Midterm 1

Name: Naman Makkar

Student ID: nbm49

- Reasoning and work must be shown to gain partial/full credit
- Please include the cover-page on your homework PDF with your name and student ID. Failure of doing so is considered bad citizenship.

In this homework, you are going to learn about optimization problems applied to energy systems. This assignment leverages some of the work you did for Homework 2. **Please note** that the IEEE-118 case has been modified from last homework and may include different branches or new features. All the data you need to run these studies is provided to you in the datasets folder that contains the following

1. *branches_ieee118_subset.csv*. This file contains information about the edges of the IEEE-118 network. The file contains the following features.

   - **fbus**: The "from" bus ID
   - **tbus**" The "to" bus ID
   - **r**: The resistance in p.u.
   - **x**: The reactance in p.u.
   - **rateA**: The thermal limit (capacity) of the line in MW.
   - **ratio**: transformer off nominal turns ratio

2. *buses_ieee118_subset.csv*. This file contains information about the nodes of the IEEE-118 network. The file contains the following features.

   - **bus_i**: ID of the bus
   - **Pd**: Real power demand in MW
   - **baseKV**: base voltage in kV

3. *generators_ieee118_subset.csv*. This file contains information about the nodes of the IEEE-118 network. The file contains the following features.

   - **bus**: ID of the bus that the generator is connected to
   - **mBase**: Total MVA base of machine
   - **c2,c1,c0**: Coefficients of the generator cost function of the form $f(x) = c_2x^2 + c_1x + c_0$
   - **Pmax,Pmin**: Capacity limits of the generator in MW.

1. (3.5 points) **Fundamentals of optimization**:

   (a) (**0.25 pts**) Indicate whether the following functions are convex or non-convex.

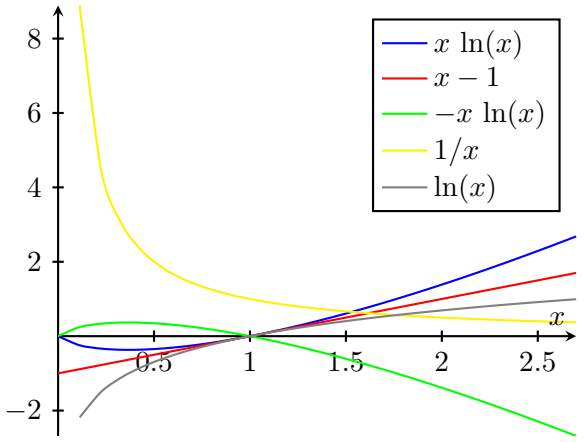

Figure 1: Convex and non-convex functions

Solution: $\mathbf{x\ln(x)}$ is a convex function on $(0,\infty)$, $\mathbf{x-1}$ is a convex function on $(-\infty,\infty)$, $\mathbf{-x\ln(x)}$ is a **non-convex function**, $\mathbf{1/x}$ is a convex function on $(0,\infty)$ and $\mathbf{\ln(x)}$ is a non-convex function.

   (b) (**0.25 pts**) Indicate whether the following sets are convex or non-convex.

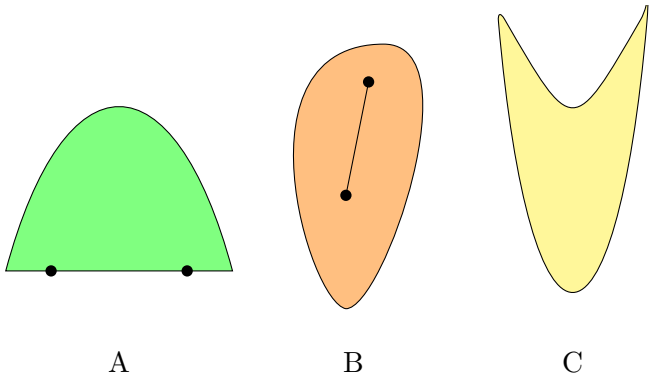

Figure 2: Sets

Solution: SET A is convex, **SET B** is convex, **SET C** is non-convex.

   (c) (**0.5 pts**) Plot the feasible region of the following problem, and indicate whether the problem is convex or non-convex. If feasible and bounded, provide the optimal solution. (**Note**: you can draw the feasible region by hand and upload a picture, or you can draw

it in LaTeX. **Hint**: The problem may have more than one optimal solution.)

$$\begin{array}{ll} \min & x_1 + x_2 \\ \text{s.t.} & x_1^2 + x_2^2 \leq 4 \\ & x_1 + x_2 \geq 1 \\ & x_1, x_2 \geq 0 \end{array} \tag{1}$$

Solution:

