AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Mathematics for Intelligent Systems - 2

Convex Optimisation and Applications

Aadharsh Aadhithya CEN - CB.EN.U4AIE20001
Anirudh Edpuganti - CB.EN.U4AIE20005
Onteddu Chaitanya Reddy - CB.EN.U4AIE20045
Pillalamarri Akshaya - CB.EN.U4AIE20049

DECLARATION

We hereby declare that the project titled “Convex Optimisation and Applications" submitted to the Center for Computational Engineering and Networking is a record of the original work done under the guidance of Dr.Neethu Mohan .

Signature of the faculty

Date: 19.08.2021

Place: Ettimadai

ACKNOWLEDGMENT

The completion of this project has been made possible by the efforts of many. We would like to thank Dr. Sasangan Ramanathan, Dean- Engineering, and our faculty for the course 19MAT117 Dr.Neethu Mohan, for their unstinting support. We would also like to extend our gratitude to our friends and family who have aided us throughout the project.

ABSTRACT

Optimization is a key analysis technique used to make quantitative decisions about real-world problems. In addition, with the development of new technologies, there are more and more complex and multi-dimensional optimization models.Convex optimization Convex optimization for several reasons seems to have several tools to solve problems. This project shall be a pedagogical first step towards convex optimisation and solving real world problems using convex optimization.

PROJECT OUTCOMES

We Aim to do this project with an intent of learning convex optimization and more importantly to get a taste of formulating optimization problems . The project shall be a stepping stone and give us motivations to explore further on this vast topic of Convex Optimization. Primary objectives of this project are as follows.

- Solving convex optimisation
- Getting acquainted with some solvers for convex optimisation
- Introductory Pedagogical applications of convex Optimisation for heuristic purposes.These include the following.

    1. Markowitz Portfolio Optimization
    2. Zero sum game
    3. Shortest Route Problem
    4. Network Flow Problem
    5. Water Filling Algorithm in Wireless Communications
    6. Transhipment Problem in Decision Making
    7. The Assignment Problem

INTRODUCTION

A convex optimization problem is a problem consisting of minimizing a convex function over a convex set .More explicitly ,a convex problem is of the form

$$\min f(x)$$
$$\text{s.t. } x \in C$$

Where C is the convex set and f is a convex function over convex set(C).This is implicit in a sense. We often regard the more explicit formula of the convex problem as a convex optimization problem in functional form, that is, a convex problem of the form

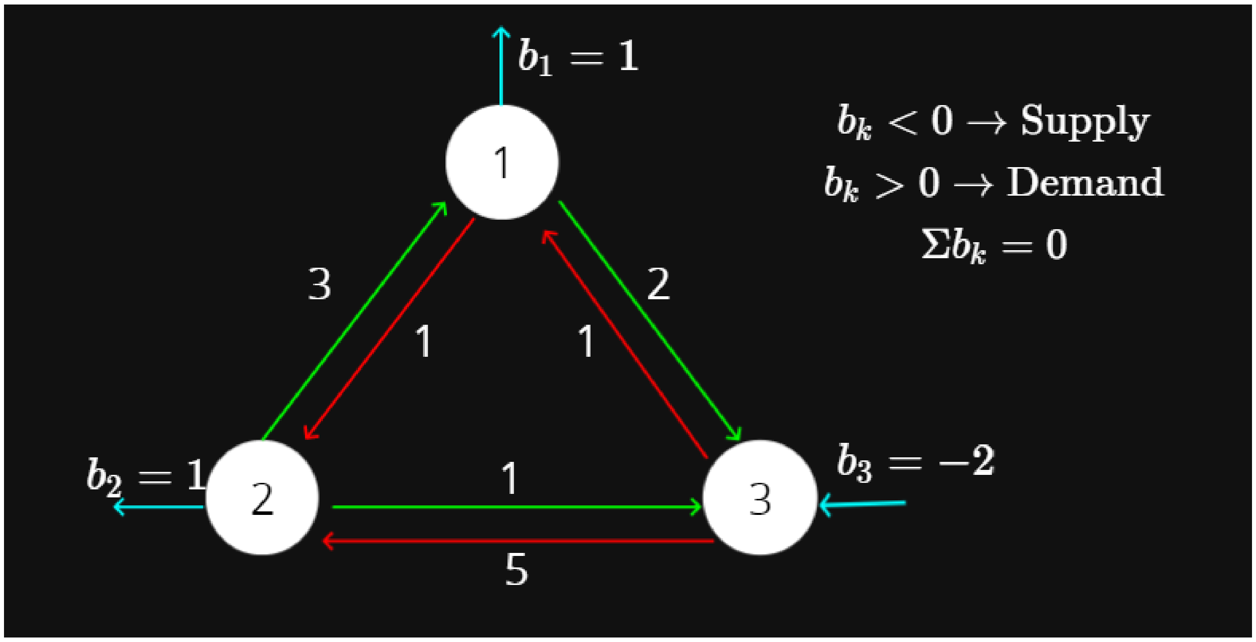
$$\begin{aligned} &\min f(x) \\ \text{S.t. } \quad & g_i(x) \le 0,\ i = 1,2,\ldots,m, \\ & h_j(x) = 0,\ j = 1,2,\ldots,p, \end{aligned}$$

