# Problem 1 (30 points)

## Problem Description

In this problem you will implement polynomial linear least squares regression on two datasets, with and without regularization. Additionally, you will use gradient descent to optimize for the model parameters.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

*You are welcome to use any of the code provided in the lecture activities.*

### Summary of deliverables:

Results:

- Print fitted model parameters `w` for the 4 models requested without regularization
- Print fitted model parameters `w` for the 2 models requested *with* $L_2$ regularization
- Print fitted model parameters `w` for the one model solved via gradient descent

Plots:

- 2 plots of each dataset along with the ground truth function
- 4 plots of the fitted function along with the respective data and ground truth function for LLS without regularization
- 2 plots of the fitted function along with the respective data and ground truth function for LLS with $L_2$ regularization
- 1 plot of the fitted function along with the respective data and ground truth function for LLS with $L_2$ regularization solved via gradient descent

Discussion:

- Discussion of challenges fitting complex models to small datasets
- Discussion of difference between the $L_2$ regularized model versus the standard model
- Dicussion of whether gradient descent could get stuck in a local minimum
- Discussion of gradient descent results versus closed form results

### Imports and Utility Functions:

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

def gt_function():
    xt = np.linspace(0,1,101)
    yt = np.sin(2 *np.pi*xt)
    return xt, yt
```

```
def plot_data(x,y,xt,yt,title = None):
    # Provide title as a string e.g. 'string'
    plt.plot(x,y,'bo',label = 'Data')
    plt.plot(xt,yt,'g-', label = 'Ground Truth')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.legend()
    if title:
        plt.title(title)
    plt.show()

