ISOM 677 Project 1
Applied Electronics

Seth Abayo, Lanston Chen, Guangming (Dola) Qiu, Yizhou (Paul) Sun, Yihua (Anthony) Wang

Introduction

Company Overview

Applied Electronics, a semiconductor chip manufacturer, serves a global customer base from six fabrication plants

Challenge

Optimize production plan for the upcoming year to address changing demands and to consider the potential of producing and shipping chips between countries to meet these demands more efficiently

Solution

With an initial plan that resulted in a total cost of $78.445 million, we want to apply Vogel's Approximation Method to yield a more cost-effective solution.

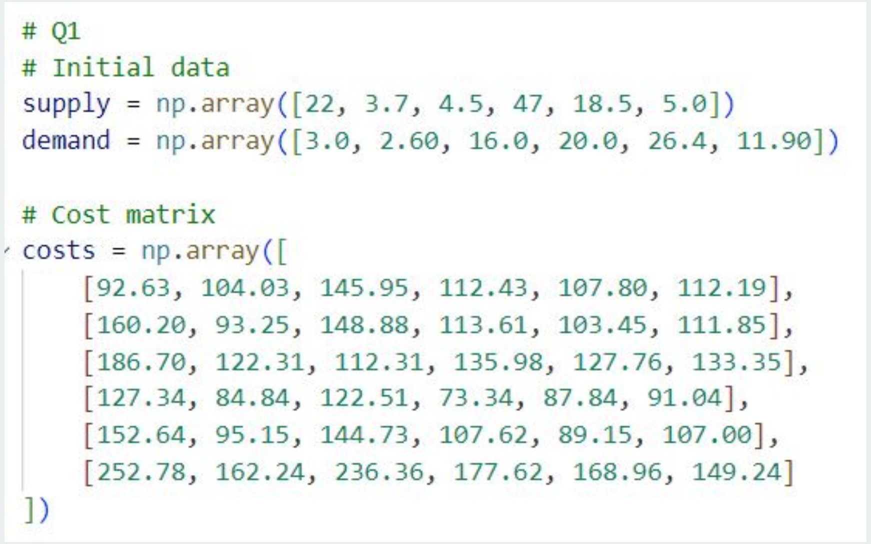
Question 1 Data Structure

In order to use VAM to produce a production plan in Python, we need those data inputs to function correctly

1. Supply and Demand
   The supply array represents the maximum units each supplier can produce, while the demand array reflects the required units at each consumer location.

2. Cost Matrix
   A two-dimensional array, where each cell [i][j] represents the cost of transportation from the i-th supplier to the j-th consumer.

Question 1 Data Structure



```
# Q1
# Initial data
supply = np.array([22, 3.7, 4.5, 47, 18.5, 5.0])
demand = np.array([3.0, 2.60, 16.0, 20.0, 26.4, 11.90])

# Cost matrix
costs = np.array([
    [92.63, 104.03, 145.95, 112.43, 107.80, 112.19],
    [160.20, 93.25, 148.88, 113.61, 103.45, 111.85],
    [186.70, 122.31, 112.31, 135.98, 127.76, 133.35],
    [127.34, 84.84, 122.51, 73.34, 87.84, 91.04],
    [152.64, 95.15, 144.73, 107.62, 89.15, 107.00],
    [252.78, 162.24, 236.36, 177.62, 168.96, 149.24]
])
```

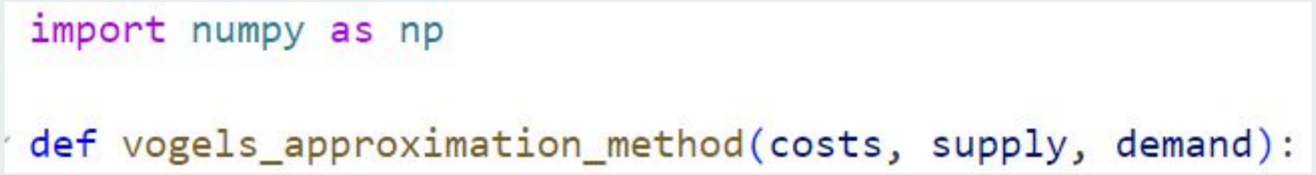
- Identify useful data and set up data array

