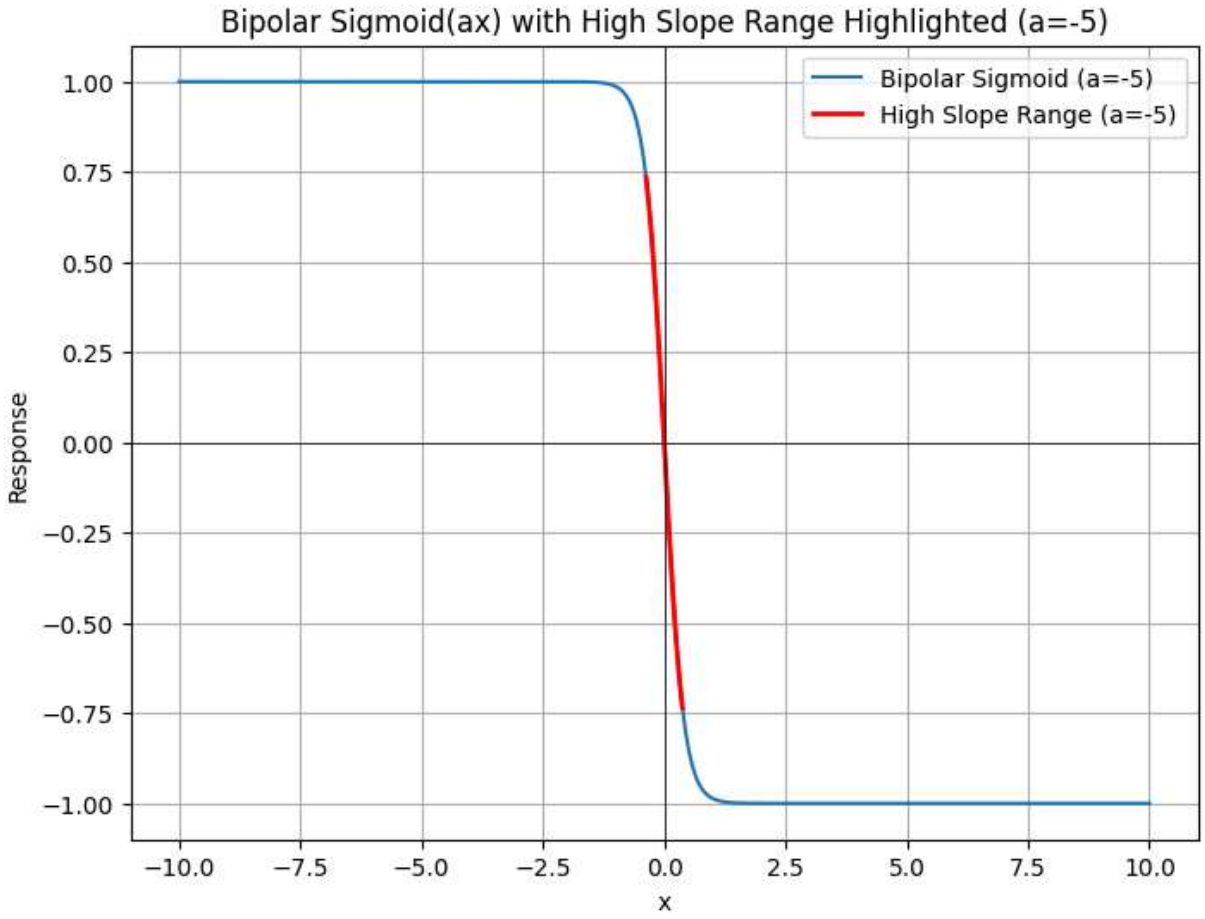


In [ ]:

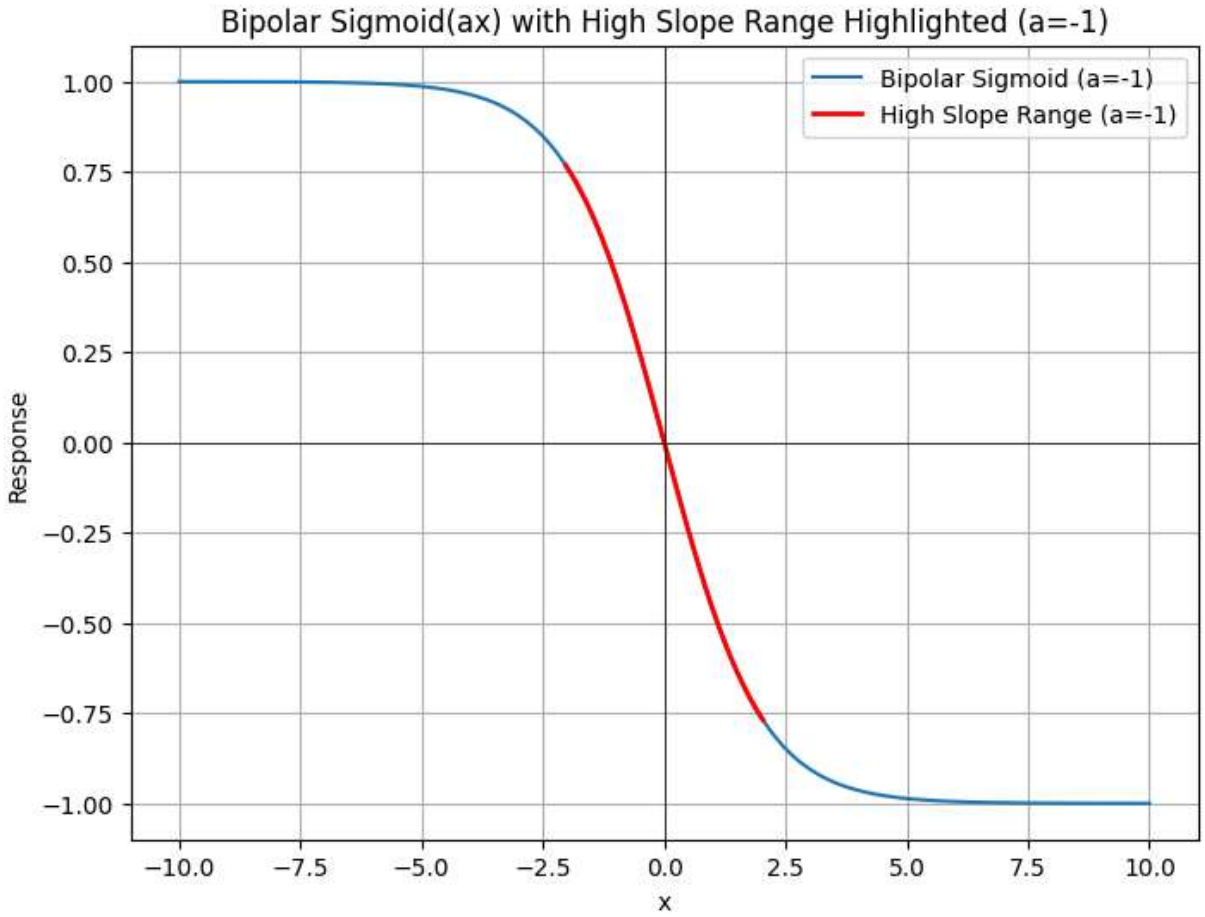
```python
import numpy as np
import matplotlib.pyplot as plt

# Unipolar sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Bipolar sigmoid function
def bipolar_sigmoid(x):
    return 2 * sigmoid(x) - 1

# Numerical derivative function
def numerical_derivative(f, x, h=1e-5):
