# Operational Research SA1

Espiritu, Joseph Raphael, Dacanay, Jordan, Empeno, Jhon Kheil Cymon

2025-03-24

# Case 1

**Summative Assessment 1**

This is a group project for 2-3 members. Show all computations and codes or software outputs used to analyze each case.

**Case 1:**

HiDec produces two models of electronic gadgets that use resistors, capacitors, and chips. The following table summarizes the data of the situation:

| | Unit resource requirements | | |
|---|---|---|---|
| Resource | *Model 1 (units)* | *Model 2 (units)* | Maximum availability (units) |
| Resistor | 2 | 3 | 1200 |
| Capacitor | 2 | 1 | 1000 |
| Chips | 0 | 4 | 800 |
| Unit price ($) | 3 | 4 | |

1. Formulate the problem as a linear program, and find the optimum solution.
2. Determine the status of each resource. (used/unused proportions or scarcity/abundance)
3. In terms of the optimal revenue, determine the dual prices for the resistors, capacitors, and chips.
4. Determine the feasibility ranges for the dual prices obtained in (3).
5. If the available number of resistors is increased to 1300 units, find the new optimum solution.
6. If the available number of chips is reduced to 350 units, will you be able to determine the new optimum solution directly from the given information? Explain.
7. If the availability of capacitors is limited by the feasibility range computed in (4), determine the corresponding range of the optimal revenue and the corresponding ranges for the numbers of units to be produced of Models 1 and 2.
8. A new contractor is offering to sell HiDec additional resistors at 40 cents each, but only if HiDec would purchase at least 500 units. Should HiDec accept the offer?

Maximize: $Z = 3x_1 + 4x_2$

Subject to:

$$2x_1 + 3x_2 \leq 1200 \text{Resistors}$$
$$2x_1 + 1x_2 \leq 1000 \text{Capacitor}$$
$$4x_2 \leq 1000 \text{Chips}$$
$$x_1, x_2 \geq 0 \text{Non-Negative}$$

```r
if (!require(lpSolve)) install.packages("lpSolve")
library(lpSolve)

# coefficients of the objective function
objective <- c(3, 4)  # Model 1 = $3, Model 2 = $4

# constraints matrix (resource usage per unit)
constraints <- matrix(c(2, 3,  # Resistor usage
                        2, 1,  # Capacitor usage
                        0, 4), # Chips usage
                      nrow = 3, byrow = TRUE)

# right-hand side of the constraints
rhs <- c(1200, 1000, 800)

# direction of the constraints (<=)
directions <- c("<=", "<=", "<=")

# Solve the lp
solution <- lp(direction = "max",
               objective.in = objective,
               const.mat = constraints,
               const.dir = directions,
               const.rhs = rhs,
               compute.sens = TRUE)
# Output the results
cat("Optimal Solution:\n")
cat(solution$solution, "Model 1 and Model 2")

shadow_prices <- solution$duals[1:length(rhs)]  # Extract duals for constraints
names(shadow_prices) <- c("Resistors", "Capacitors", "Chips")
print(shadow_prices)

cat("\nMaximum Profit: $", solution$objval, "\n")

# Analyze resource usage
cat("\nResource Usage Status:\n")
used_resources <- constraints %*% solution$solution # Matrix Multiply and Find used resources
for (i in 1:length(rhs)) {
  unused <- rhs[i] - used_resources[i]
  if (abs(unused) < 1e-12) unused <- 0 # remove Floating point Error small numbers
  cat(paste0("Resource ", i, ": Used = ", used_resources[i], ", Unused = ", unused, "\n"))
  lower_bound <- solution$duals.from[i]
  upper_bound <- solution$duals.to[i]

  if (solution$duals[i] == 0) {
```

```
    cat(paste0("Feasability Range ", i, ": Unbounded above (Shadow Price = 0)\n"))
  } else {
    cat(paste0("Feasability Range ", i, ": Lower Bound = ", lower_bound,
               ", Upper Bound = ", upper_bound, "\n"))
  }
}
```

```
## Loading required package: lpSolve
```

```
## Optimal Solution:
```

```
## 450 100 Model 1 and Model 2
```

```
##  Resistors Capacitors      Chips
##       1.25       0.25       0.00
```

```
##
## Maximum Profit: $ 1750
```

```
##
## Resource Usage Status:
```

```
## Resource 1: Used = 1200, Unused = 0
## Feasability Range 1: Lower Bound = 1000, Upper Bound = 1400
## Resource 2: Used = 1000, Unused = 0
## Feasability Range 2: Lower Bound = 800, Upper Bound = 1200
## Resource 3: Used = 400, Unused = 400
## Feasability Range 3: Unbounded above (Shadow Price = 0)
```