Question 2 Vogel’s Approximation Method Implementation

Goal

By utilizing VAM, we want to minimize the total transportation cost by avoiding the incurrence of large costs while meeting all supply and demand constraints

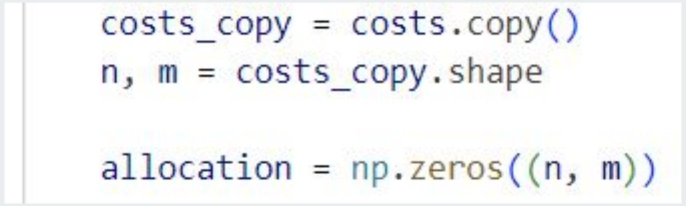
Question 2 Vogel’s Approximation Method Implementation



```
import numpy as np

def vogels_approximation_method(costs, supply, demand):
```

The code defines a function vogels_approximation_method that implements Vogel's Approximation Method (VAM) to solve a transportation problem

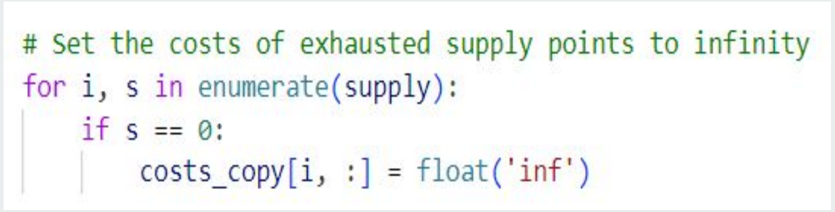
Question 2 Vogel's Approximation Method Implementation



```
    costs_copy = costs.copy()
    n, m = costs_copy.shape

    allocation = np.zeros((n, m))
```

- Creates a copy of the cost matrix to preserve the original data. This allows for the original data to remain intact for future reference or for running multiple iterations of the algorithm with different parameters.
- Initializes an allocation matrix with the same dimensions as the cost matrix, to keep track of the quantities to be transported from suppliers to consumers.

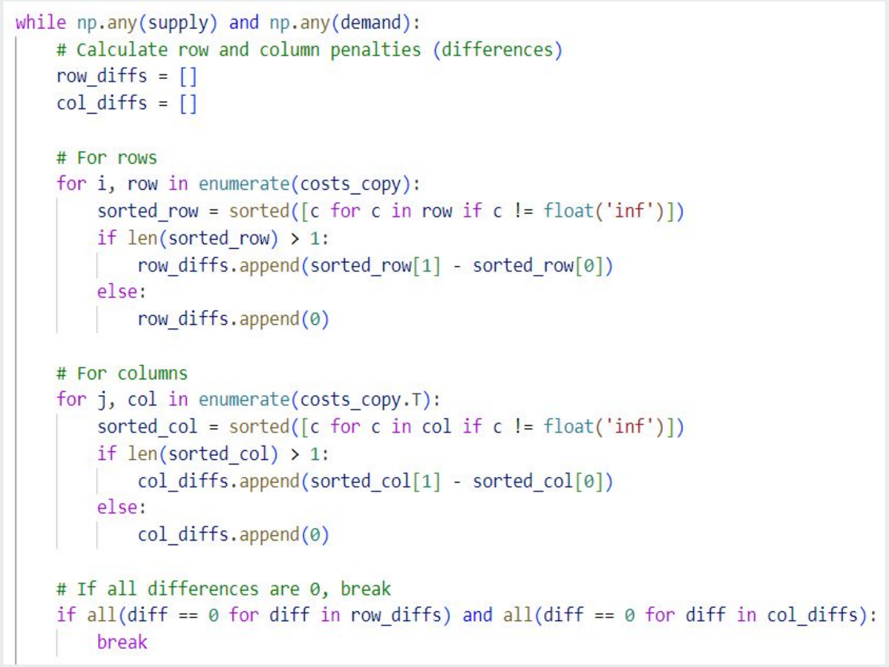
Question 2 Vogel’s Approximation Method Implementation



```python
# Set the costs of exhausted supply points to infinity
for i, s in enumerate(supply):
    if s == 0:
        costs_copy[i, :] = float('inf')
```

- Iterates through the supply array and sets the cost of transportation from any exhausted supply points (where supply is zero) to inf (infinity). This effectively removes these points from consideration.
- If a supplier has no supply left (supply is zero), it cannot provide any more goods. Setting its cost to infinity effectively removes it from the pool of available suppliers, as no further allocation should come from this supplier.

