Convex Optimization

Tutorial 12

Tanmay Garg CS20BTECH11063

In [ ]:
```
#Importing required Libraries
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
import math
```

(a)

The function is convex because $c^T x$ is linear hence it is convex. $f_0(x)$ is max or sup of convex functions, hence it is also convex

(b), (c)

Constructing F and g

$$F = \begin{bmatrix} I_{n\times n} \\ -I_{n\times n} \\ \vec{1}^T\vec{c} \\ -\vec{1}^T\vec{c} \end{bmatrix}$$

$$g = \begin{bmatrix} 1.25c_{nom} \\ -0.75c_{nom} \\ \vec{1}^T c_{nom} \\ -\vec{1}^T c_{nom} \end{bmatrix}$$

Dual Problem

The dual problem will be

$$\inf_{c} \quad c^T x + \lambda^T F c - \lambda^T g$$

such that

$$\lambda \geq 0$$

Final LP of the dual form is

The convex problem will be

$$\min_{x,\lambda} \quad \lambda^T g$$

such that

$$F^T \lambda = x$$
$$\lambda \geq 0$$
$$Ax \geq b$$

Primal problem is feasible, so the dual problem should also be feasible as strong duality holds here as it is an LP problem (affine). Hence optimal value of primal and dual are same

In [ ]:
```
np.random.seed(10)
(m, n) = (30, 10)
A = np.random.rand(m, n)
A = np.asmatrix(A)
b = np.random.rand(m, 1)
b = np.asmatrix(b)
c_nom = np.ones((n, 1)) + np.random.rand(n, 1)
c_nom = np.asmatrix(c_nom)

# print(c_nom)
```

In [ ]:
```
# import prettytable
# from prettytable import PrettyTable
# import pandas as pd
```

In [ ]:
```
F = np.hstack((np.identity(n), -np.identity(n), np.ones((n, 1)), -np.ones((n, 1))))
F = F.T

# print(F)
# print(pd.DataFrame(F))
```

```
# g = np.vstack((1.25*c_nom, -0.75*c_nom, 1.1*np.sum(c_nom)/n, -0.9*np.sum(c_nom)/n))

g = np.vstack((1.25*c_nom, -0.75*c_nom, 1.1*np.sum(c_nom), -0.9*np.sum(c_nom)))

lambd = cp.Variable(g.shape)
x = cp.Variable((n,1))

# print(F.shape)
# print(lambd.shape)
# print(g.shape)
```

In [ ]:

```
# print(F)
```

In [ ]:

```
# print(g)
```

In [ ]:

```
MyConstraints = [
    A@x >= b,
    F.T@lambd == x,
    lambd >= 0
]

MyProblem = cp.Problem(cp.Minimize(lambd.T @ g), MyConstraints)
value = MyProblem.solve()
# print(value)

robust_x_val = x.value
```

In [ ]:

```
new_x_var = cp.Variable((n,1))
MyConstraints_part2 = [
    A@new_x_var >= b
]

MyProblem2 = cp.Problem(cp.Minimize(c_nom.T@new_x_var), MyConstraints_part2)
value2 = MyProblem2.solve()

# print(value2)
new_x_val = new_x_var.value
```

In [ ]:

```
c = cp.Variable((n,1))
MyConstraints_part3 = [
    F@c <= g
]
```

```
worst_case_cost_f_robust = cp.Problem(cp.Maximize(c.T@robust_x_val), MyConstraints_part3).solve()
worst_case_cost_f_normal = cp.Problem(cp.Maximize(c.T@new_x_val), MyConstraints_part3).solve()
```

In [ ]:

```
# print(c_nom.T@robust_x_val)
# print(c_nom.T@new_x_val)
```

In [ ]:

```
print("The worst case cost of f(x) in case of robust is: ", worst_case_cost_f_robust)
print("The worst case cost of f(x) in case of nominal is: ", worst_case_cost_f_normal)
nominal_cost_robust = c_nom.T@robust_x_val
print("The nominal cost in robust case: ", (c_nom.T@robust_x_val)[0,0])
print("The nominal cost in nominal case: ", (c_nom.T@new_x_val)[0,0])
```

```
The worst case cost of f(x) in case of robust is:  3.1659610511015486
The worst case cost of f(x) in case of nominal is:  7.221562200521366
The nominal cost in robust case:  2.5232088648898556
The nominal cost in nominal case:  2.1092714620825994
```

In robust case the nominal cost is slightly greater than the nominal cost in nominal case and the worst case cost in robust is less than that of nominal case due to in robust case we are looking at a set of values and it chooses the value which fits the optimization problem the best while in nominal case it looks at only one value