MATRIX INPUT

```
import numpy as np
rows=int(input("ENTER NUMBER OF ROWS: "))
cols=int(input("ENTER NUMBER OF COLS: "))
matrix=[]
print("Enter the elements of matrix by row for first matrix:")
for i in range(rows):
    row = []
    for j in range(cols):
        element = int(input(f"Element at ({i + 1}, {j + 1}): "))
        row.append(element)
    matrix.append(row)
a=np.array(matrix)
print(a)
```

MATRIX ADDITION

```
import numpy as np
rows1 = int(input("Enter no. of rows1: "))
cols1 = int(input("Enter no. of columns1: "))
print("Enter the elements of the first matrix row-wise (space-separated):")
matrix1 = []
for i in range(rows1):
    row = list(map(int, input(f"Row {i + 1}: ").split()))
    while len(row) != cols1:
        print("Invalid number of elements. Please enter again:")
        row = list(map(int, input(f"Row {i + 1}: ").split()))
    matrix1.append(row)
matrix1 = np.array(matrix1)

rows2 = int(input("Enter no. of rows2: "))
cols2 = int(input("Enter no. of columns2: "))
if rows1 != rows2 or cols1 != cols2:
    print("Matrix addition is not possible!! Dimensions must match.")
else:
    print("Enter the elements of the second matrix row-wise (space-
separated):")
    matrix2 = []
    for i in range(rows2):
        row = list(map(int, input(f"Row {i + 1}: ").split()))
        while len(row) != cols2:
            print("Invalid number of elements. Please enter again:")
            row = list(map(int, input(f"Row {i + 1}: ").split()))
        matrix2.append(row)
    matrix2 = np.array(matrix2)
    result = matrix1 + matrix2
    print("The result of matrix addition is:")
    print(result)
```

MATRIX TRANSPOSE

```
import numpy as np
rows=int(input("ENTER NUMBER OF ROWS: "))
cols=int(input("ENTER NUMBER OF COLS: "))
matrix=[]
print("Enter the elements of matrix by row for first matrix:")
for i in range(rows):
    row = []
    for j in range(cols):
        element = int(input(f"Element at ({i + 1}, {j + 1}): "))
        row.append(element)
    matrix.append(row)
a=np.array(matrix)
print("The transpose of the matrix is:\n", a.T)
```

MATRIX MULTIPICATION

```
import numpy as np
rows1 = int(input("Enter no. of rows1: "))
cols1 = int(input("Enter no. of columns1: "))
matrix1 = []
print("Enter the elements of matrix by row for first matrix:")
for i in range(rows1):
    row = []
    for j in range(cols1):
        element = int(input(f"Element at ({i + 1}, {j + 1}): "))
        row.append(element)
    matrix1.append(row)
rows2 = int(input("Enter no. of rows2: "))
cols2 = int(input("Enter no. of columns2: "))
if cols1 != rows2:
    print("Matrix multiplication is not possible!!")
else:
    matrix2 = []
    print("Enter the elements of matrix by row for second matrix:")
    for i in range(rows2):
        row = []
        for j in range(cols2):
            element = int(input(f"Element at ({i + 1}, {j + 1}): "))
            row.append(element)
        matrix2.append(row)
    result = []
    a=np.array(matrix1)
    b=np.array(matrix2)
    result = np.dot(a, b)
    print("Result of matrix multiplication:\n", result)
```

max in array

```
arr = [12,11,5,13,6]
max_num = max(arr)
print("Maximum number in the array:", max_num)
```

#SORT

BUBBLE SORT

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
arr = [38,27,43,3,9,82,10]
print("Original array:", arr)
sorted_arr = bubble_sort(arr)
print("Sorted array:", sorted_arr)
```

```
Original array: [38, 27, 43, 3, 9, 82, 10]
Sorted array: [3, 9, 10, 27, 38, 43, 82]
```

SELECTION SORT

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]

arr = [64, 25, 12, 22, 11]
print("Original array:", arr)
selection_sort(arr)
print("Sorted array:", arr)
```

MARGE SORT

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)

        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1
        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1