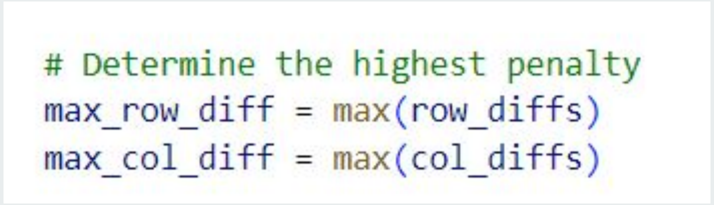
Question 2 Vogel’s Approximation Method Implementation



```python
while np.any(supply) and np.any(demand):
    # Calculate row and column penalties (differences)
    row_diffs = []
    col_diffs = []

    # For rows
    for i, row in enumerate(costs_copy):
        sorted_row = sorted([c for c in row if c != float('inf')])
        if len(sorted_row) > 1:
            row_diffs.append(sorted_row[1] - sorted_row[0])
        else:
            row_diffs.append(0)

    # For columns
    for j, col in enumerate(costs_copy.T):
        sorted_col = sorted([c for c in col if c != float('inf')])
        if len(sorted_col) > 1:
            col_diffs.append(sorted_col[1] - sorted_col[0])
        else:
            col_diffs.append(0)

    # If all differences are 0, break
    if all(diff == 0 for diff in row_diffs) and all(diff == 0 for diff in col_diffs):
        break
```

- For each row and column in the cost matrix, calculate the penalty, which is the difference between the lowest and the next lowest costs in that row or column.
- Identify which allocations would lead to the most significant potential cost increase if not selected. This step guides the algorithm to make decisions that prevent large cost increases

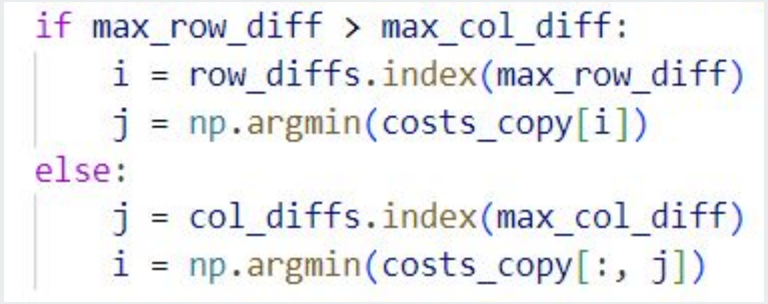
Question 2 Vogel's Approximation Method Implementation



```
# Determine the highest penalty
max_row_diff = max(row_diffs)
max_col_diff = max(col_diffs)
```

- Identifies the row or column with the highest penalty, which signifies the most critical allocation to be made next.

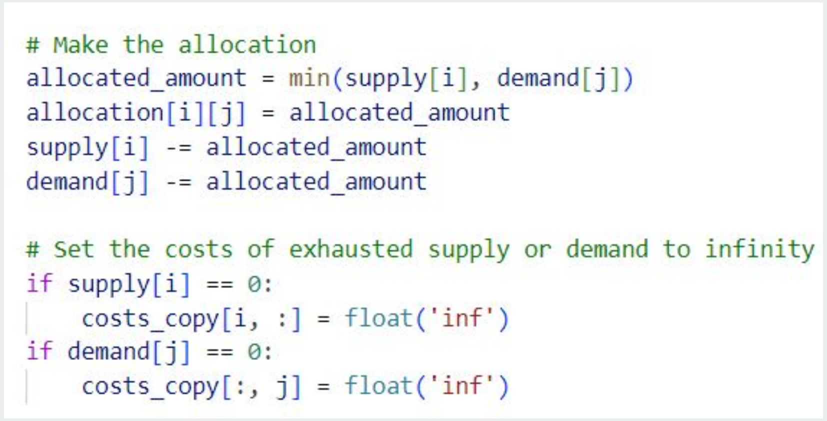
Question 2 Vogel’s Approximation Method Implementation



```
if max_row_diff > max_col_diff:
    i = row_diffs.index(max_row_diff)
    j = np.argmin(costs_copy[i])
else:
    j = col_diffs.index(max_col_diff)
    i = np.argmin(costs_copy[:, j])
```

