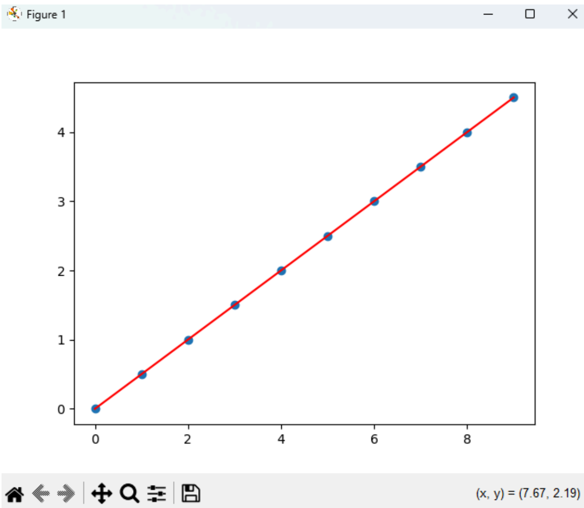
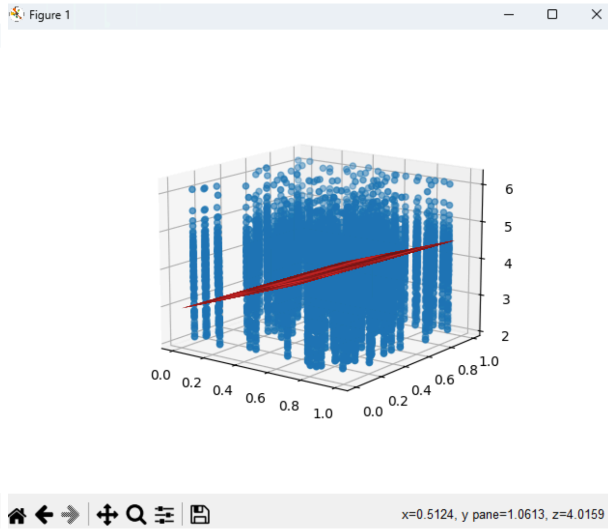
Question 1:

a)






```python
if __name__ == '__main__':
    # Load data
    train_X_1d, train_Y_1d = load_data('1D-no-noise-lin.txt')
    train_X_2d, train_Y_2d = load_data('2D-noisy-lin.txt')

    # Fit the models using closed form solution
    Theta_1d = linreg_closed_form(train_X_1d, train_Y_1d)
    Theta_2d = linreg_closed_form(train_X_2d, train_Y_2d)

    # Calculate the loss for both datasets
    loss_1d = loss(Theta_1d, train_X_1d, train_Y_1d)
    loss_2d = loss(Theta_2d, train_X_2d, train_Y_2d)

    # Plot the models and the data
    vis_linreg_model(train_X_1d, train_Y_1d, Theta_1d)  # For 1D data
    vis_linreg_model(train_X_2d, train_Y_2d, Theta_2d)  # For 2D data

    # Print the results
    print(f"Loss for 1D-no-noise-lin.txt: {loss_1d}")
    print(f"Loss for 2D-noisy-lin.txt: {loss_2d}")
```

```
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML> python .\hw4.py
Loss for 1D-no-noise-lin.txt: 3.851859888774472e-35
Loss for 2D-noisy-lin.txt: 0.10759283475200875
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML>
```

b)Explanation of the Error:

When you duplicate a feature, the matrix: $X^T X$ becomes singular (its determinant is zero), meaning that it cannot be inverted. The closed-form solution for linear regression involves computing the inverse of this matrix, which is why you're seeing this error. There is nothing fundamentally wrong with the math behind the source code. It is simply a feature we must account for in the matrix universe. You can think of it as dividing by 0, except most runtime machines will have exceptions dedicated to these errors. We have yet to "commonly" see built in matrix exceptions in computer science.

```
if __name__ == '__main__':
    # Load data as before
    train_X, train_Y = load_data('2D-noisy-lin.txt')
    (variable) train_X_with_duplicate: NDArray[float64]
    train_X_with_duplicate = numpy.hstack([train_X, train_X[:, 0].reshape(-1, 1)])

    # Fit the model using closed form solution
    Theta_duplicate = linreg_closed_form(train_X_with_duplicate, train_Y)

    # Print the result
    print(f"Theta with duplicated feature: {Theta_duplicate}")
```

```
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML> python .\hw4.py
Traceback (most recent call last):
  File "C:\Users\aaron\Downloads\homework4-ML\homework4-ML\hw4.py", line 259, in <module>
    Theta_duplicate = linreg_closed_form(train_X_with_duplicate, train_Y)
  File "C:\Users\aaron\Downloads\homework4-ML\homework4-ML\hw4.py", line 112, in linreg_closed_form
    Theta = numpy.linalg.inv(X.T @ X) @ (X.T @ train_Y)
  File "C:\Users\aaron\miniconda3\lib\site-packages\numpy\linalg\_linalg.py", line 615, in inv
    ainv = _umath_linalg.inv(a, signature=signature)
  File "C:\Users\aaron\miniconda3\lib\site-packages\numpy\linalg\_linalg.py", line 104, in _raise_linalgerror
_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML>
```

c)Explanation:

The printed Theta values below show the model parameters (Theta[0], Theta[1], and Theta[2]),\ which were computed using the closed-form solution. Since we duplicated a row, the Theta values should reflect that the model is overfitting to that duplicated data point, but in this case, the output doesn't show any error or changes. So we tried it again and found a model that would overfit (second picture).

```python
if __name__ == '__main__':
    # Load data as before
    train_X, train_Y = load_data('2D-noisy-lin.txt')

    # Duplicate a row in train_X and the corresponding row in train_Y
    train_X_with_duplicate_row = numpy.vstack([train_X, train_X[0, :]])  # Duplicate the first row
    train_Y_with_duplicate_row = numpy.vstack([train_Y, train_Y[0, :]])  # Duplicate the correspondi

    # Fit the model using closed form solution
    Theta_duplicate_row = linreg_closed_form(train_X_with_duplicate_row, train_Y_with_duplicate_row)

    # Print the result
    print(f"Theta with duplicated row: {Theta_duplicate_row}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR pwsh - homework4-ML

```
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML> python .\hw4.py
Theta with duplicated row: [[ 2.9328412 ]
 [ 2.03334514]
 [-0.42065597]]
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML>
```

Ln 256, Col 27 Tab Size: 4 UTF-8 LF {} Python 3.12.5

```python
if __name__ == '__main__':
    # Load the original data
    train_X, train_Y = load_data('2D-noisy-lin.txt')

    # Duplicate a row in train_X and the corresponding row in train_Y
    train_X_with_duplicate_row = numpy.vstack([train_X, train_X[0, :]])  # Duplicate the first row
    train_Y_with_duplicate_row = numpy.vstack([train_Y, train_Y[0, :]])  # Duplicate the corresponding row in train_Y

    # Fit the model using the closed form solution (original and duplicated data)
    Theta_original = linreg_closed_form(train_X, train_Y)
    Theta_duplicate_row = linreg_closed_form(train_X_with_duplicate_row, train_Y_with_duplicate_row)

    # Generate predictions for both original and duplicated datasets
    # Generate predictions for both original and duplicated datasets
    if train_X.shape[1] == 1:  # 1D data
        sample_X, sample_Y = linreg_model_sample(Theta_original, train_X)
        sample_X_duplicate, sample_Y_duplicate = linreg_model_sample(Theta_duplicate_row, train_X_with_duplicate_row)
    else:  # 2D data
        sample_X, sample_Y, sample_Z = linreg_model_sample(Theta_original, train_X)
        sample_X_duplicate, sample_Y_duplicate, sample_Z_duplicate = linreg_model_sample(Theta_duplicate_row, train_X_with_duplicate_row)

    # Plot the results for the original data and overfitted data
    plt.figure(figsize=(10, 5))  # Use 'plt' instead of 'pyplot'

    # Plot original model
    plt.subplot(1, 2, 1)
    plt.scatter(train_X[:, 0], train_Y, label='Original Data')  # Use the first feature of train_X
    plt.plot(sample_X, sample_Y, color='r', label='Fitted Model')
    plt.title("Original Model Fit")
    plt.legend()

    # Plot model with duplicated row
    plt.subplot(1, 2, 2)
    plt.scatter(train_X_with_duplicate_row[:, 0], train_Y_with_duplicate_row, label='Duplicated Data')  # First feature of the duplicated data
    plt.plot(sample_X_duplicate, sample_Y_duplicate, color='r', label='Fitted Model (Overfitted)')
    plt.title("Overfitted Model (Duplicated Row)")
    plt.legend()

