Report 3

Team information.

- Team leader: Nikita Tsukanov
- Team member 1: Dmitriy Tetkin
- Team member 2: Mikhail Trifonov
- Team member 3: Ilsaf Abdulkhakov

Team participation

- Nikita Tsukanov - 5/5 grade
- Dmitriy Tetkin - 5/5 grade
- Mikhail Trifonov - 5/5 grade
- Ilsaf Abdulkhakov - 5/5 grade

Link to the product.

- The product is available: https://github.com/Optimization-Innopolis/task3

Programming language.

- Programming language: Python

Transportation problem.

- Balanced problem

- Declaration of the problem:
  Supply at Sources:
  Supply $S_1 = 20$ units
  Supply $S_2 = 30$ units
  Supply $S_3 = 25$ units
  Demand at Destinations:
  Demand $D_1 = 10$ units
  Demand $D_2 = 15$ units
  Demand $D_3 = 30$ units
  Demand $D_4 = 20$ units
  Transportation Costs

	D_1	D_2	D_3	D_4
S_1	8	6	10	9
S_2	9	12	13	7
S_3	14	9	16	5

Result

Initial problem

Input

	D_1	D_2	D_3	D_4	*Supply*
S_1	8	6	10	9	20
S_2	9	12	13	7	30
S_3	14	9	16	5	25
Demand	10	15	30	20	

Output

Method	Initial Basic Feasible Solution	Cost
North-West Corner	$\begin{pmatrix} 10.0 & 10.0 & 0.0 & 0.0 \\ 0.0 & 5.0 & 25.0 & 0.0 \\ 0.0 & 0.0 & 5.0 & 20.0 \end{pmatrix}$	705.0
Vogel's Approximation	$\begin{pmatrix} 10.0 & 0.0 & 10.0 & 0.0 \\ 0.0 & 15.0 & 15.0 & 0.0 \\ 0.0 & 0.0 & 5.0 & 20.0 \end{pmatrix}$	735.0
Russell's Approximation	$\begin{pmatrix} 10.0 & 10.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 10.0 & 20.0 \\ 0.0 & 5.0 & 20.0 & 0.0 \end{pmatrix}$	775.0

Test Case 1: Simple balanced costs

Input

	D_1	D_2	D_3	D_4	*Supply*
S_1	4	8	8	6	15
S_2	6	4	3	5	25
S_3	5	7	6	4	10
Demand	5	15	15	15	

Output

Method	Initial Basic Feasible Solution	Cost
North-West Corner	$\begin{pmatrix} 5.0 & 10.0 & 0.0 & 0.0 \\ 0.0 & 5.0 & 15.0 & 5.0 \\ 0.0 & 0.0 & 0.0 & 10.0 \end{pmatrix}$	230.0
Vogel's Approximation	$\begin{pmatrix} 5.0 & 5.0 & 0.0 & 5.0 \\ 0.0 & 10.0 & 15.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 10.0 \end{pmatrix}$	215.0
Russell's Approximation	$\begin{pmatrix} 5.0 & 0.0 & 5.0 & 5.0 \\ 0.0 & 15.0 & 10.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 10.0 \end{pmatrix}$	220.0

Test Case 2: Moderate balanced costs

Input

	D_1	D_2	D_3	D_4	*Supply*
S_1	8	6	10	9	20
S_2	9	12	13	7	30
S_3	14	9	16	5	25
Demand	10	20	15	30	

Output

Method	Initial Basic Feasible Solution	Cost
North-West Corner	$\begin{pmatrix} 10.0 & 10.0 & 0.0 & 0.0 \\ 0.0 & 10.0 & 15.0 & 5.0 \\ 0.0 & 0.0 & 0.0 & 25.0 \end{pmatrix}$	615.0
Vogel's Approximation	$\begin{pmatrix} 10.0 & 0.0 & 10.0 & 0.0 \\ 0.0 & 20.0 & 5.0 & 5.0 \\ 0.0 & 0.0 & 0.0 & 25.0 \end{pmatrix}$	645.0
Russell's Approximation	$\begin{pmatrix} 10.0 & 10.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 30.0 \\ 0.0 & 10.0 & 15.0 & 0.0 \end{pmatrix}$	680.0

Test Case 3: High balanced costs

Input

	D_1	D_2	D_3	D_4	*Supply*
S_1	12	14	17	15	25
S_2	15	13	18	10	35
S_3	14	16	12	19	15
Demand	15	25	10	25	