    return (f(x + h) - f(x - h)) / (2 * h)

# Range of x and values of 'a'
x = np.linspace(-10, 10, 400)
a_values = [-5, -1, -0.1, -0.01, 0.001, 0.01, 0.1, 1, 5]

central_region = (-5, 5)



# Iterate over each 'a' value
for a in a_values:

    high_slope_threshold = abs(a)/5
    bipolar_sigmoid_a = bipolar_sigmoid(a * x)
    derivative = numerical_derivative(lambda x: bipolar_sigmoid(a * x), x)

    central_indices = np.where((x > central_region[0]) & (x < central_region[1]))[0
    high_slope_indices = np.where(np.abs(derivative) > high_slope_threshold)[0]

    high_slope_range = np.intersect1d(central_indices, high_slope_indices)

    # Get the corresponding x values for the high slope range
```

```
    high_slope_x_values = x[high_slope_range]

    if len(high_slope_x_values) > 0:
        print(f"High slope range for a={a}: x in [{high_slope_x_values[0]:.2f}, {hi
    else:
        print(f"High slope range for a={a}: No significant high slope region in the


    plt.figure(figsize=(8, 6))
    plt.plot(x, bipolar_sigmoid_a, label=f'Bipolar Sigmoid (a={a})')

    # Highlight the high slope range on the plot
    if len(high_slope_x_values) > 0:
        plt.plot(high_slope_x_values, bipolar_sigmoid_a[high_slope_range], color='r

    plt.title(f"Bipolar Sigmoid(ax) with High Slope Range (a={a})")
    plt.xlabel("x")
    plt.ylabel("Response")