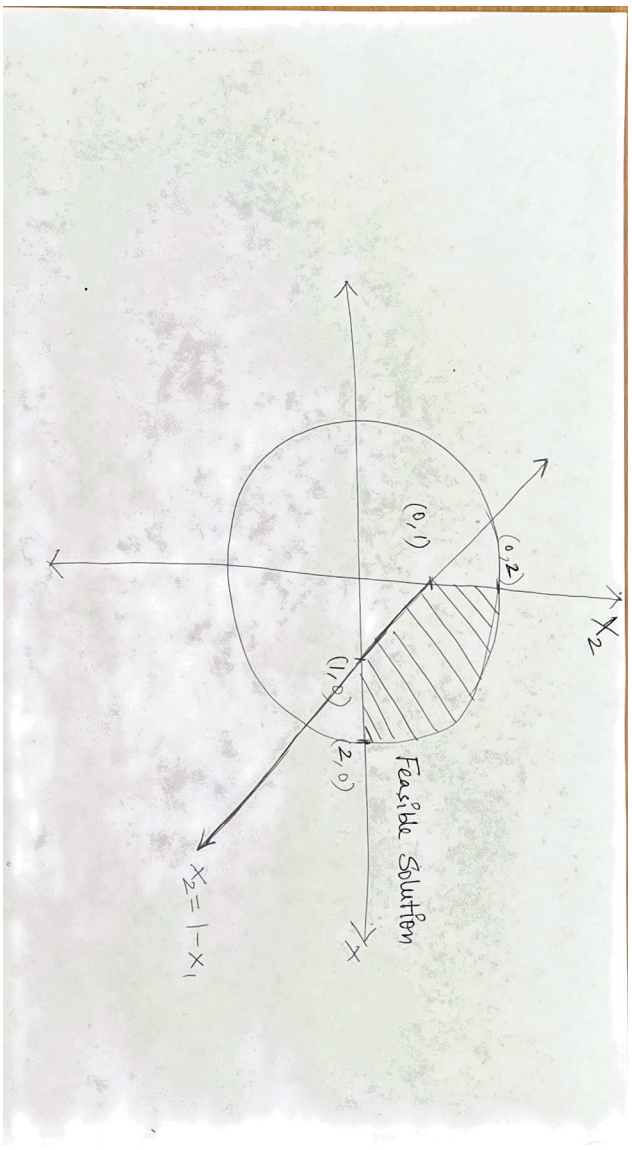

Figure 3: Plot for the feasible solution for 1.(c)

(d) (**0.5 pts**) Repeat part (c) for the following problem.

$$\begin{array}{ll} \min & (x_1-2)^2+(x_2-2)^2 \\ \text{s.t.} & x_1^2+x_2^2 \geq 4 \\ & x_1+x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{array} \tag{2}$$

Solution:

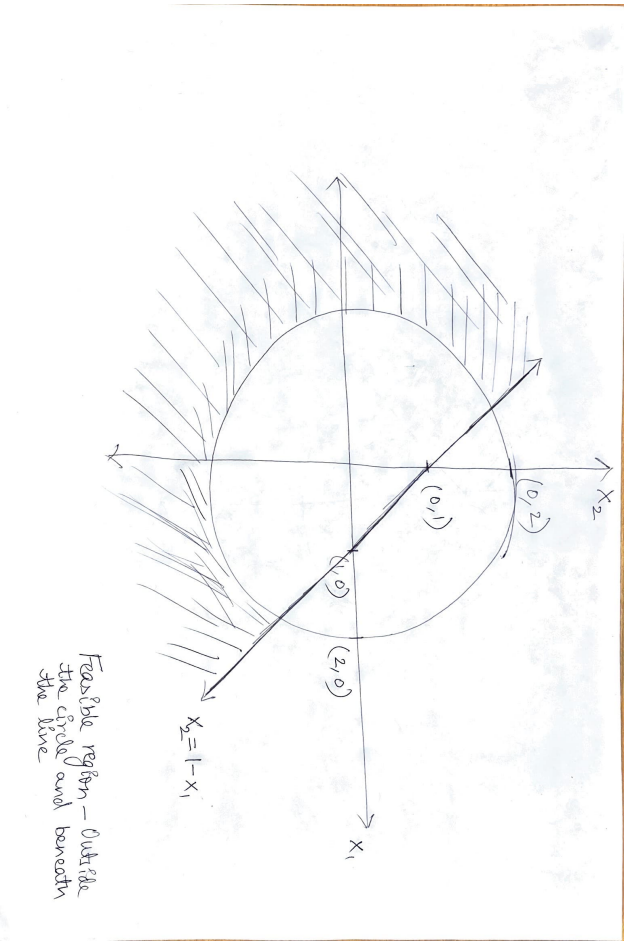

Figure 4: Plot for the feasible solution for 1.(d)

(e) (**0.5 pts**) Repeat part (c) for the following problem. In the problem below, is $x + y = 10$ a feasible solution? Why?

$$\begin{array}{ll} \max & x \\ \text{s.t.} & x+y < 10 \\ & xy <= 2 \\ & x, y \geq 0 \end{array} \tag{3}$$

Solution: $x + y = 10$ is **not a feasible solution** since it does not satisfy the first constraint which is $x + y < 10$ even though it does satisfy the second constraint $xy <= 2$ with $x = 9.9$ and $y = 0.1$ for example.