- Choose where to make the next allocation in the plan. By focusing places with higher opportunity cost, it can prevent dealing with a much higher cost in the next steps, thus potentially reducing overall costs

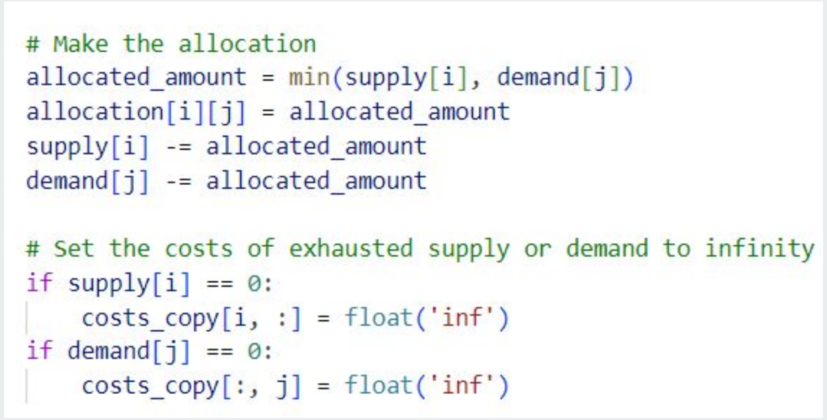
Question 2 Vogel’s Approximation Method Implementation



```
# Make the allocation
allocated_amount = min(supply[i], demand[j])
allocation[i][j] = allocated_amount
supply[i] -= allocated_amount
demand[j] -= allocated_amount

# Set the costs of exhausted supply or demand to infinity
if supply[i] == 0:
    costs_copy[i, :] = float('inf')
if demand[j] == 0:
    costs_copy[:, j] = float('inf')
```

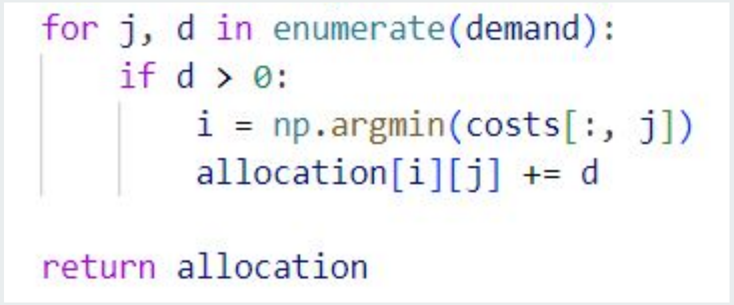
- Identifies the row or column with the highest penalty, which signifies the most critical allocation to be made next.
- Allocates as much as possible to the cell with the lowest cost in that row or column, reducing the corresponding supply and demand by the allocated amount.

Question 2 Vogel’s Approximation Method Implementation


```
# Make the allocation
allocated_amount = min(supply[i], demand[j])
allocation[i][j] = allocated_amount
supply[i] -= allocated_amount
demand[j] -= allocated_amount

# Set the costs of exhausted supply or demand to infinity
if supply[i] == 0:
    costs_copy[i, :] = float('inf')
if demand[j] == 0:
    costs_copy[:, j] = float('inf')
```

- The min(supply[i], demand[j]) function determines the maximum possible amount that can be allocated from supplier i to consumer j without exceeding either the supply available at i or the demand required at j. This ensures that allocations are made within the constraints of supply and demand.
- Once the supply from a supplier is depleted (supply[i] == 0), or the demand of a consumer is completely satisfied (demand[j] == 0), there should be no further allocation from that supplier or to that consumer.