which means that C is the intersection of the convex set as the plane set of the convex function, they must be the convex set, and also convex.

Network Flow problem

Problem:

We have a network flow in which we supply some units to a node and those units should reach its demand at each node such that the path taken to transfer those nodes has the least possible path cost.



Normal Technique

Starting from the node where the units are supplied, we need to find all the possible paths to reach the nodes in demand. After finding all the possible paths, we need to calculate the total cost of the path followed in each case. Now, we should pick the path having the minimum path cost to get the best path possible. But we might have a lot of possible combinations of paths to reach our destination and finding the cost of all the paths might not be effective i.e not an optimal way of solving the problem. So, we use convex optimization techniques to find the best way in the least time possible.

Optimization technique

Let c_{ij} is the cost function representing the cost for moving from i to j

$$c_{\mathrm{ij}} = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 0 & 1 \\ 1 & 5 & 0 \end{bmatrix}$$

Constraints

Number of units entering the node must be equal to number of units leaving the node

This can be represented mathematically as

$$\sum x_{ij} + b_i = \sum x_{ji}$$

x_{ij} represents the node leaving i and x_{ji} represents the node entering i

In linear program formulation it can be formulated in the form of $Ax = -b$

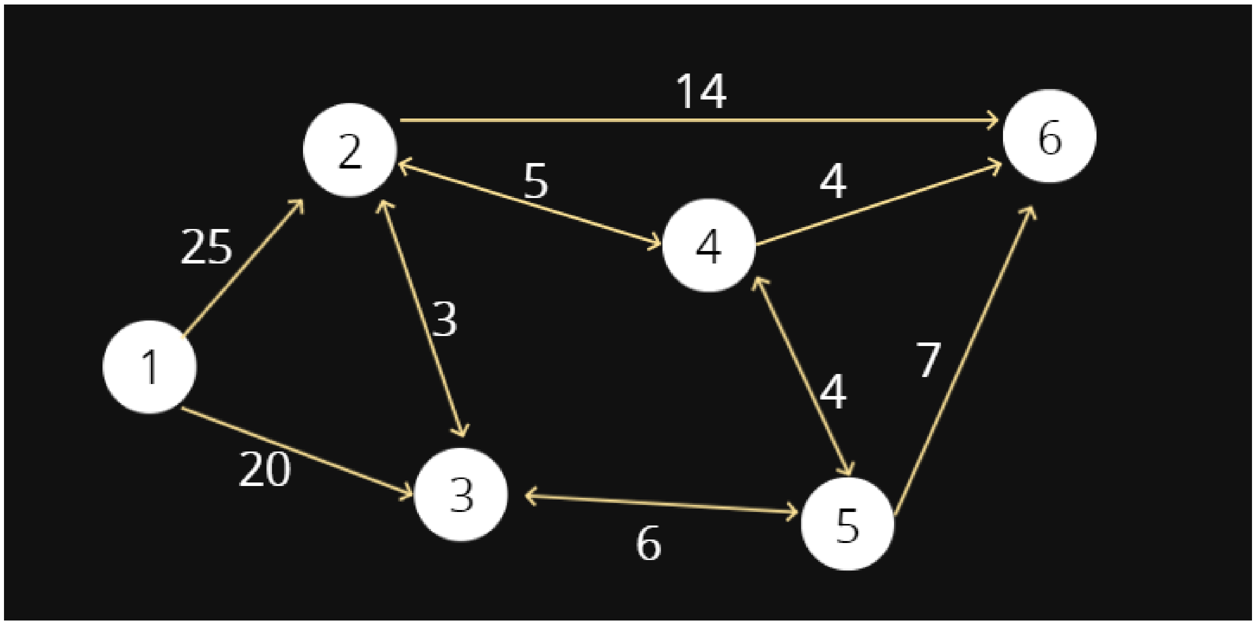
The overall problem can be represented as

$$\begin{aligned} &minimize\ c^T x \\ &w.r.t\ constraints\ Ax = -b \end{aligned}$$