def plot_model(x,y,xt,yt,xr,yr,title = None):
    # Provide title as a string e.g. 'string'
    plt.plot(x,y,'bo',label = 'Data')
    plt.plot(xt,yt,'g-', label = 'Ground Truth')
    plt.plot(xr,yr,'r-', label = 'Fitted Function')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.legend()
    if title:
        plt.title(title)
    plt.show()
```

## Load and visualize the data

The data is contained in `d10.npy` and `d100.npy` and can be loaded with `np.load()`.

Store the data as:

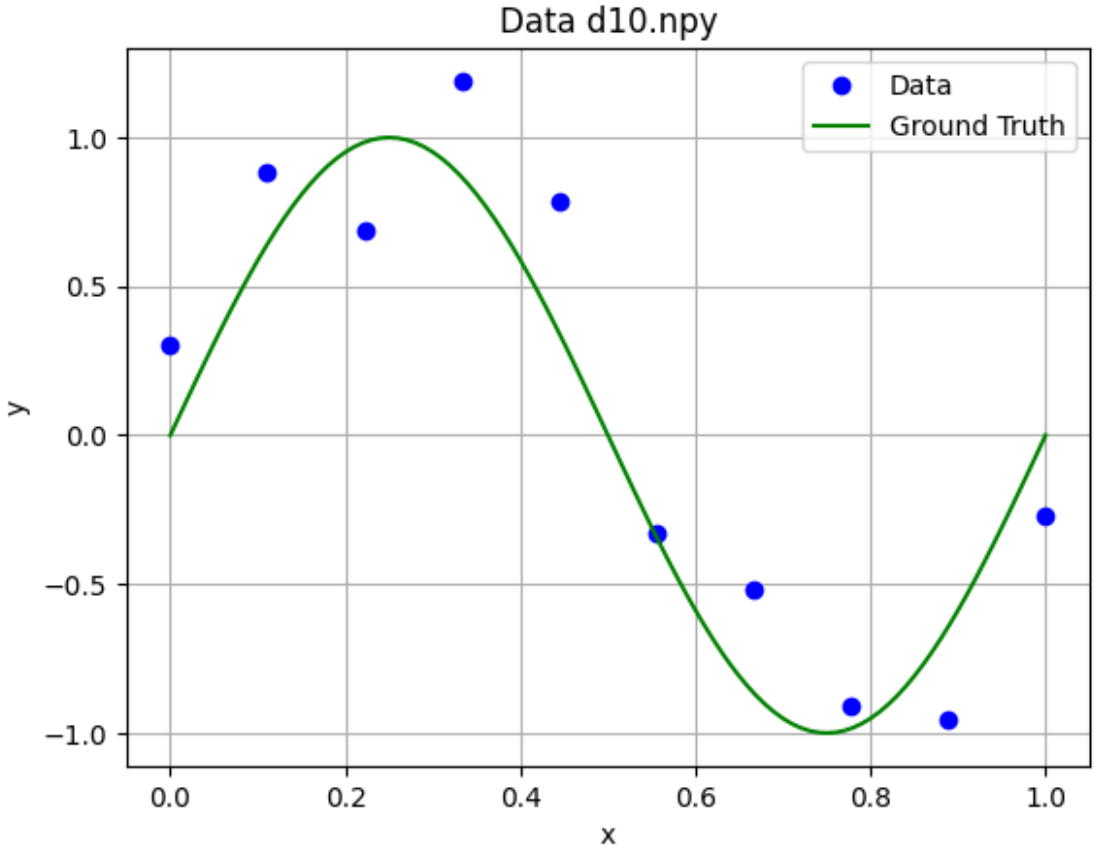
- `x10` and `x100` (the first column of the data)
- `y10` and `y100` (the second column of the data)

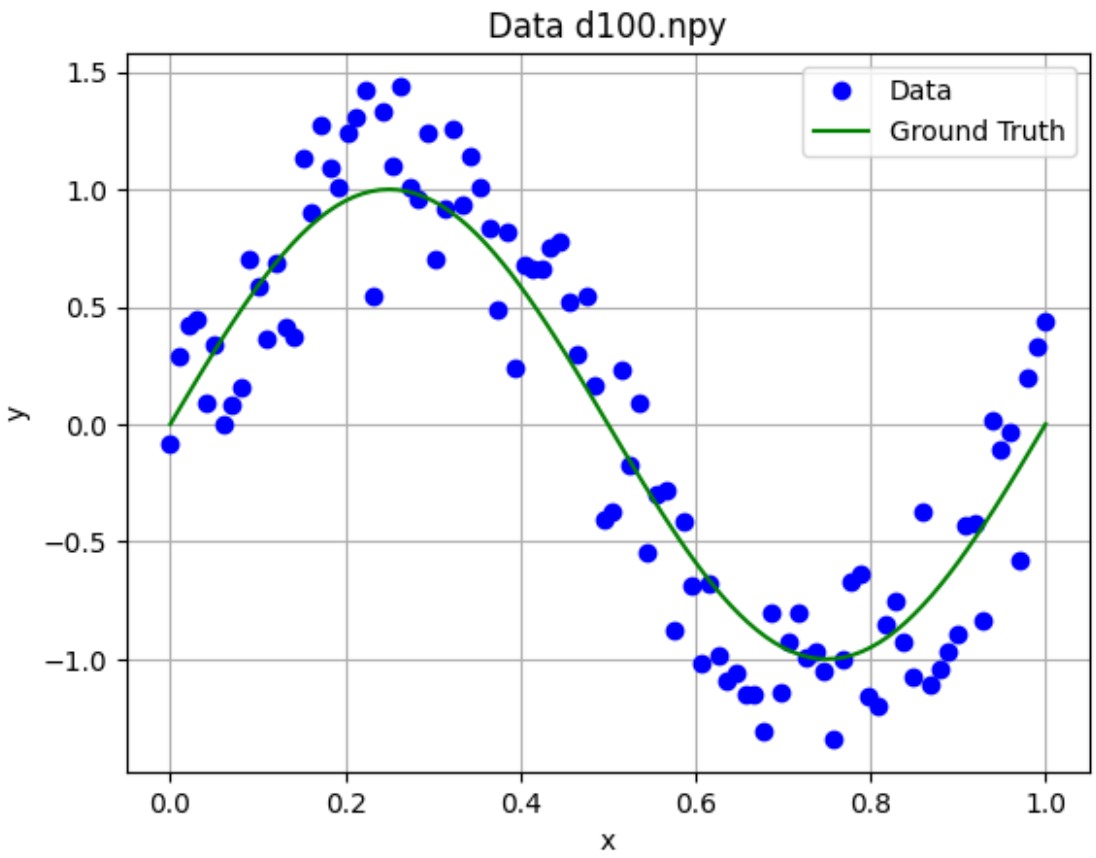
Generate the ground truth function $f(x)=\sin(2\pi x)$ using `xt, yt = gt_function()`.

Then visualize the each dataset with `plotxy(x,y,xt,yt,title)` with an appropriate title. You should generate two plots.

```
# YOUR CODE GOES HERE
# load the data
data_d10 = np.load('d10.npy')
data_d100 = np.load('d100.npy')
# store the loaded data into x and y
x10 = data_d10[:,0]     # first column
y10 = data_d10[:,1]     # second column
x100 = data_d100[:,0]   # first column
y100 = data_d100[:,1]   # second column
# ground truth fucntion
```