        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1
arr = [38, 27, 43, 3, 9, 82, 10]
print("Original array:", arr)
merge_sort(arr)
print("Sorted array:", arr)
```

LPP solution using graphical methods

i) Using Python code, examine the following linear programming problem by Graphical method Minimize $z = -x + y$ subject to: $3x + y \geq 6$, $3x + y \leq 3$ $x, y \geq 0$.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 5, 200)
y1 = 6 - 3*x
y2 = 3 - 3*x
plt.figure(figsize=(6,6))
plt.plot(x, y1, label=r'$3x + y = 6$')
plt.plot(x, y2, label=r'$3x + y = 3$')
plt.fill_between(x, 0, y2, where=(y2 >= 0), color='lightblue', alpha=0.5,
label='Below lower line')
plt.fill_between(x, y1, 5, where=(y1 <= 5), color='lightgreen', alpha=0.5,
label='Above upper line')
```

```
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Graphical Method')
plt.grid(True)
plt.show()
```

sol-i

```
from pulp import *
import matplotlib.pyplot as plt
import numpy as np
model = LpProblem("MinimumZ", LpMinimize)
x = LpVariable('x', lowBound=0)
y = LpVariable('y', lowBound=0)
model += -1*x+y
constraints = [
    (3, 1, 6, ">="),
    (3, 1, 3, "<=")
]
for i, (a, b, c, ctype) in enumerate(constraints):
    if ctype == "<=":
        model += a*x+b*y <= c
    elif ctype == ">=":
        model += a*x+b*y >= c
    else:
        model += a*x+b*y == c
status = model.solve()
print("\n--- Unified LPP Solver ---")
if LpStatus[model.status] == 'Optimal':
    print("Optimal Solution Found:")
    print(f"x = {x.varValue:.2f}")
    print(f"y = {y.varValue:.2f}")
    print(f"Z = {value(model.objective):.2f}")
else:
    print("No Optimal Solution:", LpStatus[model.status])
```

ii) Using Python code, examine the following linear programming problem by Graphical method Max $z = 60x + 50y$ subject to: $x + 2y \leq 40$, $3x + 2y \leq 60$ $x, y \geq 0$.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.linspace(0, 25, 400)
y1 = (40-x)/2
y2 = 30 - 1.5 * x
y1_clipped = np.maximum(y1, 0)
y2_clipped = np.maximum(y2, 0)
y_feasible = np.minimum(y1_clipped, y2_clipped)
plt.figure(figsize=(7,7))
plt.plot(x, y1, label=r'$x + 2y = 40$', color='blue')
plt.plot(x, y2, label=r'$3x + 2y = 60$', color='orange')
plt.fill_between(x, 0, y_feasible, where=(y_feasible > 0),
color='lightgreen', alpha=0.5)
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Graphical Solution of LPP')
plt.legend()
plt.grid(True)
plt.show()
```

solution -- (ii)

```
import pulp
import matplotlib.pyplot as plt
import numpy as np
model = LpProblem("Maximize Z", LpMaximize)
x = LpVariable('x', lowBound=0)
y = LpVariable('y', lowBound=0)
model += 60*x+50*y
constraints = [
    (1, 2, 40, "<="),
    (3, 2, 60, "<=")
]
for i, (a, b, c, ctype) in enumerate(constraints):
    if ctype == "<=":
        model += a*x+b*y <= c
    elif ctype == ">=":
        model += a*x+b*y >= c
    else:
        model += a*x+b*y == c
status = model.solve()
print("\n--- Unified LPP Solver ---")
if LpStatus[model.status] == 'Optimal':
    print("Optimal Solution Found:")
    print(f"x = {x.varValue:.2f}")
    print(f"y = {y.varValue:.2f}")
    print(f"Z = {value(model.objective):.2f}")
else:
    print("No Optimal Solution:", LpStatus[model.status])
```

iii) Using Python code, examine the following linear programming problem by Graphical method Max $z = 5x + y$ subject to: $x + y \leq 20$, $2x + 3y \leq 30$ $x, y \geq 0$.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 25, 200)
y1 = 20 - x
y2 = (30 - 2*x) / 3
y1_pos = np.maximum(y1, 0)
y2_pos = np.maximum(y2, 0)
plt.figure(figsize=(7,7))
plt.plot(x, y1_pos, label=r'$x + y = 20$')
plt.plot(x, y2_pos, label=r'$2x + 3y = 30$')
y_region = np.maximum(y1, y2)
plt.fill_between(x, y_region, 25, where=(y_region <= 25), color='lightgreen',
alpha=0.5)
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.title('Graphical Solution of LPP')
plt.show()
```