Shortest Route problem

Problem:

We have 6 cities each named from number 1 to 6 consecutively.Distances between each city is given. We have a city where the problem starts (for example, the salesman starts there) which we call the source city. This salesman needs to pass through the cities to reach the destination. We need to find the optimal path (i.e path having minimum path cost) such that the salesman reaches the destination with minimum cost.



Normal Technique

Starting from the source city (i.e city 1 in our problem), we need to find all the possible paths to reach the destination. After finding all the possible paths, we need to calculate the total cost of the path followed in each case. Now, we should pick the path having the minimum path cost to get the best path possible. But we might have a lot of possible combinations of paths to reach our destination and finding the cost of all the paths might not be effective i.e not an optimal way of solving the problem. So, we use convex optimization techniques to find the best way in the least time possible.

Optimizing the problem using the convex optimization technique

Firstly, we denote the distance between any two cities i and j as $d_{ij}$and number of units travelled from city i to j as x_{ij} . Now, we consider all the constraints faced in the problem.

Constraints

1. Source city constraints

In the problem stated above, if we consider the source city, it is connected with 2 different cities. However, we can't choose both the paths simultaneously to get the optimal solution. So, we have a condition that,the sum of the number of units travelled from source city to connected cities should be equal to 1 since the salesman cannot take both the paths in an optimal solution.

$\Sigma x_{ij} = 1$, i is the source city, j is all other connected cities.

$x_{12} + x_{13} = 1$- Source city constraint

2. Destination city constraints

This is again similar to the source city constraints case, we can reach the destination city from any of the cities that it is connected with. So, again the sum of the number of units travelled from connected cities to destination city is 1.

$\Sigma x_{ij} = 1$, i is the source city, j is all other connected cities.

$x_{26} + x_{46} + x_{56} = 1$- Destination city constraint

3. Transhipment constraints

Source nodes contain the units which are to be sent and the destination nodes receive the nodes sent from source node. But the transshipment nodes do not contain any units. They just pass the units from its previous node to the next node. So, it conserves the flow, more formally, total number of incoming units is equal to the total number of outgoing units.

$\Sigma x_{ik} = \Sigma x_{ki}$, i is the transshipment node and k is the connected nodes.

Formulation

Now, our main objective is to optimize the total cost of the path.Although, number units that we are transferring is one, in general, we consider the cost of the path between 2 cities as the number units travelling in the particular path multiplies with the distance between the cities. So, we can formulate is as follows

Minimize $\Sigma d_{ij} x_{ij}$

i.e

Min $\Sigma\, d^T x$

Subject to (All the constraints considered above).

In simple terms, this can be shown as a linear program formulation

Min $\Sigma\, d^T x$

S.T $Ax = B$

Water filling problem

Convex optimization can be used to solve the classic water filling problem. This problem is where a total amount of power P has to be assigned to n communication channels, with the objective of maximising the total communication rate. The communication rate of the ith channel is given by:

$log(\alpha_i + x_i)$

Where x_i represents the power allocated to channel i and α_i represents the floor above the baseline at which power can be added to the channel. Since −log(X) is convex, we can write the water-filling problem as a convex optimisation problem:

Minimise $\sum_{i=1}^{N} -\log(\alpha_i + x_i)$

Subject to $x_i \geq 0 \quad and \quad \sum_{i=1}^{N} x_i = P$

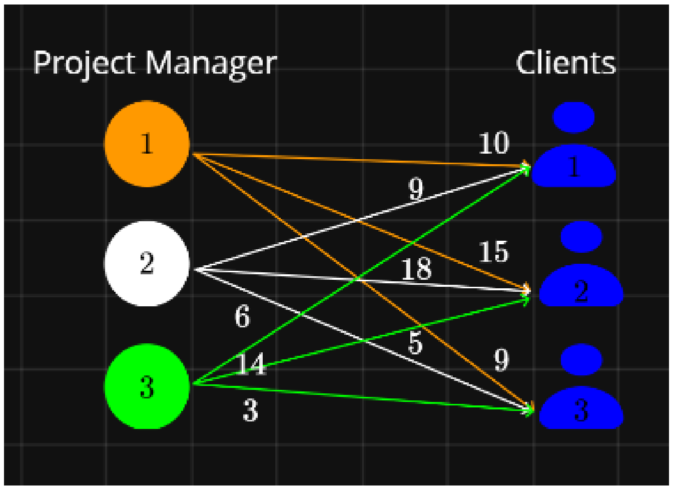
Assignment problem

Let us take there are 3 project leaders and each project leader have offered prices from 3 clients to complete a task .The company assigns project leader to each client because the company realises the time required to complete each study will depend on the experience and ability of project leader assigned and to minimise the total number of days to complete the project assigned .

Xij=1 if project leader i is assigned to client j

Xij=0 else

Where $i \epsilon \{1, 2, 3\} \& j \epsilon \{1, 2, 3\}$



	1	2	3
1	10	15	9
2	9	18	5
3	6	14	3

Cost Function to min

$$10x_{11} + 15x_{12} + 9x_{13} + 9x_{21} + 18x_{22} + 5x_{23} + 6x_{31} + 14x_{32} + 3x_{33}$$

$c^T x$

subject to:(sum of xij in column is 1 ,row is 1)

