```
import cvxpy as cp
import numpy as np
from itertools import product
```

Data of the Problem

```
shipping_cost = np.array([[1,3,2],[3,2,2]]).reshape(2,3)
production_cost = np.array([11,10]).reshape(2,1)
given_demand = np.array([[100,150,120],[120,180,150],[150,200,180]]).reshape(3,3)
selling_price = np.array([[16,16]]).reshape(2,1)
produce = np.array([100,200]).reshape(2,1)
produce_limit = np.array([300,300]).reshape(2,1)
senario = 27
senario_prob = (1/senario)*(np.ones((senario,1)))
```

Senario generation of the demand

```
senario = given_demand.T
senario_comb = []
index = np.arange(0,3)
index_permute = product(index,repeat=3)
for i in index_permute:
    temp_array =[]
    for j in range(0,3):
        temp_array.append(float(senario[j,:][i[j]]))
    senario_comb.append(temp_array)
all_demand = np.array(senario_comb).reshape(27,3)
```

Question 1.3 Solve large scale problem combining all senarios into one problem

Large scale problem solution

```
produce = cp.Variable(shape=(2,1))
sell = cp.Variable(shape=(54,3))
salvage = cp.Variable(shape=(54,1))
produce_limit = np.array([300,300]).reshape(2,1)

ls_constraints_senario = []
for index,demand in enumerate(all_demand):
    i = index*2
    ls_constraints_senario.append(cp.sum(sell[i:i+2,:],axis=1,keepdims=True) +salvage[i:i+2,:] <= produce)
    ls_constraints_senario.append(cp.sum(sell[i:i+2,:],axis=0,keepdims=True) <= demand.reshape(1,-1))


ls_constraints = [produce>=0,produce<=produce_limit,sell>= np.zeros_like(sell),salvage>= np.zeros_like(salvage)]
ls_constraints.extend(ls_constraints_senario)


ls_objective = cp.Minimize((production_cost.T@produce)-(16/27)*cp.sum(sell,keepdims=True) + (1/27)*cp.trace((((np.repeat([shipping_cost],27
ls_problem = cp.Problem(ls_objective, ls_constraints)
ls_problem.solve()
```

```
-1609.9999997418313
```

-----------------Large scale problem results ----------------------------

```
print("----Large Scale formulation results----")
print("Optimal Production plan  ", produce.value)
print("Optimal Value of the Problem",ls_problem.value)
```

```
----Large Scale formulation results----
Optimal Production plan   [[120.00000002]
 [299.99999999]]
Optimal Value of the Problem -1609.9999997418313
```

Expected value of demand

```
expected_demand = np.mean(all_demand,axis=0)


produce = cp.Variable(shape=(2,1))
sell = cp.Variable(shape=(2,3))
salvage = cp.Variable(shape=(2,1))
produce_limit = np.array([300,300]).reshape(2,1)

ed_constraints_senario = []
ed_constraints_senario.append(cp.sum(sell,axis=1,keepdims=True) +salvage <= produce)
ed_constraints_senario.append(cp.sum(sell,axis=0,keepdims=True) <= expected_demand.reshape(1,-1))


ed_constraints = [produce>=0,produce<=produce_limit,sell>= np.zeros_like(sell),salvage>= np.zeros_like(salvage)]
ed_constraints.extend(ed_constraints_senario)

ed_objective = cp.Minimize((production_cost.T@produce)-(16)*cp.sum(sell,keepdims=True) + (1)*cp.trace(((((np.repeat([shipping_cost],1,axis=6
ed_problem = cp.Problem(ed_objective, ed_constraints)
ed_problem.solve()
```

------------------------ Deterministic Expected Demand ---------------------------------

```
-1773.3333279947847
```

```
print("----- Results using deterministic expected demand----")
print("\nOptimal Production plan using",produce.value)
print("\nOptimal Value of the Problem",ed_problem.value)
```

```
----- Results using deterministic expected demand----

Optimal Production plan using [[150.0000002 ]
 [299.99999937]]

Optimal Value of the Problem -1773.3333279947847
```

