Student name: Thi Vu
Student number: 151394898

DATA.STAT.770 Dimensionality Reduction and Visualization, Spring 2025
Exercise set 1

Exercise 1:
Code:

```python
import numpy as np

def compute_proportions(dimensions, num_points=10_000_000):
    results_hypersphere = {}
    results_shell = {}

    for d in dimensions:
        # Generate points uniformly in [-1, 1]^d
        points = np.random.uniform(-1, 1, size=(num_points, d))

        # Compute squared distances from the origin
        squared_distances = np.sqrt(np.sum(points**2, axis=1))

        # a) Proportion inside the hypersphere
        inside_hypersphere = squared_distances <= 1
        proportion_hypersphere = np.mean(inside_hypersphere)
        results_hypersphere[d] = proportion_hypersphere

        # b) Proportion inside the spherical shell
        inside_shell = (squared_distances >= 0.95) & (squared_distances <= 1)
        proportion_shell = np.sum(inside_shell) / np.sum(inside_hypersphere)
        results_shell[d] = proportion_shell

    return results_hypersphere, results_shell

# Dimensions to test
dimensions = [1, 2, 3, 4, 7, 11, 16]

# Run the computation
proportions_hypersphere, proportions_shell = compute_proportions(dimensions)

# Display results
# a) Proportion of points inside the hypersphere
print("Proportion of points inside the hypersphere:")
for d, proportion in proportions_hypersphere.items():
    print(f"Dimension {d}: {proportion:.6f}")

# b) Proportion of points inside the spherical shell
print("\nProportion of points inside the spherical shell:")
for d, proportion in proportions_shell.items():
    print(f"Dimension {d}: {proportion:.6f}")
```

Result:

a) Proportion of points inside the hypersphere:

```
Dimension 1: 1.000000
Dimension 2: 0.785385
Dimension 3: 0.523193
Dimension 4: 0.308332
Dimension 7: 0.036971
Dimension 11: 0.000925
Dimension 16: 0.000004
```

b) Proportion of points inside the spherical shell:

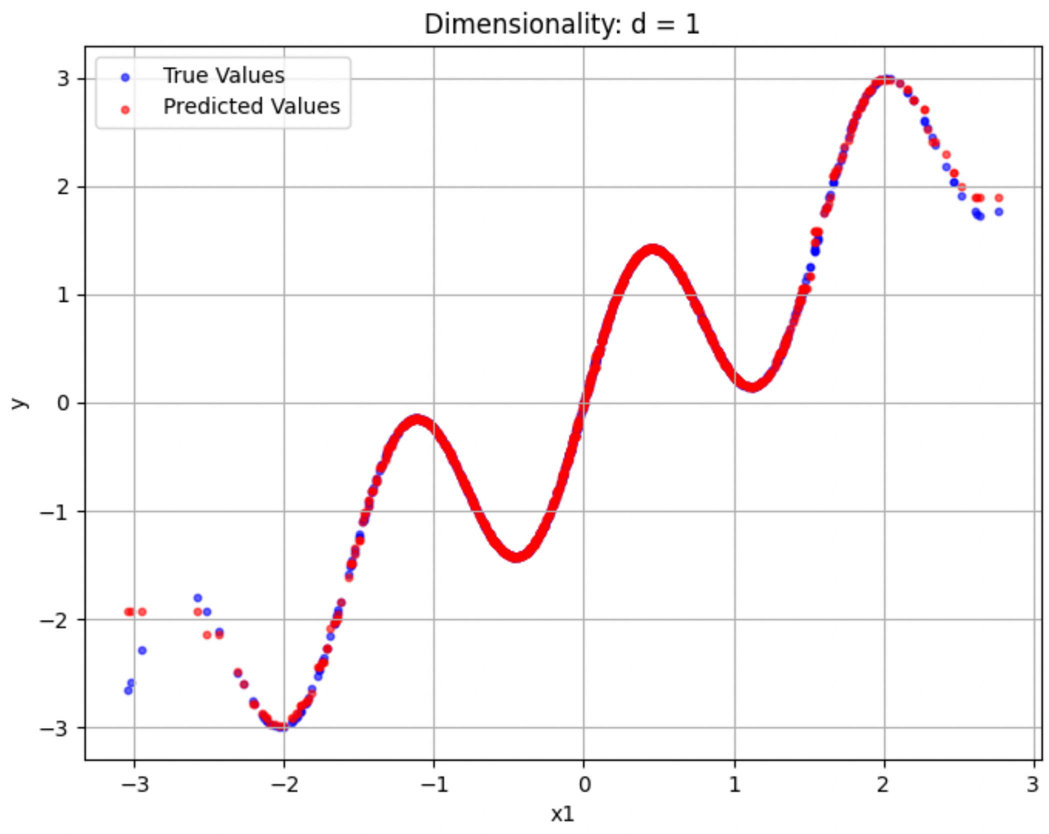
```
Dimension 1: 0.050055
Dimension 2: 0.097479
Dimension 3: 0.142634
Dimension 4: 0.185454
Dimension 7: 0.301673
Dimension 11: 0.420154
Dimension 16: 0.534884
```

