

0. Imports and Setting up Anthropic API Client

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

Mounted at /content/drive

```
!pip install python-dotenv
```

```
import os
import dotenv
```

```
dotenv.load_dotenv('/content/drive/MyDrive/.env')
```

Collecting python-dotenv
 Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
 Installing collected packages: python-dotenv
 Successfully installed python-dotenv-1.0.1
 True

```
# Load Prompts and Problem Description
```

```
prompt1_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt1_MathematicalModel.txt'
```

```
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
```

```
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/IP/IP1.txt'
```

```
prompt1_file = open(prompt1_path, "r")
```

```
prompt2_file = open(prompt2_path, "r")
```

```
problem_desc_file = open(problem_desc_path, "r")
```

```
prompt1 = prompt1_file.read()
```

```
print("Prompt 1:\n", prompt1)
```

```
prompt2 = prompt2_file.read()
```

```
print("Prompt 2:\n", prompt2)
```

```
problem_desc = problem_desc_file.read()
```

```
print("Problem Description:\n", problem_desc)
```

hematical optimization model for this problem. Include parameters, decision variables, the objective function and the constraints. Use the following Python pyomo code for this optimization problem. The data needed for the model is provided in the file named 'data.csv'. Use sample data.

:

o welcome you, our newest intern on the Analytics team of Massachusetts General Hospital! You have been placed in a challenge for determining the best treatment plan for 17 patients who require radiotherapy. Your goal is to optimize the use of treatment course of action for each patient, you will use a scoring system called the Biological Equivalent Dose (BED). This system is used to evaluate the effectiveness of different treatment plans for patients who need radiotherapy. Each patient i needs 15 fractions, which can be photon fractions, proton fractions, or a mix of both. The file 'patientData.csv' contains a 2D matrix of BED scores. It does not have an index. It was made in Excel and saved as csv. The total maximal capacity C is 100 proton fractions. To maximize the total BED scores for all the patients, which patients should be treated with proton fractions?

```
!pip install anthropic
```

Show hidden output

```
# Importing Anthropic & Setting Headers
```

```
import anthropic
```

```
client = anthropic.Anthropic(
```

```
    # defaults to os.environ.get("ANTHROPIC_API_KEY")
```

```
    api_key=os.environ.get("ANTHROPIC_API_KEY"),
```

```
)
```

1. Prompt 1: Create Mathematical Model

```

message = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt1,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)

```

```
response_p1 = message.content[0].text
```

```

# Print response
print(response_p1)

```

Thank you for the warm welcome and the challenging problem! I'm excited to work on this project and contribute to improving patient care.

To solve this problem, I'll formulate an integer linear optimization model. First, let's define the parameters, decision variables, and constraints.

Parameters:

- n : number of patients ($n = 17$)
- C : total maximal capacity of proton fractions ($C = 100$)
- $BED_i(j, 15-j)$: BED score for patient i when receiving j proton fractions and $15-j$ photon fractions

Decision Variables:

- x_{ij} : binary variable, 1 if patient i receives j proton fractions, 0 otherwise ($i = 1, 2, \dots, n$; $j = 0, 1, \dots, 15$)

Objective Function:

Maximize the total BED scores for all patients:

$$\max \sum_{i=1}^n \sum_{j=0}^{15} BED_i(j, 15-j) * x_{ij}$$

Constraints:

- Each patient must receive exactly 15 fractions in total:
$$\sum_{j=0}^{15} x_{ij} = 1, \text{ for all } i = 1, 2, \dots, n$$
- The total number of proton fractions used must not exceed the maximal capacity C :
$$\sum_{i=1}^n \sum_{j=0}^{15} j * x_{ij} \leq C$$
- Binary constraint for decision variables:
$$x_{ij} \in \{0, 1\}, \text{ for all } i = 1, 2, \dots, n; j = 0, 1, \dots, 15$$

The complete integer linear optimization model is as follows:

$$\max \sum_{i=1}^n \sum_{j=0}^{15} BED_i(j, 15-j) * x_{ij}$$

subject to:

- $\sum_{j=0}^{15} x_{ij} = 1, \text{ for all } i = 1, 2, \dots, n$
- $\sum_{i=1}^n \sum_{j=0}^{15} j * x_{ij} \leq C$
- $x_{ij} \in \{0, 1\}, \text{ for all } i = 1, 2, \dots, n; j = 0, 1, \dots, 15$

By solving this optimization model, we can determine which patients should receive proton fractions and how many each patient should receive.