solution--(iii)

```
from pulp import *
import matplotlib.pyplot as plt
import numpy as np
model = LpProblem("MaximizeZ", LpMaximize)
x = LpVariable('x', lowBound=0)
y = LpVariable('y', lowBound=0)
model += 5*x+6*y
constraints = [
    (1, 1, 20, ">="),
    (2, 3, 30, ">=")
]
for i, (a, b, c, ctype) in enumerate(constraints):
    if ctype == "<=":
        model += a*x+b*y <= c
    elif ctype == ">=":
        model += a*x+b*y >= c
    else:
        model += a*x+b*y == c
```

```python
status = model.solve()
print("\n--- Unified LPP Solver ---")
if LpStatus[model.status] == 'Optimal':
    print("Optimal Solution Found:")
    print(f"x = {x.varValue:.2f}")
    print(f"y = {y.varValue:.2f}")
    print(f"Z = {value(model.objective):.2f}")
else:
    print("No Optimal Solution:", LpStatus[model.status])
```

LPP SOLUTION

i) Using Python code, solve the following linear programming problem Minimize $Z* = 2\, x + y + 5z$ subject to: $x + y + z \leq 6$ $2x + y + 3z \leq 12$ $x + y + 3z \leq 12$ $x, y, z \geq 0$.

```python
from pulp import *
import matplotlib.pyplot as plt
import numpy as np
model = LpProblem("Minimize Z", LpMinimize)
x = LpVariable('x', lowBound=0)
y = LpVariable('y', lowBound=0)
z = LpVariable('z', lowBound=0)
model += 2 * x + 1 * y + 5*z
constraints = [
    (1, 1, 1,6, "<="),
    (2, 1, 3,12,"<="),
    (1, 1, 3,12, "<=")
]
for i, (a, b, d, c, ctype) in enumerate(constraints):
    if ctype == "<=":
        model += a*x+b*y+d*z<= c
    elif ctype == ">=":
        model += a*x+b*y+d*z >= c
    else:
        model += a*x+b*y+d*z==c
status = model.solve()
print("\n--- Unified LPP Solver ---")
if LpStatus[model.status] == 'Optimal':
    print("Optimal Solution Found:")
    print(f"x = {x.varValue:.2f}")
    print(f"y = {y.varValue:.2f}")
    print(f"z = {z.varValue:.2f}")
    print(f"Z = {value(model.objective):.2f}")
else:
    print("No Optimal Solution:", LpStatus[model.status])
```

ii) Using Python to solve the following Linear Programming Problem (L.P.P). Maximize $Z = 2\ x1 + 3x2 + 5x3$ subject to: $2\ x1 + 3x2 + 5x3 \leq 1\ 2\ x1 + 3x2 + 5x3 \geq 12\ x1, x2, x3 \geq 0$.

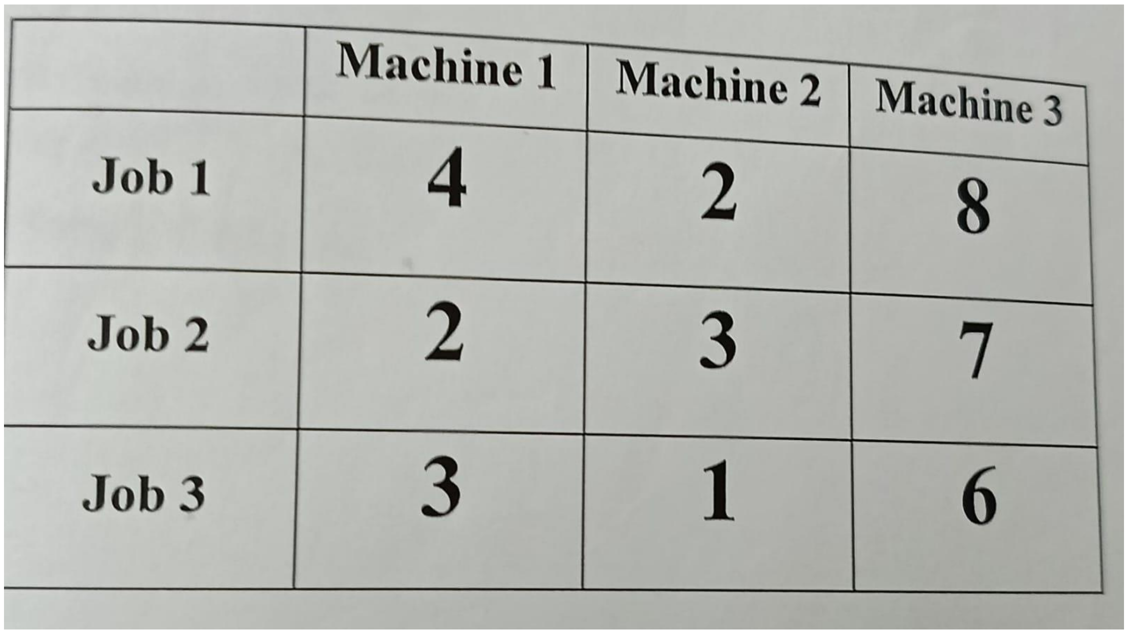
```
from pulp import *
import matplotlib.pyplot as plt
import numpy as np
model = LpProblem("Max Z", LpMaximize)
x = LpVariable('x', lowBound=0)
y = LpVariable('y', lowBound=0)
z = LpVariable('z', lowBound=0)
model += 2 * x + 3 * y + 5*z
constraints = [
    (2, 3, 5,1, "<="),
    (2, 3, 5,12,">=")

]
for i, (a, b, d, c, ctype) in enumerate(constraints):
    if ctype == "<=":
        model += a*x+b*y+d*z<= c
    elif ctype == ">=":
        model += a*x+b*y+d*z >= c
    else:
        model += a*x+b*y+d*z==c
status = model.solve()
print("\n--- Unified LPP Solver ---")
if LpStatus[model.status] == 'Optimal':
    print("Optimal Solution Found:")
    print(f"x = {x.varValue:.2f}")
    print(f"y = {y.varValue:.2f}")
    print(f"z = {z.varValue:.2f}")
    print(f"Z = {value(model.objective):.2f}")
else:
    print("No Optimal Solution:", LpStatus[model.status])
```