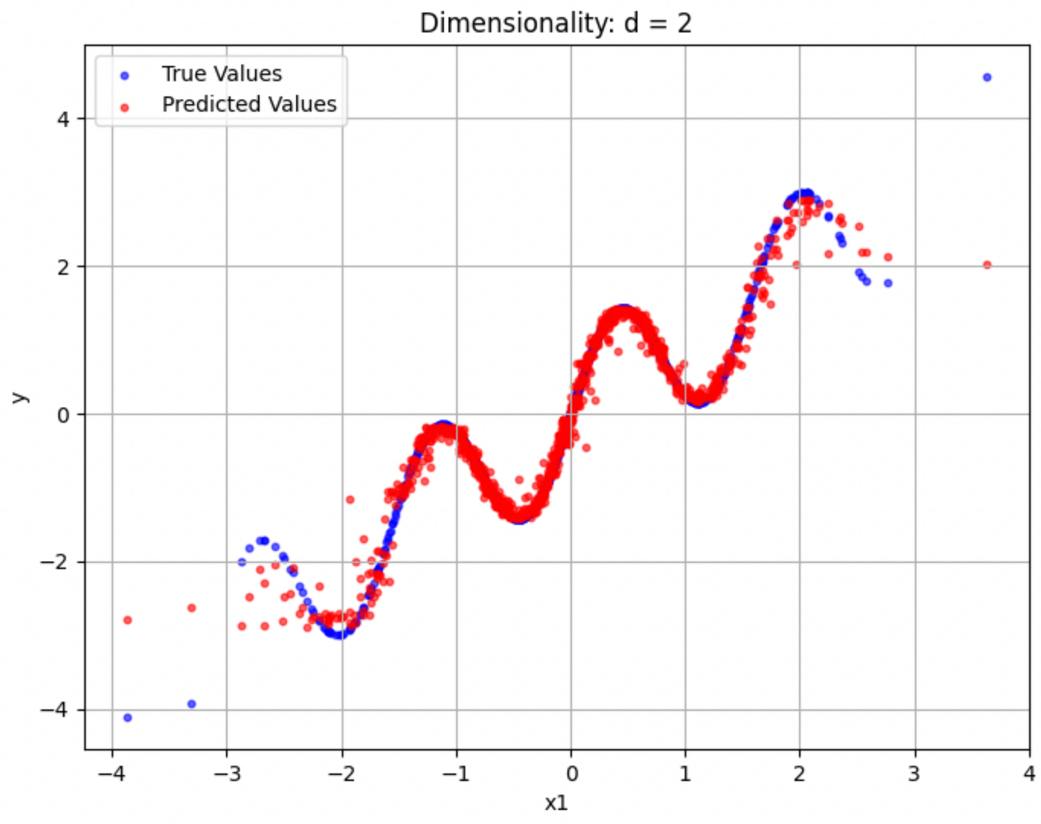
Exercise 2:

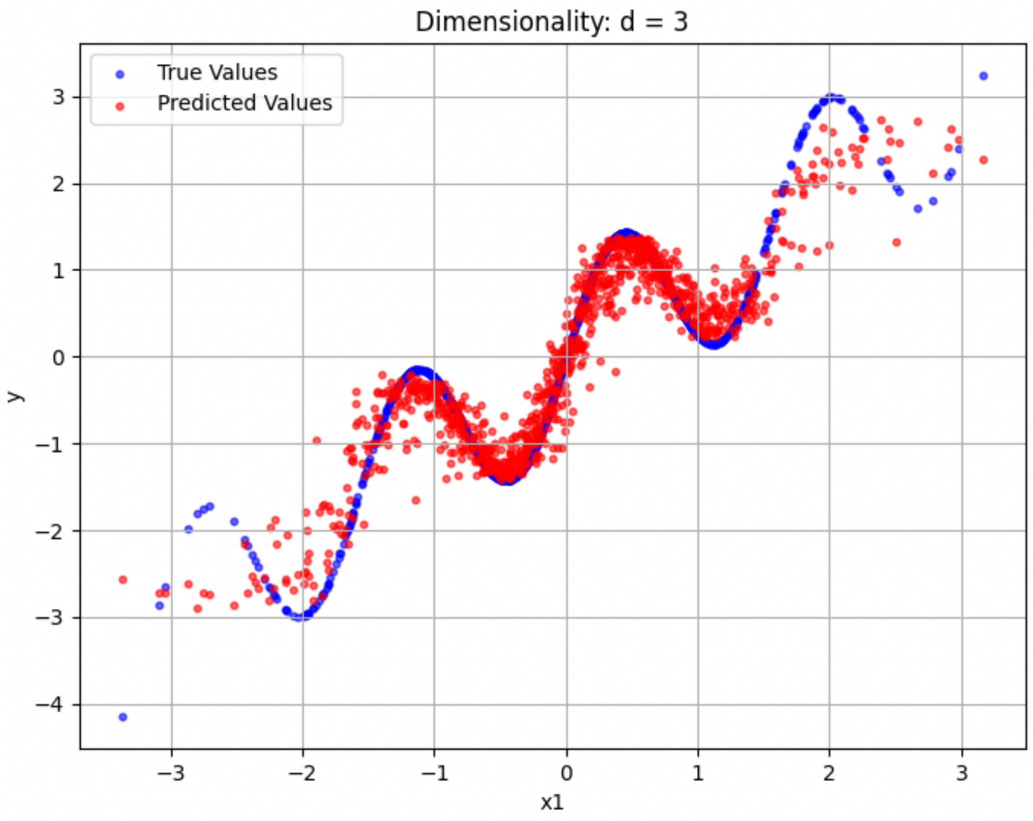
Code:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor

def generate_data(d, num_points=2000):
    # Generate data points normally distributed around the origin
    X = np.random.normal(0, 1, size=(num_points, d))
    # Compute the target variable y
    y = X[:, 0] + np.sin(4 * X[:, 0])
    return X, y

def split_data(X, y):
    # Split into 1000 training and 1000 testing points
    return X[:1000], y[:1000], X[1000:], y[1000:]

def train_and_predict_knn(X_train, y_train, X_test, n_neighbors=5):
    # Train a 5-Nearest Neighbor Regressor
    knn = KNeighborsRegressor(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # Predict on the test set
    y_pred = knn.predict(X_test)
    return y_pred

def compute_mse(y_true, y_pred):
    # Compute Mean Squared Error
    return np.mean((y_true - y_pred)**2)

def plot_results(x1_test, y_test, y_pred, d):
    plt.figure(figsize=(8, 6))
    plt.scatter(x1_test, y_test, color='blue', label='True Values', alpha=0.6,
s=10)
    plt.scatter(x1_test, y_pred, color='red', label='Predicted Values', alpha=0.6,
s=10)
    plt.title(f"Dimensionality: d = {d}")
    plt.xlabel("x1")
    plt.ylabel("y")
    plt.legend()
    plt.grid(True)
    plt.show()

def main():
    dimensions = [1, 2, 3, 4, 7, 11, 16]
    mse_results = {}

    for d in dimensions:
        # Generate data
        X, y = generate_data(d)

        # Split into training and testing sets
        X_train, y_train, X_test, y_test = split_data(X, y)

        # Train and predict using 5-NN
        y_pred = train_and_predict_knn(X_train, y_train, X_test)

        # Compute mean squared error
        mse = compute_mse(y_test, y_pred)
        mse_results[d] = mse

        # Plot results
        plot_results(X_test[:, 0], y_test, y_pred, d)

    # Print MSE results
    print("Mean Squared Error for each dimensionality:")
    for d, mse in mse_results.items():
        print(f"Dimension {d}: MSE = {mse:.6f}")