```
xt, yt = gt_function()
# plot the data
plot_data(x10,y10,xt,yt,'Data d10.npy')
plot_data(x100,y100,xt,yt,'Data d100.npy')
```






# Implement polynomial linear regression

Now you will implement polynomial linear least squares regression without regularization using the closed form solution from lecture to compute the model parameters. You will consider the following 4 cases:

1. Data: data10.txt, Model: 2nd order polynomial (highest power of $x$ in your regression model = 2)

2. Data: data100.txt, Model: 2nd order polynomial (highest power of $x$ in your regression model = 2)

3. Data: data10.txt, Model: 9th order polynomial (highest power of $x$ in your regression model = 9)

4. Data: data100.txt, Model: 9th order polynomial (highest power of $x$ in your regression model = 9)

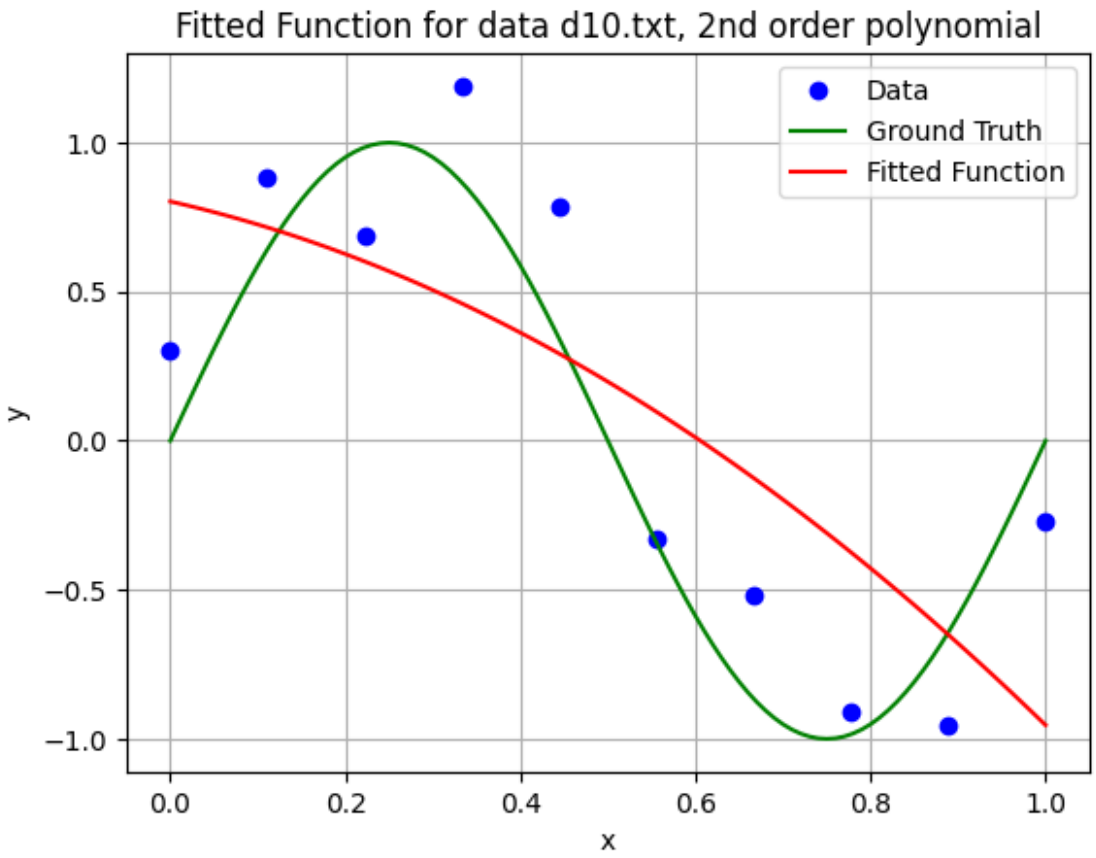
For each model:

- Print the learned model parameters `w`
- Use the model parameters `w` to predict `yr` values over a range of x values given by `xr = np.linspace(0,1,101)`

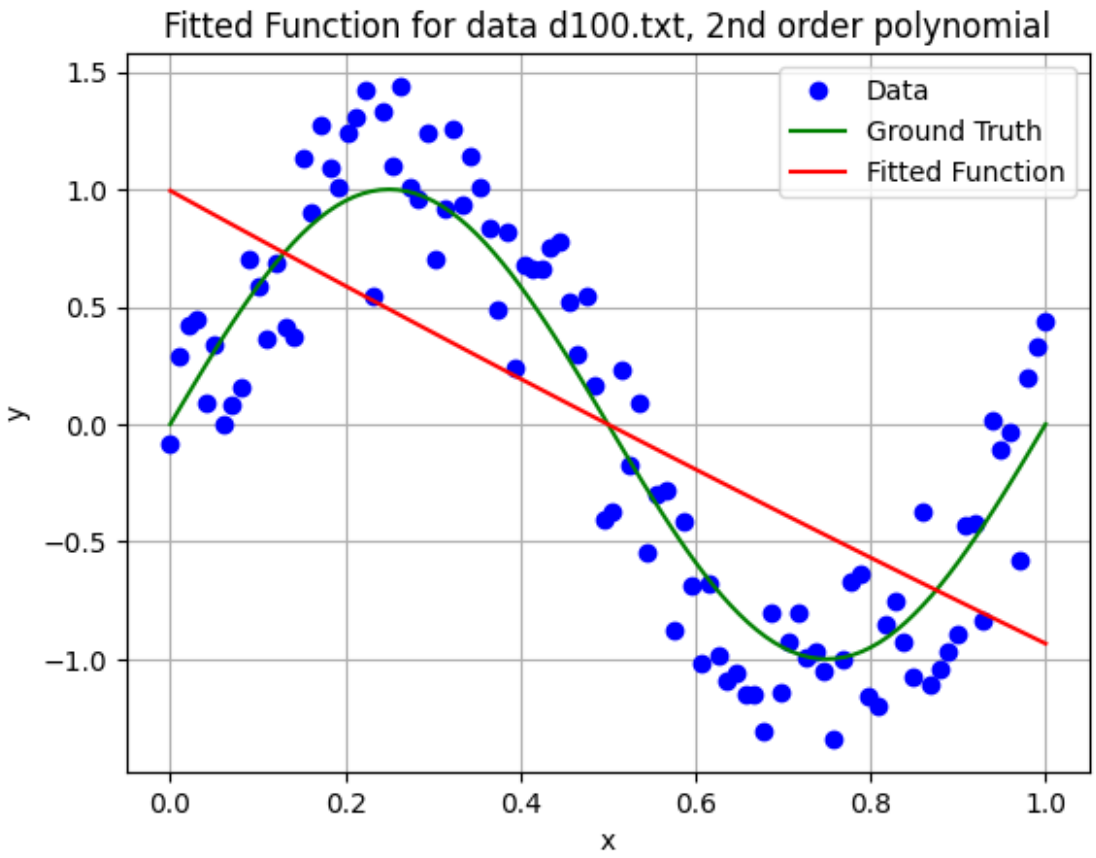
- Plot the data, ground truth function, and regressed model using `plot_model(x,y,xt,yt,xr,yr,title)` with an appropriate title.

