OR_Assignment2

September 27, 2025

[1]:
```python
import gurobipy as gp
from gurobipy import GRB
```

[2]:
```python
# Data from Table 2.4 [cite: 61, 62]
fruits = ["Apple", "Banana", "Blueberries", "Durian", "Tangerine"]
nutrients = ["Calories", "Carbohydrate", "Fiber", "Vitamin A", "Vitamin C"]

# Price per 100g
prices = {"Apple": 0.5, "Banana": 0.3, "Blueberries": 2.5, "Durian": 10,␣
 ↪"Tangerine": 0.5}

# Nutrition requirements (min, max)
requirements = {
    "Calories": (500, 3000), "Carbohydrate": (50, 400), "Fiber": (20, 30),
    "Vitamin A": (2000, 3500), "Vitamin C": (75, 150)
}

# Nutrition content per 100g of fruit
content = {
    ("Calories", "Apple"): 52, ("Calories", "Banana"): 89, ("Calories",␣
 ↪"Blueberries"): 57, ("Calories", "Durian"): 147, ("Calories", "Tangerine"):␣
 ↪53,
    ("Carbohydrate", "Apple"): 14, ("Carbohydrate", "Banana"): 23,␣
 ↪("Carbohydrate", "Blueberries"): 14, ("Carbohydrate", "Durian"): 27,␣
 ↪("Carbohydrate", "Tangerine"): 13,
    ("Fiber", "Apple"): 2.4, ("Fiber", "Banana"): 2.6, ("Fiber", "Blueberries"):
 ↪ 2.4, ("Fiber", "Durian"): 3.8, ("Fiber", "Tangerine"): 1.8,
    ("Vitamin A", "Apple"): 54, ("Vitamin A", "Banana"): 64, ("Vitamin A",␣
 ↪"Blueberries"): 54, ("Vitamin A", "Durian"): 44, ("Vitamin A", "Tangerine"):␣
 ↪681,
    ("Vitamin C", "Apple"): 4.6, ("Vitamin C", "Banana"): 8.7, ("Vitamin C",␣
 ↪"Blueberries"): 9.7, ("Vitamin C", "Durian"): 19.7, ("Vitamin C",␣
 ↪"Tangerine"): 26.7,
}

# Create a new model
m = gp.Model("diet_a")
```

```
# Create decision variables: quantity of each fruit in 100g units
x = m.addVars(fruits, name="quantity", lb=0)

# Set objective: minimize total cost
m.setObjective(gp.quicksum(prices[f] * x[f] for f in fruits), GRB.MINIMIZE)

# Add nutrient constraints
for n in nutrients:
    m.addRange(gp.quicksum(content[n, f] * x[f] for f in fruits),
               requirements[n][0], requirements[n][1], name=n)

# Optimize the model
m.optimize()
```

```
Set parameter Username
Set parameter LicenseID to value 2697524
Academic license - for non-commercial use only - expires 2026-08-21
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[arm] - Darwin 24.6.0
24G84)

CPU model: Apple M4
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 5 rows, 10 columns and 30 nonzeros
Model fingerprint: 0x5c18266f
Coefficient statistics:
  Matrix range     [1e+00, 7e+02]
  Objective range  [3e-01, 1e+01]
  Bounds range     [1e+01, 2e+03]
  RHS range        [3e+01, 4e+03]
Presolve time: 0.00s
Presolved: 5 rows, 10 columns, 30 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   1.143750e+02   0.000000e+00      0s
       3    2.9998792e+00   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.00 seconds (0.00 work units)
Optimal objective  2.999879183e+00
```

[3]:
```
# Print results
if m.status == GRB.OPTIMAL:
    print(f"\nOptimal solution found.")
    print(f"Minimum cost: ${m.objVal:.2f}")
    print("\nOptimal diet (in 100g units):")
    for f in fruits:
        if x[f].X > 0.001:
            print(f"- {f}: {x[f].X:.2f} units (i.e., {x[f].X * 100:.1f} grams)")
else:
    print("No optimal solution found.")
```

```
Optimal solution found.
Minimum cost: $3.00

Optimal diet (in 100g units):
- Banana: 6.05 units (i.e., 605.3 grams)
- Tangerine: 2.37 units (i.e., 236.8 grams)
```

##Result Interpretation: This means the cheapest combination of fruits that satisfies all dietary constraints is approximately 605 grams of bananas and 237 grams of tangerines.