$$x_{11}+x_{12}+x_{13}=1$$

$$x_{12}+x_{22}+x_{32}=1$$

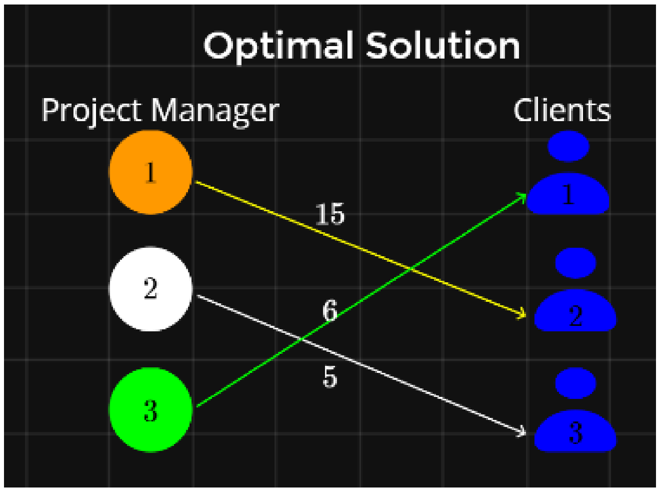
$$x_{13}+x_{23}+x_{13}=1$$

$$x_{11}+x_{21}+x_{31}=1$$

$$x_{21}+x_{22}+x_{23}=1$$

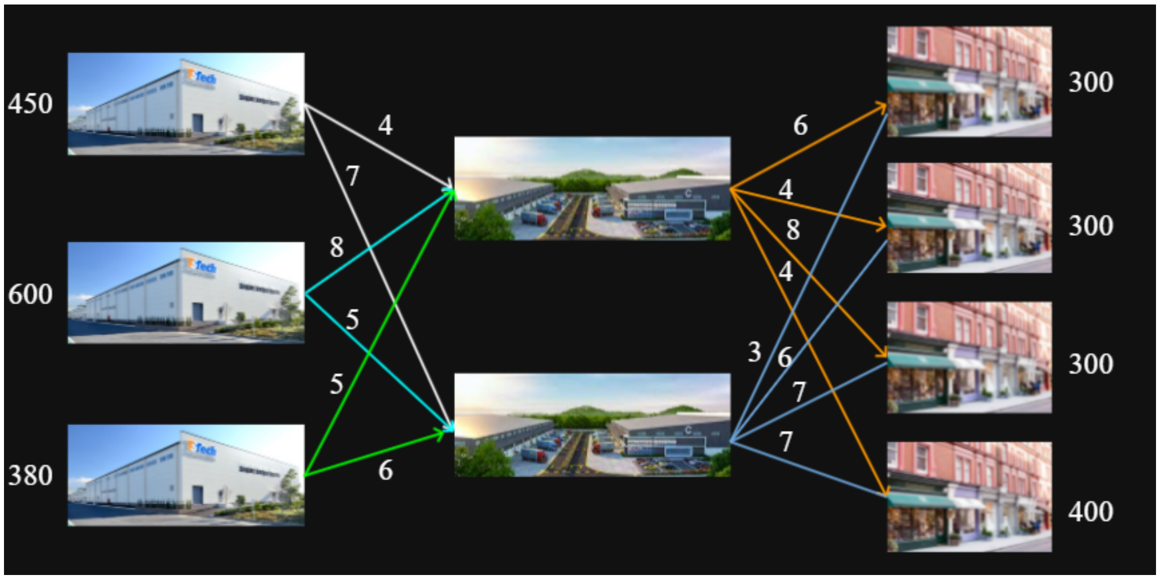
$$x_{31}+x_{32}+x_{33}=1$$

Of the form Ax=b



Transshipment problem in decision making

This is a type of transportation problem .Which includes source and destination with intermediate in between .Our goal is to ship from source to destination cheaply or as fast as possible. Each path comes with cost .The objective is to minimise total cost w.r.t X and Y
Everything which comes inside source should go to destination through intermediate
So,here if we see there are 3 plants and they are shipped to 2 warehouses and then to 4 dealers .



Formulation:

Plant	1	2	Capacity of plant
1	4	2	450
2	8	5	600
3	5	6	380

Warehouse	1	2	3	4
1	4	4	8	4
2	8	6	7	7
Demand	300	300	300	400

Constraints :

Each plant will produce some capacity of goods so the goods produced by plant 1 will be at most 450,the goods produced by plant 2 will be at most 600 and the goods produced by plant 3 will be at most 380.

$$x_{11} + x_{12} \leq 450$$

$$x_{21} + x_{22} \leq 600$$

$$x_{31} + x_{32} \leq 380$$

Each dealer demands a specific quantity of goods i.e dealer 1 demands 300 , 2 demand 300,3 demand 300 and 4 demands 400.

$$y_{11} + y_{21} = 300$$

$$y_{21} + y_{22} = 300$$

$$y_{13} + y_{23} = 300$$

$$y_{14} + y_{24} = 400$$

Minimize

Total Cost : $\sum_{i=1}^{3} \sum_{j=1}^{2} CijXij \ + \sum_{i=1}^{2} \sum_{j=1}^{4} \mathrm{CijYij}$

This includes cost from plants to warehouse and warehouse to dealer

Where C is cost extracted and X and Y are the number of units transported

All the goods which are given to warehouse should reach dealers

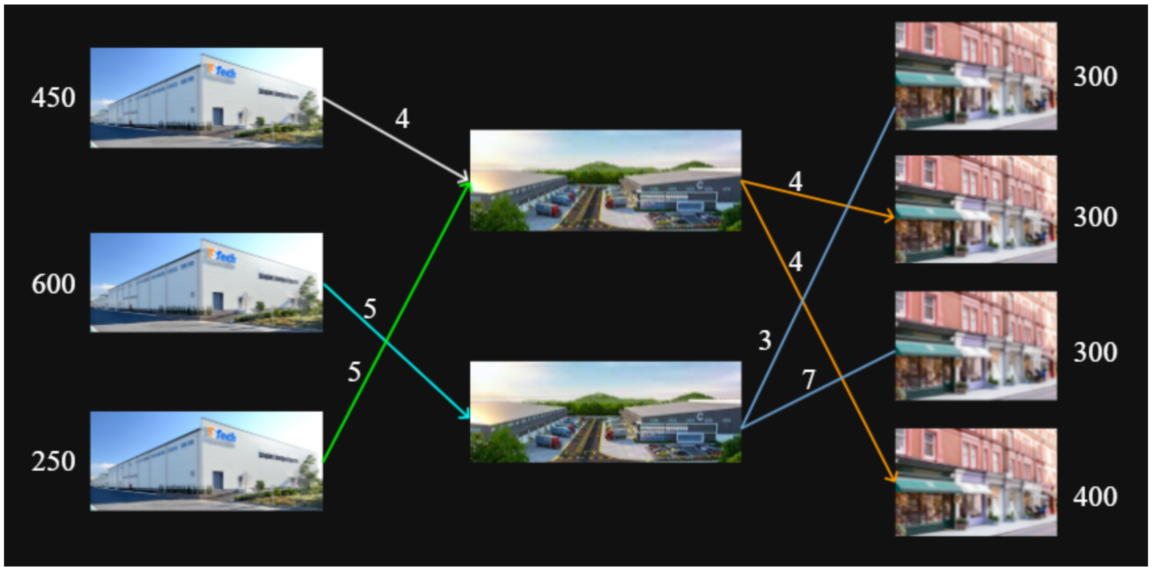
Everything leaving from warehouse to dealer - Everything coming inside warehouse=0