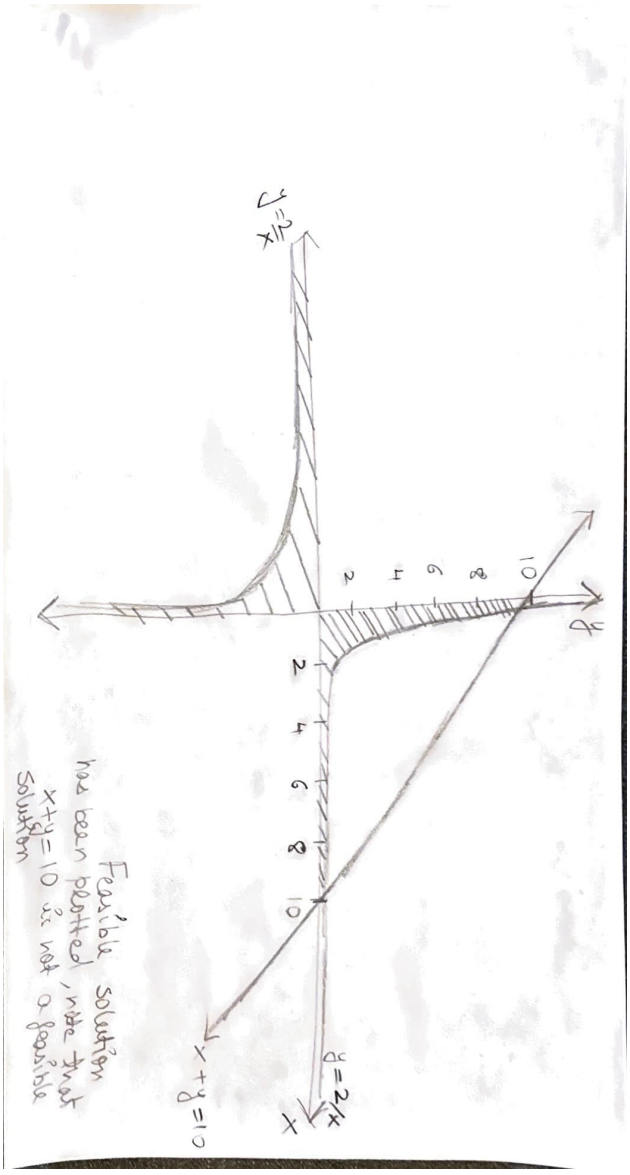

Figure 5: Plot for the feasible solution for 1.(e)

(f) (**1.5 pts**) Consider the linear program

$$\begin{array}{ll} \min & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array} \tag{4}$$

- (**0.25 pts**) Find the dual problem
- (**0.25 pts**) Provide the expression of the Lagrangian of the problem

- (**1 pts**) Provide the KKT conditions
  1. (**0.25 pts**) Primal Feasibility
  2. (**0.25 pts**) Dual Feasibility
  3. (**0.25 pts**) Complementary Slackness
  4. (**0.25 pts**) Stationarity of the Lagrangian

Solution: The dual problem is:

$$\begin{array}{cc} \max & \mathbf{b}^{\mathrm{T}}\mathbf{y} \text{ s.t.} \\ \mathbf{A}^{\mathrm{T}}\mathbf{y} \leq \mathbf{c} & \mathbf{y} \geq 0 \end{array} \tag{5}$$

The Lagrangian of the problem is:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) = \mathbf{c}^{\mathrm{T}}\mathbf{x} + \boldsymbol{\lambda}^{\mathrm{T}}(\mathbf{b} - \mathbf{A}\mathbf{x}) - \mathbf{y}^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}}\mathbf{y} - \mathbf{c}) \tag{6}$$

The KKT conditions are:

Primal Feasibility:

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq 0 \tag{7}$$

Dual Feasibility:

$$\mathbf{A}^{\mathrm{T}}\mathbf{y} \leq \mathbf{c}, \mathbf{y} \geq 0 \tag{8}$$

Complementary Slackness:

$$\boldsymbol{\lambda}^{\mathrm{T}}(\mathbf{b} - \mathbf{A}\mathbf{x}) = 0, \mathbf{y}^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}}\mathbf{y} - \mathbf{c}) = 0 \tag{9}$$

Stationarity of the Lagrangian:

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) = \mathbf{c} - \mathbf{A}^{\mathrm{T}}\boldsymbol{\lambda} = 0, \nabla_{\mathbf{y}}\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) = \mathbf{A}^{\mathrm{T}}\mathbf{y} - \mathbf{c} = 0 \tag{10}$$

2. (2.5 points) **DC Optimal Power Flow**: In this problem, we are going to use the DC approximation to run an Optimal Power Flow (OPF) problem and calculate the generator setpoints. Power networks can be denoted as a graph $\mathcal{G} := \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Each edge $e := \{i, j\} \in \mathcal{E}$ connects two buses $i, j \in \mathcal{V}$. Also, $N := |\mathcal{V}|$ denotes the number of buses and $E := |\mathcal{E}|$ is the number of deges (the operator $|.|$ denotes the cardinality of a set). The DC OPF is a problem of the form

$$\begin{array}{lll} \min_{\mathbf{p}} & \mathbf{p}^\intercal \mathbf{C}_2 \mathbf{p} + \mathbf{c}_1^\intercal \mathbf{p} + \mathbb{1}^\intercal \mathbf{c}_0 & \\ \text{s.t.} & \mathbf{p} - \mathbf{d} = \mathbf{B}\boldsymbol{\theta} & (\boldsymbol{\lambda}) \\ & \underline{\mathbf{p}} \leq \mathbf{p} \leq \overline{\mathbf{p}} & (\boldsymbol{\beta}) \\ & \underline{\mathbf{f}} \leq \text{diag}(\mathbf{b})\mathbf{M}^\intercal \boldsymbol{\theta} \leq \overline{\mathbf{f}} & (\boldsymbol{\mu}) \end{array} \tag{11}$$