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

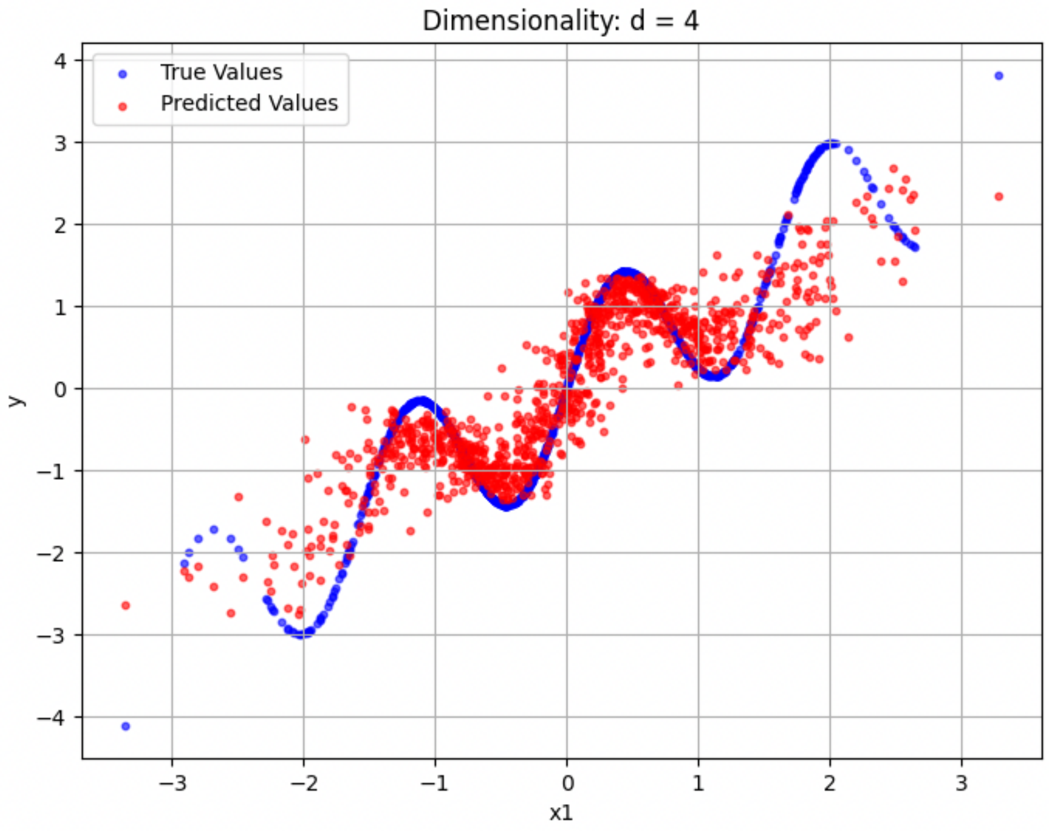
The plot:


Dimensionality: d = 1
True Values
Predicted Values
y
x1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

