Instructions: Try to solve all problems on your own. If you have difficulties, ask the instructor or TAs. Please use Python.

Instructions: (Please read carefully and follow them!)

Try to solve all problems on your own. If you have difficulties, ask the instructor or TAs.
In this session, we will apply the methods we have developed in the previous labs, to solve a practical problem. The scalability analysis performed in previous labs will be carried out in this lab as well. The implementation of the optimization algorithms in this lab will involve extensive use of the `numpy` Python package. It would be useful for you to get to know some of the functionalities of `numpy` package. For details on `numpy` Python package, please consult https://numpy.org/doc/stable/index.html
For plotting purposes, please use `matplotlib.pyplot` package. You can find examples in the site https://matplotlib.org/examples/.

Please follow the instructions given below to prepare your solution notebooks:

- Please use different notebooks for solving different Exercise problems.
- The notebook name for Exercise 1 should be `YOURROLLNUMBER_IE684_Lab7_Ex1.ipynb`.
- Similarly, the notebook name for Exercise 2 should be `YOURROLLNUMBER_IE684_Lab7_Ex2.ipynb`, etc.
- Please post your doubts in MS Teams Discussion Forum channel so that TAs can clarify.

There are only 3 exercises in this lab. Try to solve all problems on your own. If you have difficulties, ask the Instructors or TAs.

Only the questions marked [**R**] need to be answered in the notebook. You can either print the answers using `print` command in your code or you can write the text in a separate text tab. To add text in your notebook, click `+Text`. Some questions require you to provide proper explanations; for such questions, write proper explanations in a text tab. Some questions require the answers to be written in LaTeX notation. Please see the demo video (posted in Lab 1) to know how to write LaTeX in Google notebooks. Some questions require plotting certain graphs. Please make sure that the plots are present in the submitted notebooks. Please include all answers in your `.pynb` files.

After completing this lab's Exercise 1 and Exercise 2, click File → Download .ipynb and save your files to your local laptop/desktop. Create a folder with name `YOURROLLNUMBER_IE684_Lab7_Ex1_Ex2` and copy your `.ipynb` files to the folder.
Then zip the folder to create `YOURROLLNUMBER_IE684_Lab7_Ex1_Ex2.zip`. Then upload only the `.zip` file to Moodle.
Similarly, after completing this lab's Exercise 3, click File → Download .ipynb and save your files to your local laptop/desktop. Create a folder with name `YOURROLLNUMBER_IE684_Lab7_Ex3` and copy your `.ipynb` files to the folder. Then zip the folder to create `YOURROLLNUMBER_IE684_Lab7_Ex3.zip`. Then upload only the `.zip` file to Moodle.
There will be extra marks for students who follow the proper naming conventions in their submissions.

Please check the **submission deadline announced in moodle.**

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

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

  Let $x_f^* = \text{argmin}_x f(x)$.

- Regularized OLSLR

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

  Let $x_{f_\lambda}^* = \text{argmin}_x f_\lambda(x)$.

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:

```
import numpy as np

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. [**R**] 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 $\lambda = 0.001$ and starting point $x = \mathbf{0}$. Explain the reason for your observation. Report the values of x_f^* and $x_{f_\lambda}^*$.

2. [**R**] 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 $\lambda = 0.001$ and starting point $x = \mathbf{0}$). Explain your observations. Report the values of x_f^* and $x_{f_\lambda}^*$.

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 10-dimensional data set. We will check the behavior of Newton and BFGS methods on large data sets in this exercise.
Caution: The scalability experiments might impose excessive memory demands. Please carefully watch the memory usage of Google colab.
Use the following code for the scalability experiments.

```
#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 = 0.001
eps = np.random.randn(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)
    #Normalize the columns
    for j in range(A.shape[1]):
      A[:,j] = A[:,j]/np.linalg.norm(A[:,j])

    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
```

```
#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 = 0.001
eps = np.random.randn(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)
    #Normalize the columns
```

```
for j in range(A.shape[1]):
  A[:,j] = A[:,j]/np.linalg.norm(A[:,j])
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.
[R] 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 and x_{orig} is used in the code.

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.

[R] 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

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

   [**R**] Find an appropriate choice of $f_i(x)$.

2. [**R**] 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 = 0.001
eps = np.random.randn(N,1) #random noise

#Create data matrix, label vector
A = np.random.randn(N,d)
#Normalize the columns
for j in range(A.shape[1]):
   A[:,j] = A[:,j]/np.linalg.norm(A[:,j])

xorig = np.ones( (d,1) )
y = np.dot(A,xorig) + eps

#initialize the optimization variable to be used in the new algo ALG-LAB8
x = np.zeros((d,1))
epochs = 1e4 #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
    if t>1e4:
      t = 1
alglab7time = timeit.default_timer() - start #time is in seconds
x_alglab7 = x
#print the time taken, ||Ax_alglab8 - y||^2, ||x_alglab8 - xorig||^2
```

   [**R**] Report the time taken for ALG-LAB7, $\|\nabla f_\lambda(x^*)\|$, $\|Ax^* - y\|_2^2$ and $\|x^* - x_{orig}\|_2^2$ where x^* is the optimizer obtained by ALG-LAB7.

4. [**R**] Fix $\lambda = 0.001$ and repeat the experiment for 10^6, 10^8 and 10^{10} epochs and report the time taken for ALG-LAB7 and $\|\nabla f_\lambda(x^*)\|$, $\|Ax^* - y\|_2^2$ and $\|x^* - x_{orig}\|_2^2$. Explain your observations.

5. [**R**] Fix 10^9 epochs and try $\lambda \in \{1000, 100, 10, 1, 0.1, 10^{-2}, 10^{-3}\}$ and report the time taken for ALG-LAB7 and $\|\nabla f_\lambda(x^*)\|$, $\|Ax^* - y\|_2^2$ and $\|x^* - x_{orig}\|_2^2$. Explain your observations.

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

7. [**R**] Explain your understanding of ALG-LAB7.