**NUMBER 1**

**Optimal Solution:**

- The optimal solution is 450 units of Model 1 and 100 units of Model 2
- With this solution, the optimal revenue is $1750.

**NUMBER 2**

| Resource | Used | Unused | Dual Price | Status |
|---|---|---|---|---|
| Resistors | 1200 | 0 | 1.25 | Scarce (Binding) |
| Capacitors | 1000 | 0 | 0.25 | Scarce (Binding) |
| Chips | 400 | 400 | 0 | Abundant (Non-Binding) |

- **For Resistors and Capacitors**
  Both resources are fully utilized, as the zero unused units indicate. Additionally, since both resources have a Dual Price > 0, then the **status of resistors and capacitors is scarce**.
- **For Chips**
  This resource has 400 unused units. Also, it has a dual price of 0. Therefore, the **status of chips is abundant.**

**NUMBER 3**

| Resource | Dual Price (Shadow Price) |
|---|---|
| Resistors | 1.25 |
| Capacitors | 0.25 |
| Chips | 0 |

- **Resistors:** Increasing available resistors by one unit increases profit by $1.25.
- **Capacitors:** Increasing available capacitors by one unit increases profit by $0.25.
- **Chips**: Increasing chips has no effect on profit (not fully utilized).

**NUMBER 4**

| Resource | Feasibility Range (Valid for Dual Price) |
|---|---|
| Resistors | 1000 to 1400 |
| Capacitors | 800 to 1200 |
| Chips | Unbounded |

- **Resistors:** The dual price remains $1.25 per unit for available resistor units between 1000 and 1400
- **Capacitors:** The dual price remains $0.25/unit for available capacitor units between 800 and 1200
- **Chips:** There is no upper bound because it is not a constraint.

**Question 5** Increasing the Resistor availability to 1300, based on the dual prices and feasibility analysis from Q3 and Q4, results in a profit increase with minimal impact on the overall resource capacity. The change primarily leads to higher usage of Chips while shifting production priorities—producing more of Model 2 and slightly less of Model 1 to optimize profits. This adjustment effectively leverages the additional Resistors for greater profitability.

```
## Optimal Solution:

## 425 150 Model 1 and Model 2

##  Resistors Capacitors      Chips
##       1.25       0.25       0.00

##
## Maximum Profit: $ 1875

##
## Resource Usage Status:

## Resource 1: Used = 1300, Unused = 0
## Feasability Range 1: Lower Bound = 1000, Upper Bound = 1400
## Resource 2: Used = 1000, Unused = 0
## Feasability Range 2: Lower Bound = 900, Upper Bound = 1300
## Resource 3: Used = 600, Unused = 200
## Feasability Range 3: Unbounded above (Shadow Price = 0)
```

**Question 6** Reducing the Chips capacity to 350, we can intuitively predict that it will become the limiting factor for profit, as it previously had 400 units of unused capacity based on earlier analysis. This reduction and adjust the optimal solution to be highly one-sided and leads to unbalanced productions of both models and lower profits. To better understand the impact, here is an updated output that highlights the effect

```
## Optimal Solution:

## 456.25 87.5 Model 1 and Model 2

##  Resistors Capacitors      Chips
##      0.000      1.500      0.625

##
## Maximum Profit: $ 1718.75

##
## Resource Usage Status:

## Resource 1: Used = 1175, Unused = 25
## Feasability Range 1: Unbounded above (Shadow Price = 0)
## Resource 2: Used = 1000, Unused = 0
## Feasability Range 2: Lower Bound = 87.5000000000001, Upper Bound = 1025
## Resource 3: Used = 350, Unused = 0
## Feasability Range 3: Lower Bound = 0, Upper Bound = 400
```