Equations for 1 and 2 warehouses

$$y_{11} + y_{12} + y_{13} + y_{14} - (x_{11} + x_{21} + x_{31}) = 0$$

$$y_{21} + y_{22} + y_{23} + y_{24} - \left(x_{12} + x_{22} + x_{32}\right) = 0$$

The optimal path followed is



Markowitz optimisation

Markowitz optimization problem focuses on giving the optimal split for minimal risk. Suppose there are i assets. The gain on asset i can be told as ,

$$P_i = \frac{S_{t+1}}{S_t}$$

Let x be the positions on asset , the n return on asset i can be given as ,

$$r_i = p_i x_i$$

Total returns can be ,

$$\sum_i r_i = \sum_i p_i x_i =$$

$$R = P^T X$$

$$P = [P_1 P_2 \cdots P_n]^T \quad X = [X_1 X_2 \cdots X_n]^T$$

Covariance on returns can be written as ,

$$E((r - \bar{r})^2) = x^T E((p - \bar{p})(p - \bar{p})^T)x = x^T \Sigma x$$

Therefore , we have our problem formulation as ,

$$\begin{aligned} &minimize \quad risk \\ &S.T\ minimum\ return \end{aligned}$$

Hence, Our optimisation problem is

$$\begin{aligned} &minimize \quad x^T \Sigma x \\ &S.T\ p^T x \geq r_{min},\ x^T 1 = 1 \end{aligned}$$

Zero Sum Game

Zero sum game is prevalent in economics and game theory , closely related to nash equilibria problems. Let us consider a two player zero sum game, which will eventually lead us to a min-max sort of problem.

Suppose we build a “Payoff” matrix. Where the entries correspond to the cost of player one. That is ,

$A = \{a_{ij} :\ cost\ player\ 1\ should\ pay,\ if\ player\ one\ chooses\ move\ i\ and\ player\ 2\ chooses\ move\ j\}$

Let us suppose , their choices are independent. Then “Expected” Payoff for player 1 will be $\Sigma u_i v_j P_{ij}$, where u_i is the probability player 1 chooses move i , and , v_i is the probability player 2 chooses move j.

$\vec{u} = [u_1 u_2 \cdots u_n]$, $\vec{v} = [v_1 v_2 \cdots v_m]$

$Expected\ Payoff\ from\ Player\ 1 = \sum u_i v_j P_{ij} = U^T PV$

This is an expectation. Player one only has control over his moves. That is he can only control the U.

To minimize the Payoff , he should minimize the maximum possible expectation. That is ,

$$minimize \quad max(U^T P)$$

This can be formulated as a linear programme

$$\begin{aligned} &minimize \qquad t \\ &U^T P \le t \\ &U^T 1 = 1 \end{aligned}$$

CVX and CVXpy

CVX was initially started by stephan boyd and his group as a matlab based modeling language for modelling and solving convex optimisation problems.CVX was initially built on the paradigm of Disciplined convex programming , Under this approach, convex functions and sets are built up from a small set of rules from convex analysis, starting from a base library of convex functions and sets. Constraints and objectives that are expressed using these rules are automatically transformed to a canonical form and solved.

CVXPy was a python equivalent for CVX , built by Stephan Boyd and his group. This builts on top of the same paradigm of DCP.

Solutions for above problems

Markowitz portfolio optimisation

```
import cvxpy as cp
import numpy as np
import pandas as pd



bitcoin = pd.read_csv("/content/coin_Bitcoin.csv")
bitcoin['Date'] = pd.to_datetime(bitcoin['Date'])

cardano = pd.read_csv("/content/coin_Cardano.csv")
cardano['Date'] = pd.to_datetime(cardano['Date'])



ethereum = pd.read_csv("/content/coin_Ethereum.csv")
ethereum['Date'] = pd.to_datetime(ethereum['Date'])

bitcoin = bitcoin[['Date' , 'Close' ]]

cardano = cardano[['Date' , 'Close' ]]
```

```
ethereum = ethereum[['Date' , 'Close' ]]

bitcoin = bitcoin.groupby(pd.PeriodIndex(bitcoin['Date'] , freq =
'M'))['Close'].mean().reset_index()

cardano = cardano.groupby(pd.PeriodIndex(cardano['Date'] , freq =
'M'))['Close'].mean().reset_index()

ethereum = ethereum.groupby(pd.PeriodIndex(ethereum['Date'] , freq =
'M'))['Close'].mean().reset_index()



df = pd.merge((pd.merge(bitcoin,cardano,how='inner',on='Date',suffixes=('_bitcoin' ,
'_cardano'))),ethereum)
df = df.rename(columns={'Close':'Close_ethereum'})

df.set_index('Date' , inplace=True)
df = df.pct_change().dropna()
df
df = df[['Close_ethereum' , 'Close_cardano']]
df.cov()*252
returns = df.to_numpy()
ret = returns.mean(axis=0)
covariance =  df.cov().to_numpy()
covariance*252
currencies = ['bitcoin'  , 'ethereum']

n= len(currencies)

x = cp.Variable((n,1))
```

```
# minimum reutrn
min_return =0.2




#return
r = ret@x

#risk
risk = cp.quad_form(x,covariance)

prob = cp.Problem(cp.Minimize(risk), [cp.sum(x)==1 ,  r >= min_return , x>=0])

prob.solve()
prob.value
x.value
```