Using Python to solve the following Linear Programming Problem (L.P.P). Maximize $Z = 3\ x1 + 2x2 + 4x3$ subject to: $x1 + x2 + x3 \leq 6\ 2\ x1 + x2 + 3x3 \leq 12\ x1 + 3x2 + 2x3 \leq 6\ x1, x2, x3 \geq 0$.

```
from pulp import *
import matplotlib.pyplot as plt
import numpy as np
model = LpProblem("Max Z", LpMaximize)
x = LpVariable('x', lowBound=0)
```

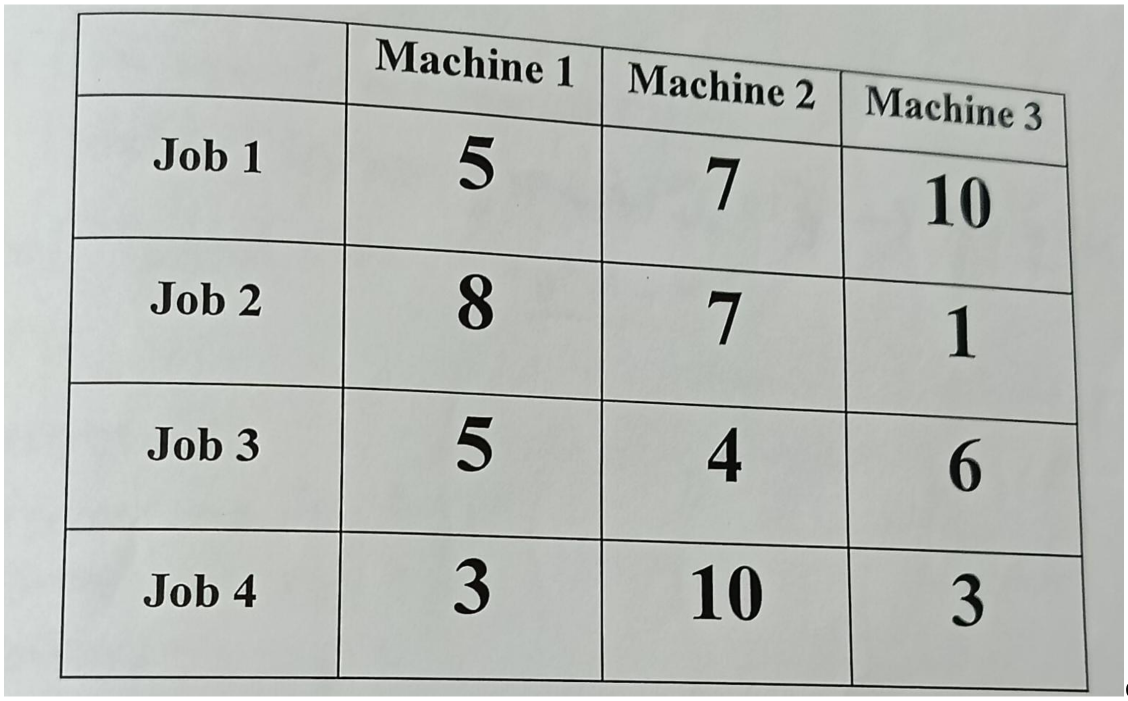
```python
y = LpVariable('y', lowBound=0)
z = LpVariable('z', lowBound=0)
model += 3* x + 2 * y + 4*z
constraints = [
    (1, 1, 1,6, "<="),
    (2, 1, 3,12,"<="),
    (1, 3, 2,6,"<=")

]
for i, (a, b, d, c, ctype) in enumerate(constraints):
    if ctype == "<=":
        model += a*x+b*y+d*z<= c
    elif ctype == ">=":
        model += a*x+b*y+d*z >= c
    else:
        model += a*x+b*y+d*z==c
status = model.solve()
print("\n--- Unified LPP Solver ---")
if LpStatus[model.status] == 'Optimal':
    print("Optimal Solution Found:")
    print(f"x = {x.varValue:.2f}")
    print(f"y = {y.varValue:.2f}")
    print(f"z = {z.varValue:.2f}")
    print(f"Z = {value(model.objective):.2f}")
else:
    print("No Optimal Solution:", LpStatus[model.status])
```