```
# YOUR CODE GOES HERE
# linear regression
def ploy_linear_regression(x,y,deg):
    X = np.vander(x, deg+1)
    w = np.linalg.inv(X.T @ X) @ X.T @ y.reshape(-1,1)

    # print model parameters (w)
    print('Model parameters [w]: \n', w)

    # plot the fitted function
    xr = np.linspace(0,1,101)
    XR = np.vander(xr, deg+1)
    yr = XR @ w
    plot_model(x,y,xt,yt,xr,yr, f'Fitted Function for data {data},
{deg}nd order polynomial')

    return w

# [case 1] data10.txt, model: 2nd order polynomial
data = 'd10.txt'
deg = 2
w = ploy_linear_regression(x10,y10,deg)

# [case 2] data100.txt, model: 2th order polynomial
data = 'd100.txt'
deg = 2
w = ploy_linear_regression(x100,y100,deg)

# [case 3] data10.txt, model: 9nd order polynomial
data = 'd10.txt'
deg = 9
w = ploy_linear_regression(x10,y10,deg)

# [case 4] data100.txt, model: 9th order polynomial
data = 'd100.txt'
deg = 9
w = ploy_linear_regression(x100,y100,deg)
```

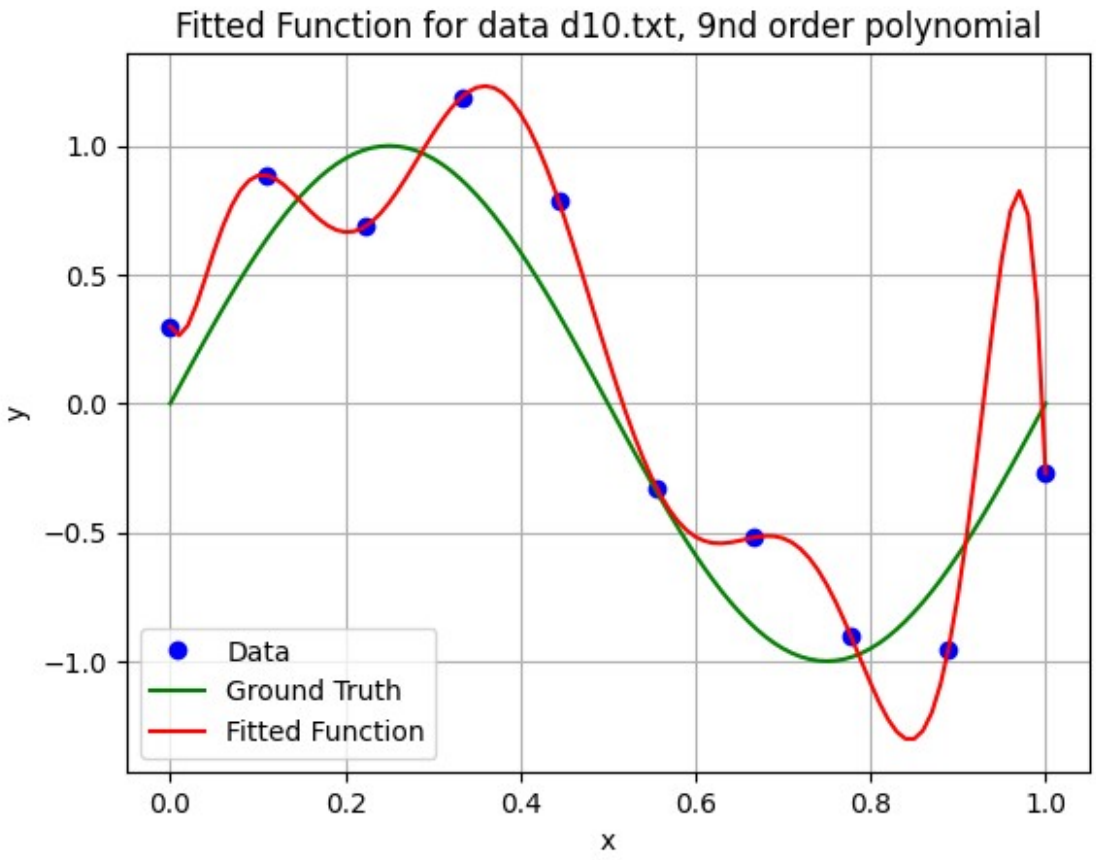
```
Model parameters [w]:
 [[-1.09384447]
 [-0.66283292]
 [ 0.80276877]]
```



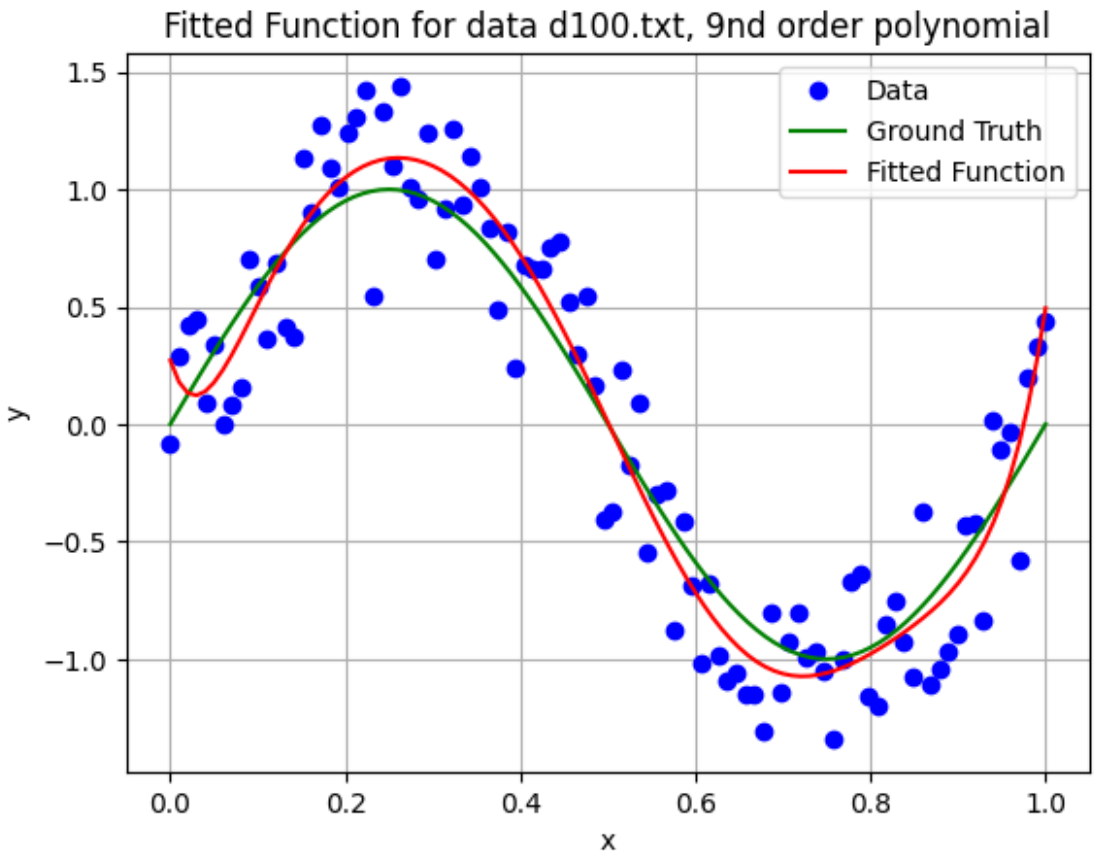

```
Model parameters [w]:
 [[ 0.12128272]
 [-2.04993418]
 [ 0.99435046]]
```




```
Model parameters [w]:
 [[-3.92430472e+04]
 [ 1.67896289e+05]
 [-2.96801430e+05]
 [ 2.79608057e+05]
 [-1.50834125e+05]
 [ 4.64985814e+04]
 [-7.69080481e+03]
 [ 5.74539370e+02]
 [-8.63097293e+00]
 [ 3.00002255e-01]]
```