[4]:
```
#Part B

# Data from Table 2.4 [cite: 61, 62]
fruits = ["Apple", "Banana", "Blueberries", "Durian", "Tangerine"]
nutrients = ["Calories", "Carbohydrate", "Fiber", "Vitamin A", "Vitamin C"]

# Price per 100g
prices = {"Apple": 0.5, "Banana": 0.3, "Blueberries": 2.5, "Durian": 10,␣
 ↪"Tangerine": 0.5}

# Nutrition requirements (min, max)
requirements = {
    "Calories": (500, 3000), "Carbohydrate": (50, 400), "Fiber": (20, 30),
    "Vitamin A": (2000, 3500), "Vitamin C": (75, 150)
}

# Nutrition content per 100g of fruit
content = {
    ("Calories", "Apple"): 52, ("Calories", "Banana"): 89, ("Calories",␣
 ↪"Blueberries"): 57, ("Calories", "Durian"): 147, ("Calories", "Tangerine"):␣
 ↪53,
    ("Carbohydrate", "Apple"): 14, ("Carbohydrate", "Banana"): 23,␣
 ↪("Carbohydrate", "Blueberries"): 14, ("Carbohydrate", "Durian"): 27,␣
 ↪("Carbohydrate", "Tangerine"): 13,
    ("Fiber", "Apple"): 2.4, ("Fiber", "Banana"): 2.6, ("Fiber", "Blueberries"):
 ↪ 2.4, ("Fiber", "Durian"): 3.8, ("Fiber", "Tangerine"): 1.8,
    ("Vitamin A", "Apple"): 54, ("Vitamin A", "Banana"): 64, ("Vitamin A",␣
 ↪"Blueberries"): 54, ("Vitamin A", "Durian"): 44, ("Vitamin A", "Tangerine"):␣
 ↪681,
    ("Vitamin C", "Apple"): 4.6, ("Vitamin C", "Banana"): 8.7, ("Vitamin C",␣
 ↪"Blueberries"): 9.7, ("Vitamin C", "Durian"): 19.7, ("Vitamin C",␣
 ↪"Tangerine"): 26.7,
}

# Add fat data to the model
requirements['Fat'] = (0, 10)
content[('Fat', "Apple")] = 0.2
content[('Fat', "Banana")] = 0.3
content[('Fat', "Blueberries")] = 0.3
content[('Fat', "Durian")] = 5
content[('Fat', "Tangerine")] = 0.3

# Add the new fat constraint to the model before optimizing
m.addRange(gp.quicksum(content['Fat', f] * x[f] for f in fruits),
           requirements['Fat'][0], requirements['Fat'][1], name='Fat')

# Create a new model
m = gp.Model("diet_a")

# Create decision variables: quantity of each fruit in 100g units
x = m.addVars(fruits, name="quantity", lb=0)

# Set objective: minimize total cost
m.setObjective(gp.quicksum(prices[f] * x[f] for f in fruits), GRB.MINIMIZE)

# Add nutrient constraints
for n in nutrients:
    m.addRange(gp.quicksum(content[n, f] * x[f] for f in fruits),
               requirements[n][0], requirements[n][1], name=n)

# Optimize the model
m.optimize()
```

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[arm] - Darwin 24.6.0
24G84)

CPU model: Apple M4
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 5 rows, 10 columns and 30 nonzeros
Model fingerprint: 0x5c18266f
Coefficient statistics:
  Matrix range     [1e+00, 7e+02]
  Objective range  [3e-01, 1e+01]
  Bounds range     [1e+01, 2e+03]
  RHS range        [3e+01, 4e+03]
Presolve time: 0.00s
Presolved: 5 rows, 10 columns, 30 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
```

```
       0    0.0000000e+00   1.143750e+02   0.000000e+00      0s
       3    2.9998792e+00   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.01 seconds (0.00 work units)
Optimal objective  2.999879183e+00
```

[5]:
```
# Print results
if m.status == GRB.OPTIMAL:
    print(f"\nOptimal solution found.")
    print(f"Minimum cost: ${m.objVal:.2f}")
    print("\nOptimal diet (in 100g units):")
    for f in fruits:
        if x[f].X > 0.001:
            print(f"- {f}: {x[f].X:.2f} units (i.e., {x[f].X * 100:.1f} grams)")
else:
    print("No optimal solution found.")
```

```

Optimal solution found.
Minimum cost: $3.00

Optimal diet (in 100g units):
- Banana: 6.05 units (i.e., 605.3 grams)
- Tangerine: 2.37 units (i.e., 236.8 grams)
```

##Verification: As we can see the min cost remain the same as do the banana and Tangerine values. Hence no impact of adding fat verified.

[15]:
```
#6 (a),(b)

import numpy as np

I0 = 2000
first_week_sales = 25
T = 15
prices = [60,54,48,36]
demands = [125,162.5,217.5,348.8]
salvage = 25

I_r = I0 - first_week_sales
T_r = T - 1

m = gp.Model()
t = m.addVars(4, lb=0, name="t")
m.addConstr(sum(t[i] for i in range(4)) == T_r)
m.addConstr(sum(demands[i]*t[i] for i in range(4)) <= I_r)
# objective: realized week1 revenue + revenue from remaining sales + salvage of␣
 ↪leftover
```

```
sales = gp.quicksum(prices[i]*demands[i]*t[i] for i in range(4))
salvage_term = salvage * (I_r - gp.quicksum(demands[i]*t[i] for i in range(4)))
m.setObjective(prices[0]*first_week_sales + sales + salvage_term, GRB.MAXIMIZE)
m.setParam('OutputFlag', 0)
m.optimize()

print("t:", [t[i].X for i in range(4)])
print("Objective:", m.objVal)
```

```
t: [8.0, 6.0, 0.0, 0.0]
Objective: 114150.0
```

##Results: As we can see that compared to the benchmark (116,750), revenue drops by 2,600 because of poor first-week sales.

But if we stuck with original policy: then Revenue is- 116750-{100*(60-25)}= 116,750 − 3,500 = 113,250(since only 25 units were sold and rest were salvaged)

so the improvement is- 114150-113250=900

[14]:
```
#6(c)

# ---------- (c) Fine-grained grid ----------
prices_grid = list(range(60, 35, -2))  # 60, 58, ..., 36
# Linear interpolation of demands
demands_grid = [125 + (348.8 - 125) * (60 - p) / (60 - 36) for p in prices_grid]

sol_c, rev_c = solve_markdown(prices_grid, demands_grid, inventory, weeks,␣
  ↪salvage_value)
print("Part (c):", sol_c, rev_c, "Improvement:", rev_c - rev_a)
```

```
Part (c): {0: 8.297587131367294, 1: 6.702412868632706, 2: 0.0, 3: 0.0, 4: 0.0,
5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0, 12: 0.0}
118074.39678284182 Improvement: 1324.3967828418245
```

##Results:

As we can see that with new policy the revenue increases by: 118074.39678284182 Improvement: 1324.396782841824

Unlike the benchmark (only 2 price levels used), the fine grid allows smoother demand matching, reducing salvage and extracting more consumer surplus.