ASSIMENT



	Machine 1	Machine 2	Machine 3
Job 1	4	2	8
Job 2	2	3	7
Job 3	3	1	6

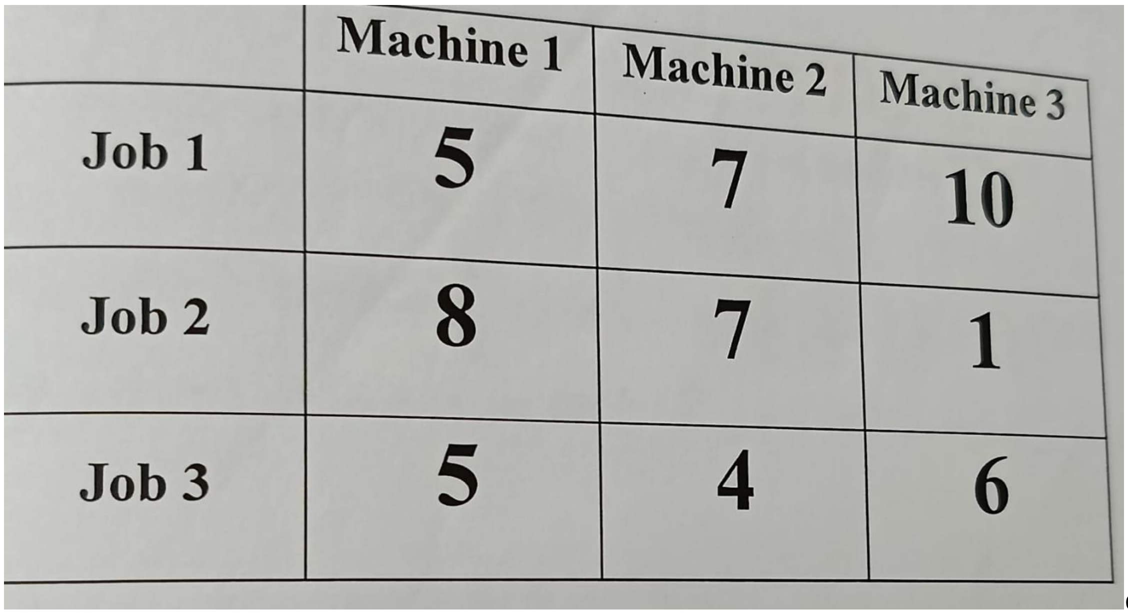
```
from scipy.optimize import linear_sum_assignment
import numpy as np
cost = np.array([[4, 2, 8], [2, 3, 7], [3, 1, 6]])
row_ind,col_ind=linear_sum_assignment(cost)
total_cost = cost[row_ind, col_ind].sum()
print("Assignments:")
for i in range(len(row_ind)):
  print (f"Worker {row_ind[i]+1} -> Job {col_ind[i]+1}")
print("Total Cost:",total_cost)
```