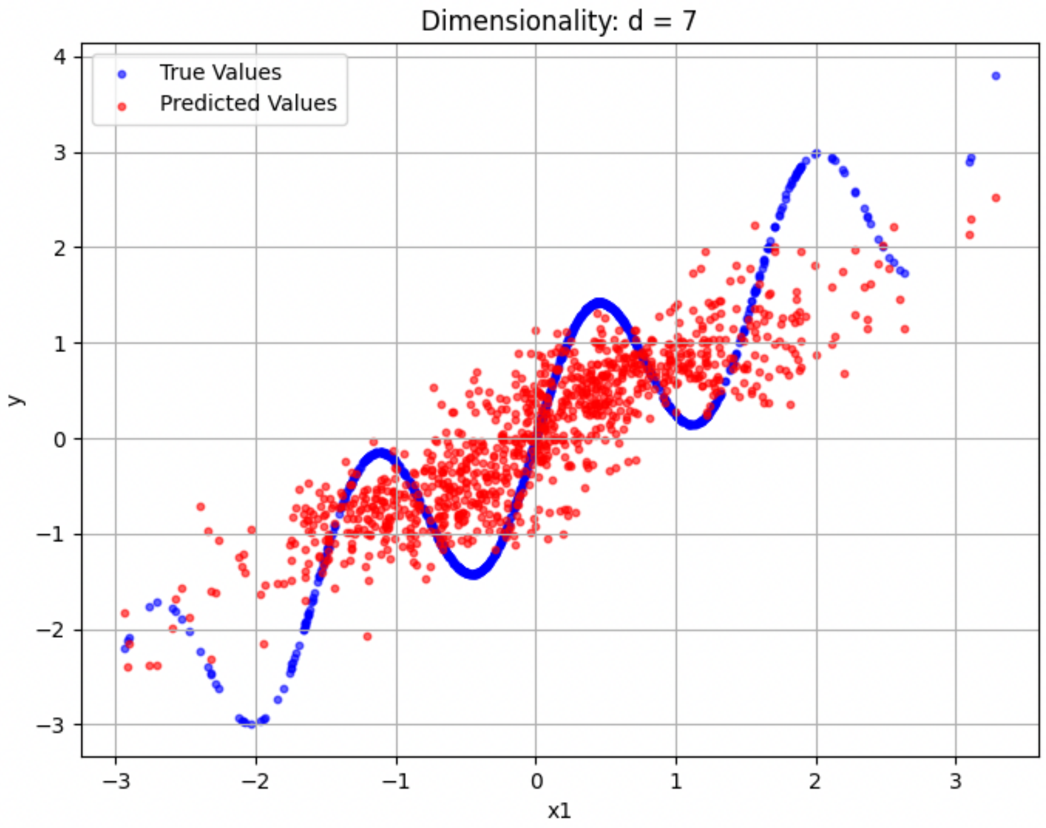

Dimensionality: d = 2
True Values
Predicted Values
y
x1
4
2
0
−2
−4
−4
−3
−2
−1
0
1
2
3
4