```
Model parameters [w]:
 [[-2.21754283e+03]
 [ 1.15791219e+04]
 [-2.48684201e+04]
 [ 2.86926291e+04]
 [-1.95212853e+04]
 [ 8.13280715e+03]
 [-2.07018765e+03]
 [ 2.84952056e+02]
 [-1.18519096e+01]
 [ 2.71581706e-01]]
```




# Discussion:

When the sample size (number of data points) is small, what issues or tendencies do you see with complex models?

## *Your answer goes here*

When the sample size is small, we can observe overfitting with complex models. Furthermore, the model is more sensitive to the given training data and tends to have higher variance in its predictions.

# Implement polynomial linear regression with $L_2$ regularization

You will repeat the previous section, but this time using $L_2$ regularization. Your regularization term should be $\lambda\, w' I_m w$, where $\lambda = e^{-10}$, and $I_m$ is the modified identity matrix that masks out the bias term from regularization.

You will consider only two cases:

1. Data: data10.txt, Model: 9th order polynomial (highest power of $x$ in your regression model = 9)

2. Data: data100.txt, Model: 9th order polynomial (highest power of $x$ in your regression model = 9)

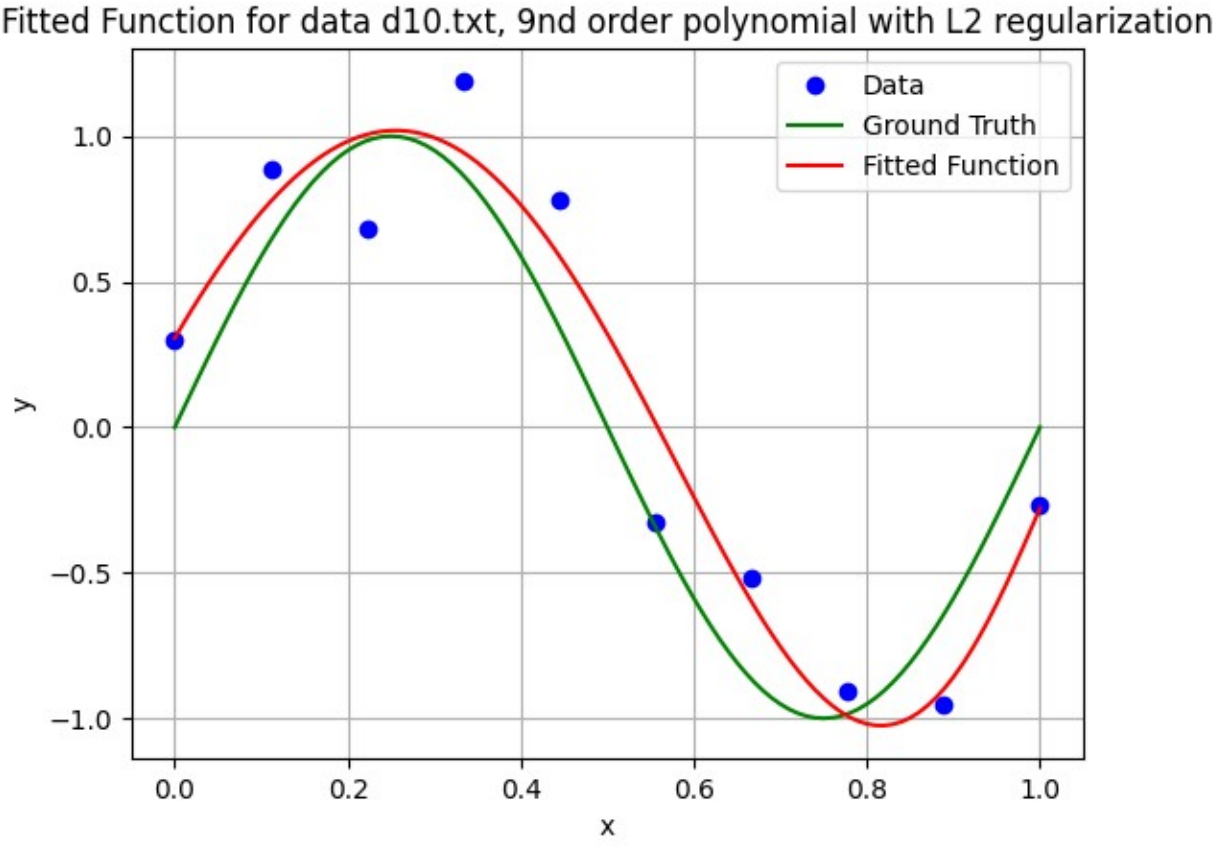
For each model:

- Print the learned model parameters `w`
- Use the model parameters `w` to predict `yr` values over a range of x values given by `xr = np.linspace(0,1,101)`
- Plot the data, ground truth function, and regressed model using `plot_model(x,y,xt,yt,xr,yr,title)` with an appropriate title.

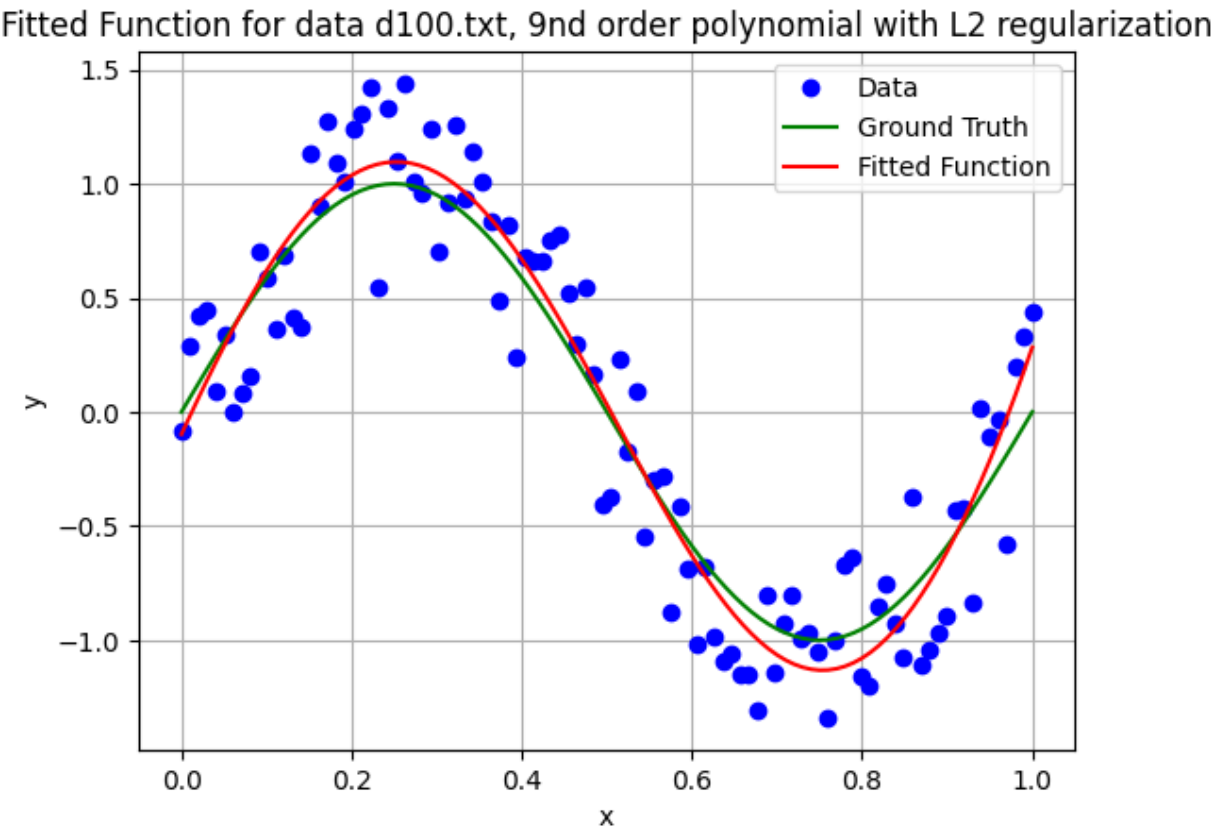
```
# YOUR CODE GOES HERE
# linear regression with L2 regularization
def ploy_linear_regression_L2(x,y,deg, L= np.exp(-10)):
    I_m = np.eye(deg+1)
    I_m[-1,-1] = 0
    X = np.vander(x, deg+1)
    regularized_w = np.linalg.inv(X.T @ X + L*I_m) @ X.T @ y.reshape(-
1,1)

