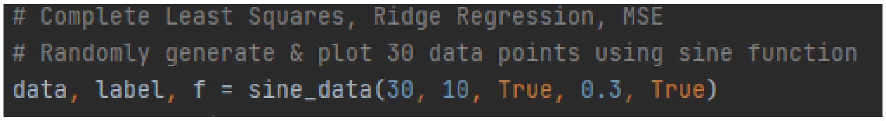
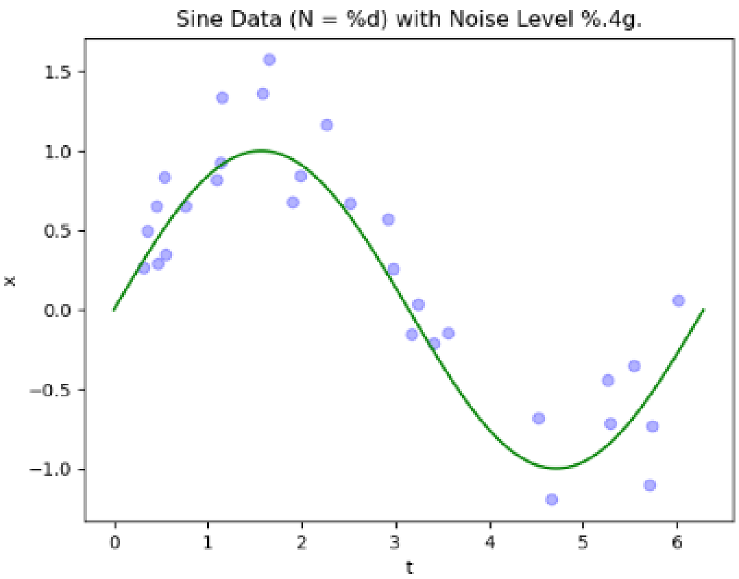
1. Linear regression
    a. Randomly generate 30 data points
        i.

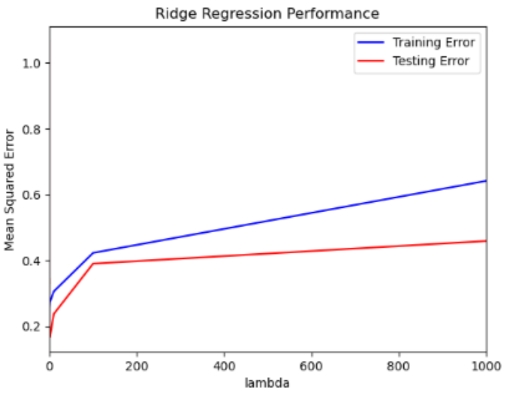


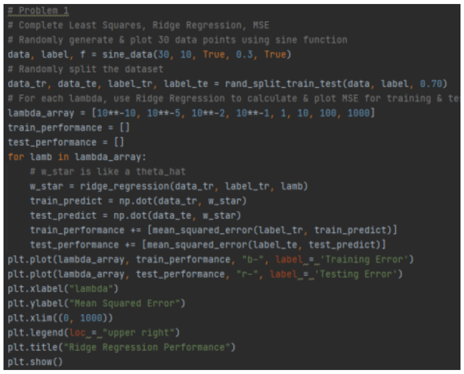
```
# Complete Least Squares, Ridge Regression, MSE
# Randomly generate & plot 30 data points using sine function
data, label, f = sine_data(30, 10, True, 0.3, True)
```

b.



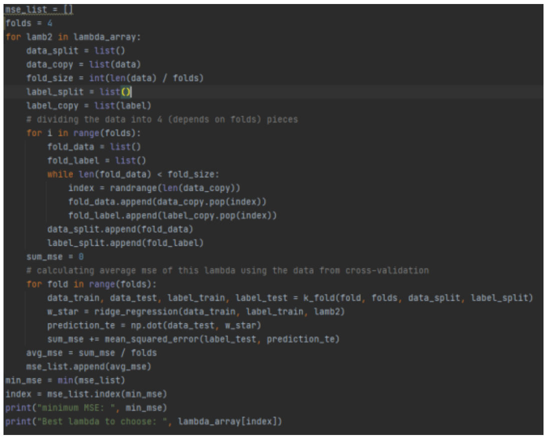
c.



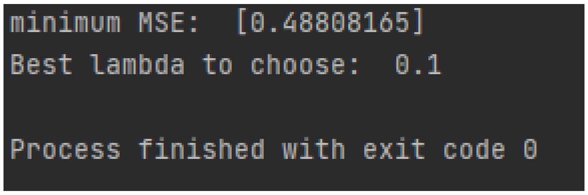


```
# Problem 1
# Complete Least Squares, Ridge Regression, MSE
# Randomly generate & plot 30 data points using sine function
data, label, f = sine_data(30, 10, True, 0.3, True)
# Randomly split the dataset
data_tr, data_te, label_tr, label_te = rand_split_train_test(data, label, 0.70)
# For each lambda, use Ridge Regression to calculate & plot MSE for training & te
lambda_array = [10**-10, 10**-5, 10**-2, 10**-1, 1, 10, 100, 1000]
train_performance = []
test_performance = []
for lamb in lambda_array:
    # w_star is like a theta_hat
    w_star = ridge_regression(data_tr, label_tr, lamb)
    train_predict = np.dot(data_tr, w_star)
    test_predict = np.dot(data_te, w_star)
    train_performance += [mean_squared_error(label_tr, train_predict)]
    test_performance += [mean_squared_error(label_te, test_predict)]
plt.plot(lambda_array, train_performance, "b-", label = 'Training Error')
plt.plot(lambda_array, test_performance, "r-", label = 'Testing Error')
plt.xlabel("lambda")
plt.ylabel("Mean Squared Error")
plt.xlim((0, 1000))
plt.legend(loc = "upper right")
plt.title("Ridge Regression Performance")
plt.show()
```

