Instructions: Try to solve all problems on your own. If you have difficulties, ask the instructor or TAs. Please use Python.
Important Note: Due to overwhelming requests, this lab exercise has been designed such that it can be completed within the lab hours. However the deadline for submission is set to tomorrow 11.55 PM. Please note that there will be no extension to this deadline.

In this lab, we will continue with the ordinary least squares linear regression (OLSLR) problem introduced in the previous lab. Recall that we considered two versions of optimization problems, as follows:

- Direct OLSLR

$$\min_x f(x) = \frac{1}{2}\|Ax - y\|_2^2. \tag{1}$$

- Regularized OLSLR

$$\min_x f_\lambda(x) = \frac{\lambda}{2}x^\top x + \frac{1}{2}\|Ax - y\|_2^2. \tag{2}$$

However, the need for regularized version was possibly not clear. In the first exercise, we will try to motivate the need for the regularized version in problem (2).

Exercise 1: The need for regularization

Load the digits dataset from `scikit-learn` package using the following code. Create the data matrix A and the label vector y as described in the code:

```python
import numpy as np
#for the following statement to compile successfully, you need the scikit-learn package.
#You can install it using pip install -U scikit-learn or conda install scikit-learn

from sklearn.datasets import load_digits

digits = load_digits()

#check the shape of digits data
print(digits.data.shape)

#check the shape of digits target
print(digits.target.shape)

#let us use the linear regression used in the previous lab

#N = digits.data.shape[0] #Number of data points
#d = digits.data.shape[1] #Dimension of data points

A = digits.data

#In the following code, we create a Nx1 vector of target labels
y = 1.0*np.ones([A.shape[0],1])
for i in range(digits.target.shape[0]):
   y[i] = digits.target[i]
```

1. Now use your Newton method to solve problem (1), which is the direct OLSLR. Use the starting point $x = \mathbf{0}$. Indicate the difficulties you encounter. Check if you face similar difficulties when you use Newton method to solve problem (2), the regularized OLSLR with $\lambda = 0.1$ and starting point $x = \mathbf{0}$. Explain the reason for your observation.

2. Use BFGS method with starting point $x = \mathbf{0}$, to solve problem (1) and describe if you observe any difficulty. Check if solving the regularized problem (2) helps (use $\lambda = 0.1$ and starting point $x = \mathbf{0}$). Explain your observations.

Exercise 2: Scalability

Having understood the need for regularization, we will focus on the regularized problem (2) from now on.
The experiments you performed in the last lab would have given an impression that Newton and BFGS methods work extremely well. However, recall that in the last lab, you performed experiments on a simple 2-dimensional data set. We will check the behavior of Newton and LBFGS methods on large data sets in this exercise.
Caution: The scalability experiments might impose excessive memory demands. Please close (or stop) other programs. Please carefully watch the memory usage and take appropriate actions.
Use the following code for the scalability experiments.

```python
#Code for Newton method
import numpy as np
import timeit
np.random.seed(1000) #for repeatability

N = 200
ds = [1000, 5000, 10000, 20000, 25000, 50000, 100000, 200000, 500000, 1000000]
lambda_reg = 0.1
eps = np.random.rand(N,1) #random noise

#For each value of dimension in the ds array, we will check the behavior of Newton method
for i in range(np.size(ds)):
    d=ds[i]
    A = np.random.randn(N,d)
    xorig = np.ones((d,1))
    y = np.dot(A,xorig) + eps

    start = timeit.default_timer()
    #call Newton method with A,y,lambda and obtain the optimal solution x_opt
    #x_opt = Newton(A,y,lambda_reg)
    newtontime = timeit.default_timer() - start #time is in seconds
    #print the total time and the L2 norm difference || x_opt - xorig|| for Newton method
```

```python
#Code for BFGS method
import numpy as np
import timeit
np.random.seed(1000) #for repeatability

N = 200
ds = [1000, 5000, 10000, 20000, 25000, 50000, 100000, 200000, 500000, 1000000]
lambda_reg = 0.1
eps = np.random.rand(N,1) #random noise

#For each value of dimension in the ds array, we will check the behavior of BFGS method
```

```
for i in range(np.size(ds)):
    d=ds[i]
    A = np.random.randn(N,d)
    xorig = np.ones((d,1))
    y = np.dot(A,xorig) + eps

    start = timeit.default_timer()
    #call Newton method with A,y,lambda and obtain the optimal solution x_opt
    #x_opt = Newton(A,y,lambda_reg)
    newtontime = timeit.default_timer() - start #time is in seconds
    #print the total time, ||Ax_opt-y||^2 and the L2 norm difference || x_opt - xorig||^2
        for Newton method
    start = timeit.default_timer()
    #call BFGS method with A,y,lambda and obtain the optimal solution x_opt_bfgs
    #x_opt_bfgs = BFGS(A,y,lambda_reg)
    bfgstime = timeit.default_timer() - start #time is in seconds
    #print the total time, ||Ax_opt_bfgs-y||^2 and the L2 norm difference || x_opt_bfgs -
        xorig||^2 for BFGS method
```