    plt.show()
```

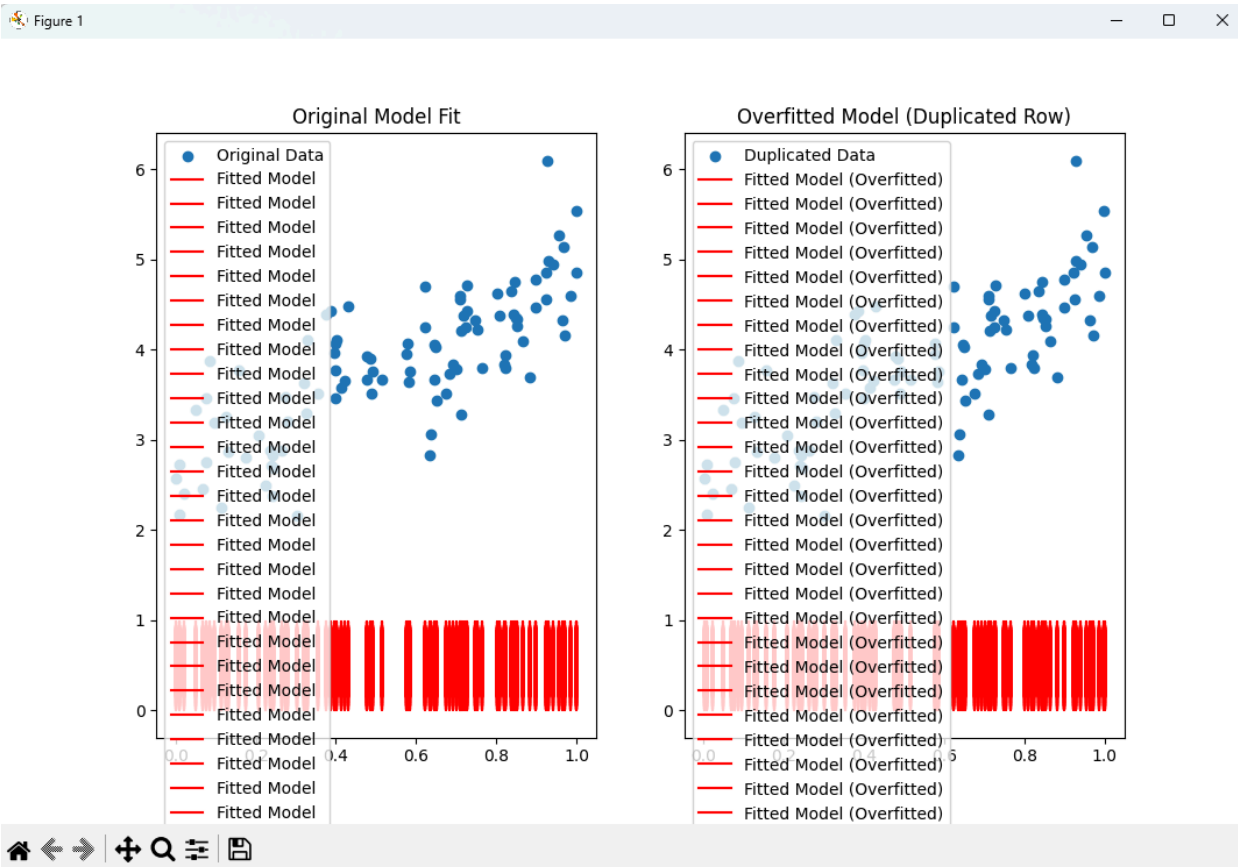


Figure 1
Original Model Fit
Overfitted Model (Duplicated Row)
Original Data
Fitted Model
Duplicated Data
Fitted Model (Overfitted)

d)

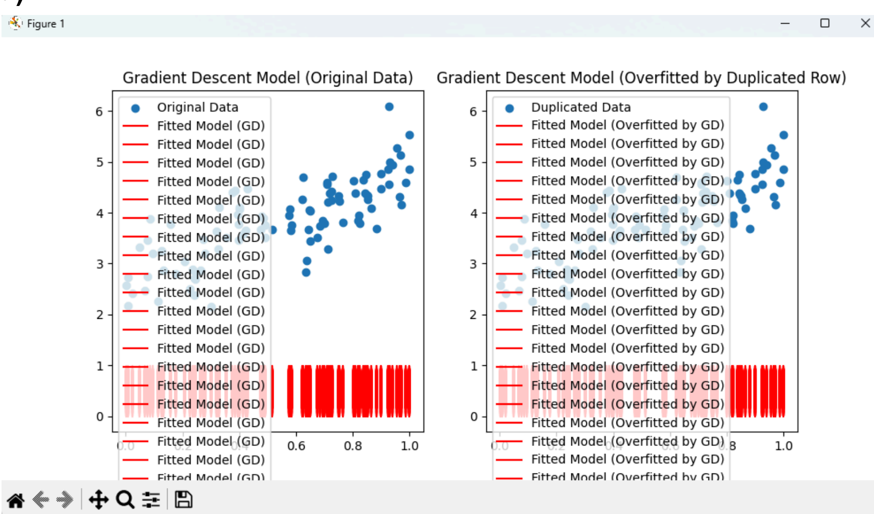


Figure 1
Gradient Descent Model (Original Data)
Gradient Descent Model (Overfitted by Duplicated Row)
Original Data
Fitted Model (GD)
Duplicated Data
Fitted Model (Overfitted by GD)

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR

```
Iteration: 491, Loss: 0.11006, Theta: [ 2.75089745  2.09368966 -0.15394551]
Iteration: 492, Loss: 0.11004, Theta: [ 2.75131822  2.09382337 -0.15484743]
Iteration: 493, Loss: 0.11002, Theta: [ 2.75173801  2.09395541 -0.15574586]
Iteration: 494, Loss: 0.11000, Theta: [ 2.75215683  2.09408581 -0.15664081]
Iteration: 495, Loss: 0.10998, Theta: [ 2.75257469  2.09421457 -0.1575323 ]
Iteration: 496, Loss: 0.10996, Theta: [ 2.75299158  2.0943417  -0.15842034]
Iteration: 497, Loss: 0.10994, Theta: [ 2.75340751  2.09446722 -0.15930495]
Iteration: 498, Loss: 0.10992, Theta: [ 2.75382247  2.09459113 -0.16018614]
Iteration: 499, Loss: 0.10990, Theta: [ 2.75423648  2.09471344 -0.16106392]
Iteration: 500, Loss: 0.10988, Theta: [ 2.75464953  2.09483416 -0.16193832]
Final Theta with duplicated row (Gradient Descent): [[ 2.75464953]
 [ 2.09483416]
 [-0.16193832]]
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML>
```