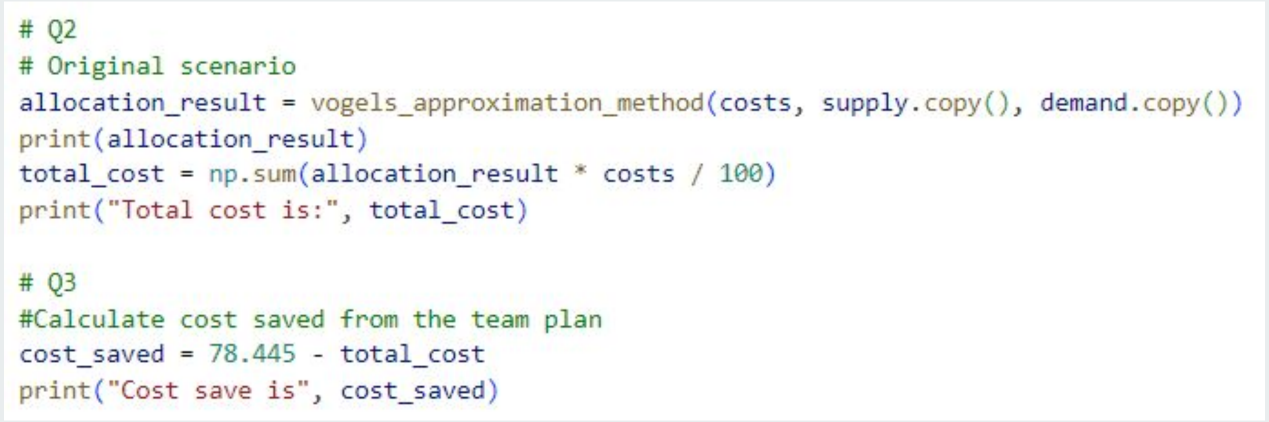
Question 2 Vogel’s Approximation Method Implementation



```
for j, d in enumerate(demand):
    if d > 0:
        i = np.argmin(costs[:, j])
        allocation[i][j] += d

return allocation
```

- If there's still unmet demand after the main allocation process, it must be fulfilled to complete the production plan. The remaining demand is allocated to the suppliers with the lowest costs to satisfy all demands while still aiming for cost-effectiveness.

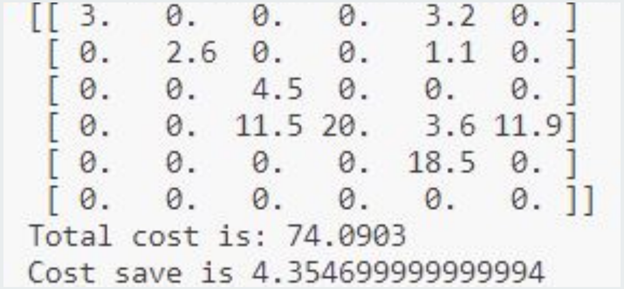
Question 3 What's the new plan using 100% capacity?



```
# Q2
# Original scenario
allocation_result = vogels_approximation_method(costs, supply.copy(), demand.copy())
print(allocation_result)
total_cost = np.sum(allocation_result * costs / 100)
print("Total cost is:", total_cost)

# Q3
#Calculate cost saved from the team plan
cost_saved = 78.445 - total_cost
print("Cost save is", cost_saved)
```

- The total cost for AE's production plan using VAM is calculated by multiplying the allocation result by the cost matrix and dividing by 100 to handle unit cost scaling.
- The cost saved by implementing the VAM production plan is evaluated by comparing it against the team's current plan, which has a cost of $78.445