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)
    plt.legend()
    plt.grid(True)
    plt.show()
```

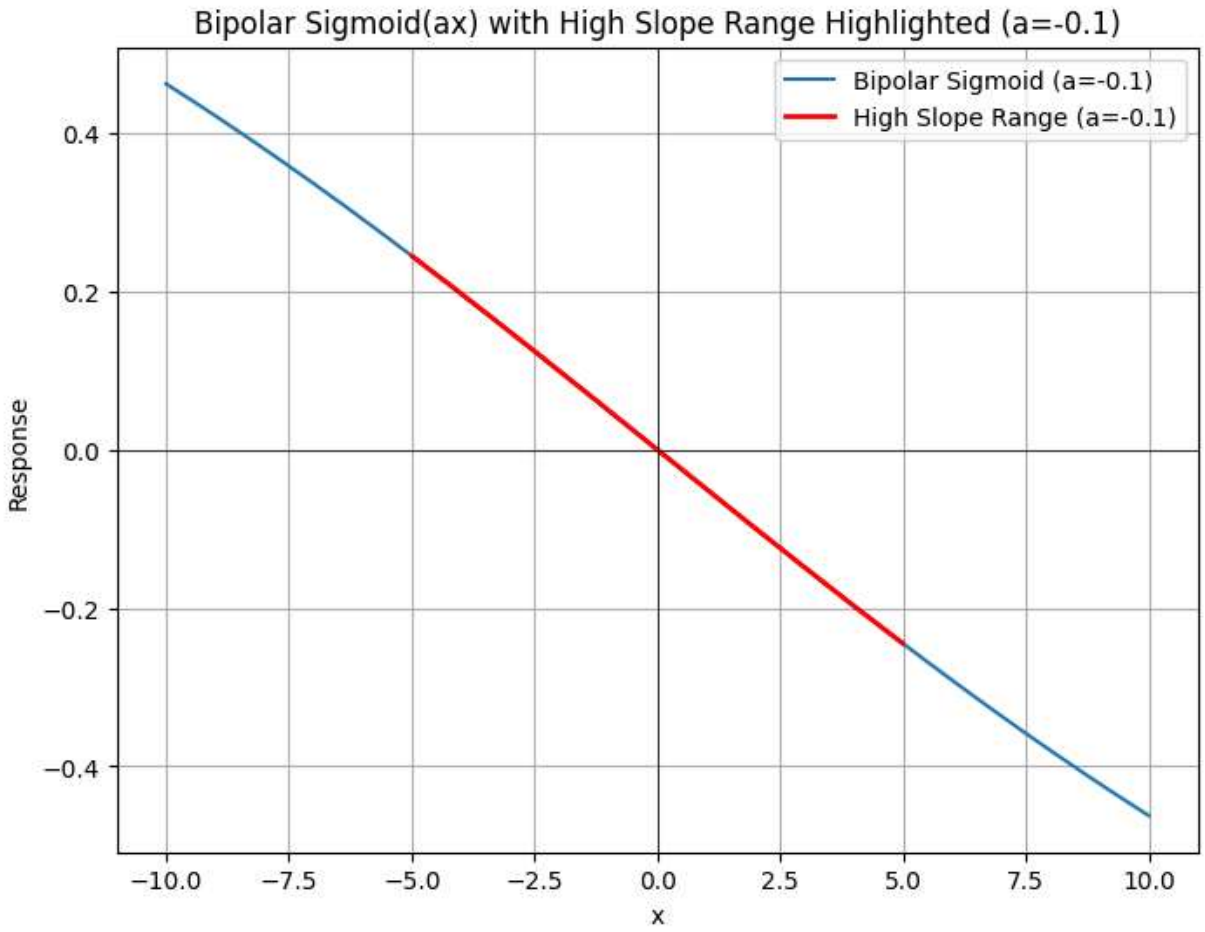
```
High slope range for a=-5: x in [-0.38, 0.38]
```



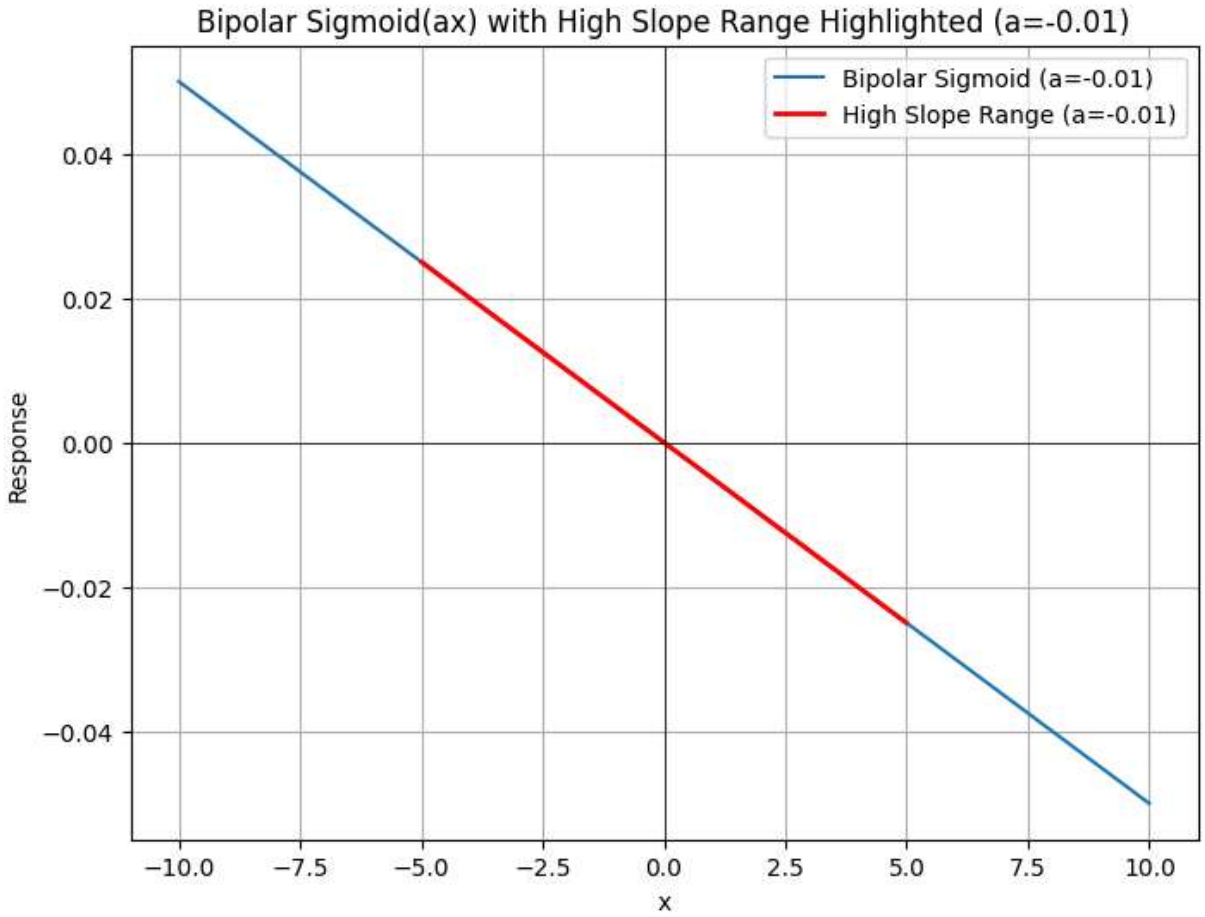

```
High slope range for a=-1: x in [-2.03, 2.03]
```



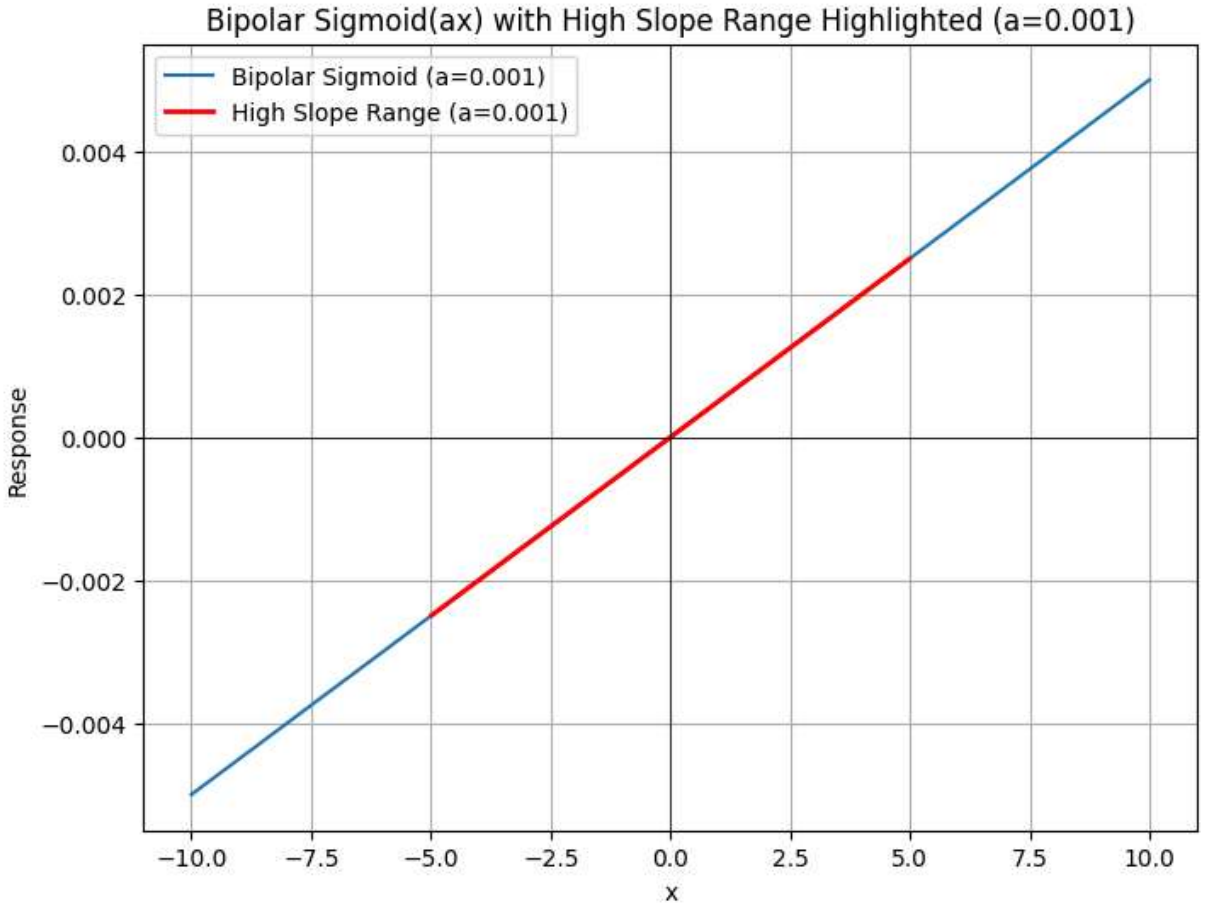

```