where $\mathbf{p} \in \mathbb{R}^N$ is the vector of power injections, $\mathbf{d}$ is the vector of power demands. The generator cost constants are given by $\mathbf{C}_2 := \text{diag}(\mathbf{c}_2)$ where $\mathbf{c}_2$ as well as $\mathbf{c}_1$ and $\mathbf{c}_0$ are the vectors of constants. Also, $\mathbf{B} \in \mathbb{R}^{N \times N}$ is the susceptance matrix and $\boldsymbol{\theta} \in \mathbb{R}^N$ is the vector of bus angles. $\mathbf{M} \in \{-1, 0, 1\}^{N \times E}$ is the incidence of the graph, $\mathbf{b} := [b_1, \ldots, b_E]^\intercal \in \mathbb{R}^E$ is the vector of line susceptances where $b_e := 1/(\tau_e x_e), \forall e \in \mathcal{E}$ (See Homework 2). Also, $\overline{\mathbf{p}}$, $\underline{\mathbf{p}}$, $\overline{\mathbf{f}}$ and $\underline{\mathbf{f}}$ are the maximum and minimum generator capacity and line capacity limits. It should be noted that $\underline{\mathbf{f}} := -\overline{\mathbf{f}}$. The vectors $\boldsymbol{\lambda}$, $\boldsymbol{\beta}$ and $\boldsymbol{\mu}$ are the dual variables of their respective constraints, and $\mathbb{1}$ is the all-ones vector.

(a) (**0.25 pts**) What are you trying to calculate when solving a DC optimal power flow problem? What can you obtain as a byproduct of solving the dual problem?

Solution: When solving a DC optimal power flow problem we aim to calculate the optimal power output of generators and the power flow through transmission lines by subjecting them to constraints such as generator capacity limits, line flow limits, and power balance limits. We can calculate marginal energy prices as a byproduct of solving the dual problem.

(b) (**0.25 pts**) Using your code from Homework 2, obtain $\mathbf{B}$, $\mathbf{M}$, $\mathbf{b}$ and $\mathbf{d}$.

Solution:

In [1]:

```
import numpy as np
import pandas as pd
branches = pd.read_csv('/kaggle/input/energy-systems-hw2/datasets/branches_ieee118_subset.csv')
buses = pd.read_csv('/kaggle/input/energy-systems-hw2/datasets/buses_ieee118_subset.csv')
generators = pd.read_csv('/kaggle/input/energy-systems-hw2/datasets/generators_ieee118_subset.csv')

# Incidence Matrix Calculation
E = len(branches['fbus'])
N = max(branches['tbus'])
M = [[0 for _ in range(N)] for _ in range(E)]
fbus = list(branches['fbus'])
tbus = list(branches['tbus'])

for edge_num in range(E):
    node1 = fbus[edge_num]
    node2 = tbus[edge_num]
    M[edge_num][node1-1] = -1
    M[edge_num][node2-1] = 1
M = np.array(M)
M = M.T
print(M)
```

Out[1]:

```

[[-1 -1  0 ...  0  0  0]
 [ 1  0  0 ...  0  0  0]
 [ 0  1  0 ...  0  0  0]
 ...
 [ 0  0  0 ...  0  0  0]
 [ 0  0  0 ...  1  0  0]
 [ 0  0  0 ...  0  1  1]]
```

In [2]:

```
transformer_turn_ratios = list(branches['ratio'])
reactance = list(branches['x'])
susceptances = [0 for _ in range(E)]

for edge_num in range(E):
    susceptances[edge_num] = 1/(transformer_turn_ratios[edge_num]*reactance[edge_num])
b = np.array(susceptances)
b_diag = np.diag(b)
print(b_diag)
```

Out[2]:

```
array([[ 10.01001001,   0.        ,   0.        , ...,   0.        ,
          0.        ,   0.        ],
       [  0.        ,  23.58490566,   0.        , ...,   0.        ,
          0.        ,   0.        ],
       [  0.        ,   0.        , 125.31328321, ...,   0.        ,
          0.        ,   0.        ],
       ...,
       [  0.        ,   0.        ,   0.        , ...,   7.14285714,
          0.        ,   0.        ],
       [  0.        ,   0.        ,   0.        , ...,   0.        ,
         20.79002079,   0.        ],
       [  0.        ,   0.        ,   0.        , ...,   0.        ,
          0.        ,  18.38235294]])
```

In [3]:

```
# Susceptance Matrix calculation
B = M @ np.diag(b) @ M.T
print(B)
```

Out[3]:

```
[[ 33.59491567 -10.01001001 -23.58490566 ...   0.           0.
    0.        ]
 [-10.01001001  26.24377624   0.         ...   0.           0.
    0.        ]
 [-23.58490566   0.          39.09416492 ...   0.           0.
    0.        ]
 ...
 [  0.           0.           0.         ... 246.91358025   0.
    0.        ]
 [  0.           0.           0.         ...   0.
    7.14285714
    0.        ]
 [  0.           0.           0.         ...   0.           0.
   39.17237373]]
```

