

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
```

```
# Variables Prompt
```

```
prompt11_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt11_MathematicalModel.txt'
```

```
# Objective Prompt
```

```
prompt12_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt12_MathematicalModel.txt'
```

```
# Constraint Prompt
```

```
prompt13_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt13_MathematicalModel.txt'
```

```
# Code Prompt
```

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

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

```
prompt11_file = open(prompt11_path, "r")
prompt12_file = open(prompt12_path, "r")
prompt13_file = open(prompt13_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")
```

```
prompt11 = prompt11_file.read()
print("Prompt 1.1 (Variables):\n", prompt11)
```

```
prompt12 = prompt12_file.read()
print("Prompt 1.2 (Objective):\n", prompt12)
```

```
prompt13 = prompt13_file.read()
print("Prompt 1.3 (Constraints):\n", prompt13)
```

```
prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)
```

```
problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

Prompt 1.1 (Variables):
 Please formulate only the variables for this mathematical optimization problem.
 Prompt 1.2 (Objective):
 Please formulate only the objective function for this mathematical optimization problem.
 Prompt 1.3 (Constraints):
 Please formulate only the constraints for this mathematical optimization problem.
 Prompt 2:
 Please write a python pyomo code for this optimization problem.
 Use sample data where needed.
 Indicate where you use sample data.
 Problem Description:
 We are delighted to welcome you, our newest intern on the Analytics team of Massachusetts General Hospital! You have been assigned to the role of a data analyst. You are responsible for determining the best treatment plan for 17 patients who require radiotherapy. Your goal is to optimize the treatment plan. To determine the best course of action for each patient, you will use a scoring system called the Biological Equivalent Dose (BED). We have $n=17$ patients who need radiotherapy. Each patient i needs 15 fractions, which can be photon fractions, proton fractions, or a combination of both. The data file "ProblemData.csv" contains a 2D matrix of BED scores. It does not have an index. It was made in Excel and is located in the "Data" folder. Suppose that the total maximal capacity C is 100 proton fractions. To maximize the total BED scores for all the patients

```
!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 1.1: Create Variables for Mathematical Model

```
message11 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt11,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)
```

```
response_p11 = message11.content[0].text
```

```
# Print response
```

```
print(response_p11)
```

 Great! Let's formulate the variables for this integer linear optimization problem.

Decision Variables:

Let x_{ij} be a binary variable that indicates whether patient i receives j proton fractions.

– $x_{ij} = 1$ if patient i receives j proton fractions (and $15-j$ photon fractions)

– $x_{ij} = 0$ otherwise

where:

$i = 1, 2, \dots, 17$ (representing the 17 patients)

$j = 0, 1, 2, \dots, 15$ (representing the possible number of proton fractions)

These variables will help determine the optimal treatment plan for each patient, specifying the number of proton and pho


✓ 1. Prompt 1.2: Create Objective for Mathematical Model

```
message12 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt12,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11)}
    ]
)
```

```
response_p12 = message12.content[0].text
```

```
# Print response
```

```
print(response_p12)
```

 Objective Function:
Maximize the total BED scores for all patients:

Maximize: $\sum_{i=1}^{17} \sum_{j=0}^{15} \text{BED}_i(j, 15-j) * x_{ij}$

The objective function aims to maximize the sum of the BED scores for all patients, considering the number of proton fra

✓ 1. Prompt 1.3: Create Constraints for Mathematical Model

```

message13 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt13,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11 + response_p12)}
    ]
)

response_p13 = message13.content[0].text

# Print response
print(response_p13)

```

→ Constraints:

1. Each patient must receive exactly 15 fractions in total (either proton or photon):
 $\sum_{j=0 \text{ to } 15} x_{ij} = 1$, for each patient i

This constraint ensures that each patient receives a total of 15 fractions, and only one treatment plan (combination

2. The total number of proton fractions used across all patients cannot exceed the maximum capacity C :
 $\sum_{i=1 \text{ to } 17} \sum_{j=0 \text{ to } 15} j * x_{ij} \leq C$

This constraint limits the total number of proton fractions used across all patients to be within the maximum capacity

3. Binary variable constraint:
 $x_{ij} \in \{0, 1\}$, for all patients i and proton fractions j

This constraint ensures that the decision variables x_{ij} can only take binary values (0 or 1), indicating whether a pa

These constraints, along with the objective function and decision variables, complete the formulation of the integer lin

✓ 2. Prompt 2: Write Code based on Mathematical Model

```

message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11 + response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text