d.



```
mse_list = []
folds = 4
for lamb2 in lambda_array:
    data_split = list()
    data_copy = list(data)
    fold_size = int(len(data) / folds)
    label_split = list()
    label_copy = list(label)
    # dividing the data into 4 (depends on folds) pieces
    for i in range(folds):
        fold_data = list()
        fold_label = list()
        while len(fold_data) < fold_size:
            index = randrange(len(data_copy))
            fold_data.append(data_copy.pop(index))
            fold_label.append(label_copy.pop(index))
        data_split.append(fold_data)
        label_split.append(fold_label)
    sum_mse = 0
    # calculating average mse of this lambda using the data from cross-validation
    for fold in range(folds):
        data_train, data_test, label_train, label_test = k_fold(fold, folds, data_split, label_split)
        w_star = ridge_regression(data_train, label_train, lamb2)
        prediction_te = np.dot(data_test, w_star)
        sum_mse += mean_squared_error(label_test, prediction_te)
    avg_mse = sum_mse / folds
    mse_list.append(avg_mse)
min_mse = min(mse_list)
index = mse_list.index(min_mse)
print("minimum MSE: ", min_mse)
print("Best lambda to choose: ", lambda_array[index])
```

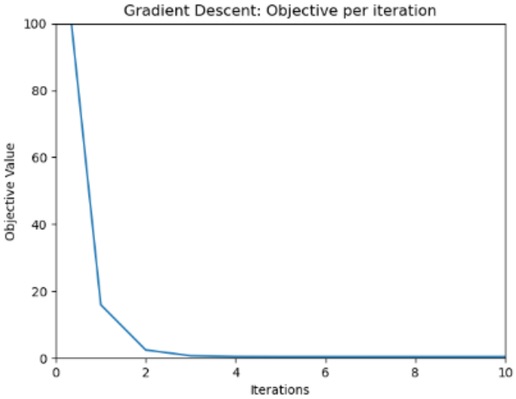


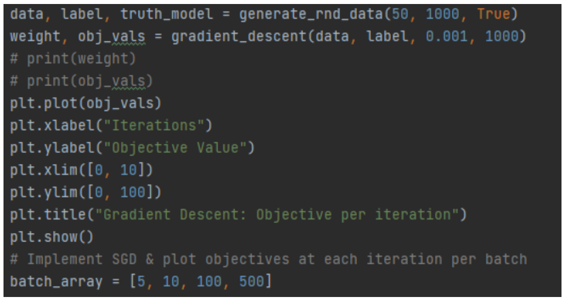
```
minimum MSE:  [0.48808165]
Best lambda to choose:  0.1

Process finished with exit code 0
```

i. (the best lambda value changes depends on the datasets.)

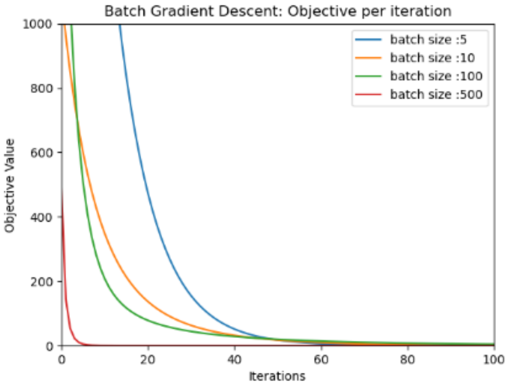
2. Gradient Descent

a.

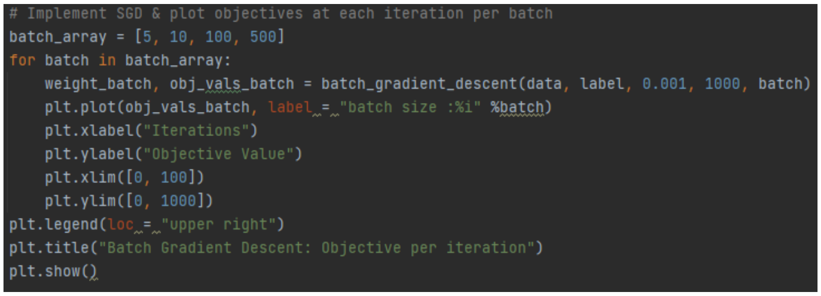




```
data, label, truth_model = generate_rnd_data(50, 1000, True)
weight, obj_vals = gradient_descent(data, label, 0.001, 1000)
# print(weight)
# print(obj_vals)
plt.plot(obj_vals)
plt.xlabel("Iterations")
plt.ylabel("Objective Value")
plt.xlim([0, 10])
plt.ylim([0, 100])
plt.title("Gradient Descent: Objective per iteration")
plt.show()
# Implement SGD & plot objectives at each iteration per batch
batch_array = [5, 10, 100, 500]
```



b.



```
# Implement SGD & plot objectives at each iteration per batch
batch_array = [5, 10, 100, 500]
for batch in batch_array:
    weight_batch, obj_vals_batch = batch_gradient_descent(data, label, 0.001, 1000, batch)
    plt.plot(obj_vals_batch, label = "batch size :%i" %batch)
    plt.xlabel("Iterations")
    plt.ylabel("Objective Value")
    plt.xlim([0, 100])
    plt.ylim([0, 1000])
plt.legend(loc = "upper right")
plt.title("Batch Gradient Descent: Objective per iteration")
plt.show()
```