Output

Method	Initial Basic Feasible Solution	Cost
North-West Corner	$\begin{pmatrix} 15.0 & 10.0 & 0.0 & 0.0 \\ 0.0 & 15.0 & 10.0 & 10.0 \\ 0.0 & 0.0 & 0.0 & 15.0 \end{pmatrix}$	1080.0
Vogel's Approximation	$\begin{pmatrix} 15.0 & 0.0 & 0.0 & 10.0 \\ 0.0 & 25.0 & 0.0 & 10.0 \\ 0.0 & 0.0 & 10.0 & 5.0 \end{pmatrix}$	970.0
Russell's Approximation	$\begin{pmatrix} 15.0 & 10.0 & 0.0 & 0.0 \\ 0.0 & 15.0 & 0.0 & 20.0 \\ 0.0 & 0.0 & 10.0 & 5.0 \end{pmatrix}$	930.0

Code

```
import numpy as np
import pandas as pd


def check_balanced(supply, demand):
    if sum(supply) != sum(demand):
        return False
    return True


def check_applicability(supply, demand, costs):
    if any(s < 0 for s in supply) or any(d < 0 for d in demand) or \
```

```
            any(cost < 0 for row in costs for cost in row) or \
            len(supply) == 0 or len(demand) == 0 or \
            costs.shape != (len(supply), len(demand)):
        print("The method is not applicable!")
        return False
    return True


def print_input_table(costs, supply, demand):
    table = pd.DataFrame(costs, columns=[f'D{i + 1}' for i in range(len(demand))])
    table['Supply'] = supply
    table.loc['Demand'] = demand + [np.nan]
    print("Input Parameter Table:")
    print(table)
    print("\n")


def north_west_corner(supply, demand):
    x = np.zeros((len(supply), len(demand)))
    i = j = 0
    while i < len(supply) and j < len(demand):
        min_val = min(supply[i], demand[j])
        x[i][j] = min_val
        supply[i] -= min_val
        demand[j] -= min_val
        if supply[i] == 0:
            i += 1
        if demand[j] == 0:
            j += 1
    return x


def vogel_approximation(supply, demand, costs):
    x = np.zeros((len(supply), len(demand)))
    supply_copy = supply.copy()
    demand_copy = demand.copy()

    while sum(supply_copy) > 0:
        penalties = []
        for i, s in enumerate(supply_copy):
            if s > 0:
                row = [cost for j, cost in enumerate(costs[i]) if demand_copy[j] > 0]
                row.sort()
                penalties.append(row[1] - row[0] if len(row) > 1 else row[0])
            else:
                penalties.append(float('inf'))

        for j, d in enumerate(demand_copy):
            if d > 0:
                col = [costs[i][j] for i in range(len(supply_copy)) if supply_copy[i] > 0]
                col.sort()
                penalties.append(col[1] - col[0] if len(col) > 1 else col[0])
            else:
                penalties.append(float('inf'))

        index = penalties.index(min(penalties))
        if index < len(supply_copy):
            i = index
            j = np.argmin([costs[i][j] if demand_copy[j] > 0 else float('inf') for j in ra
        else:
            j = index - len(supply_copy)
```

```
            i = np.argmin([costs[i][j] if supply_copy[i] > 0 else float('inf') for i in ra

        min_val = min(supply_copy[i], demand_copy[j])
        x[i][j] = min_val
        supply_copy[i] -= min_val
        demand_copy[j] -= min_val
    return x


def russell_approximation(supply, demand, costs):
    x = np.zeros((len(supply), len(demand)))
    u = np.zeros(len(supply))
    v = np.zeros(len(demand))
    for i in range(len(supply)):
        u[i] = min(costs[i])
    for j in range(len(demand)):
        v[j] = min(costs[:, j])

    while sum(supply) > 0:
        i, j = np.unravel_index(np.argmax(u.reshape(-1, 1) + v - costs), costs.shape)
        min_val = min(supply[i], demand[j])
        x[i][j] = min_val
        supply[i] -= min_val
        demand[j] -= min_val
        if supply[i] == 0:
            u[i] = -float('inf')
        if demand[j] == 0:
            v[j] = -float('inf')
    return x


def solve_transportation_problem(supply, demand, costs):
    if not check_applicability(supply, demand, costs):
        return

    if not check_balanced(supply, demand):
        print("The problem is not balanced!")
        return

    print_input_table(costs, supply, demand)

    nw_corner_sol = north_west_corner(supply.copy(), demand.copy())
    vogel_sol = vogel_approximation(supply.copy(), demand.copy(), costs)
    russell_sol = russell_approximation(supply.copy(), demand.copy(), costs)

    print("Initial Basic Feasible Solution using North-West Corner Method:")
    print(nw_corner_sol)
    print("\nInitial Basic Feasible Solution using Vogel's Approximation Method:")
    print(vogel_sol)
    print("\nInitial Basic Feasible Solution using Russell's Approximation Method:")
    print(russell_sol)
```