High slope range for a=-0.1: x in [-4.99, 4.99]
```



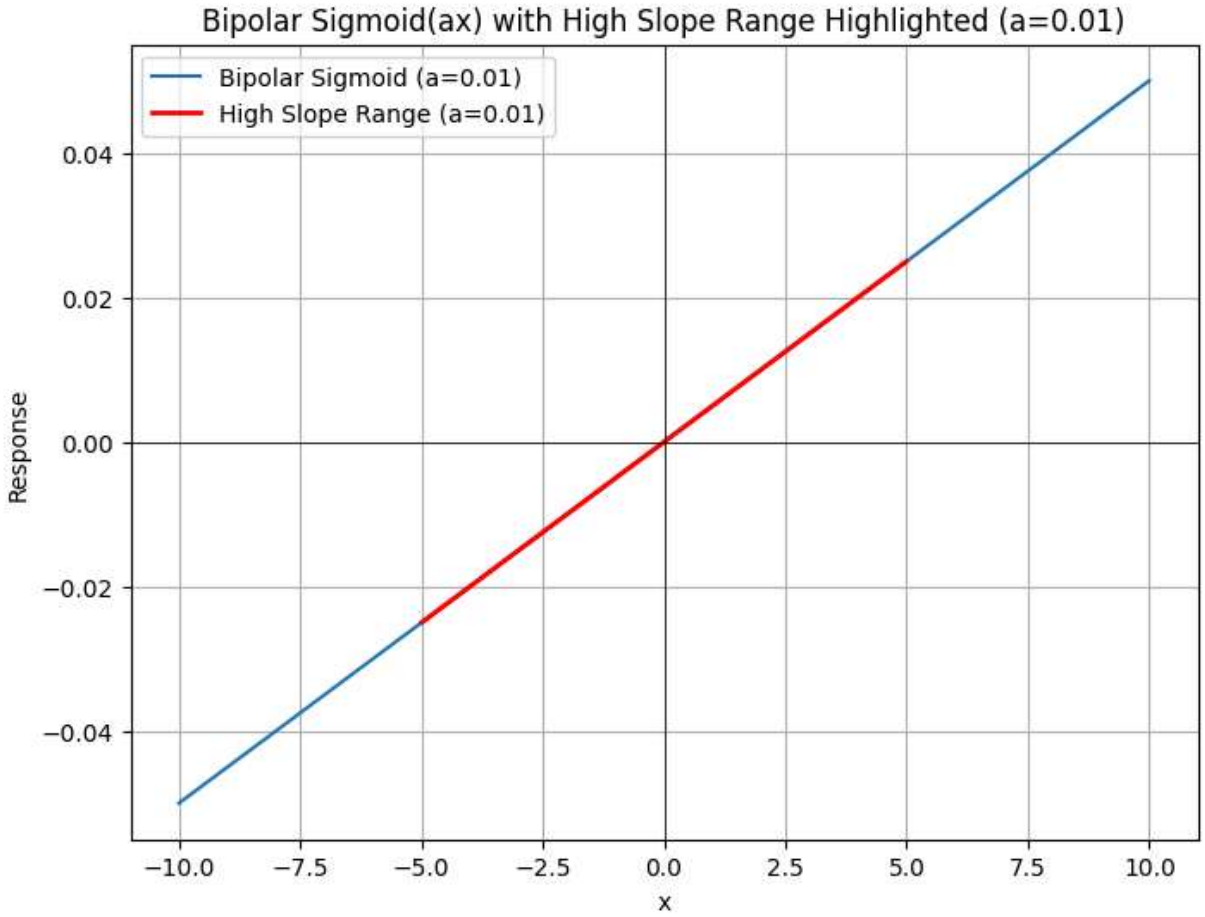

```
High slope range for a=-0.01: x in [-4.99, 4.99]
```



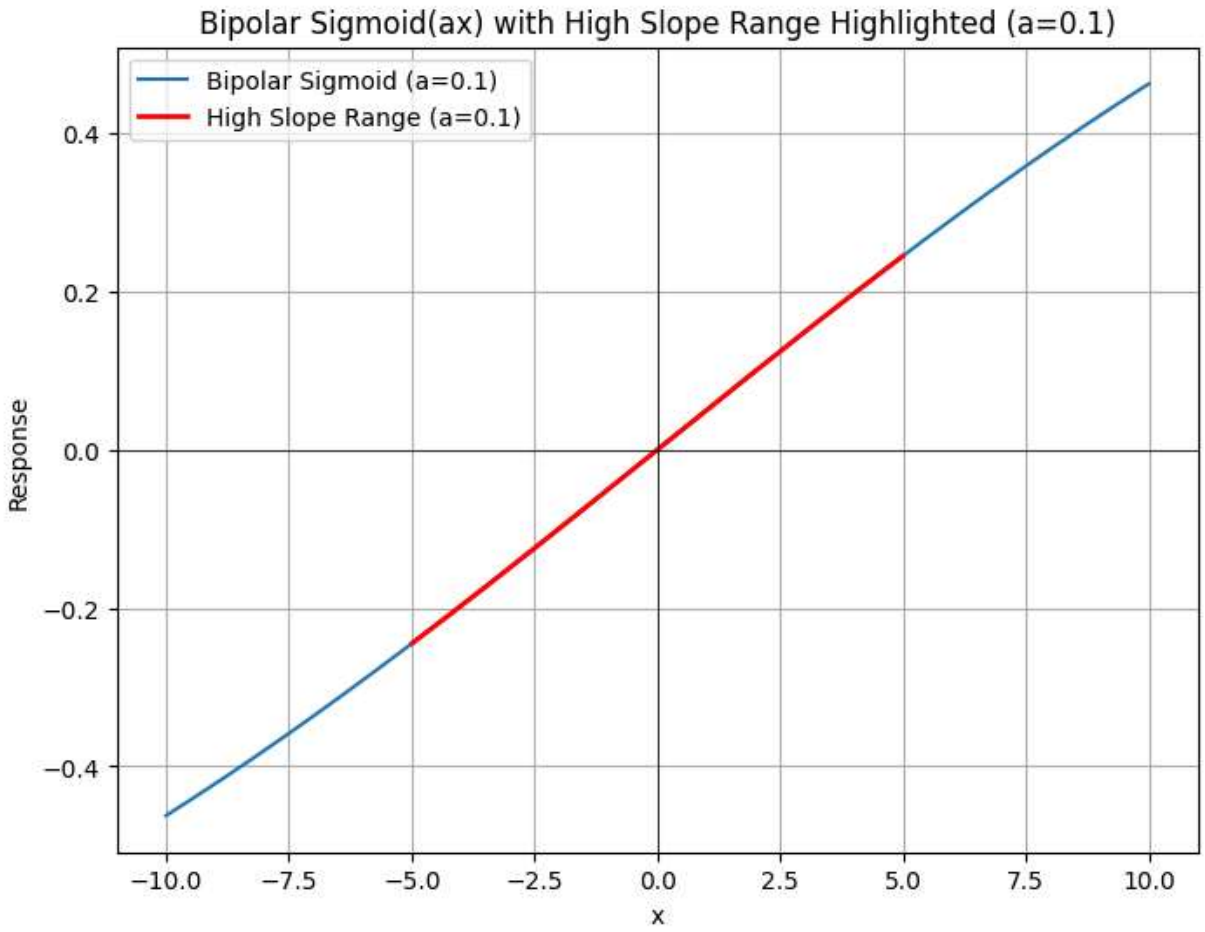
High slope range for a=0.001: x in [-4.99, 4.99]



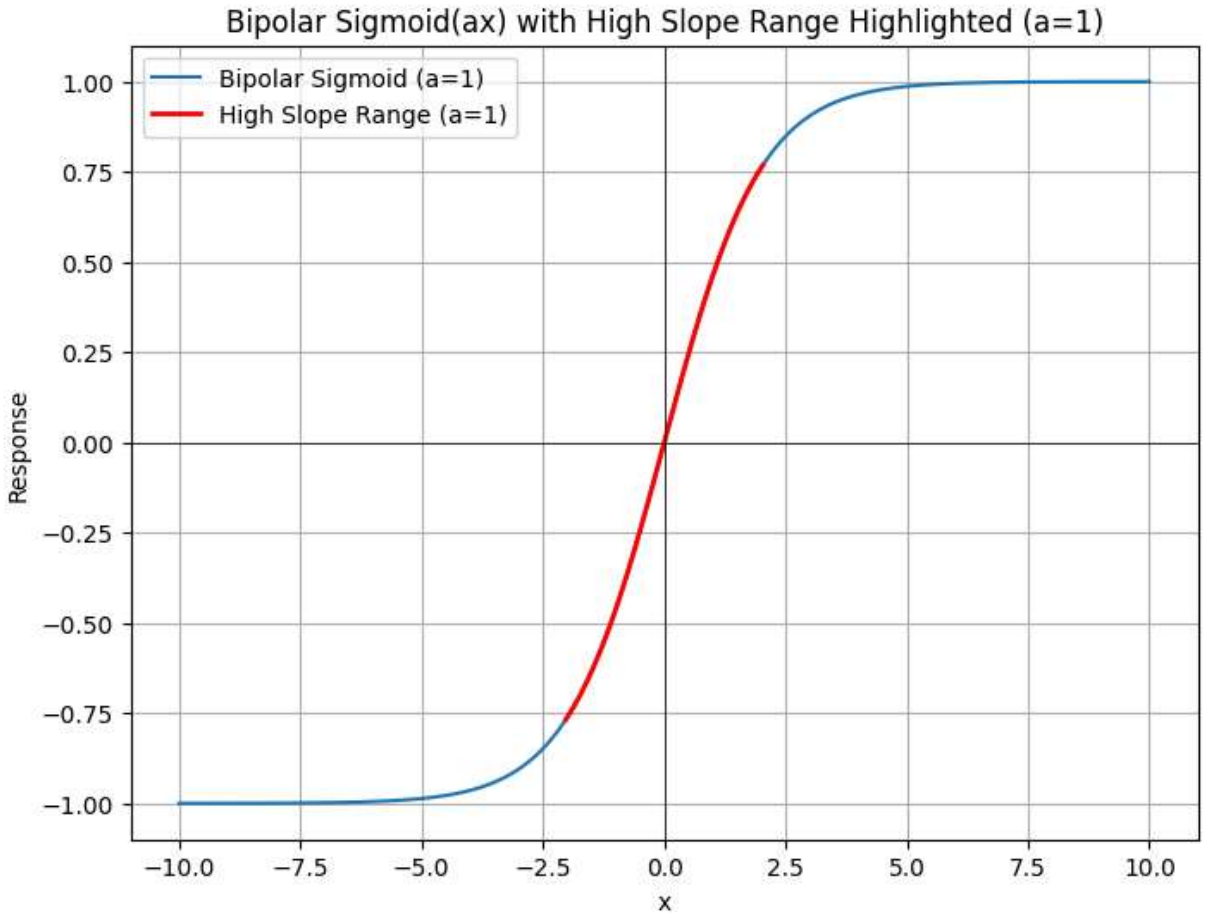
High slope range for a=0.01: x in [-4.99, 4.99]



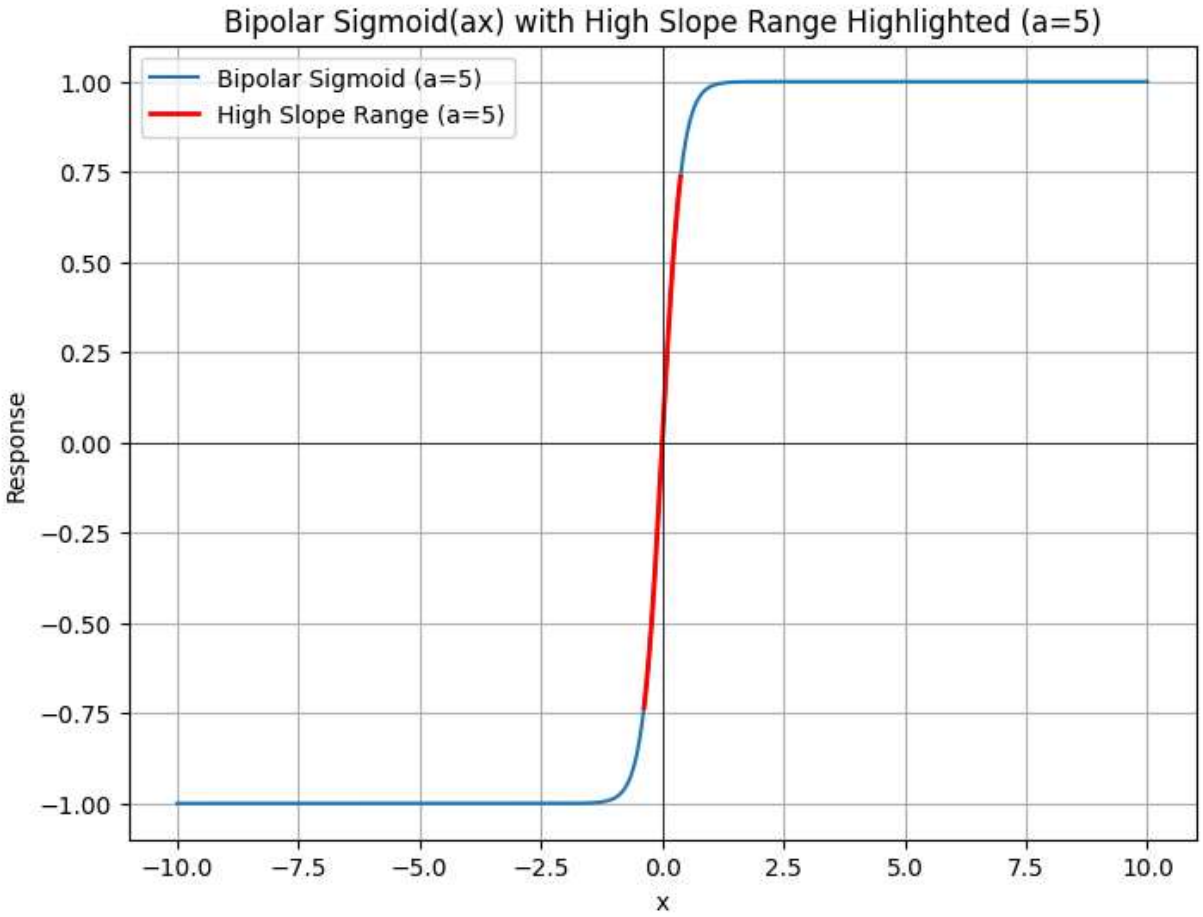
High slope range for a=0.1: x in [-4.99, 4.99]




```
High slope range for a=1: x in [-2.03, 2.03]
```




```
High slope range for a=5: x in [-0.38, 0.38]
```


Bipolar Sigmoid(ax) with High Slope Range Highlighted (a=5)
Bipolar Sigmoid (a=5)
High Slope Range (a=5)
Response
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0