**Question 7** Determine the corresponding range of the optimal revenue and the corresponding ranges for the number of units to be produced in Models 1 and 2

```
## Current Coefficients are 3 and 4 x1,x2 respectively

## Sensitivity Range for Objective Coefficients:

## Variable 1: Lower = 2.66666666666667, Upper = 8
## Variable 2: Lower = 1.5, Upper = 4.5
```

The current optimal production quantities are **450 units for Model 1** ($x_1$) and **100 units for Model 2** ($x_2$), as long as the **objective function coefficients** remain within the specified sensitivity ranges:

$$c_1 \text{ (profit per unit for Model 1)}: \text{ Lower Bound } = 2.67, \text{ Upper Bound } = 8$$
$$c_2 \text{ (profit per unit for Model 2)}: \text{ Lower Bound } = 1.5, \text{ Upper Bound } = 4.5$$

- Sensitivity ranges represent the limits where the **current basis (set of active constraints)** remains valid.
- As long as the coefficients ($c_1$ and $c_2$) stay within these ranges, the solution $x_1 = 450$ and $x_2 = 100$ will continue to maximize profit.
- However, if the coefficients exceed these bounds, the optimal production plan may no longer maximize profit, and a new solution will need to be calculated.

**Question 8** if HiDec should accept the offer, we need to evaluate the profitability of purchasing the additional resistors at 40 cents each under the given conditions.

-

```
## 450 100 Current Optimal Model 1 and Model 2

## 1750 Current Profit Value

## 1.25 0.25 0 0 0 Current Shadow Prices

## 2.666667 1.5 Current Lower Bound Coefficients

## 8 4.5 Current Upper Bound Coefficients

## 375 250 New Optimal Model 1 and Model 2

## 2125 New Profit Value

## 0 1.5 0.625 0 0 New Shadow Prices

## 0 1.5 New Lower Bound Coefficients

## 8 1e+30 New Upper Bound Coefficients
```

The shadow price is 1.25 for resistors, meaning each additional resistor within the feasibility range (1000 $<$ 1200 $<$ 1500) would increase the profit by 1.25 per unit. However, with the contractor's offer to sell resistors at 40 cents per unit, the net profit per resistor decreases to 1.25 - 0.40 = 0.85 per unit. If HiDec purchases an additional 500 resistors, the new total would be 1200 + 500 = 1700, which exceeds the feasibility range. Therefore, the shadow price of 1.25 would no longer apply for the units beyond 1500. Total profit increase for these 300 resistors: 300×0.85=255 Cost of these 200 resistors: 200×0.40=80 Net change in profit: 255−80=175

**HiDec should accept the offer event if it changes the shadow prices and feasability range because the new profit overall will still be higher at 2125 than 1750 but more model 2 will be made instead of the focus at model 1 production in the previous profit**

**Case 2:**

Three refineries with daily capacities of 6, 5, and 8 million gallons, respectively, supply three distribution areas with daily demands of 4, 8, and 7 million gallons, respectively. Gasoline is transported to the three distribution areas through a network of pipelines. The transportation cost is 10 cents per 1000 gallons per pipeline mile. The table gives the mileage between the refineries and the distribution areas. Refinery 1 is not connected to distribution area 3.

TABLE 5.26 Mileage Chart for Problem 5-8

| | Distribution area | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| Refinery 1 | 180 | 180 | — |
| Refinery 2 | 300 | 800 | 900 |
| Refinery 3 | 220 | 200 | 120 |

1. Construct the associated transportation model.
2. Determine the optimum shipping schedule in the network.
3. Suppose that the capacity of refinery 3 is 6 million gallons only and that distribution area 1 must receive all its demand. Additionally, any shortages at areas 2 and 3 will incur a penalty of 5 cents per gallon. Determine the optimum shipping schedule.