Q-1.4

Solving problem treating each demand individually

```
produce_over_scenario = []
objective_over_scenario = []
for demand in all_demand:
    produce = cp.Variable(shape=(2,1))
    sell = cp.Variable(shape=(2,3))
    salvage = cp.Variable(shape=(2,1))
    produce_limit = np.array([300,300]).reshape(2,1)

    constraints_senario = []
    constraints_senario.append(cp.sum(sell,axis=1,keepdims=True) +salvage <= produce)
    constraints_senario.append(cp.sum(sell,axis=0,keepdims=True) <= demand.reshape(1,-1))


    constraints = [produce>=0,produce<=produce_limit,sell>= np.zeros_like(sell),salvage>= np.zeros_like(salvage)]
    constraints.extend(constraints_senario)


    objective = cp.Minimize((production_cost.T@produce)-(16)*cp.sum(sell,keepdims=True) + (1)*cp.trace(((((np.repeat([shipping_cost],1,axis=
    problem = cp.Problem(objective, constraints)
    problem.solve()

    produce_over_scenario.append(produce.value)
    objective_over_scenario.append(problem.value)


print("Production plan considering each demand as determinstic:\n")
for index,demand in enumerate(all_demand):
    print(f"\n-----Senario {index}-----")
    print("\n Demand : ",demand)
    print("Optimal production",produce_over_scenario[index])
```

```
print("Optimal objective value",objective_over_scenario[index])
```

---------------------- Production plan for each individual demand -----------------------------

```
Production plan considering each demand as determinstic:


-----Senario 0-----

 Demand :  [100. 150. 120.]
Optimal production [100.00000039 269.99999935]
Optimal objective value [-1479.99999364]

-----Senario 1-----

 Demand :  [100. 150. 150.]
Optimal production [100.00000002 299.99999981]
Optimal objective value [-1599.99999832]

-----Senario 2-----

 Demand :  [100. 150. 180.]
Optimal production [130.00000026 299.99999889]
Optimal objective value [-1689.99999141]

-----Senario 3-----

 Demand :  [100. 180. 120.]
Optimal production [100.00000005 299.99999988]
Optimal objective value [-1599.99999891]

-----Senario 4-----

 Demand :  [100. 180. 150.]
Optimal production [130.00000023 299.99999923]
Optimal objective value [-1689.99999382]

-----Senario 5-----

 Demand :  [100. 180. 180.]
Optimal production [159.99999998 299.99999967]
Optimal objective value [-1779.9999969]

-----Senario 6-----

 Demand :  [100. 200. 120.]
Optimal production [120.00000025 299.99999945]
Optimal objective value [-1659.99999698]

-----Senario 7-----

 Demand :  [100. 200. 150.]
Optimal production [149.99999993 299.99999961]
Optimal objective value [-1749.99999596]

-----Senario 8-----

 Demand :  [100. 200. 180.]
Optimal production [180.00000003 299.99999979]
Optimal objective value [-1839.99999805]

-----Senario 9-----
```

Calculating value of perfect information

------------------------ value of perfect information ---------------------

```
produce_over_scenario = np.array(produce_over_scenario).reshape(27,2)
objective_over_scenario = np.array(objective_over_scenario).reshape(27,1)
value_of_perfect_info = ls_problem.value - np.mean(objective_over_scenario)

print("Value of value of perfect information ",value_of_perfect_info)
```

```
Value of value of perfect information  159.9999967386409
```

Q 1.5

Solve the scenario formulation with nonanticipativity constraints.

```
produce = cp.Variable(shape=(2,27))
sell = cp.Variable(shape=(54,3))
salvage = cp.Variable(shape=(54,1))
produce_limit = (np.array([300,300]).reshape(2,1))
produce_limit_ext = np.repeat([produce_limit],27,axis=0).reshape(2,27)

na_constraints = []

# Nonanticipativity constraints
for index,demand in enumerate(all_demand):
    na_constraints.append(cp.reshape(produce[:,index],shape=(2,1))==(1/27)*cp.sum(produce,axis=1,keepdims=True))


for index,demand in enumerate(all_demand):
    i = index*2
    na_constraints.append(cp.sum(sell[i:i+2,:],axis=1,keepdims=True) +salvage[i:i+2,:] <= cp.reshape(produce[:,index],shape=(2,1)))
    na_constraints.append(cp.sum(sell[i:i+2,:],axis=0,keepdims=True) <= demand.reshape(1,-1))


constraints = [produce>=0,produce<=produce_limit,sell>= np.zeros_like(sell),salvage>= np.zeros_like(salvage)]
na_constraints.extend(constraints)


na_objective = cp.Minimize((1/27)*cp.sum(production_cost.T@produce)-(16/27)*cp.sum(sell,keepdims=True) + (1/27)*cp.trace((((np.repeat([ship
na_problem = cp.Problem(na_objective, na_constraints)
na_problem.solve()
```

```
-1609.9999999266336
```