```python
if __name__ == '__main__':
    # Load the original data
    train_X, train_Y = load_data('2D-noisy-lin.txt')

    # Duplicate a row in train_X and the corresponding row in train_Y
    train_X_with_duplicate_row = numpy.vstack([train_X, train_X[0, :]])  # Duplicate the first row
    train_Y_with_duplicate_row = numpy.vstack([train_Y, train_Y[0, :]])  # Duplicate the corresponding row in train_Y

    # Fit the model using the closed form solution (original and duplicated data)
    Theta_original = linreg_closed_form(train_X, train_Y)
    Theta_duplicate_row = linreg_closed_form(train_X_with_duplicate_row, train_Y_with_duplicate_row)

    # Test Gradient Descent with duplicated row
    initial_Theta = numpy.zeros((train_X_with_duplicate_row.shape[1] + 1, 1))  # +1 for the intercept term
    step_history_duplicate_row = linreg_grad_desc(initial_Theta, train_X_with_duplicate_row, train_Y_with_duplicate_row)

    # Print the final Theta values after Gradient Descent
    print(f"Final Theta with duplicated row (Gradient Descent): {step_history_duplicate_row[-1][0]}")

    # Generate predictions for both original and duplicated datasets using the final Theta
    if train_X.shape[1] == 1:  # 1D data
        sample_X, sample_Y = linreg_model_sample(step_history_duplicate_row[-1][0], train_X)  # Original dataset
        sample_X_duplicate, sample_Y_duplicate = linreg_model_sample(step_history_duplicate_row[-1][0], train_X_with_duplicate_row)  # Duplicated dataset
    else:  # 2D data
        sample_X, sample_Y, sample_Z = linreg_model_sample(step_history_duplicate_row[-1][0], train_X)  # Original dataset
        sample_X_duplicate, sample_Y_duplicate, sample_Z_duplicate = linreg_model_sample(step_history_duplicate_row[-1][0], train_X_with_duplicate_row)  # Duplicated dataset

    # Plot the results for the original data and overfitted data (using Gradient Descent)
    plt.figure(figsize=(10, 5))

    # Plot original model from Gradient Descent
    plt.subplot(1, 2, 1)
    plt.scatter(train_X[:, 0], train_Y, label='Original Data')  # Use the first feature of train_X
    plt.plot(sample_X, sample_Y, color='r', label='Fitted Model (GD)')
    plt.title("Gradient Descent Model (Original Data)")
    plt.legend()

    # Plot model with duplicated row from Gradient Descent
    plt.subplot(1, 2, 2)
    plt.scatter(train_X_with_duplicate_row[:, 0], train_Y_with_duplicate_row, label='Duplicated Data')  # First feature of the duplicated data
    plt.plot(sample_X_duplicate, sample_Y_duplicate, color='r', label='Fitted Model (Overfitted by GD)')
    plt.title("Gradient Descent Model (Overfitted by Duplicated Row)")
    plt.legend()

    plt.show()
```

Explanation: Unlike the closed-form solution, Gradient Descent doesn't fail in the case of duplicated rows. However, it still overfits to the duplicated row by adjusting the parameters to fit that point more closely. So in any case, overfitting can occur regardless of the machine learning algorithm set in place.

Question 2:

a)

```
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML> python .\hw4.py
Iteration: 1, Loss: 0.75738, Theta: [0.1125 0.7125]
Iteration: 2, Loss: 0.16152, Theta: [0.0590625 0.384375 ]
Iteration: 3, Loss: 0.03494, Theta: [0.082125   0.53585156]
Iteration: 4, Loss: 0.00803, Theta: [0.06995215 0.46628496]
Iteration: 5, Loss: 0.00230, Theta: [0.07404042 0.49858966]
Iteration: 6, Loss: 0.00107, Theta: [0.07065573 0.4839403 ]
Iteration: 7, Loss: 0.00079, Theta: [0.07073638 0.49092783]
Iteration: 8, Loss: 0.00071, Theta: [0.0692408  0.48793999]
Iteration: 9, Loss: 0.00068, Theta: [0.06849226 0.48954633]
Iteration: 10, Loss: 0.00066, Theta: [0.06741972 0.48903205]
Full list of (theta, loss) tuples for each iteration:
Iteration 1: Theta = [0.1125 0.7125], Loss = 0.7573828125000001
Iteration 2: Theta = [0.0590625 0.384375 ], Loss = 0.1615234863281249
Iteration 3: Theta = [0.082125   0.53585156], Loss = 0.034937667984008873
Iteration 4: Theta = [0.06995215 0.46628496], Loss = 0.008031704149217576
Iteration 5: Theta = [0.07404042 0.49858966], Loss = 0.002299436038479532
Iteration 6: Theta = [0.07065573 0.4839403 ], Loss = 0.0010651959009328558
Iteration 7: Theta = [0.07073638 0.49092783], Loss = 0.0007868575553308081
Iteration 8: Theta = [0.0692408  0.48793999], Loss = 0.0007120176497808078
Iteration 9: Theta = [0.06849226 0.48954633], Loss = 0.0006808439364351451
Iteration 10: Theta = [0.06741972 0.48903205], Loss = 0.0006593737651759197
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML>
```