## Transportation Model

The transportation problem can be formulated as follows:

**Objective Function:**

$$\text{Minimize } Z = 180x_{11} + 180x_{12} + 300x_{21} + 800x_{22} + 900x_{23} + 220x_{31} + 200x_{32} + 120x_{33}$$

**Subject to:**

1. **Supply Constraints** (capacities of refineries):

$$\begin{aligned} x_{11} + x_{12} &\leq 6 \quad \text{(Refinery 1 capacity)} \\ x_{21} + x_{22} + x_{23} &\leq 5 \quad \text{(Refinery 2 capacity)} \\ x_{31} + x_{32} + x_{33} &\leq 8 \quad \text{(Refinery 3 capacity)} \end{aligned}$$

2. **Demand Constraints** (demands of distribution areas):

$$\begin{aligned} y_{11} + y_{21} + y_{31} &= 4 \quad \text{(Distribution Area 1 demand)} \\ y_{12} + y_{22} + y_{32} &= 8 \quad \text{(Distribution Area 2 demand)} \\ y_{23} + y_{33} &= 7 \quad \text{(Distribution Area 3 demand)} \end{aligned}$$

3. **Non-negativity Constraint**:

$$x_{ij} \geq 0 \quad \forall i, j$$

4. **Additional Condition**:

- $x_{13} = 0$ (Refinery 1 is not connected to Distribution Area 3).

**Transportation Table:**

| Refinery → Distribution Area | 1 | 2 | 3 | Supply (million gallons) |
|---|---|---|---|---|
| **Refinery 1** | 180 | 180 | – | 6 |
| **Refinery 2** | 300 | 800 | 900 | 5 |
| **Refinery 3** | 220 | 200 | 120 | 8 |
| **Demand (million gallons)** | 4 | 8 | 7 | – |

```r
# Install lpSolve package if not already installed
if (!require(lpSolve)) install.packages("lpSolve")

# Load lpSolve library
library(lpSolve)

# Define the cost matrix with large finite values for infeasible routes
cost_matrix <- matrix(c(
  180, 180, 9999999999999,     # Refinery 1 to Distribution Areas
  300, 800, 900,     # Refinery 2 to Distribution Areas
  220, 200, 120      # Refinery 3 to Distribution Areas
), nrow = 3, byrow = TRUE)

# Define supply (capacities of the refineries in millions of gallons)
supply <- c(6, 5, 8)

# Define demand (requirements of distribution areas in millions of gallons)
demand <- c(4, 8, 7)

# Solve the transportation problem
solution <- lp.transport(cost_matrix, "min", row.signs = rep("<=", 3), row.rhs = supply,
                         col.signs = rep(">=", 3), col.rhs = demand)

# Output the optimal shipping schedule
cat("Optimal Shipping Schedule (in millions of gallons):\n")
shipping_schedule <- matrix(solution$solution, nrow = 3, byrow = TRUE)
rownames(shipping_schedule) <- c("Refinery 1", "Refinery 2", "Refinery 3")
colnames(shipping_schedule) <- c("Distribution Area 1", "Distribution Area 2", "Distribution Area 3")
print(shipping_schedule)

# Output the total transportation cost
cat("Minimum Transportation Cost: $", solution$objval / 1000, "\n")  # Divide by 1000 to adjust for per
```

```
## Optimal Shipping Schedule (in millions of gallons):
```

```
##            Distribution Area 1 Distribution Area 2 Distribution Area 3
## Refinery 1                   0                   4                   0
## Refinery 2                   6                   1                   1
## Refinery 3                   0                   0                   7
```

```
## Minimum Transportation Cost: $ 4.12
```

**Interpretation**

The optimal shipping schedule ensures the allocation satisfies the supply capacities of the refineries (6, 5, and 8 million gallons) and the demand requirements of the distribution areas (4, 8, and 7 million gallons) without any surplus or shortage. The minimum transportation cost for this optimal allocation is $4.12 million, based on mileage and transportation rates.