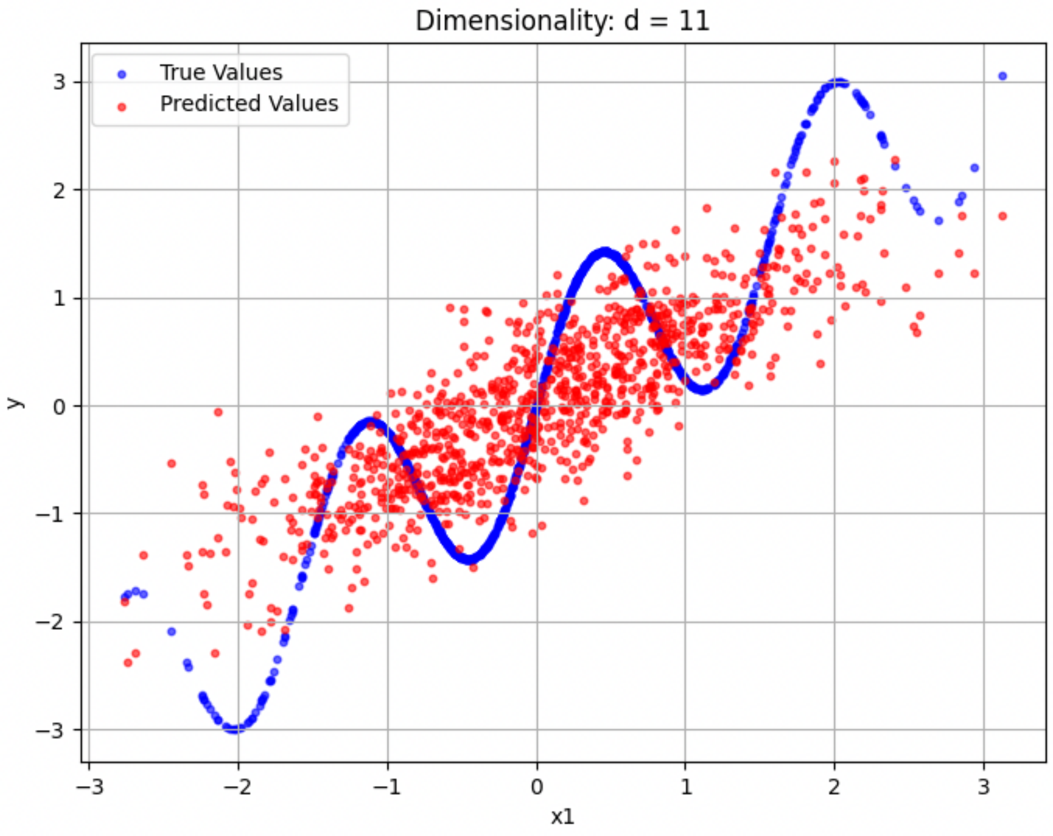


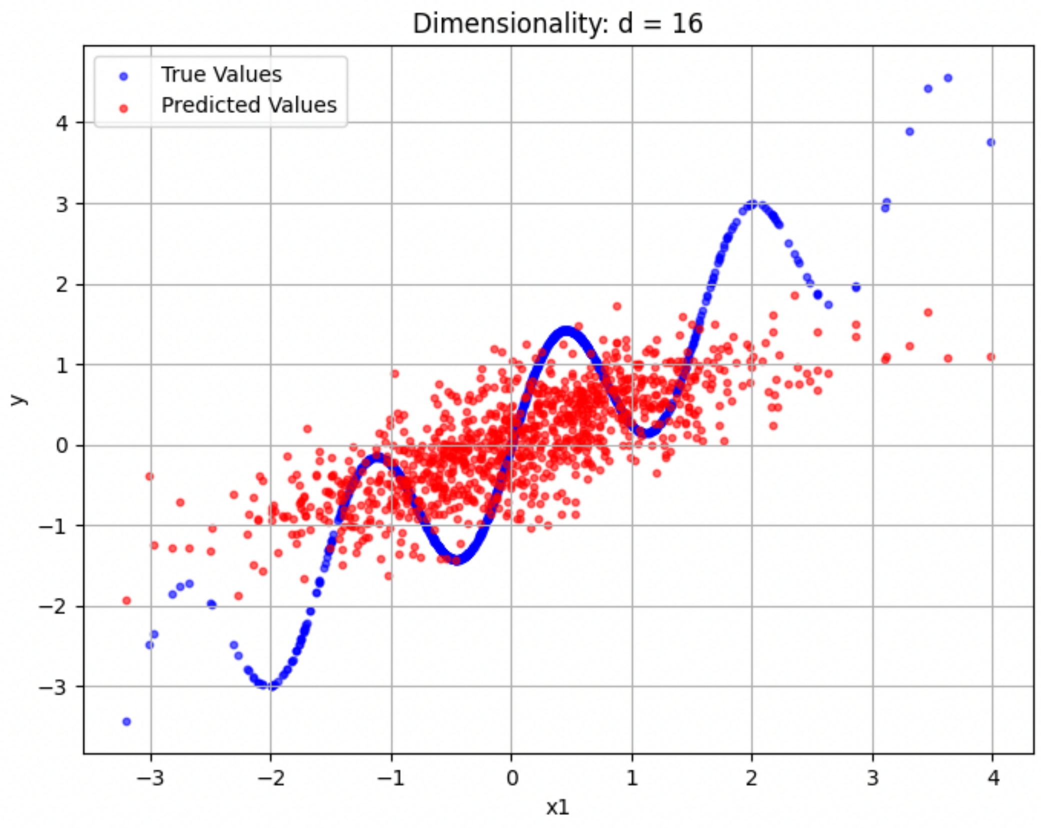
Dimensionality: d = 3
True Values
Predicted Values
y
x1
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3



Dimensionality: d = 4
True Values
Predicted Values
y
x1
4
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3


Dimensionality: d = 7
True Values
Predicted Values
y
x1
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Dimensionality: d = 11
True Values
Predicted Values
y
x1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3



```
Mean Squared Error for each dimensionality:
Dimension 1: MSE = 0.001607
Dimension 2: MSE = 0.043631
Dimension 3: MSE = 0.143781
Dimension 4: MSE = 0.362643
Dimension 7: MSE = 0.654743
Dimension 11: MSE = 0.762133
Dimension 16: MSE = 0.979365
```