Note that in the code fragments, we experiment with different sizes of data set dimension. Prepare a tabulation where you report the following quantities for each dimension for Newton and BFGS methods:

1. The total CPU time taken to solve the respective method
2. The value $\|Ax^* - y\|_2^2$, where x^* is the respective optimizer obtained by Newton and BFGS methods.
3. ℓ_2 norm difference values $\|x^* - x_{orig}\|_2^2$ where x^* is the respective optimizer obtained by Newton and BFGS methods.

Run the experiments until you face either of the following situations (which we flag as *failure*):

- You sense that the code is taking much longer to execute for a particular dimension (say more than 20 minutes)
- Your machine faces memory issues.

Report the dimension where *failure* occurs respectively for Newton and BFGS method.

Exercise 3: A possible approach to handle scalability

In this exercise, we will discuss one possible approach to handle the scalability issue faced in the previous exercise.

1. First note that the regularized problem (2) can be written as

$$\min_x f_\lambda(x) = \min_x \sum_{i=1}^{N} f_i(x). \tag{3}$$

   Find an appropriate choice of $f_i(x)$.

2. Write an expression to compute the gradient of $f_i(x)$. Let us denote it by $g_i(x) = \nabla_x f_i(x)$.
3. Consider the dimension where you observed *failure* in the previous exercise. Implement the following algorithm (ALG-LAB7) to solve (3):

```
#Code for ALG-LAB7
import numpy as np
import timeit
np.random.seed(1000) #for repeatability

N = 200
d = ??? #Consider the dimension which caused failure in the previous experiment
lambda_reg = 0.1
eps = np.random.rand(N,1) #random noise

#Create data matrix, label vector
A = np.random.randn(N,d)
xorig = np.ones( (d,1) )
y = np.dot(A,xorig) + eps

#initialize the optimization variable to be used in the new algo ALG-LAB7
x = np.zeros((d,1))
epochs = 20 #initialize the number of rounds needed to process
t = 1
arr = np.arange(N) #index array

start = timeit.default_timer() #start the timer
for epoch in range(epochs):
  np.random.shuffle(arr) #shuffle every epoch
  for i in np.nditer(arr): #Pass through the data points
    # Update x using x <- x - 1/t * g_i (x)
    t = t+1
alglab7time = timeit.default_timer() - start #time is in seconds
x_alglab7 = x
#print the time taken, ||Ax_alglab7 - y||^2, ||x_alglab7 - xorig||^2
```

Report the time taken for ALG-LAB7, $\|Ax^* - y\|_2^2$ and $\|x^* - x_{orig}\|_2^2$ where x^* is the optimizer obtained by ALG-LAB7.

4. Fix $\lambda = 0.1$ and repeat the experiment for 30, 40 and 50 epochs and report the time taken for ALG-LAB7 and $\|Ax^* - y\|_2^2$ and $\|x^* - x_{orig}\|_2^2$. Explain your observations.

5. Fix 20 epochs and try $\lambda \in \{1000, 100, 10, 1, 0.1, 10^{-2}, 10^{-3}\}$ and report the time taken for ALG-LAB7 and $\|Ax^* - y\|_2^2$ and $\|x^* - x_{orig}\|_2^2$. Explain your observations.

6. Does ALG-LAB7 work for the *failure* dimension?

7. Explain your understanding of ALG-LAB7.