Zero Sum Game

```
import numpy as np
import pandas as pd
import cvxpy as cp

P = np.asarray([[30,-10,20] , [-10,20,-20]])
f = np.asarray([1,0,0])
A = np.asarray([0,1,1])
```

```
b = np.ones((1,1))
G = np.hstack((np.ones((3,1)) , P.T))
h = np.zeros((3,1))
lb = np.zeros((3,1))
ub = 100*np.zeros((3,1))

x = cp.Variable((3,1))



constraints = [x>=0 , A@x==b, G@x >= h]
prob = cp.Problem(cp.Minimize(f.T@x) , constraints)
prob.solve()

x.value
```

Shortest Route Problem

```
import numpy as np
import cvxpy as cp

d = np.array([[0, 25, 20, 0, 0, 0], [25, 0, 3, 5, 0, 14], [20, 3, 0, 0, 6, 0], [0, 5, 0, 0, 4, 4], [0, 0, 6,
4, 0, 7], [0, 14, 0, 4, 7, 0]])
b = np.array([[1],[0],[0],[0],[0],[1]])

def route2Index(i,j):
  index = 6*(i-1) + j
  return index

def index2Route(Index):
  route = np.array([0,0])
  route[0] = np.ceil(Index/6)
  route[1] = Index - 6*(route[0]-1)
  return route

A = np.zeros((6,36))

#Source Constraints
A[1-1][route2Index(1,2)-1] = 1
A[1-1][route2Index(1,3)-1] = 1

#Transhipment Constraints Node2
A[2-1][route2Index(1,2)-1] = 1
A[2-1][route2Index(3,2)-1] = 1
A[2-1][route2Index(4,2)-1] = 1
A[2-1][route2Index(2,3)-1] = -1
A[2-1][route2Index(2,4)-1] = -1
A[2-1][route2Index(2,6)-1] = -1

#Transhipment Constraints Node3
A[3-1][route2Index(1,3)-1] = 1
A[3-1][route2Index(2,3)-1] = 1
A[3-1][route2Index(5,3)-1] = 1
A[3-1][route2Index(3,2)-1] = -1
A[3-1][route2Index(3,5)-1] = -1

#Transhipment Constraints Node4
A[4-1][route2Index(2,4)-1] = 1
A[4-1][route2Index(5,4)-1] = 1
A[4-1][route2Index(4,2)-1] = -1
```

```
A[4-1][route2Index(4,5)-1] = -1
A[4-1][route2Index(4,6)-1] = -1

#Transhipment Constraints Node5
A[5-1][route2Index(3,5)-1] = 1
A[5-1][route2Index(4,5)-1] = 1
A[5-1][route2Index(5,3)-1] = -1
A[5-1][route2Index(5,4)-1] = -1
A[5-1][route2Index(5,6)-1] = -1

#Destination Constraints
A[6-1][route2Index(2,6)-1] = 1
A[6-1][route2Index(4,6)-1] = 1
A[6-1][route2Index(5,6)-1] = 1

d = d.ravel()

x = cp.Variable((36,1), integer=True)
constraints = [A@x == b, x>=0]
sol = cp.Problem(cp.Minimize(d@x),constraints)
sol.solve()
index = np.where(x.value == 1)
index = index[0]

for i in range(0,4):
  print(index2Route(index[i]+1))
```

