# Optimisation for Machine Learning

Ayush Abrol B20AI052

---

## Question 1

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

```
# Solve the following problem using Newton method. Use stopping criterion: ||
# ∇F(x^(k+1)) + A^T μk+1|| ≤ 10^-3.

# a) min F(x) = Sum(xi)(1->5)xilogxi
#     s.t Sum(xi) = 1
# and choose x0 = (1/5, 1/5, 1/5, 1/5, 1/5).

# b) min F(x) = Sum(xi)(1->4)xi exp(-xi)
#     s.t x1 + x2 + x3 + x4 = 1
#     x1 - 2x2 + 3x3 - 4x4 = 0
#     choose x0 = (2/3, 1/3, 0, 0).

def newton_method(f, grad_f, hessian_f, x0, eps=1e-3):
    x = x0
    while True:
        grad = grad_f(x)
        hessian = hessian_f(x)
        delta = np.linalg.solve(hessian, grad)
        x = x - delta
        if np.linalg.norm(grad) <= eps:
            break
    return x

def f_a(x):
    return np.sum(x * np.log(x))

def grad_f_a(x):
    return np.array([np.log(x[0]) + 1, np.log(x[1]) + 1, np.log(x[2]) + 1,
                     np.log(x[3]) + 1, np.log(x[4]) + 1])

def hessian_f_a(x):
    return np.array([[1/x[0], 0, 0, 0, 0], [0, 1/x[1], 0, 0, 0],
                     [0, 0, 1/x[2], 0, 0], [0, 0, 0, 1/x[3], 0],
                     [0, 0, 0, 0, 1/x[4]]])

def f_b(x):
    return np.sum(x * np.exp(-x))
```

```
def grad_f_b(x):
    return np.array([np.exp(-x[0]) - 1, np.exp(-x[1]) - 1,
                     np.exp(-x[2]) - 1, np.exp(-x[3]) - 1])

def hessian_f_b(x):
    return np.array([[-np.exp(-x[0]), 0, 0, 0], [0, -np.exp(-x[1]), 0, 0],
                     [0, 0, -np.exp(-x[2]), 0], [0, 0, 0, -np.exp(-x[3])]])
```

```
x0_a = np.array([1/5, 1/5, 1/5, 1/5, 1/5])
x0_b = np.array([2/3, 1/3, 0, 0])
x_a = newton_method(f_a, grad_f_a, hessian_f_a, x0_a)
x_b = newton_method(f_b, grad_f_b, hessian_f_b, x0_b)
print("a) x = ", x_a)
print("b) x = ", x_b)
```

```
a) x =  [0.36787944 0.36787944 0.36787944 0.36787944 0.36787944]
b) x =  [-2.05960932e-07 -1.62596206e-12  0.00000000e+00  0.00000000e+00]
```

## Question 2

```
# Solve the following problem using log barrier method.
# Choose σ0 = 1, R = 10, x0 → Strictly feasible point.
# Stopping criterion m/σk < 10^-3

# min F(x) = x1 + 2x2 + 5x3 − 8x4 + 7x5 − 11x6
# s.t x1 − x2 + x3 = 0
# x1 − 2x2 + 2x3 + x4 + x6 = 3
# 2x3 + x4 − 5x5 + x6 = −2
# x2 + x3 + 2x4 − 3x5 + 2x6 = 1
# x1 + 3x3 − x4 + 2x6 = 2

def f(x):
    return x[0] + 2 * x[1] + 5 * x[2] - 8 * x[3] + 7 * x[4] - 11 * x[5]

def grad_f(x):
    return np.array([1, 2, 5, -8, 7, -11])

def log_barrier_method(f, grad_f, hessian_f, x0, sigma0=1, R=10, eps=1e-3):
    x = x0
    sigma = sigma0
    m = 6
    while True:
        grad = grad_f(x)
        hessian = hessian_f(x)
        delta = np.linalg.solve(hessian, grad)
        x = x - delta
        sigma = sigma * R
        if m / sigma < eps:
```

```
            break
    return x
```