**Another way to solve this is using Python and doing the algorithm of Vogels Approximation**

```
[180, 180, inf]
[300, 800, 900]
[220, 200, 120]
----------
[180, 180, inf]
[300, 800, inf]
[220, 200, inf]
----------
[inf, 180, inf]
[inf, 800, inf]
[inf, 200, inf]
----------
[inf, inf, inf]
[inf, 800, inf]
[inf, 200, inf]
----------
[inf, inf, inf]
[inf, inf, inf]
[inf, 200, inf]
----------
[inf, inf, inf]
[inf, inf, inf]
[inf, inf, inf]
----------
[inf, inf, inf]
[inf, inf, inf]
[inf, inf, inf]
----------
[7, 120] [2, 2]
[4, 300] [1, 0]
[6, 180] [0, 1]
[1, 800] [1, 1]
[1, 200] [2, 1]
(4120, [[2, 2], [1, 0], [0, 1], [1, 1], [2, 1]])
```

with thise console output you can see the same values outputted by the R and Python code

**Question 2** Changes in supply and demand with cost reduction for missing gallons

**Objective Function:**

$$\text{Minimize } Z = 180x_{11} + 180x_{12} + 300x_{21} + 800x_{22} + 900x_{23} + 220x_{31} + 200x_{32} + 120x_{33} + 5s_2 + 5s_3$$

Where $s_2$ and $s_3$ represent the shortages (in millions of gallons) for Distribution Areas 2 and 3, respectively.

---

**Subject to:**

1. **Supply Constraints** (capacities of refineries):

$$\begin{aligned} x_{11} + x_{12} &\leq 6 \quad \text{(Refinery 1 capacity)} \\ x_{21} + x_{22} + x_{23} &\leq 5 \quad \text{(Refinery 2 capacity)} \\ x_{31} + x_{32} + x_{33} &\leq 6 \quad \text{(Adjusted Refinery 3 capacity)} \end{aligned}$$

2. **Demand Constraints** (requirements of distribution areas):

$$\begin{aligned} x_{11} + x_{21} + x_{31} &= 4 \quad \text{(Distribution Area 1 demand must be fully met)} \\ x_{12} + x_{22} + x_{32} + s_2 &= 8 \quad \text{(Distribution Area 2 demand)} \\ x_{13} + x_{23} + x_{33} + s_3 &= 7 \quad \text{(Distribution Area 3 demand)} \end{aligned}$$

3. **Shortages**:

$$s_2, s_3 \geq 0$$

4. **Non-negativity Constraint**:

$$x_{ij} \geq 0 \quad \forall i, j$$

5. **Additional Condition**:

- $x_{13} = 0$ (Refinery 1 is not connected to Distribution Area 3).

---

**Transportation Table:**

| Refinery -> Distribution Area | 1 | 2 | 3 | Supply (million gallons) |
|---|---|---|---|---|
| **Refinery 1** | 180 | 180 | – | 6 |
| **Refinery 2** | 300 | 800 | 900 | 5 |
| **Refinery 3** | 220 | 200 | 120 | 6 |
| **Demand (million gallons)** | 4 | 8 | 7 | – |

```r
# Install lpSolve package if not already installed
if (!require(lpSolve)) install.packages("lpSolve")

# Load lpSolve library
library(lpSolve)

# Define the cost matrix including penalty costs for shortages
# Large number (9999999999999) is used for the prohibited route (Refinery 1 to Distribution Area 3)
cost_matrix <- matrix(c(
  180, 180, 9999999999999,  # Refinery 1 to Distribution Areas
  300, 800, 900,            # Refinery 2 to Distribution Areas
  220, 200, 120,            # Refinery 3 to Distribution Areas
  0, 5, 5                   # Shortages for Distribution Areas 2 and 3
), nrow = 4, byrow = TRUE)

# Define supply (capacities of the refineries in millions of gallons, including shortages)
supply <- c(6, 5, 6, 2)  # Adjusted Refinery 3 capacity to 6; last row for shortage supply
```

```r
# Define demand (requirements of distribution areas in millions of gallons)
demand <- c(4, 8, 7)

# Solve the transportation problem
solution <- lp.transport(cost_matrix, "min", row.signs = rep("<=", 4), row.rhs = supply,
                         col.signs = rep(">=", 3), col.rhs = demand)

# Output the optimal shipping schedule
cat("Optimal Shipping Schedule (in millions of gallons):\n")
shipping_schedule <- matrix(solution$solution, nrow = 4, byrow = TRUE)
rownames(shipping_schedule) <- c("Refinery 1", "Refinery 2", "Refinery 3", "Shortages")
colnames(shipping_schedule) <- c("Distribution Area 1", "Distribution Area 2", "Distribution Area 3")
print(shipping_schedule)

# Output the total transportation cost
cat("Minimum Transportation Cost: $", solution$objval / 1000, "\n")  # Divide by 1000 to adjust for per
```

```
## Optimal Shipping Schedule (in millions of gallons):
```

```
##            Distribution Area 1 Distribution Area 2 Distribution Area 3
## Refinery 1                   0                   4                   0
## Refinery 2                   0                   6                   1
## Refinery 3                   0                   1                   0
## Shortages                    0                   6                   1
```

```
## Minimum Transportation Cost: $ 3.81
```