	Machine 1	Machine 2	Machine 3
Job 1	5	7	10
Job 2	8	7	1
Job 3	5	4	6
Job 4	3	10	3

q2)

```
from scipy.optimize import linear_sum_assignment
import numpy as np
cost = np.array([[5,7,10,0], [8,7,1,0], [5,4,6,0],[3,10,3,0]])
row_ind,col_ind=linear_sum_assignment(cost)
total_cost = cost[row_ind, col_ind].sum()
print("Assignments:")
for i in range(len(row_ind)):
  print (f"Worker {row_ind[i]+1} -> Job {col_ind[i]+1}")
print("Total Cost:",total_cost)
```

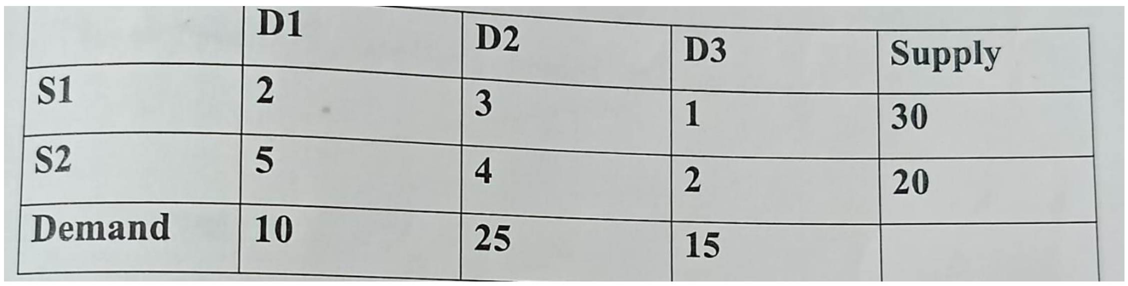


	Machine 1	Machine 2	Machine 3
Job 1	5	7	10
Job 2	8	7	1
Job 3	5	4	6

q3)

```python
import numpy as np
from scipy.optimize import linear_sum_assignment
max_cost_matrix = np.array([[5,7,10], [8,7,1], [5,4,6]])
max_value = np.max(max_cost_matrix)
subtracted_cost_matrix = max_value - max_cost_matrix
row_ind, col_ind = linear_sum_assignment(subtracted_cost_matrix)
total_max_cost = max_cost_matrix[row_ind, col_ind].sum()
print("\nAssignments for Maximization:")
for i in range(len(row_ind)):
    print(f"Worker {row_ind[i]+1} -> Job {col_ind[i]+1}")
print("\nTotal Maximum Cost:", total_max_cost)
```