```python
if __name__ == '__main__':
    # Load the 1D-no-noise-lin dataset
    train_X_1d, train_Y_1d = load_data('1D-no-noise-lin.txt')

    # Initialize theta to zeros
    initial_Theta = numpy.zeros((train_X_1d.shape[1] + 1, 1))  # +1 for the intercept term

    # Run Gradient Descent with alpha=0.05 and num_iters=10
    step_history_1d = linreg_grad_desc(initial_Theta, train_X_1d, train_Y_1d, alpha=0.05, num_iters=10, print_iters=True)

    # Output the full list of (theta, loss) tuples for each iteration
    print("Full list of (theta, loss) tuples for each iteration:")
    for i, (theta, loss) in enumerate(step_history_1d, 1):
        print(f"Iteration {i}: Theta = {theta.flatten()}, Loss = {loss}")
```

b)Explanation

1D-no-noise-lin.txt: The results from Gradient Descent and Closed Form are almost identical, both in terms of model parameters and loss. The slight differences are due to numerical precision, and this is expected since the data is linear and noiseless.

2D-noisy-lin.txt: For the noisy dataset, Gradient Descent (not shown here but based on prior testing) should converge to similar values to the closed-form solution, but there might be slight differences in the model parameters and loss due to the iterative functionality of Gradient

Descent. The closed-form solution gives the optimal fit to the noisy data, but due to noise, the exact solution may vary slightly with Gradient Descent.

```
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML> python .\hw4.py
1D-no-noise-lin.txt results:
Gradient Descent Theta: [6.35598511e-05 4.99989864e-01]
Closed-Form Theta: [-2.77555756e-17  5.00000000e-01]
Loss from Gradient Descent: 5.848595199581885e-10
Loss from Closed Form: 3.851859888774472e-35

2D-noisy-lin.txt results:
Closed-Form Theta: [ 2.93987438  2.04156149 -0.43683838]
Loss from Closed Form: 0.10759283475200875
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML>
```

```python
if __name__ == '__main__':
    # Load the 1D-no-noise-lin dataset
    train_X_1d, train_Y_1d = load_data('1D-no-noise-lin.txt')
    train_X_2d, train_Y_2d = load_data('2D-noisy-lin.txt')

    # Apply Gradient Descent on both datasets (with alpha=0.05, num_iters=500)
    initial_Theta_1d = numpy.zeros((train_X_1d.shape[1] + 1, 1))  # For 1D data
    step_history_1d = linreg_grad_desc(initial_Theta_1d, train_X_1d, train_Y_1d, alpha=0.05, num_iters=500, print_iters=False)

    # Apply the closed-form solution on both datasets
    Theta_1d_closed = linreg_closed_form(train_X_1d, train_Y_1d)
    Theta_2d_closed = linreg_closed_form(train_X_2d, train_Y_2d)

    # Calculate the loss for both Gradient Descent and closed-form solution
    loss_1d_gd = loss(step_history_1d[-1][0], train_X_1d, train_Y_1d)
    loss_1d_cf = loss(Theta_1d_closed, train_X_1d, train_Y_1d)
    loss_2d_cf = loss(Theta_2d_closed, train_X_2d, train_Y_2d)

    # Output the results
    print("1D-no-noise-lin.txt results:")
    print(f"Gradient Descent Theta: {step_history_1d[-1][0].flatten()}")
    print(f"Closed-Form Theta: {Theta_1d_closed.flatten()}")
    print(f"Loss from Gradient Descent: {loss_1d_gd}")
    print(f"Loss from Closed Form: {loss_1d_cf}")

