GPINA-ISYE6501-HW7

July 1, 2025

1 Diet Optimization using Gurobi

In this notebook, we solve a diet optimization problem using Gurobi. The objective is to find the cheapest diet that meets all nutritional requirements. We'll also explore extensions with logical and variety constraints.

1.1 Step 1: Load Data

[31]:
```
import pandas as pd
import gurobipy as gp
from gurobipy import GRB

# Load Excel file
xls = pd.read_excel("diet.xls", sheet_name=None)
foods_df = xls['Sheet1']
nutrients_df = xls['Sheet2']
foods_df.head()
```

[31]:

	Foods	Price/ Serving	Serving Size	Calories	\
0	Frozen Broccoli	0.16	10 Oz Pkg	73.8	
1	Carrots,Raw	0.07	1/2 Cup Shredded	23.7	
2	Celery, Raw	0.04	1 Stalk	6.4	
3	Frozen Corn	0.18	1/2 Cup	72.2	
4	Lettuce,Iceberg,Raw	0.02	1 Leaf	2.6	

	Cholesterol mg	Total_Fat g	Sodium mg	Carbohydrates g	Dietary_Fiber g	\
0	0.0	0.8	68.2	13.6	8.5	
1	0.0	0.1	19.2	5.6	1.6	
2	0.0	0.1	34.8	1.5	0.7	
3	0.0	0.6	2.5	17.1	2.0	
4	0.0	0.0	1.8	0.4	0.3	

	Protein g	Vit_A IU	Vit_C IU	Calcium mg	Iron mg
0	8.0	5867.4	160.2	159.0	2.3
1	0.6	15471.0	5.1	14.9	0.3
2	0.3	53.6	2.8	16.0	0.2
3	2.5	106.6	5.2	3.3	0.3
4	0.2	66.0	0.8	3.8	0.1

1.1.1 Interpretation of Food Data Table for the Diet Optimization Problem

The table above lists several foods along with their nutritional content and price per serving. Each row represents a food item, and the columns provide information such as serving size, calories, cholesterol, fat, sodium, carbohydrates, fiber, protein, and key vitamins and minerals.

Implications for Diet Optimization:

- **Cost Efficiency:** Foods like "Lettuce, Iceberg, Raw" and "Celery, Raw" have very low prices per serving ($0.02 and $0.04, respectively), making them attractive for minimizing total diet cost.
- **Nutrient Density:** "Frozen Broccoli" and "Carrots, Raw" provide high amounts of vitamins (e.g., Vitamin A and C) and fiber per serving, which helps meet daily nutritional requirements efficiently.
- **Variety and Balance:** While some foods are cheap or nutrient-dense, no single food provides all required nutrients in sufficient quantities. For example, "Celery, Raw" is low in calories and protein, so relying solely on it would not meet energy or protein needs.
- **Optimization Trade-offs:** The goal of the diet problem is to select a combination of these foods (and others in the full dataset) that meets all nutritional constraints (minimum and maximum daily intake for each nutrient) at the lowest possible cost. The optimizer will balance cheap, low-calorie foods with more nutrient-rich or calorie-dense options to satisfy all requirements.

Conclusion:
This table provides the foundational data for formulating and solving the diet optimization problem. The optimizer will use these values to determine the optimal quantities of each food to include in the diet, ensuring nutritional adequacy at minimal cost.

1.2 Step 2: Basic LP Model (Minimize Cost with Nutrition Constraints)

[10]:
```
# Create model
model = gp.Model("BasicDiet")

# Filter out constraint rows and get clean food data
foods_df_clean = foods_df.dropna(subset=['Foods'])
foods = foods_df_clean['Foods']
cost = dict(zip(foods, foods_df_clean['Price/ Serving']))
food_vars = model.addVars(foods, name="Servings", lb=0)

# Objective: minimize cost
model.setObjective(gp.quicksum(cost[f] * food_vars[f] for f in foods), GRB.
 ↪MINIMIZE)

# Extract constraint data from the foods DataFrame
# Row 65 has minimum values, Row 66 has maximum values
min_constraints = foods_df.iloc[65]  # Minimum daily intake row
max_constraints = foods_df.iloc[66]  # Maximum daily intake row

# Get nutrient column names (excluding non-nutrient columns)
```

```
nutrient_cols = [col for col in foods_df.columns if col not in ['Foods', 'Price/
 ↪ Serving', 'Serving Size']]

# Add nutrition constraints
for nutrient in nutrient_cols:
    min_val = min_constraints[nutrient]
    max_val = max_constraints[nutrient]

    # Create constraint coefficients
    coef = dict(zip(foods, foods_df_clean[nutrient]))

    # Add minimum constraint
    model.addConstr(gp.quicksum(coef[f] * food_vars[f] for f in foods) >=␣
 ↪min_val, f"{nutrient}_min")

    # Add maximum constraint
    model.addConstr(gp.quicksum(coef[f] * food_vars[f] for f in foods) <=␣
 ↪max_val, f"{nutrient}_max")

model.optimize()

# Print results
if model.status == GRB.OPTIMAL:
    print(f"Optimal cost: ${model.objVal:.2f}")
    print("\nOptimal diet:")
    for f in foods:
        if food_vars[f].x > 0.001:  # Only show foods with significant␣
 ↪quantities
            print(f"{f}: {food_vars[f].x:.3f} servings")
```