---------------------Results using non anticaptivity constraints---------------------------------------

```
print("----- Results using nonanticipativity ----")
print("\nOptimal Production plan using",produce.value)
print("\nOptimal Value of the Problem",na_problem.value)
```

```
----- Results using nonanticipativity ----

Optimal Production plan using [[120.00000001 120.00000001 120.00000001 120.00000001 120.00000001
  120.00000001 120.00000001 120.00000001 120.00000001 120.00000001
  120.00000001 120.00000001 120.00000001 120.00000001 120.00000001
  120.00000001 120.00000001 120.00000001 120.00000001 120.00000001
  120.00000001 120.00000001 120.00000001 120.00000001 120.00000001
  120.00000001 120.00000001]
 [299.99999999 299.99999999 299.99999999 299.99999999 299.99999999
  299.99999999 299.99999999 299.99999999 299.99999999 299.99999999
  299.99999999 299.99999999 299.99999999 299.99999999 299.99999999
  299.99999999 299.99999999 299.99999999 299.99999999 299.99999999
  299.99999999 299.99999999 299.99999999 299.99999999 299.99999999
  299.99999999 299.99999999]]

Optimal Value of the Problem -1609.9999999266336
```

-----------------------Dual Values----------------

```
for i in range(0,27):
    print(f"\nSecinario-{i}-------")
    print("Dual values for non anticipativiaty constraints")
    print(na_problem.constraints[i].dual_value)
```

```
Secinario-20-------
Dual values for non anticipativiaty constraints
[[0.14818783]
 [0.13000983]]

Secinario-21-------
Dual values for non anticipativiaty constraints
[[0.14818783]
 [0.10750179]]

Secinario-22-------
Dual values for non anticipativiaty constraints
[[0.14818783]
 [0.13000983]]

Secinario-23-------
Dual values for non anticipativiaty constraints
[[0.14818783]
 [0.13000983]]

Secinario-24-------
Dual values for non anticipativiaty constraints
[[0.14818783]
 [0.13000984]]

Secinario-25-------
Dual values for non anticipativiaty constraints
[[0.14818783]
 [0.13000984]]

Secinario-26-------
Dual values for non anticipativiaty constraints
[[0.14818783]
 [0.13000984]]
```

Q-1.6

Solve the problem by the cutting plane method in the basic version (Benders decomposition)

```
def solve_second_stage_problem(produce,demand,shipping_cost):

    sell = cp.Variable(shape=(2,3))
    salvage = cp.Variable(shape=(2,1))

    objective = cp.Minimize(-16*cp.sum(sell,keepdims=True) + cp.trace((((shipping_cost@(sell.T))))))
    constraints = [cp.sum(sell,axis=1,keepdims=True) +salvage <= produce,cp.sum(sell,axis=0,keepdims=True) <= demand.reshape(1,-1), sell>= r

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

    return problem


def solve_master_problem(produce,produce_limit,g_ks,alpha_ks):

    production_cost = np.array([11,10]).reshape(2,1)
    produce  = cp.Variable(shape=(2,1),name="produce")
    v = cp.Variable(shape=(1,1),name="value")

    constraints = []
    for i in range(0,len(g_ks)):
        constraints.append(np.array(g_ks[i]).T@produce+np.array(alpha_ks[i])<=v)

    constraints.extend([produce>=0,v>=-100000,produce<=produce_limit])

    objective = cp.Minimize(production_cost.T@produce + v)

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

    return problem
```

```
produce = np.array([100,200]).reshape(2,1)  # Initial Guess
g_ks = []
alpha_ks = []

objctive_values  = [np.nan]
epsilon = 10**(-4)
iter = 0

while True:

    # Solve Second stage problem for each demand and store its duals and objective values
    duals = []
    objs= []
    for demand in all_demand:

        second_stage_sol = solve_second_stage_problem(produce,demand,shipping_cost)

        temp_dual = second_stage_sol.constraints[0].dual_value   # Take the duals of 1st contraint
        temp_obj = second_stage_sol.value                        # Take the objective value of second stage problem

        # Store duals and objective values for each senario
        duals.append(temp_dual)
        objs.append(temp_obj)

    # Reshaping the values
    duals = np.array(duals).reshape(-1,2)
    objs = np.array(objs).reshape(-1,1)

    g_ks_temp = (-senario_prob.T@duals).T
    alpha_ks_temp = senario_prob.T@objs - g_ks_temp.T@produce

    g_ks.append(g_ks_temp)
    alpha_ks.append(alpha_ks_temp)

    first_stage_sol = solve_master_problem(produce,produce_limit,g_ks,alpha_ks)
    obj_value = first_stage_sol.value
    new_produce = first_stage_sol.var_dict["produce"].value
    new_limit = first_stage_sol.var_dict["value"].value

    if np.abs(obj_value - objctive_values[-1])<= epsilon:
        print("Terminating condition satisfied !")
        break
    else:
        pass

    objctive_values.append(obj_value)
    produce,limit = new_produce,new_limit # swap the values
    iter = iter+1

    print(f"\n----------Iteration no.  {iter}--------------------")
    print(f"\nproduction is {produce}")
    print(f"\nobjctive value is {first_stage_sol.value}\n")
```