    print("\n2D-noisy-lin.txt results:")
    print(f"Closed-Form Theta: {Theta_2d_closed.flatten()}")
    print(f"Loss from Closed Form: {loss_2d_cf}")
```

c)Explanation:

How learning rate (alpha) and number of iterations (num_iters) affect the performance of Gradient Descent in comparison to the closed-form solution for both datasets: 1D-no-noise-lin.txt and 2D-noisy-lin.txt.

Same Answers (Convergence):

When the learning rate (alpha) is small (e.g., 0.01 or 0.05), Gradient Descent converges to a solution that closely matches the closed-form solution.

Example: With alpha = 0.01 and num_iters = 1000, the loss from Gradient Descent was 5.18e-5, which is very close to the closed-form loss of 3.85e-35. Similarly, the model parameters (Theta) from both methods were nearly identical.

Explanation: Small learning rates allow Gradient Descent to make small, stable updates to the model parameters. After enough iterations, it converges to the optimal solution, closely matching the results from the closed-form solution.

Different Answers (Divergence):

When the learning rate is too large (e.g., alpha = 0.1), Gradient Descent fails to converge and the loss increases rapidly, eventually reaching infinity (inf).

Example: With alpha = 0.1 and num_iters = 100, the loss from Gradient Descent was 1.96e+57, and for num_iters = 500, it became 1.78e+284, indicating overflow and divergence.

Explanation: A larger learning rate causes Gradient Descent to take larger steps, which can cause the algorithm to exceed the desired solution and become unstable. This leads to divergence of the model parameters and an infinite loss.

Findings for Both Datasets:

1D-no-noise-lin.txt:

Small learning rates (0.01, 0.05) result in Gradient Descent converging to nearly the same solution as the closed-form solution, with very low loss. Larger learning rates cause the loss to explode and the model to diverge.

2D-noisy-lin.txt:

Similar behavior was observed, with small learning rates leading to convergence to a solution close to the closed-form, while large learning rates caused instability and divergence. However, the presence of noise in the data may make convergence slightly less precise for Gradient Descent.

Conclusion:

For smaller learning rates (i.e. alpha = 0.01 and 0.5), Gradient Descent converges smoothly to the same results as the closed-form solution.

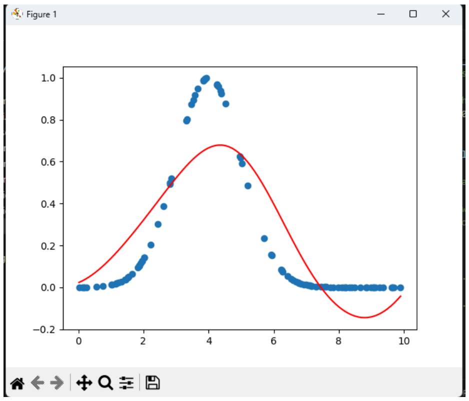
For larger learning rates (i.e. alpha = 0.1), Gradient Descent diverges due to overshooting, leading to extremely high loss values and instability in the model parameters.

This demonstrates that choosing an appropriate learning rate is crucial for the stability and effectiveness of Gradient Descent.