In [4]:

```
d = np.array(buses.Pd) / 100
d
```

Out[4]:

```
array([0.51, 0.2 , 0.39, 0.39, 0.  , 0.52, 0.19, 0.28, 0.  , 0.  ,
       0.7 ,
          0.47, 0.34, 0.14, 0.9 , 0.25, 0.11, 0.6 , 0.45, 0.18, 0.14,
       0.1 ,
          0.07, 0.13, 0.  , 0.  , 0.71, 0.17, 0.24, 0.  , 0.43, 0.59,
       0.23,
          0.59, 0.33, 0.31, 0.  , 0.  , 0.27, 0.66, 0.37, 0.96, 0.18,
       0.16,
          0.53, 0.28, 0.34, 0.2 , 0.87, 0.17, 0.17, 0.18, 0.23, 1.13,
       0.63,
          0.84, 0.12, 0.12, 2.77, 0.78, 0.  , 0.77, 0.  , 0.  , 0.  ,
       0.39,
          0.28, 0.  , 0.  , 0.66, 0.  , 0.12, 0.06, 0.68, 0.47, 0.68,
       0.61,
          0.71, 0.39, 1.3 , 0.  , 0.54, 0.2 , 0.11, 0.24, 0.21, 0.  ,
       0.48,
          0.  , 1.63, 0.1 , 0.65, 0.12, 0.3 , 0.42, 0.38, 0.15, 0.34,
       0.42,
          0.37, 0.22, 0.05, 0.23, 0.38, 0.31, 0.43, 0.5 , 0.02, 0.08,
       0.39,
          0.  , 0.68, 0.06, 0.08, 0.22, 1.84, 0.2 , 0.33])
```

(c) (**0.25 pts**) Create $\mathbf{C}_2$, $\mathbf{c}_1$ and $\mathbf{c}_0$ using the data provided.

Solution:

In [5]:

```
c2=np.array(merged_bus_gen.c2)
c2
```

Out[5]:

```
array([0.01      , 0.        , 0.        , 0.01      , 0.        ,
       0.01      , 0.        , 0.01      , 0.        , 0.02222222,
       0.        , 0.11764706, 0.        , 0.        , 0.01      ,
       0.        , 0.        , 0.01      , 0.01      , 0.        ,
       0.        , 0.        , 0.        , 0.01      , 0.04545455,
       0.03184713, 0.01      , 0.        , 0.        , 0.        ,
       1.42857143, 0.01      , 0.        , 0.01      , 0.        ,
       0.01      , 0.        , 0.        , 0.        , 0.01      ,
       0.        , 0.01      , 0.        , 0.        , 0.        ,
       0.52631579, 0.        , 0.        , 0.04901961, 0.        ,
       0.        , 0.        , 0.        , 0.20833333, 0.01      ,
       0.01      , 0.        , 0.        , 0.06451613, 0.        ,
       0.0625    , 0.01      , 0.        , 0.        , 0.02557545,
       0.0255102 , 0.        , 0.        , 0.01936483, 0.01      ,
       0.        , 0.01      , 0.01      , 0.01      , 0.        ,
       0.01      , 0.01      , 0.        , 0.        , 0.02096436,
       0.        , 0.        , 0.        , 0.        , 0.01      ,
       0.        , 2.5       , 0.        , 0.01647447, 0.01      ,
       0.01      , 0.01      , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.01      , 0.03968254,
       0.        , 0.        , 0.25      , 0.01      , 0.01      ,
       0.        , 0.01      , 0.        , 0.        , 0.01      ,
       0.27777778, 0.01      , 0.01      , 0.        , 0.        ,
       0.01      , 0.        , 0.        ])
```

In [6]:

```
C2 = np.diag(generators.c2)
C2
```

Out[6]:

```
array([[0.01, 0.  , 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.01, 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.01, ..., 0.  , 0.  , 0.  ],
       ...,
       [0.  , 0.  , 0.  , ..., 0.01, 0.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 0.01, 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 0.  , 0.01]])
```

In [7]:

```
c1 = np.array(generators.c1)
c1
```

Out[7]:

```
array([40, 40, 40, 40, 20, 20, 40, 40, 40, 40, 20, 20, 40, 20, 40,
    40, 40,
       40, 40, 20, 20, 20, 40, 40, 20, 20, 40, 20, 20, 20, 40, 40,
    40, 40,
       40, 40, 20, 40, 20, 20, 40, 40, 40, 40, 20, 20, 40, 40, 40,
    40, 20,
       40, 40, 40])
```

In [8]:

```
c0 = np.array(generators.c0)
c0
```

Out[8]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

(d) (**0.25 pts**) Build the vectors of capacity limits $\overline{\mathbf{p}}$, $\underline{\mathbf{p}}$, $\overline{\mathbf{f}}$ and $\underline{\mathbf{f}}$. (**Note**: You need to divide the values in MW by 100 to match the per-unit system)

Solution:

In [9]:
```
p_max = np.array(generators.Pmax) / 100
p_min = np.array(generators.Pmin) / 100
p_max
```