-------------------- Results using Bender algorithm --------------------------------

```
----------Iteration no.  1--------------------

production is [[300.00000004]
 [299.99999992]]

objctive value is -2351.895943550797


----------Iteration no.  2--------------------

production is [[149.00805118]
 [299.9999992 ]]

objctive value is -1784.2447749212179


----------Iteration no.  3--------------------

production is [[112.70470909]
 [300.00000003]]

objctive value is -1647.7631011106232
```

```
----------Iteration no.  4--------------------

production is [[125.23365091]
 [299.99999495]]

objctive value is -1618.5289057328064


----------Iteration no.  5--------------------

production is [[119.99999869]
 [299.99999991]]

objctive value is -1609.9999959895367

Terminating condition satisfied !
```

```
print("----- Results using Bender Decompostion ----")
print(f"\nproduction is {produce}")
print(f"\nobjctive value is {first_stage_sol.value}\n")
```

```
----- Results using Bender Decompostion ----

production is [[119.99999869]
 [299.99999991]]

objctive value is -1609.9999957406962
```

Q-1.7 (Solve the problem by the multicut method)

```
def solve_master_problem_ml(produce,g_ks,alpha_ks,senario_prob,production_cost,produce_limit):


    produce  = cp.Variable(shape=(2,1),name="produce")
    v = cp.Variable(shape=(len(senario_prob),1),name="value")

    constraints = []
    for i in range(0,len(g_ks)):
        for j in range(0,len(g_ks[i])):
            constraints.append(np.array([g_ks[i][j]])@produce+np.array([alpha_ks[i][j]])<=v[j])

    constraints.extend([produce>=0,v>=-100000,produce<=produce_limit])

    objective = cp.Minimize(production_cost.T@produce + senario_prob.T@v)

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

    return problem


g_ks = []
alpha_ks = []
senario = 27
objctive_values  = [np.nan]
epsilon = 10**(-4)
iter = 0
while True:

    # Solve Second stage problem for each demand and store its duals and objective values
    duals = []
    objs= []
    for demand in all_demand:

        second_stage_sol = solve_second_stage_problem(produce,demand,shipping_cost)

        temp_dual = second_stage_sol.constraints[0].dual_value   # Take the duals of 1st contraint
        temp_obj = second_stage_sol.value                        # Take the objective value of second stage problem

        # Store duals and objective values for each senario
        duals.append(temp_dual)
```

```
        objs.append(temp_obj)

     # Reshaping the values
    duals = np.array(duals).reshape(-1,2)
    objs = np.array(objs).reshape(-1,1)

    gks_batch = []
    alpha_ks_batch = []

    for i in range(0,senario):
        gks_batch.append(-duals[i])
        alpha_ks_batch.append(objs[i]+duals[i].T@produce)

    g_ks.append(gks_batch)
    alpha_ks.append(alpha_ks_batch)

    first_stage_sol = solve_master_problem_ml(produce,g_ks,alpha_ks,senario_prob,production_cost,produce_limit)
    obj_value = first_stage_sol.value
    new_produce = first_stage_sol.var_dict["produce"].value
    new_limit = first_stage_sol.var_dict["value"].value

    if np.abs(obj_value - objctive_values[-1])<= epsilon:
        print("Terminating condition satisfied !")
        break

    else:
        pass

    objctive_values.append(obj_value)
    produce,limit = new_produce,new_limit # swap the values
    iter = iter+1

    print(f"\n----------Iteration no.  {iter}--------------------")
    print(f"\nproduction is {produce}")
    print(f"\nobjctive value is {first_stage_sol.value}\n")
```

---------------- Results using multicut method ----------------------------

```
----------Iteration no.  1--------------------

production is [[299.99999989]
 [299.99999992]]

objctive value is -1727.7034315843757


----------Iteration no.  2--------------------

production is [[120.00000283]
 [299.99999952]]

objctive value is -1609.9999916085671

Terminating condition satisfied !
```

```
print("----- Results using Multicut Method----")
print(f"\nproduction is {produce}")
print(f"\nobjctive value is {first_stage_sol.value}\n")
```

```
----- Results using Multicut Method----

production is [[120.00000283]
 [299.99999952]]

objctive value is -1609.999988313576
```