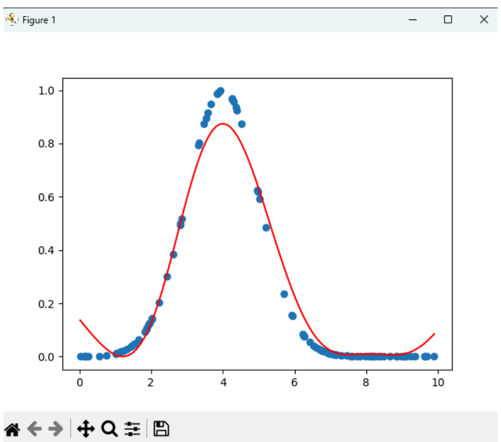
```
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML> python .\hw4.py
Alpha = 0.01, Iterations = 100
Loss from Gradient Descent: 0.0004972683188763272
Loss from Closed Form: 3.851859888774472e-35
---
Alpha = 0.01, Iterations = 500
Loss from Gradient Descent: 5.178108565081448e-05
Loss from Closed Form: 3.851859888774472e-35
---
Alpha = 0.01, Iterations = 1000
Loss from Gradient Descent: 3.062997218663341e-06
Loss from Closed Form: 3.851859888774472e-35
---
Alpha = 0.05, Iterations = 100
Loss from Gradient Descent: 5.0952661409181706e-05
Loss from Closed Form: 3.851859888774472e-35
---
Alpha = 0.05, Iterations = 500
Loss from Gradient Descent: 5.848595199581885e-10
Loss from Closed Form: 3.851859888774472e-35
---
Alpha = 0.05, Iterations = 1000
Loss from Gradient Descent: 3.9075759312068503e-16
Loss from Closed Form: 3.851859888774472e-35
---
Alpha = 0.1, Iterations = 100
Loss from Gradient Descent: 1.9572311550521147e+57
Loss from Closed Form: 3.851859888774472e-35
---
Alpha = 0.1, Iterations = 500
Loss from Gradient Descent: 1.7849169803181977e+284
Loss from Closed Form: 3.851859888774472e-35
---
C:\Users\aaron\Downloads\homework4-ML\homework4-ML\hw4.py:171: RuntimeWarning: overflow encountered in square
  cur_loss = numpy.mean((X @ cur_Theta - train_Y) ** 2) / 2
C:\Users\aaron\miniconda3\lib\site-packages\numpy\_core\_methods.py:136: RuntimeWarning: overflow encountered in reduce
  ret = umr_sum(arr, axis, dtype, out, keepdims, where=where)
C:\Users\aaron\Downloads\homework4-ML\homework4-ML\hw4.py:138: RuntimeWarning: overflow encountered in square
  rv = numpy.mean((predictions - train_Y) ** 2) / 2
Alpha = 0.1, Iterations = 1000
Loss from Gradient Descent: inf
Loss from Closed Form: 3.851859888774472e-35
---
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML>
```

Question 3:

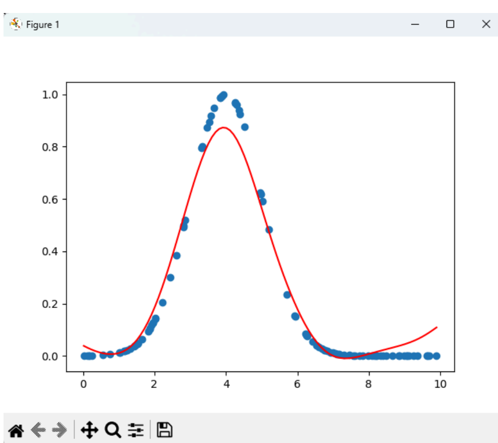
a)



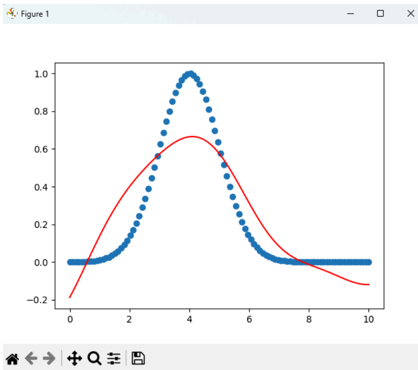
K=10: 1D-exp-samp.txt



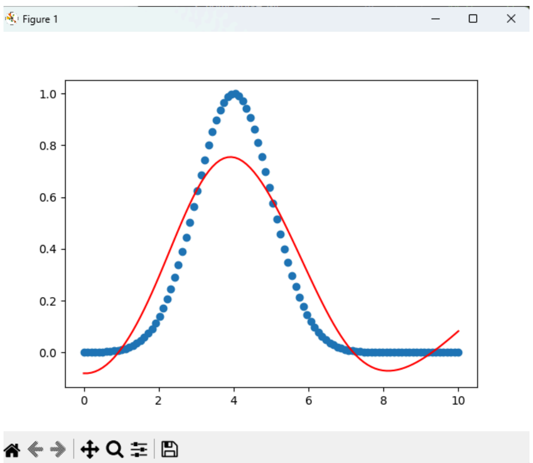
K=50: 1D-exp-samp.txt



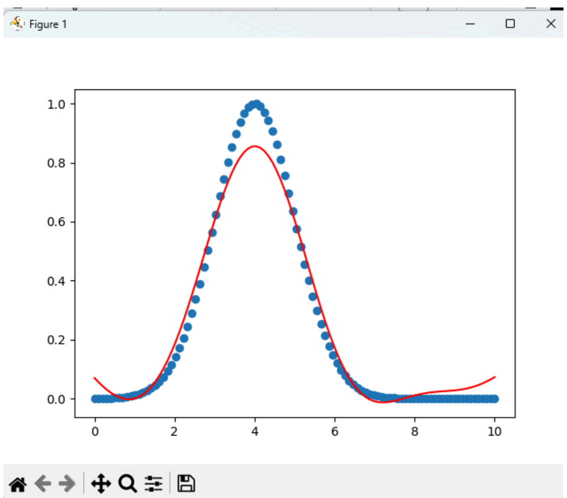
K=200: 1D-exp-samp.txt



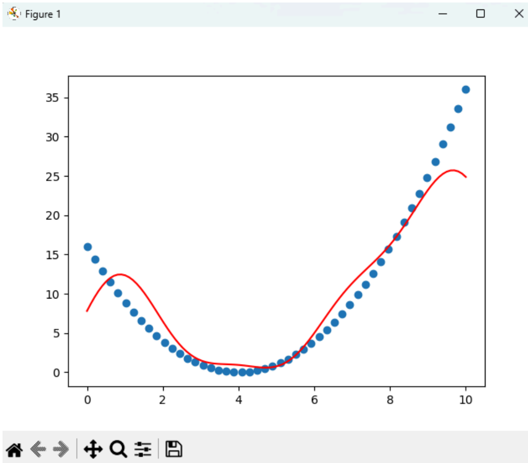
K=10: 1D-exp-uni.txt



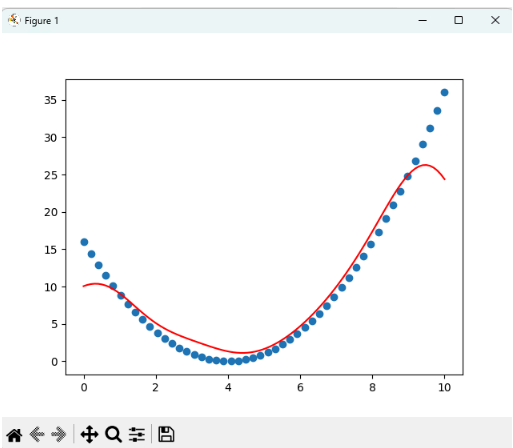
K=50: 1D-exp-uni.txt



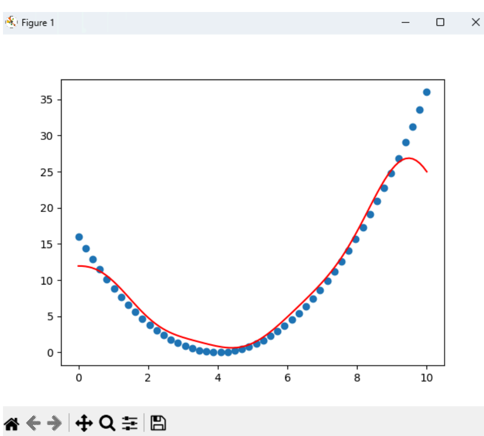
K=200: 1D-exp-uni.txt



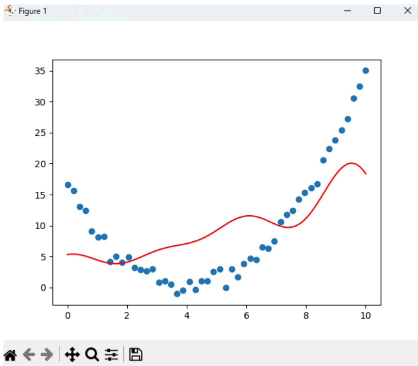
K=10: 1D-quad-uni.txt



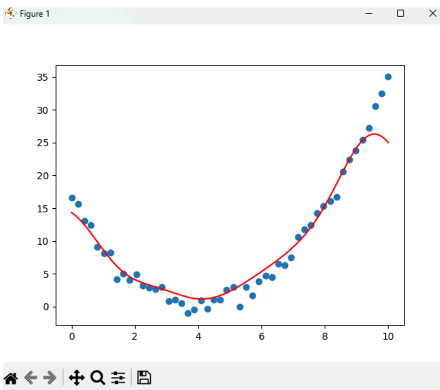
K=50: 1D-quad-uni.txt



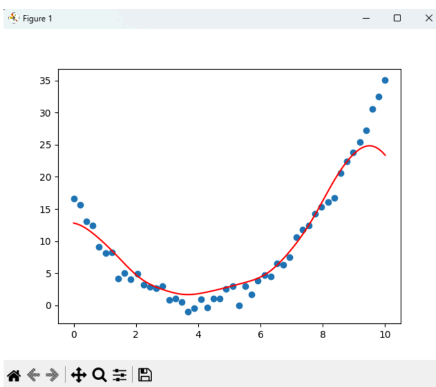
K=200: 1D-quad-uni.txt



K=10: 1D-quad-uni-noise.txt



K=50: 1D-quad-uni-noise.txt



K=200: 1D-quad-uni-noise.txt