Out[9]:
```
array([1.   , 1.   , 1.   , 1.   , 5.5  , 1.85 , 1.   , 1.   , 1.
       ,
          1.   , 3.2  , 4.14 , 1.   , 1.07 , 1.   , 1.   , 1.   , 1.
       ,
          1.   , 1.19 , 3.04 , 1.48 , 1.   , 1.   , 2.55 , 2.6  , 1.
       ,
          4.91 , 4.92 , 8.052, 1.   , 1.   , 1.   , 1.   , 1.   , 1.
       ,
          5.77 , 1.   , 1.04 , 7.07 , 1.   , 1.   , 1.   , 1.   , 3.52
       ,
          1.4  , 1.   , 1.   , 1.   , 1.   , 1.36 , 1.   , 1.   , 1.
       ])
```

In [10]:
```
p_min
```

Out[10]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0.,
          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0.,
          0., 0., 0.])
```

In [11]:
```
f_max = np.array(branches.rateA)
f_max
```

Out[11]:

```
array([175, 175, 175, 175, 175, 175, 440, 440, 550, 440, 440, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 200, 175,
       175,
          175, 175, 300, 440, 440, 175, 175, 175, 175, 175, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 350, 175, 175, 440, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175,
       400,
          440, 200, 440, 440, 175, 175, 175, 175, 200, 175, 440, 440,
       175,
          175, 350, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 440,
       440,
          200, 175, 175, 175, 175, 175, 175, 175, 175, 200, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 175,
       175,
          175, 175, 175, 175, 175, 175, 175, 175, 175, 175, 440, 440,
       175,
          175, 175])
```

In [12]:

```
f_min = - f_max
f_min
```

Out[12]:

```
array([-175, -175, -175, -175, -175, -175, -440, -440, -550, -440,
       -440,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -200, -175, -175, -175, -175, -300, -440,
       -440,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -350, -175, -175, -440, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -400, -440, -200, -440, -440, -175,
       -175,
          -175, -175, -200, -175, -440, -440, -175, -175, -350, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -440,
       -440,
          -200, -175, -175, -175, -175, -175, -175, -175, -175, -200,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -175, -175, -175,
       -175,
          -175, -175, -175, -175, -175, -175, -175, -440, -440, -175,
       -175,
          -175])
```

(e) (**0.75 pts**) Install the Python package *cvxpy*[1] and solve the problem provided in Eq. (11). Provide the optimal cost of the problem and a table containing the generators producing more than 0.1 MWs (note that you need to multiple your results by 100, to go from per-unit base back to MWs). The table should contain three columns, 1) generator id, 2) the bus id where the generator is connected and 3) the generation in MWs. (**Hint**: Eq. (11) is a quadratic program. Use this example[2] as a reference.)

Solution:

[1] https://www.cvxpy.org/

[2] https://www.cvxpy.org/examples/basic/quadratic_program.html

In [13]:

```
!pip install cvxpy
import cvxpy as cp
import numpy as np

Egen = np.zeros((118, 54))
for idx, bus in enumerate(generators.bus):
    Egen[bus-1, idx] = 1

n = 118
m = 210
p = cp.Variable(54)
theta = cp.Variable(n)

constraints = [
    Egen @ p - d == B @ theta,
    p <= p_max,
    p >= p_min,
    f_min <= b_diag @ M.T @ theta,
    f_max >= b_diag @ M.T @ theta,
]

objective = cp.Minimize(p.T @ C2 @ p + c1 @ p + np.ones((1, len
    (c0))) @ c0)

problem = cp.Problem(objective, constraints)
problem.solve()
```

Out[13]:

```
852.5107881843143
```

In [14]:

```
print(f'Optimal Value: {problem.value}')
print(f'Value of power Injection p is: {p.value}')
print(f'A dual solution corresponding to the 1st constraints is
     : {problem.constraints[0].dual_value}')
```

Out[14]:

```
Optimal Value: 852.5107881843143
Value of power Injection p is: [-1.75082603e-14 -1.71881674e-14
    -1.73974437e-14 -1.60971520e-14
  4.36080834e+00  8.23708235e-01 -1.40283234e-14 -1.38926479e-14
 -1.36334832e-14 -9.30861701e-15  2.13195065e+00  3.04287503e+00
 -1.41196197e-14  6.78347945e-02 -1.39087316e-14 -9.60191343e-15
 -9.72456050e-15 -7.38611193e-15 -4.88388019e-15  1.84122998e-01
  1.97689955e+00  4.65152835e-01  3.26783347e-16  3.05910412e-16
  1.50205603e+00  1.55050945e+00  5.25904425e-16  3.78905740e+00
  3.79874816e+00  5.00426904e+00  3.62466005e-16 -4.70304007e-15
 -4.32907159e-16  2.16059365e-15  4.06188846e-15  5.81358169e-15
  4.62245603e+00  1.12995585e-14  3.87627335e-02  5.88224462e+00
  1.23463406e-14  1.23417211e-14  1.20436571e-14  1.01169678e-14
  2.44205219e+00  3.87627332e-01  1.32130871e-14  1.35007691e-14
  1.35020932e-14  1.42241263e-14  3.48864595e-01  1.43274206e-14
 -1.40797626e-14  2.09992059e-15]
A dual solution corresponding to the 1st constraints is :
    [-20.1938137  -20.1938137  -20.1938137  -20.1938137  -20.1938137
 -20.1938137  -20.1938137  -20.1938137  -20.1938137  -20.1938137
 -20.1938137  -20.1938137  -20.1938137  -20.1938137  -20.1938137
 -20.1938137  -20.1938137  -20.1938137  -20.1938137  -20.1938137
 -20.1938137  -20.1938137  -20.1938137  -20.19381369 -20.1938137
 -20.1938137  -20.1938137  -20.1938137  -20.1938137  -20.1938137
 -20.1938137  -20.1938137  -20.1938137  -20.19381369 -20.19381369
 -20.19381369 -20.19381369 -20.19381369 -20.19381369 -20.19381369
 -20.19381369 -20.19381369 -20.19381369 -20.19381369 -20.19381368
 -20.19381368 -20.19381368 -20.19381368 -20.19381368 -20.19381368
 -20.19381368 -20.19381368 -20.19381368 -20.19381368 -20.19381368
 -20.19381368 -20.19381368 -20.19381368 -20.19381368 -20.19381368
 -20.19381368 -20.19381368 -20.19381368 -20.19381368 -20.19381368
 -20.19381368 -20.19381369 -20.19381368 -20.19381368 -20.19381368
 -20.19381368 -20.19381367 -20.19381367 -20.19381367 -20.19381367
 -20.19381368 -20.19381367 -20.19381367 -20.19381367 -20.19381367
 -20.19381367 -20.19381367 -20.19381367 -20.19381367 -20.19381367
 -20.19381367 -20.19381367 -20.19381367 -20.19381367 -20.19381367
 -20.19381367 -20.19381367 -20.19381367 -20.19381367 -20.19381367
 -20.19381367 -20.19381367 -20.19381366 -20.19381366 -20.19381366
 -20.19381366 -20.19381366 -20.1938137  -20.1938137  -20.1938137
 -20.19381368 -20.1938137  -20.19381368]
```

In [15]:

```
print(f'A dual solution corresponding to the 2nd constraints is
    : {problem.constraints[1].dual_value}')
```

Out[15]:

```
A dual solution corresponding to the 2nd constraints is : [0. 0. 0.
   0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
   0. 0.
 0. 0. 0. 0. 0. 0.]
```

In [16]:

```python
print(f'A dual solution corresponding to the 3rd constraints is : {problem.constraints[2].dual_value}')
```

Out[16]:

```
A dual solution corresponding to the 3rd constraints is : [19.8061863  19.8061863  19.8061863  19.8061863   0.          0.
 19.8061863  19.8061863  19.8061863  19.80618631  0.          0.
 19.8061863   0.         19.8061863  19.80618631 19.80618631 19.80618631
 19.80618631  0.          0.          0.         19.80618632 19.80618632
  0.          0.         19.80618632  0.          0.          0.
 19.80618632 19.80618631 19.80618632 19.80618632 19.80618632 19.80618633
  0.         19.80618633  0.          0.         19.80618633 19.80618633
 19.80618633 19.80618633  0.          0.         19.80618633 19.80618633
 19.80618633 19.80618634  0.         19.80618634 19.8061863  19.80618632]
```

In [17]:

```python
print(f'A dual solution corresponding to the 4th constraints is : {problem.constraints[3].dual_value}')
```

Out[17]:

```
A dual solution corresponding to the 4th constraints is : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [18]:

```python
print(f'A dual solution corresponding to the 5th constraints is : {problem.constraints[4].dual_value}')
```

```
Out[18]: A dual solution corresponding to the 5th constraints is : [0. 0. 0.
    0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

It can be observed from the dual solution obtained that, $\boldsymbol{\mu}$ is 0 vector while $\boldsymbol{\lambda}$ and $\boldsymbol{\beta}$ are not. And $\boldsymbol{\beta}$ provides us the list of binding generator capacity constraints.

(f) (**0.75 pts**) Provide a list of binding transmission lines and generator capacity constraints. Why are these constraints binding? In the case of transmission line constraints, provide the power flowing through the line of the binding constraints. In the case of the generator capacity constraints, provide the generation of the constraints that are binding. Create a table like you did in part (e) (**Note**: Do not forget to consider the maximum and minimum capacity constraints)

Solution: Using the minimum power capacity constraints we obtain a list of generators that are binding for the constraints, i.e the values of the dual variable of the constraint that are non-zero. We do not obtain a list of binding transmission lines due to the fact that the dual variable for the maximum and minimum line transmission capacity constraints are 0 vectors.