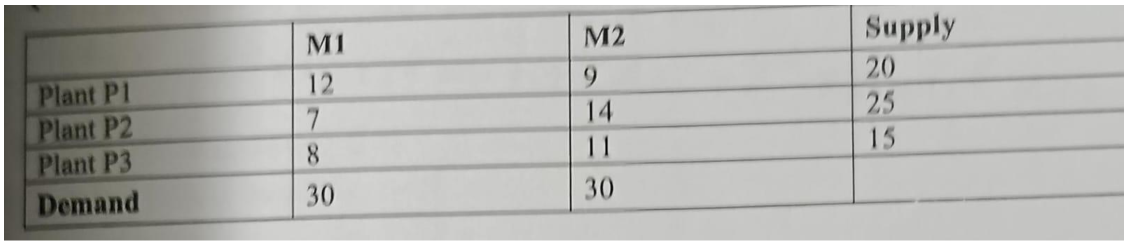
TRANSPORT



	D1	D2	D3	Supply
S1	2	3	1	30
S2	5	4	2	20
Demand	10	25	15	

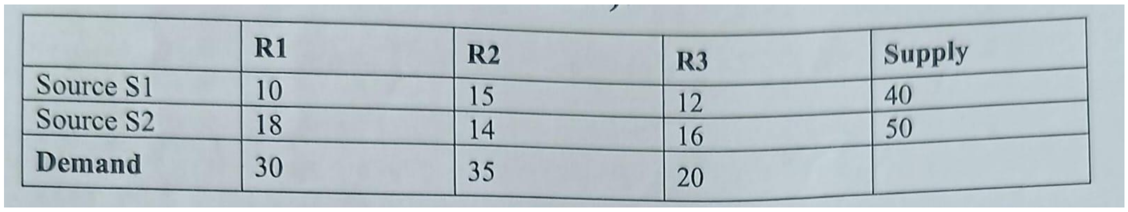
```python
# pip install pulp
from pulp import *
model = LpProblem("Transportation_Minimize_Cost", LpMinimize)
x11 = LpVariable("S1_to_D1", lowBound=0)
x12 = LpVariable("S1_to_D2", lowBound=0)
x13 = LpVariable("S1_to_D3", lowBound=0)
x21 = LpVariable("S2_to_D1", lowBound=0)
x22 = LpVariable("S2_to_D2", lowBound=0)
```

```
x23 = LpVariable("S2_to_D3", lowBound=0)
model += 2*x11 + 3*x12 + 1*x13 + 5*x21 + 4*x22 + 2*x23
model += x11 + x12 + x13 == 30
model += x21 + x22 + x23 == 20
model += x11 + x21 == 10
model += x12 + x22 == 25
model += x13 + x23 == 15
model.solve()
print("Status:", LpStatus[model.status])
print("Optimal Transportation Plan:")
print(f"S1 -> D1: {x11.varValue}")
print(f"S1 -> D2: {x12.varValue}")
print(f"S1 -> D3: {x13.varValue}")
print(f"S2 -> D1: {x21.varValue}")
print(f"S2 -> D2: {x22.varValue}")
print(f"S2 -> D3: {x23.varValue}")
print("Minimum Total Cost:", value(model.objective))
```



	M1	M2	Supply
Plant P1	12	9	20
Plant P2	7	14	25
Plant P3	8	11	15
Demand	30	30	

```
# !pip install pulp
from pulp import *
model = LpProblem("Max_Profit_Transportation", LpMaximize)
x11 = LpVariable("P1_to_M1", lowBound=0)
x12 = LpVariable("P1_to_M2", lowBound=0)
x21 = LpVariable("P2_to_M1", lowBound=0)
x22 = LpVariable("P2_to_M2", lowBound=0)
x31 = LpVariable("P3_to_M1", lowBound=0)
x32 = LpVariable("P3_to_M2", lowBound=0)
model += 12*x11 + 9*x12 + 7*x21 + 14*x22 + 8*x31 + 11*x32
model += x11 + x12 == 20
model += x21 + x22 == 25
model += x31 + x32 == 15
model += x11 + x21 + x31 == 30
model += x12 + x22 + x32 == 30
model.solve()
print("Status:", LpStatus[model.status])
for v in model.variables():
    print(v.name, "=", v.varValue)
print("Maximum Profit = ", value(model.objective))
```



	R1	R2	R3	Supply
Source S1	10	15	12	40
Source S2	18	14	16	50
Demand	30	35	20	

```
# !pip install pulp
from pulp import *
model = LpProblem("Unbalanced_Transportation", LpMinimize)
x11 = LpVariable("S1_to_R1", lowBound=0)
x12 = LpVariable("S1_to_R2", lowBound=0)
x13 = LpVariable("S1_to_R3", lowBound=0)
x14 = LpVariable("S1_to_R4_dummy", lowBound=0)
x21 = LpVariable("S2_to_R1", lowBound=0)
x22 = LpVariable("S2_to_R2", lowBound=0)
x23 = LpVariable("S2_to_R3", lowBound=0)
x24 = LpVariable("S2_to_R4_dummy", lowBound=0)
model += (10*x11 + 15*x12 + 12*x13 + 0*x14 +
          18*x21 + 14*x22 + 16*x23 + 0*x24)
model += x11 + x12 + x13 + x14 == 40
model += x21 + x22 + x23 + x24 == 50
model += x11 + x21 == 30
model += x12 + x22 == 35
model += x13 + x23 == 20
model += x14 + x24 == 5
model.solve()
print("Status:", LpStatus[model.status])
for v in model.variables():
    print(v.name, "=", v.varValue)
print("Minimum Transportation Cost = ", value(model.objective))
```