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[x86] - Darwin 22.6.0
22H625)

CPU model: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 22 rows, 64 columns and 1194 nonzeros
Model fingerprint: 0xcae440ed
Coefficient statistics:
  Matrix range     [1e-01, 2e+04]
  Objective range  [2e-02, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+01, 1e+04]
Presolve removed 11 rows and 0 columns
Presolve time: 0.03s
Presolved: 11 rows, 75 columns, 608 nonzeros
```

```
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   1.384375e+02   0.000000e+00      0s
      12    4.3371168e+00   0.000000e+00   0.000000e+00      0s

Solved in 12 iterations and 0.04 seconds (0.00 work units)
Optimal objective  4.337116810e+00
Optimal cost: $4.34

Optimal diet:
Frozen Broccoli: 0.260 servings
Celery, Raw: 52.644 servings
Lettuce,Iceberg,Raw: 63.989 servings
Oranges: 2.293 servings
Poached Eggs: 0.142 servings
Popcorn,Air-Popped: 13.869 servings
```

1.3 Step 3: Results of Basic Model

[11]:
```python
if model.status == GRB.OPTIMAL:
    print(f"Optimal Cost: ${model.ObjVal:.2f}")
    for f in foods:
        if food_vars[f].X > 1e-4:
            print(f"{f}: {food_vars[f].X:.2f} servings")
```

```
Optimal Cost: $4.34
Frozen Broccoli: 0.26 servings
Celery, Raw: 52.64 servings
Lettuce,Iceberg,Raw: 63.99 servings
Oranges: 2.29 servings
Poached Eggs: 0.14 servings
Popcorn,Air-Popped: 13.87 servings
```

1.3.1 Interpretation of Basic LP Model Output for the Diet Problem

The output summarizes the results of solving the basic linear programming (LP) model for the diet optimization problem using Gurobi.

Key Points:

- **Model Structure:**
  - 22 constraints (rows): These represent nutritional requirements (minimums and/or maximums for nutrients).
  - 64 variables (columns): Each variable corresponds to the number of servings of a particular food.
  - 1194 nonzero coefficients: Indicates the complexity and density of the constraint matrix.
- **Optimization Process:**
  - Presolve reduced the problem size by removing redundant constraints, making the model more efficient.
  - The solver found the optimal solution in 12 iterations and 0.04 seconds.

- **Solution:**
  - **Optimal Cost:** The minimum cost to meet all nutritional requirements is **$4.34 per day**.
  - **Optimal Diet:** The solution specifies the exact servings of each food to include in the diet. For example:
    * 0.26 servings of Frozen Broccoli
    * 52.64 servings of Celery, Raw
    * 63.99 servings of Lettuce, Iceberg, Raw
    * 2.29 servings of Oranges
    * 0.14 servings of Poached Eggs
    * 13.87 servings of Popcorn, Air-Popped

Implications:

- The optimizer selects mostly very low-cost foods (like celery and lettuce) in large quantities to minimize cost, while including small amounts of other foods to meet specific nutrient requirements.
- The resulting diet, while cost-effective, may not be practical or palatable due to the extremely high servings of certain foods.
- This highlights a limitation of the basic LP model: without additional constraints (e.g., upper bounds on servings, variety, or palatability), the solution may not be realistic for actual consumption.

Conclusion:
The basic LP model successfully finds the lowest-cost diet that meets all nutritional requirements, but further refinements are needed to ensure the solution is practical and acceptable for real-world use.

1.4 Step 4: Extended Model with Binary Logic and Variety Constraints

[29]:
```
model2 = gp.Model("ExtendedDiet")

servings = model2.addVars(foods, name="Servings", lb=0)
chosen = model2.addVars(foods, vtype=GRB.BINARY, name="Chosen")
model2.setObjective(gp.quicksum(cost[f] * servings[f] for f in foods), GRB.
 ↪MINIMIZE)