Network Flow Problem

```
import numpy as np
import cvxpy as cp

c = np.array([0.0, 1.0, 2.0, 3.0, 0.0, 1.0, 1.0, 5.0, 0.0])
b = np.array([1.0, 1.0, -2.0])
A = np.array([[0.0, 1.0, 1.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0],[0.0, -1.0, 0.0, 1.0, 0.0, 1.0, 0.0, -1.0,
0.0],[0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 1.0, 1.0, 0.0]])
x = cp.Variable((9), integer=True)
c = c.T
constraints = [A@x == -b, x>=0]
sol = cp.Problem(cp.Minimize(c@x),constraints)
sol.solve()

print(x.value)
```

Water filling Algorithm

```
def water_filling(n, a, sum_x):



    x = cp.Variable(shape=n)
    alpha = cp.Parameter(n, nonneg=True)
    alpha.value = a



    obj = cp.Maximize(cp.sum(cp.log(alpha + x)))


    constraints = [x >= 0, cp.sum(x) - sum_x == 0]


    prob = cp.Problem(obj, constraints)
    prob.solve()
    if(prob.status=='optimal'):
        return prob.status, prob.value, x.value
    else:
        return prob.status, np.nan, np.nan



alpha = np.array([7, 15, 19])
bucket_size = np.size(alpha)
stat, prob,x = water_filling(bucket_size,alpha,40)
print("Status: " + format(stat))
print("Optimal Communication Rate: "+ format(prob))
print("Transmittng power:"+format(x))
np.sum(x)
```

Transhipment Problem

```
import numpy as np
import cvxpy as cp
```

```
c = np.array([4, 7, 8, 5, 5, 6, 6, 4, 8, 4, 3, 6, 7, 7])

#Inequality Constraints
Ai = np.zeros((3,14))
Ai[0][0:2] = 1
Ai[1][2:4] = 1
Ai[2][4:6] = 1
Bi = np.array([[450],[600],[380]])

#Equality Constraints
Ae = np.zeros((6,14))
Ae[0][6] = 1
Ae[0][10] = 1
Ae[1][7] = 1
Ae[1][11] = 1
Ae[2][8] = 1
Ae[2][12] = 1
Ae[3][9] = 1
Ae[3][13] = 1
Ae[4][6:10] = 1
Ae[4][0] = -1
Ae[4][2] = -1
Ae[4][4] = -1
Ae[5][10:14] = 1
Ae[5][1] = -1
Ae[5][3] = -1
Ae[5][5] = -1
Be = np.array([[300],[300],[300],[400],[0],[0]])
```

```
x = cp.Variable((14,1),integer=True)
constraint = [(Ae @ x == Be), (Ai @ x <= Bi), x>=0]
prob = cp.Problem(cp.Minimize(c@x),constraint)
prob.solve()
solution = x.value
print(solution)
x_values = solution[0:6]
y_values = solution[6:]
Total_cost = c@solution
print(Total_cost)
```

The Assignment Problem

```
import numpy as np
import cvxpy as cp

c = np.array([10, 15, 9, 9, 18, 5, 6, 14, 3])
N = 3
Aeq = np.zeros((6,9))
beq = np.array([[1], [1], [1], [1], [1], [1]])

def couple2Index(i,j,N):
  index = N*(i-1) + j
  return index

def index2couple(ind,N):
  couple = np.array([0,0])
  couple[0] = np.ceil(ind/N)
  couple[1] = ind - N*(couple[0] - 1)
  return couple

for k in range(1,N+1):
  Aeq[k-1][couple2Index(1,k,N)-1] = 1;
  Aeq[k-1][couple2Index(2,k,N)-1] = 1;
  Aeq[k-1][couple2Index(3,k,N)-1] = 1;
  Aeq[k+3-1][couple2Index(k,1,N)-1] = 1;
  Aeq[k+3-1][couple2Index(k,2,N)-1] = 1;
  Aeq[k+3-1][couple2Index(k,3,N)-1] = 1;

x = cp.Variable((9,1),integer=True)
constraint = [Aeq @ x == beq, x >= 0]
prob = cp.Problem(cp.Minimize(c.T@x),constraints=constraint )
prob.solve()
x_values = x.value
idx = np.where(x_values == 1)
idx = idx[0]
idx = idx + 1
for i in range(3):
   k = index2couple(idx[i],N)
   print(k)
```

Conclusion

This project proved to be a great pedagogical process to get our hands wet with some pseudo real world applications of convex optimisations and provided motivation for us to explore more in this field.Several Problems were analysed and formulated as Convex optimization problems, and subsequently solved using CVXPy.