Golden Section Method

i) Use Golden Section Method to minimize the function $f(x)$ = $(x - 2)2$ over the interval [0, 5]. Perform the search with tolerance level ϵ = 0.00001 (based on Golden Ratio). Write a Python program to implement the method and find the approximate minimizer and the corresponding minimum value of the function.

```
def golden_section_search(func, a, b,tal=1e-8):
  phi=(1+5**0.5)/2
  resphi=2-phi
  x1=a+resphi*(b-a)
  x2=b-resphi*(b-a)
  f1=func(x1)
  f2=func(x2)
  while abs(b-a)>tal:
    if f1<f2:
      b=x2
      x2=x1
      f2=f1
      x1=a+resphi*(b-a)
      f1=func(x1)
    else:
      a,x1,f1=x1,x2,f2
      x2=b-resphi*(b-a)
      f2=func(x2)
  return (a+b)/2
f=lambda x:(x-2)**2
result=golden_section_search(f,0,5,1e-8)
min=f(result)
print("Approximated minimum at X =",result)
print("Approximated minimum value is",min)
```

ii) Use Golden Section Method to maximize the function $f(x)$ = 4 + 4x – 6x2 + 4x3 – x4 over the interval [0, 2]. Perform the search with tolerance level ϵ = 0.00000002 (based on Golden Ratio). Write a Python program to implement the method and find the approximate maximizer and the corresponding maximum value of the function.

```
def golden_section_maximize(func, a, b, tol=2e-8):
    phi = (1 + 5 ** 0.5) / 2
    resphi = 2 - phi
    x1 = a + resphi * (b - a)
    x2 = b - resphi * (b - a)
    f1 = func(x1)
    f2 = func(x2)
    while abs(b - a) > tol:
        if f1 < f2:
            a = x1
            x1 = x2
            f1 = f2
            x2 = b - resphi * (b - a)
            f2 = func(x2)
        else:
            b = x2
            x2 = x1
            f2 = f1
            x1 = a + resphi * (b - a)
            f1 = func(x1)
    x_max = (a + b) / 2
    f_max = func(x_max)
    return x_max, f_max
f = lambda x: 4 + 4*x - 6*x**2 + 4*x**3 - x**4
x_max, f_max = golden_section_maximize(f, 0, 2, 2e-8)
```

```
print("Approximate maximizer x =", x_max)
print("Approximate maximum value f(x) =", f_max)
```

FIBONICCI SEARCH METHOD

III) USE THE FIBONACCI SEARCH METHOD TO MINIMIZE THE FUNCTION F(X)=(X-2)**2 OVER THENINTERVAL [0,5].PERFORM THE THE SEARCH USINIG N=10 ITRITION(BASED ON THE FIBONICCI SEQUQNCE).

```
def fibonacci_search(func, a, b, n):
    F =[0,1]
    for i in range(2,n+1):
      F.append(F[i-1]+F[i-2])
    k=0
    x1=a+(F[n-2]/F[n])*(b-a)
    x2=a+(F[n-1]/F[n])*(b-a)
    f1,f2=func(x1),func(x2)
    while k<n-2:
      k+=1
      if f1>f2:
        a=x1
        x1,f1=x2,f2
        x2=a+(F[n-k-1]/F[n-k])*(b-a)
        f2=func(x2)
      else:
        b=x2
        x2,f2=x1,f1
        x1=a+(F[n-k-2]/F[n-k])*(b-a)
        f1=func(x1)
    return (a+b)/2
f=lambda x:(x-2)**2
result_fibonacci=fibonacci_search(f,0,5,1000)
min=f(result_fibonacci)
print("Approximated minimum at X =",result_fibonacci)
print("Approximated minimum value =",min)
```

iv) Use the Fibonacci Search Method to maximize the function $f(x)$ = 4 + 4x −6x2+4x3−x4 over the interval [0, 2]. Perform the search using n=1000 iterations (based on the Fibonacci sequence). Write a Python program to implement the method and find the approximate maximizer and the corresponding maximum value of the function.

```
def fibonacci_seaarch(func, a, b, n):
    F= [0, 1]
    for i in range(2, n+1):
        F.append(F[i - 1] + F[i - 2])

    k = 0
    x1 = a + (F[n -2] / F[n]) * (b -a)
    x2 = a + (F[n -1] / F[n]) * (b -a)
    f1, f2 = func(x1), func(x2)

    while k < n-2:
        k += 1
        if f1 < f2:
            a = x1
            x1, f1 = x2, f2
            x2 = a + (F[n -k -1] / F[n - k]) * (b -a)
            f2 = func(x2)

        else:
            b = x2
            x2, f2 = x1, f1
            x1 = a + (F[n -k -2] / F[n - k]) * (b -a)
            f1 = func(x1)

    return (a+b)/2

f = lambda x: 4 + 4*x - 6*x**2 + 4*x**3 - x**4
result_fibonacci = fibonacci_seaarch(f, 0, 2, 1000)
Max = f(result_fibonacci)

print("Approximated maximized value at x =", result_fibonacci)
print("Approximate maximum value is ", Max)
```