**Interpretation** The 15 cents (0.15 million) added to the cost represents the penalty incurred due to the shortages in Distribution Areas 2 and 3. These penalties are a direct result of the refineries being unable to meet the full demand for gasoline. Therefore, the total cost of 3.81 million includes both the actual shipping cost and the penalty for shortages. So, in essence:

Actual transportation cost = $3.66 million.

Penalty for shortages = $0.15 million.

Final total cost = $3.81 million.

- The supply cost reduction comes from the decision to allow shortages in Distribution Areas 2 and 3 instead of fully satisfying their demands through additional transportation.
- Consider increasing the capacities of Refinery 2 and Refinery 3 to reduce shortages in the future, especially for high-demand areas like Distribution Areas 2 and 3.
- Reassess the transportation network to connect Refinery 1 with Distribution Area 3 if feasible, as it could further reduce costs.

also attach is the python code to execute in IPYNB, Jupyter Notebook

```python
def vogelApproximation(costValueMatrix, supply, demand):
    if len(costValueMatrix) != len(demand):
        y = supply.copy()
        x = demand.copy()
    else:
        y = demand.copy()
        x = supply.copy()
    penalty_y = []
    penalty_x = []
```

```
    values = []
    keyPairValues = []
    x_index = 0
    y_index = 0
    Done = False
    for row in costValueMatrix:
        penalty_y.append(sorted(row)[1] - sorted(row)[0])
    for i in range(len(costValueMatrix[0])):
        column = [row[i] for row in costValueMatrix]
        penalty_x.append(sorted(column)[1] - sorted(column)[0])
    while not Done:
        if max(penalty_y) > max(penalty_x):
            y_index = penalty_y.index(max(penalty_y))
            minCost = min(costValueMatrix[y_index])
            x_index = costValueMatrix[y_index].index(minCost)
            if x[x_index] > y[y_index]:  # choose y
                values.append([y[y_index], minCost])
                keyPairValues.append([y_index, x_index])
                x[x_index] -= y[y_index]
                for i in range(len(costValueMatrix[0])):
                    costValueMatrix[y_index][i] = float('inf')
                y[y_index] = 0
            elif x[x_index] < y[y_index]:  # choose x
                values.append([x[x_index], minCost])
                keyPairValues.append([y_index, x_index])
                y[y_index] -= x[x_index]
                for i in range(len(costValueMatrix)):
                    costValueMatrix[i][x_index] = float('inf')
                x[x_index] = 0
            elif x[x_index] == y[y_index]:  # choose x
                values.append([x[x_index], minCost])
                keyPairValues.append([y_index, x_index])
                y[y_index] -= x[x_index]
                for i in range(len(costValueMatrix)):
                    costValueMatrix[i][x_index] = float('inf')
                x[x_index] = 0
            penalty_y[y_index] = float('-inf')
            for row in costValueMatrix:
                print(row)
            print(f'-'*10)
        elif max(penalty_y) < max(penalty_x):
            x_index = penalty_x.index(max(penalty_x))
            minCost = float('inf')
            for row in costValueMatrix:
                if min(row) < minCost:
                    minCost = min(row)
                    y_index = costValueMatrix.index(row)
            if x[x_index] > y[y_index]:  # choose y
                values.append([y[y_index], minCost])
                keyPairValues.append([y_index, x_index])
                x[x_index] -= y[y_index]