```
In [19]:
a = np.array(problem.constraints[2].dual_value)
a /= a.max()
a
```

```
Out[19]: array([1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1.,
    1., 1.,
       1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 1.,
    1., 1.,
       1., 1., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1.,
    1., 0.,
       1., 1., 1.])
```

```
In [20]:
b = np.array(generators.bus)
l = np.round(a * b)
l[l != 0]
```

Out[20]:
```
array([  1.,   4.,   6.,   8.,  15.,  18.,  19.,  24.,  27.,  32.,
    34.,
        36.,  40.,  42.,  55.,  56.,  62.,  70.,  72.,  73.,  74.,
    76.,
        77.,  85.,  90.,  91.,  92.,  99., 104., 105., 107., 110.,
    112.,
       113., 116.])
```

Therefore, we obtain a list of binding generator buses for the constraint $\underline{\mathbf{p}} \leq \mathbf{p}$

3. (2 points) **AC Power Flow**: AC problems are non-linear and non-convex and thus, are hard to solve. In this problem, we are going to solve a simple network by hand to gain an understanding of the problem. The network we are going to use is shown below where the resistance and reactance are in per-unit values.

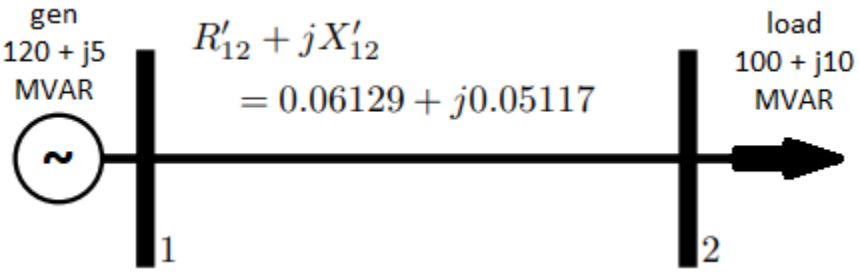

Figure 6: 2-bus network

(a) (**0.5 pts**) Formulate the active power flow equations on both nodes.
Solution: $\mathbf{P_1} = \mathbf{V_1V_2}(\mathbf{G_{12}}\cos\theta_{12} + \mathbf{B_{12}}\sin\theta_{12}) + \mathbf{V_1V_1}(\mathbf{G_{11}}\cos\theta_{11} + \mathbf{B_{11}}\sin\theta_{11})$
$\mathbf{P_2} = \mathbf{V_2V_1}(\mathbf{G_{21}}\cos\theta_{21} + \mathbf{B_{21}}\sin\theta_{21}) + \mathbf{V_2V_2}(\mathbf{G_{22}}\cos\theta_{22} + \mathbf{B_{22}}\sin\theta_{22})$
Using the admittance matrix and the resistance and reactance without a shunt we know that $\mathbf{G_{11}} + \mathbf{B_{11}} = \frac{1}{0.06129+0.05117j} = 9.6 - 8j$, therefore $\mathbf{G_{11}} = 9.6, \mathbf{B_{11}} = -8j$ $\mathbf{G_{12}} + \mathbf{B_{12}} = -\frac{1}{0.06129+0.05117j} = -9.6 + 8j$, therefore $\mathbf{G_{12}} = -9.6, \mathbf{B_{12}} = 8j$
Using the admittance matrix, we know that $\mathbf{G_{11}} + \mathbf{B_{11}} = \mathbf{G_{22}} + \mathbf{B_{22}}$ and $\mathbf{G_{12}} + \mathbf{B_{12}} = \mathbf{G_{21}} + \mathbf{B_{21}}$ Therefore we can write the above equations as -

$$120 = \mathbf{V_1V_2}(-9.6\cos\theta_{12} + 8j\sin\theta_{12}) + \mathbf{V_1V_1}(9.6\cos\theta_{11} - 8j\sin\theta_{11})$$

$$100 = \mathbf{V_2V_1}(-9.6\cos\theta_{21} + 8j\sin\theta_{21}) + \mathbf{V_2V_2}(9.6\cos\theta_{22} - 8j\sin\theta_{22})$$

(b) (**0.5 pts**) Formulate the reactive power flow equations on both nodes.
Solution: $\mathbf{Q_1} = \mathbf{V_1V_2}(\mathbf{G_{12}}\sin\theta_{12} - \mathbf{B_{12}}\cos\theta_{12}) + \mathbf{V_1V_1}(\mathbf{G_{11}}\sin\theta_{11} - \mathbf{B_{11}}\cos\theta_{11})$
$\mathbf{Q_2} = \mathbf{V_2V_1}(\mathbf{G_{21}}\sin\theta_{21} - \mathbf{B_{21}}\cos\theta_{21}) + \mathbf{V_2V_2}(\mathbf{G_{22}}\sin\theta_{22} - \mathbf{B_{22}}\cos\theta_{22})$

Using the values of $\mathbf{G_{11}}, \mathbf{G_{12}}, \mathbf{G_{22}}, \mathbf{G_21}, \mathbf{B_{11}}, \mathbf{B_{12}}, \mathbf{B_{21}}, \mathbf{B_{22}}$ calculated above, we can calculate the reactive powerflow as -

$$5j = \mathbf{V_1V_2}(-9.6\sin\theta_{12} - 8j\cos\theta_{12}) + \mathbf{V_1V_1}(9.6\sin\theta_{11} + 8j\cos\theta_{11})$$

$$10j = \mathbf{V_2V_1}(-9.6\sin\theta_{21} - 8j\cos\theta_{21}) + \mathbf{V_2V_2}(9.6\sin\theta_{22} + 8j\cos\theta_{22})$$

(c) (**1 pts**) Fix the voltage magnitude and angle at one of the nodes (you can do 1 p.u. voltage magnitude and 0 degree angle) and solve the equations. The method you chose to solve the equations is your choice. Provide a drawing where you show the the voltage magnitude and angle on both nodes, and the power flow on the line.