2. Prompt 2: Write Code based on Mathematical Model

```

message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": response_p1}
    ]
)

```

```
response_p2 = message2.content[0].text
```

```

# Print response
print(response_p2)

```

Here's a Python code using Pyomo to solve the optimization problem you described. I'll use sample data for the BED scores.

```

```python
from pyomo.environ import *

Sample data
n = 17 # Number of patients
C = 100 # Total maximal capacity of proton fractions

Sample BED scores (replace with actual data)
BED = {
 (1, 0): 50, (1, 1): 55, (1, 2): 60, (1, 3): 65, (1, 4): 70, (1, 5): 75, (1, 6): 80, (1, 7): 85, (1, 8): 90, (1, 9): 95,
 (2, 0): 60, (2, 1): 65, (2, 2): 70, (2, 3): 75, (2, 4): 80, (2, 5): 85, (2, 6): 90, (2, 7): 95, (2, 8): 100, (2, 9): 105,
 # ... add sample data for the remaining patients
}

```

```

Create the model
model = ConcreteModel()

Define sets
model.I = RangeSet(1, n) # Set of patients
model.J = RangeSet(0, 15) # Set of proton fractions

Define decision variables
model.x = Var(model.I, model.J, domain=Binary)

Define objective function
def obj_rule(model):
 return sum(BED[i, j] * model.x[i, j] for i in model.I for j in model.J)
model.obj = Objective(rule=obj_rule, sense=maximize)

Define constraints
def fraction_constraint_rule(model, i):
 return sum(model.x[i, j] for j in model.J) == 1
model.fraction_constraint = Constraint(model.I, rule=fraction_constraint_rule)

def capacity_constraint_rule(model):
 return sum(j * model.x[i, j] for i in model.I for j in model.J) <= C
model.capacity_constraint = Constraint(rule=capacity_constraint_rule)

Solve the model
solver = SolverFactory('glpk')
results = solver.solve(model)

Print the results
print("Optimal solution found with objective value:", model.obj())
print("\nPatient allocation:")
for i in model.I:
 for j in model.J:
 if model.x[i, j]() == 1:
 print(f"Patient {i} receives {j} proton fractions and {15-j} photon fractions")
 ...

In this code:
- I used sample data for the BED scores, represented by the `BED` dictionary. Replace this with the actual BED scores
- The `n` and `C` parameters are set to the values you provided (17 patients and 100 proton fractions).
- The decision variables `x[i, j]` are defined as binary variables.
 The objective function maximizes the total BED scores for all patients

```

## 4. Input Problem Data and Test Model Code

```

Installing pyomo and solver
!pip install -q pyomo
!pip install pandas
!apt-get install -y -qq glpk-utils
!pip install glpk

```

 [Show hidden output](#)

```

from pyomo.environ import *

Sample data
n = 17 # Number of patients
C = 100 # Total maximal capacity of proton fractions

Sample BED scores (replace with actual data)
Read the problem data from the CSV file
data = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/IP/IP1/ProblemData.csv", header=None)
data = pd.DataFrame(data)

BED = {(i + 1, j): value for i, row in data.iterrows() for j, value in enumerate(row)}

Create the model
model = ConcreteModel()

Define sets
model.I = RangeSet(1, n) # Set of patients
model.J = RangeSet(0, 15) # Set of proton fractions

Define decision variables
model.x = Var(model.I, model.J, domain=Binary)

Define objective function
def obj_rule(model):
 return sum(BED[i, j] * model.x[i, j] for i in model.I for j in model.J)
model.obj = Objective(rule=obj_rule, sense=maximize)

Define constraints
def fraction_constraint_rule(model, i):
 return sum(model.x[i, j] for j in model.J) == 1
model.fraction_constraint = Constraint(model.I, rule=fraction_constraint_rule)

def capacity_constraint_rule(model):
 return sum(j * model.x[i, j] for i in model.I for j in model.J) <= C
model.capacity_constraint = Constraint(rule=capacity_constraint_rule)


Solve the model
solver = SolverFactory('glpk')

```

## ✓ 5. Correct The Model Code to Test Mathematical Model (if applicable)

```
print("\nPatient allocation:")
```

```
 for i in range(1, n+1):
```

 Optimal solution found with objective value: 8.239999999999998

```

Patient allocation:
Patient 1 receives 8 proton fractions and 7 photon fractions
Patient 2 receives 8 proton fractions and 7 photon fractions
Patient 3 receives 3 proton fractions and 12 photon fractions
Patient 4 receives 0 proton fractions and 15 photon fractions
Patient 5 receives 5 proton fractions and 10 photon fractions
Patient 6 receives 0 proton fractions and 15 photon fractions
Patient 7 receives 4 proton fractions and 11 photon fractions
Patient 8 receives 15 proton fractions and 0 photon fractions
Patient 9 receives 4 proton fractions and 11 photon fractions
Patient 10 receives 5 proton fractions and 10 photon fractions
Patient 11 receives 6 proton fractions and 9 photon fractions
Patient 12 receives 0 proton fractions and 15 photon fractions
Patient 13 receives 10 proton fractions and 5 photon fractions
Patient 14 receives 0 proton fractions and 15 photon fractions
Patient 15 receives 10 proton fractions and 5 photon fractions
Patient 16 receives 10 proton fractions and 5 photon fractions
Patient 17 receives 12 proton fractions and 3 photon fractions

```