                for i in range(len(costValueMatrix[0])):
                    costValueMatrix[y_index][i] = float('inf')
```

```
            y[y_index] = 0
        elif x[x_index] < y[y_index]:  # choose x
            values.append([x[x_index], minCost])
            keyPairValues.append([y_index, x_index])
            y[y_index] -= x[x_index]
            for i in range(len(costValueMatrix)):
                costValueMatrix[i][x_index] = float('inf')
            x[x_index] = 0
        elif x[x_index] == y[y_index]:  # choose x
            values.append([x[x_index], minCost])
            keyPairValues.append([y_index, x_index])
            y[y_index] -= x[x_index]
            for i in range(len(costValueMatrix)):
                costValueMatrix[i][x_index] = float('inf')
            x[x_index] = 0
        penalty_x[x_index] = float('-inf')
        for row in costValueMatrix:
            print(row)
        print(f'-'*10)
    elif max(penalty_y) == max(penalty_x):
        y_index = penalty_y.index(max(penalty_y))
        minCost = min(costValueMatrix[y_index])
        x_index = costValueMatrix[y_index].index(minCost)
        if x[x_index] > y[y_index]:  # choose y
            values.append([y[y_index], minCost])
            keyPairValues.append([y_index, x_index])
            x[x_index] -= y[y_index]
            for i in range(len(costValueMatrix[0])):
                costValueMatrix[y_index][i] = float('inf')
            y[y_index] = 0
        elif x[x_index] < y[y_index]:  # choose x
            values.append([x[x_index], minCost])
            keyPairValues.append([y_index, x_index])
            y[y_index] -= x[x_index]
            for i in range(len(costValueMatrix)):
                costValueMatrix[i][x_index] = float('inf')
            x[x_index] = 0
        elif x[x_index] == y[y_index]:  # choose x
            values.append([x[x_index], minCost])
            keyPairValues.append([y_index, x_index])
            y[y_index] -= x[x_index]
            for i in range(len(costValueMatrix)):
                costValueMatrix[i][x_index] = float('inf')
            x[x_index] = 0
        penalty_y[y_index] = float('-inf')
        for row in costValueMatrix:
            print(row)
        print(f'-'*10)
    if sum(x) <= 0 and sum(y) <= 0:
        Done = True
    for row in costValueMatrix:
        if y[costValueMatrix.index(row)] > 0:
            penalty_y[costValueMatrix.index(row)] = sorted(row)[1] - sorted(row)[0]
```

```
        for i in range(len(costValueMatrix[0])):
            if x[i] > 0:
                column = [row[i] for row in costValueMatrix]
                penalty_x[i] = sorted(column)[1] - sorted(column)[0]
    for row in costValueMatrix:
                print(row)
    print(f'-'*10)
    results = [value[0] * value[1] for value in values]
    for values, coordinates in zip(values, keyPairValues):
        print(values, coordinates)
    return sum(results), keyPairValues

refinery = [[180, 180, float('inf')], [300, 800, 900], [220, 200, 120]]
refineryCopy = [[180, 180, float('inf')], [300, 800, 900], [220, 200, 120]]
demand, supply = [4, 8, 7], [6, 5, 8]
for row in refinery:
    print(row)
print(f'-'10)
vogels = vogelApproximation(refinery, demand, supply)

print(vogels)
print(f'-'10)


for vogel in vogels[1]:
    refineryCopy[vogel[0]][vogel[1]] = 0
for row in refineryCopy:
    print(row)
print(f'-'*10)
sums = vogels[0]
for row in refineryCopy:
    for elements in row:
        if elements != 0 and elements != float('inf') and row.index(elements) != 0:
            sums -= 0.05*elements
print(sums)
```