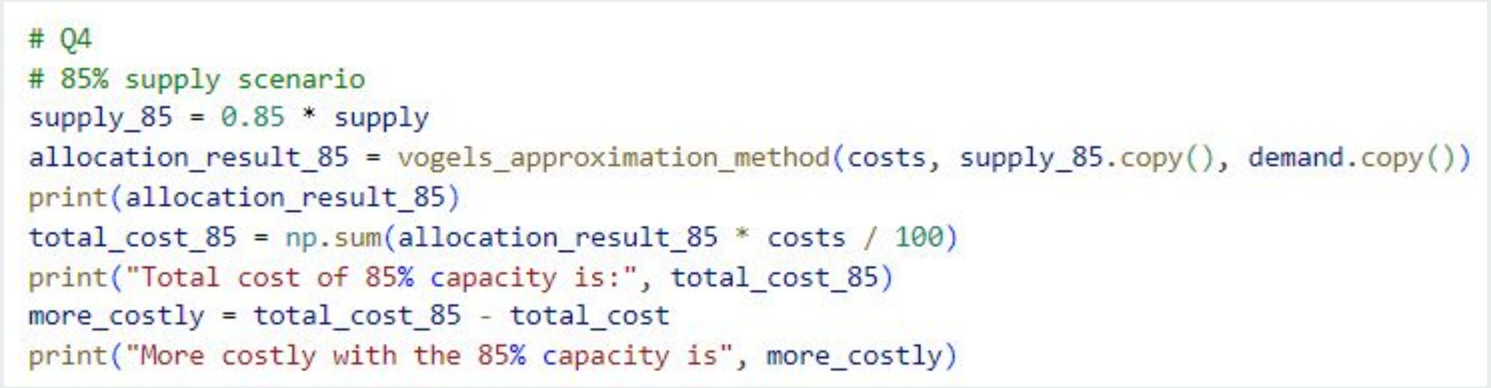
Question 3 What's the new plan using 100% capacity?



```
[[ 3.   0.   0.   0.   3.2  0. ]
 [ 0.   2.6  0.   0.   1.1  0. ]
 [ 0.   0.   4.5  0.   0.   0. ]
 [ 0.   0.  11.5 20.   3.6 11.9]
 [ 0.   0.   0.   0.  18.5  0. ]
 [ 0.   0.   0.   0.   0.   0. ]]
Total cost is: 74.0903
Cost save is 4.354699999999994
```

- The production plan was calculated using VAM, resulting in a total transportation cost of $74.09.
- When compared to the existing plan or baseline, VAM achieves a cost saving of $4.35.

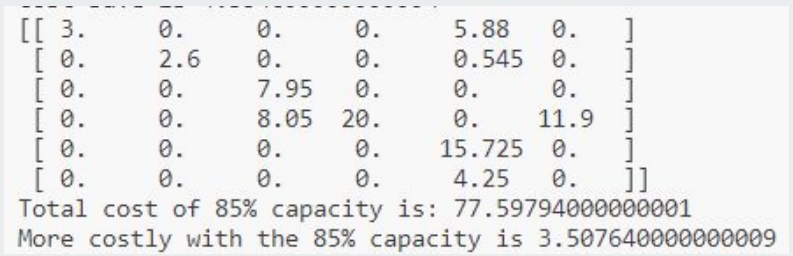
Question 4 Production plan using only 85% capacity



```
# Q4
# 85% supply scenario
supply_85 = 0.85 * supply
allocation_result_85 = vogels_approximation_method(costs, supply_85.copy(), demand.copy())
print(allocation_result_85)
total_cost_85 = np.sum(allocation_result_85 * costs / 100)
print("Total cost of 85% capacity is:", total_cost_85)
more_costly = total_cost_85 - total_cost
print("More costly with the 85% capacity is", more_costly)
```

- The scenario adjusts AE's production capacity to 85%, reflecting a more realistic operational constraint.

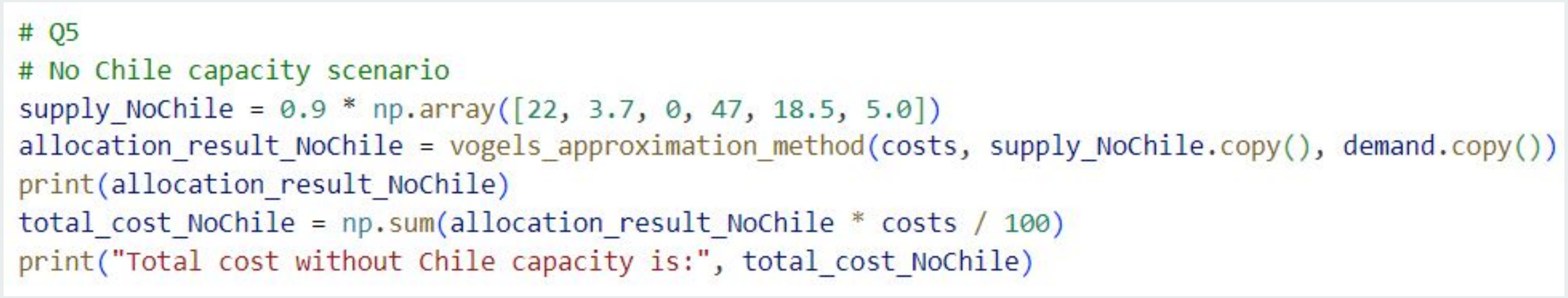
Question 4 Production plan using only 85% capacity



```
[[ 3.     0.     0.     0.     5.88   0.    ]
 [ 0.     2.6    0.     0.     0.545  0.    ]
 [ 0.     0.     7.95   0.     0.     0.    ]
 [ 0.     0.     8.05  20.     0.    11.9   ]
 [ 0.     0.     0.     0.    15.725  0.    ]
 [ 0.     0.     0.     0.     4.25   0.    ]]
Total cost of 85% capacity is: 77.59794000000001
More costly with the 85% capacity is 3.507640000000009
```