# Print response
print(response_p2)

```

→ Here's the Python Pyomo code for the optimization problem:

```

```python
from pyomo.environ import *

Sample data (replace with actual data)
num_patients = 17
max_fractions = 15
max_proton_capacity = 100

Sample BED scores (replace with actual BED scores)
BED_scores = {
 (1, 0): 10.0, (1, 1): 11.5, (1, 2): 13.0, ..., (1, 15): 25.0,
 (2, 0): 9.5, (2, 1): 11.0, (2, 2): 12.5, ..., (2, 15): 24.5,
 ...,
 (17, 0): 11.0, (17, 1): 12.5, (17, 2): 14.0, ..., (17, 15): 26.0
}

Create the Pyomo model
model = ConcreteModel()

Define the decision variables
model.x = Var(range(1, num_patients + 1), range(max_fractions + 1), within=Binary)

Define the objective function
def obj_rule(model):
 return sum(BED_scores[i, j] * model.x[i, j] for i in range(1, num_patients + 1) for j in range(max_fractions + 1))
model.obj = Objective(rule=obj_rule, sense=maximize)

Constraint: Each patient must receive exactly 15 fractions in total
def fraction_constraint_rule(model, i):
 return sum(model.x[i, j] for j in range(max_fractions + 1)) == 1
model.fraction_constraint = Constraint(range(1, num_patients + 1), rule=fraction_constraint_rule)

```

```

Constraint: Total number of proton fractions used cannot exceed the maximum capacity
def proton_capacity_constraint_rule(model):
 return sum(j * model.x[i, j] for i in range(1, num_patients + 1) for j in range(max_fractions + 1)) <= max_proton_ca
model.proton_capacity_constraint = Constraint(rule=proton_capacity_constraint_rule)

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

Print the optimal solution
print("Optimal treatment plan:")
for i in range(1, num_patients + 1):
 for j in range(max_fractions + 1):
 if value(model.x[i, j]) == 1:
 print(f"Patient {i}: {j} proton fractions, {max_fractions - j} photon fractions")

print(f"Total BED score: {value(model.obj)}")

```

In this code, I used sample data for the number of patients, maximum fractions, maximum proton capacity, and BED scores.

The code creates a Pyomo model and defines the decision variables `x[i, j]` as binary variables. The objective function

maximizes the total BED score, which is the sum of the BED scores for each patient, weighted by the number of proton fractions used.

## 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 *
import pandas as pd

Sample data (replace with actual data)
num_patients = 17
max_fractions = 15
max_proton_capacity = 100

Sample BED scores (replace with actual BED scores)
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_scores = {(i + 1, j): value for i, row in data.iterrows() for j, value in enumerate(row)} # ADJUSTED TO CORRECTLY LOAD TH

Create the Pyomo model
model = ConcreteModel()

Define the decision variables
model.x = Var(range(1, num_patients + 1), range(max_fractions + 1), within=Binary)

Define the objective function
def obj_rule(model):
 return sum(BED_scores[i, j] * model.x[i, j] for i in range(1, num_patients + 1) for j in range(max_fractions + 1))
model.obj = Objective(rule=obj_rule, sense=maximize)

Constraint: Each patient must receive exactly 15 fractions in total
def fraction_constraint_rule(model, i):
 return sum(model.x[i, j] for j in range(max_fractions + 1)) == 15
model.fraction_constraint = Constraint(range(1, num_patients + 1), rule=fraction_constraint_rule)

Constraint: Total number of proton fractions used cannot exceed the maximum capacity
def proton_capacity_constraint_rule(model):
 return sum(j * model.x[i, j] for i in range(1, num_patients + 1) for j in range(max_fractions + 1)) <= max_proton_capacit
model.proton_capacity_constraint = Constraint(rule=proton_capacity_constraint_rule)

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

Print the optimal solution
print("Optimal treatment plan:")
for i in range(1, num_patients + 1):
 for j in range(max_fractions + 1):
 if value(model.x[i, j]) == 1:
 print(f"Patient {i}: {j} proton fractions, {max_fractions - j} photon fractions")

```

```
print(f"Total BED score: {value(model.obj)}")
```

```
↗ Optimal treatment plan:
Patient 1: 8 proton fractions, 7 photon fractions
Patient 2: 8 proton fractions, 7 photon fractions
Patient 3: 3 proton fractions, 12 photon fractions
Patient 4: 0 proton fractions, 15 photon fractions
Patient 5: 5 proton fractions, 10 photon fractions
Patient 6: 0 proton fractions, 15 photon fractions
Patient 7: 4 proton fractions, 11 photon fractions
Patient 8: 15 proton fractions, 0 photon fractions
Patient 9: 4 proton fractions, 11 photon fractions
Patient 10: 5 proton fractions, 10 photon fractions
Patient 11: 6 proton fractions, 9 photon fractions
Patient 12: 0 proton fractions, 15 photon fractions
Patient 13: 10 proton fractions, 5 photon fractions
Patient 14: 0 proton fractions, 15 photon fractions
Patient 15: 10 proton fractions, 5 photon fractions
Patient 16: 10 proton fractions, 5 photon fractions
Patient 17: 12 proton fractions, 3 photon fractions
Total BED score: 8.239999999999998
```

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