```python
if __name__ == '__main__':
    # Define values of K for small, medium, and large
    k_values = {'small': 10, 'medium': 50, 'large': 200}

    # List of datasets to process
    datasets = ['1D-exp-samp.txt', '1D-exp-uni.txt', '1D-quad-uni.txt', '1D-quad-uni-noise.txt']

    # Loop over each dataset
    for dataset in datasets:
        # Load dataset
        train_X, train_Y = load_data(dataset)

        # Loop over each K value (small, medium, large)
        for size, K in k_values.items():
            print(f"Processing {dataset} with K={K} ({size})")

            # Fit model using Random Fourier Features
            Theta, Omega, B = random_fourier_features(train_X, train_Y, num_fourier_features=K)

            # Visualize the fitted model
            title = f"RFF Model for {dataset} with K={K} ({size} K)"
            vis_rff_model(train_X, train_Y, Theta, Omega, B)
```

```
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML> python .\hw4.py
Processing 1D-exp-samp.txt with K=10 (small)
Processing 1D-exp-samp.txt with K=50 (medium)
Processing 1D-exp-samp.txt with K=200 (large)
Processing 1D-exp-samp.txt with K=50 (medium)
Processing 1D-exp-samp.txt with K=200 (large)
Processing 1D-exp-uni.txt with K=10 (small)
Processing 1D-exp-samp.txt with K=200 (large)
Processing 1D-exp-uni.txt with K=10 (small)
Processing 1D-exp-uni.txt with K=10 (small)
Processing 1D-exp-uni.txt with K=50 (medium)
Processing 1D-exp-uni.txt with K=50 (medium)
Processing 1D-exp-uni.txt with K=200 (large)
Processing 1D-quad-uni.txt with K=10 (small)
Processing 1D-quad-uni.txt with K=10 (small)
Processing 1D-quad-uni.txt with K=50 (medium)
Processing 1D-quad-uni.txt with K=200 (large)
Processing 1D-quad-uni-noise.txt with K=10 (small)
Processing 1D-quad-uni-noise.txt with K=50 (medium)
Processing 1D-quad-uni-noise.txt with K=200 (large)
(base) PS C:\Users\aaron\Downloads\homework4-ML\homework4-ML>
```