- The total cost for AE's production plan, with the 85% supply capacity, is $77.60, which represents the additional cost incurred due to reduced capacity

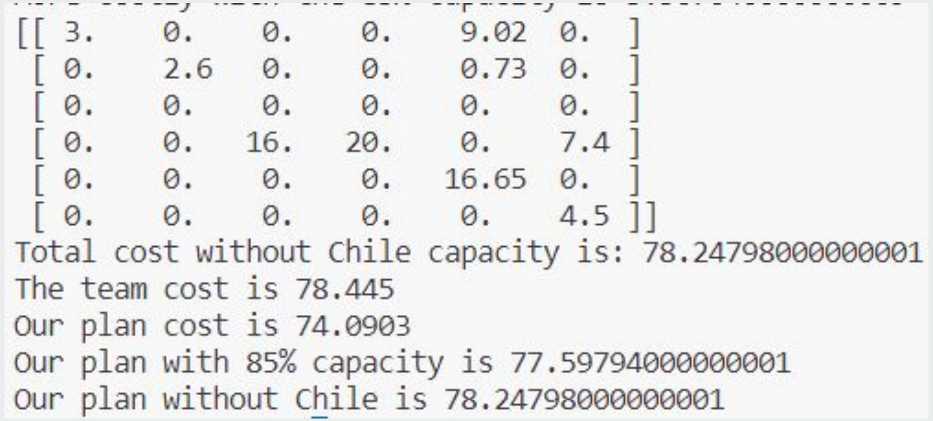
Question 5 Production plan which excludes Chile and use 90% capacity of other plants



```python
# Q5
# No Chile capacity scenario
supply_NoChile = 0.9 * np.array([22, 3.7, 0, 47, 18.5, 5.0])
allocation_result_NoChile = vogels_approximation_method(costs, supply_NoChile.copy(), demand.copy())
print(allocation_result_NoChile)
total_cost_NoChile = np.sum(allocation_result_NoChile * costs / 100)
print("Total cost without Chile capacity is:", total_cost_NoChile)
```

- The production plan has been recalculated without considering the supply from Chile.
- This scenario simulates a situation where AE must rely entirely on other plants, operating at 90% of their capacity to meet demands.

Question 5 Production plan which excludes Chile and use 90% capacity of other plants



```
[illegible]
[[ 3.    0.    0.    0.    9.02  0.  ]
 [ 0.    2.6   0.    0.    0.73  0.  ]
 [ 0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.   16.   20.    0.    7.4 ]
 [ 0.    0.    0.    0.   16.65  0.  ]
 [ 0.    0.    0.    0.    0.    4.5 ]]
Total cost without Chile capacity is: 78.24798000000001
The team cost is 78.445
Our plan cost is 74.0903
Our plan with 85% capacity is 77.59794000000001
Our plan without Chile is 78.24798000000001
```

- The total cost incurred without the Chile plant's capacity is calculated to be $78.25
- Highest Cost: The VAM plan without Chile is the most expensive at $78.24798.
- Lowest Cost: The VAM Optimized Plan is the least expensive, providing the most savings at $74.0903.

Conclusion

VAM was applied under several scenarios:

1. Using full capacity at each location.
2. Restricting production to a more realistic 85% of each plant's capacity.
3. Excluding the Chile plant completely and redistributing its demand to the remaining plants at up to 90% of their capacity.