# Extract constraint data from foods DataFrame
min_constraints = foods_df.iloc[65]
max_constraints = foods_df.iloc[66]
nutrient_cols = [col for col in foods_df.columns if col not in ['Foods', 'Price/
 ↪ Serving', 'Serving Size']]

# Nutrition constraints
for nutrient in nutrient_cols:
    min_val = min_constraints[nutrient]
    max_val = max_constraints[nutrient]
    coef = dict(zip(foods, foods_df_clean[nutrient]))
```

```
    model2.addConstr(gp.quicksum(coef[f] * servings[f] for f in foods) >=␣
 ↪min_val, f"{nutrient}_min")
    model2.addConstr(gp.quicksum(coef[f] * servings[f] for f in foods) <=␣
 ↪max_val, f"{nutrient}_max")

# Linking constraints (minimum 0.1 serving if chosen, food-specific upper␣
 ↪bounds)
celery_max = 30
lettuce_max = 30
default_max = 40
for f in foods:
    model2.addConstr(servings[f] >= 0.1 * chosen[f], f"{f}_minlink")
    if f == "Celery, Raw":
        model2.addConstr(servings[f] <= celery_max * chosen[f], f"{f}_maxlink")
    elif f == "Lettuce,Iceberg,Raw":
        model2.addConstr(servings[f] <= lettuce_max * chosen[f], f"{f}_maxlink")
    else:
        model2.addConstr(servings[f] <= default_max * chosen[f], f"{f}_maxlink")

# Celery/Broccoli constraint (at most one)
celery_food = "Celery, Raw"
broccoli_food = "Frozen Broccoli"
if celery_food in foods.values and broccoli_food in foods.values:
    model2.addConstr(chosen[celery_food] + chosen[broccoli_food] <= 1,␣
 ↪"celery_broccoli_limit")

# Protein variety constraint (at least 3 sources)
protein_keywords = ["beef", "chicken", "egg", "ham", "fish", "turkey", "pork"]
protein_sources = [f for f in foods if any(p in f.lower() for p in␣
 ↪protein_keywords)]
print(f"Found protein sources: {protein_sources}")

if len(protein_sources) >= 3:
    model2.addConstr(gp.quicksum(chosen[f] for f in protein_sources) >= 3,␣
 ↪"protein_variety")
else:
    print("Warning: Less than 3 protein sources found. Relaxing constraint.")

# Solver parameters
model2.setParam('OutputFlag', 1)
model2.setParam('MIPGap', 0.01)
model2.setParam('TimeLimit', 300)

model2.optimize()

# Enhanced result reporting
```

```python
if model2.status == GRB.OPTIMAL:
    print(f"\nOptimal cost: ${model2.objVal:.2f}")
    print(f"Number of foods chosen: {sum(1 for f in foods if chosen[f].x > 0.
 ↪5)}")
    print("\nOptimal diet:")
    total_servings = 0
    for f in foods:
        if chosen[f].x > 0.5:
            servings_val = servings[f].x
            total_servings += servings_val
            print(f"{f}: {servings_val:.3f} servings")
    print(f"\nTotal servings per day: {total_servings:.1f}")

elif model2.status == GRB.INFEASIBLE:
    print("Model is infeasible")
    model2.computeIIS()
    print("Conflicting constraints:")
    for c in model2.getConstrs():
        if c.IISConstr:
            print(f"  {c.constrName}")
```

```
Found protein sources: ['Roasted Chicken', 'Poached Eggs', 'Scrambled Eggs',
'Bologna,Turkey', 'Frankfurter, Beef', 'Ham,Sliced,Extralean', 'Hamburger
W/Toppings', 'Pork', 'Splt Pea&Hamsoup', 'Vegetbeef Soup']
Set parameter OutputFlag to value 1
Set parameter MIPGap to value 0.01
Set parameter TimeLimit to value 300
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[x86] - Darwin 22.6.0
22H625)

CPU model: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Non-default parameters:
TimeLimit  300
MIPGap  0.01

Optimize a model with 152 rows, 128 columns and 1462 nonzeros
Model fingerprint: 0x7ec2b93a
Variable types: 64 continuous, 64 integer (64 binary)
Coefficient statistics:
  Matrix range     [1e-01, 2e+04]
  Objective range  [2e-02, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
Presolve removed 3 rows and 0 columns
Presolve time: 0.02s
Presolved: 149 rows, 128 columns, 1321 nonzeros
```