    # print model parameters (w)
    print('Model parameters [w]: \n', regularized_w)

    # plot the fitted function
    xr = np.linspace(0,1,101)
    XR = np.vander(xr, deg+1)
    yr = XR @ regularized_w
    plot_model(x,y,xt,yt,xr,yr, f'Fitted Function for data {data},
{deg}nd order polynomial with L2 regularization')

    return regularized_w

# [case 1] data10.txt, model: 9nd order polynomial
data = 'd10.txt'
deg = 9
w = ploy_linear_regression_L2(x10,y10,deg)

# [case 2] data100.txt, model: 9th order polynomial
data = 'd100.txt'
deg = 9
w = ploy_linear_regression_L2(x100,y100,deg)
```

```
Model parameters [w]:
 [[-3.5338486 ]
 [-1.84896842]
 [ 2.29997278]
 [ 6.47141056]
 [ 6.80582425]
```

```
 [ 0.25321228]
 [-9.30526316]
 [-6.83132438]
 [ 5.10343135]
 [ 0.30633599]]
```




```
Model parameters [w]:
 [[ 17.53893754]
 [-22.00647043]
 [-21.06911985]
 [  7.0398566 ]
 [ 33.8281617 ]
 [ 20.79696185]
 [-38.38285113]
 [ -5.43884905]
 [  8.07411308]
 [ -0.0974112 ]]
```




# Discussion:

What differences between the regularized and standard 9th order models fit to `d10` do you notice? How does regularization affect the fitted function?

## *Your answer goes here*

The regularized 9th-order model eliminated the overfitting issue faced in the standard 9th-order models. The overall prediction in regularized models is more precise with the ground truth. Regularization will minimize the loss function and prevent models from overfitting/underfitting.

# LLS with $L_2$ regularization and gradient descent

For complex models, the size of $X'X$ can be large, making matrix inversion computationally demanding. Instead, one can use gradient descent to compute $w$. In our notes, we derived the gradient descent approach both for unregularized as well as $L_2$ regularized linear regression. The formula for the gradient descent approach with $L_2$ regularization is:

$ \frac{\partial obj}{\partial w} = X'Xw - X'y + \lambda \mathbb{I}_m w$

$ w^{new} \leftarrow w^{cur} - \alpha \frac{\partial obj}{\partial w}$

In this problem, could gradient descent get stuck in a local minimum? Explain why / why not?

*Your answer goes here*

In this problem, the gradient descent will not get stuck in a local minimum. The objective function (quadratic) in the given problem is convex, which has no local minimize or saddle points.

You will consider just a single case in the following question:

1. Data: data10.txt, Model: 9th order polynomial.

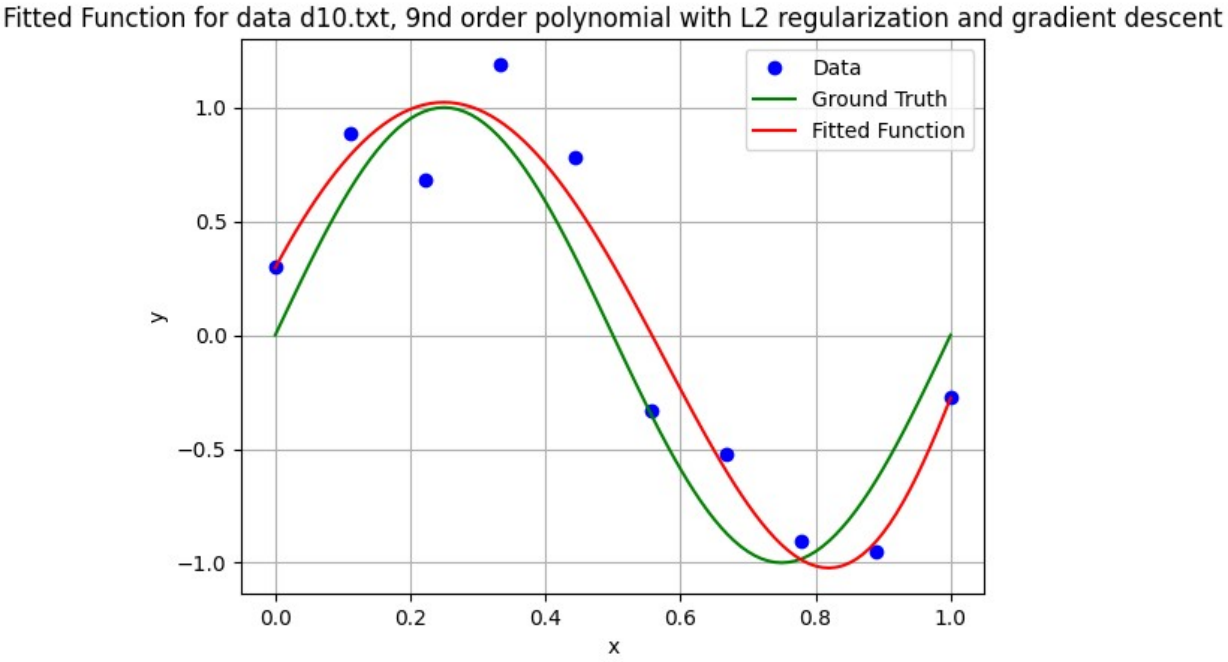
Starting with a weight vector of zeros as the initial guess, and $\lambda = e^{-10}$, $\alpha = 0.075$, apply 50000 iterations of gradient descent to find the optimal model parameters. In practice, when you train your own models you will have to determine these parameters yourself!

For the trained model:

- Print the learned model parameters `w`
- Use the model parameters `w` to predict `yr` values over a range of x values given by `xr = np.linspace(0,1,101)`
- Plot the data, ground truth function, and regressed model using `plot_model(x,y,xt,yt,xr,yr,title)` with an appropriate title.

```
# YOUR CODE GOES HERE
# linear regression with L2 regularization and gradient descent
def ploy_linear_regression_L2_GD(x,y,deg, L= np.exp(-10), a = 0.075,
iterations = 50000):
    I_m = np.eye(deg+1)
    I_m[-1,-1] = 0
    X = np.vander(x, deg+1)
    w = np.zeros((deg+1,1))
    for i in range(iterations):
        grad = X.T @ (X @ w - y.reshape(-1,1)) + L * I_m @ w
        w = w - a * grad

    # plot the fitted function
    xr = np.linspace(0,1,101)
    XR = np.vander(xr, deg+1)
    yr = XR @ w

    # print model parameters (w)
    print('Model parameters [w]: \n', w)

    # print model GD
    # print('Model gradianet: \n', grad)

    # print the predicted yr values
    # print('Predicted yr values: \n', yr)

    plot_model(x,y,xt,yt,xr,yr, f'Fitted Function for data {data},
{deg}nd order polynomial with L2 regularization and gradient descent')
```

```
    return w

# [case] data10.txt, model: 9nd order polynomial
data = 'd10.txt'
deg = 9
w = ploy_linear_regression_L2_GD(x10,y10,deg)

Model parameters [w]:
 [[-3.85457875]
 [-0.40351789]
 [ 2.81217121]
 [ 4.9128699 ]
 [ 4.63900381]
 [ 0.854452  ]
 [-5.76006075]
 [-9.34031661]
 [ 5.56763369]
 [ 0.29561747]]
```




# Discussion:

Visually compare the result you just obtained to the same 9th order polynomial model with $L_2$ regularization where you solved for `w` directly in the previous section. They should be very similar. Comment on whether gradient descent has converged.

## *Your answer goes here*

From the model results, the gradient descent has converged. The gradient value has reached near 0, which further proves the previous statement.