```
Variable types: 64 continuous, 64 integer (64 binary)

Root relaxation: objective 1.942235e+01, 41 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   19.42235    0    2          -   19.42235      -     -    0s
H    0     0                      19.6834328   19.42235  1.33%     -    0s
H    0     0                      19.6355985   19.42235  1.09%     -    0s

Explored 1 nodes (41 simplex iterations) in 0.14 seconds (0.01 work units)
Thread count was 8 (of 8 available processors)

Solution count 2: 19.6356 19.6834

Optimal solution found (tolerance 1.00e-02)
Best objective 1.963559852854e+01, best bound 1.963559852854e+01, gap 0.0000%

Optimal cost: $19.64
Number of foods chosen: 11

Optimal diet:
Celery, Raw: 30.000 servings
Lettuce,Iceberg,Raw: 30.000 servings
Peppers, Sweet, Raw: 3.128 servings
Tofu: 1.711 servings
Roasted Chicken: 0.100 servings
Tomato,Red,Ripe,Raw: 0.494 servings
Kiwifruit,Raw,Fresh: 30.630 servings
Poached Eggs: 0.100 servings
Bologna,Turkey: 0.100 servings
Peanut Butter: 2.577 servings
Beanbacn Soup,W/Watr: 0.325 servings

Total servings per day: 99.2
```

1.5 Step 5: Results of Extended Model

[32]:
```python
if model2.status == GRB.OPTIMAL:
    print(f"Optimal Cost (Extended): ${model2.ObjVal:.2f}")
    for f in foods:
        if servings[f].X > 1e-4:
            print(f"{f}: {servings[f].X:.2f} servings")
```

```
Optimal Cost (Extended): $19.64
Celery, Raw: 30.00 servings
```

```
Lettuce,Iceberg,Raw: 30.00 servings
Peppers, Sweet, Raw: 3.13 servings
Tofu: 1.71 servings
Roasted Chicken: 0.10 servings
Tomato,Red,Ripe,Raw: 0.49 servings
Kiwifruit,Raw,Fresh: 30.63 servings
Poached Eggs: 0.10 servings
Bologna,Turkey: 0.10 servings
Peanut Butter: 2.58 servings
Beanbacn Soup,W/Watr: 0.33 servings
```

1.5.1 Interpretation of Extended Model Output with Binary Logic and Variety Constraints

This output summarizes the results of solving an extended mixed-integer programming (MIP) model for the diet problem, incorporating binary variables and variety constraints.

Key Points:

- **Model Structure:**
  - 152 constraints (rows): More constraints than the basic model, reflecting additional requirements (e.g., variety, minimum/maximum servings, binary logic).
  - 128 variables (columns): 64 continuous (servings), 64 binary (whether a food is chosen).
  - 1462 nonzero coefficients: Increased complexity due to new constraints.
  - Binary variables ensure that if a food is chosen, it must be included at a minimum serving size, and variety constraints limit the number of different foods selected.
- **Optimization Process:**
  - MIPGap set to 0.01 (1% optimality gap), TimeLimit set to 300 seconds.
  - The solver found the optimal solution quickly, with a final cost of **$19.64 per day**.
  - 11 different foods were selected, ensuring dietary variety.
- **Solution:**
  - **Optimal Cost:** $19.64 per day, higher than the basic LP due to variety and minimum serving constraints.
  - **Optimal Diet:** The solution includes a mix of vegetables, protein sources, and other foods, e.g.:
    - 30 servings each of Celery, Raw and Lettuce, Iceberg, Raw
    - 3.13 servings of Peppers, Sweet, Raw
    - 1.71 servings of Tofu
    - 0.1 servings each of Roasted Chicken, Poached Eggs, Bologna, Turkey
    - 2.58 servings of Peanut Butter
    - 30.63 servings of Kiwifruit, Raw, Fresh
    - Other foods in smaller amounts
- **Implications:**
  - The model enforces a more realistic and varied diet by limiting the number of foods and requiring minimum servings for selected foods.
  - The cost increases compared to the basic LP, but the solution is more practical and palatable.
  - The inclusion of multiple protein sources and a variety of fruits and vegetables reflects a balanced approach.

Conclusion:
The extended model produces a more realistic and diverse diet plan at a higher cost, demonstrating the trade-off between cost minimization and dietary variety/palatability in diet optimization problems.