

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)
```

 To formulate the integer linear optimization model for this problem, I will define the decision variables, objective fun

Decision Variables:

Let $x[i,j]$ be a binary variable that indicates whether patient i receives j proton fractions.

– i ranges from 1 to 17 (the number of patients)

– j ranges from 0 to 15 (the possible number of proton fractions per patient)

$x[i,j] = 1$ if patient i receives j proton fractions, and 0 otherwise.


These are the variables that need to be determined by the optimization model to maximize the total BED scores while sati

✓ 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:

$$\sum_{i=1}^{17} \sum_{j=0}^{15} \text{BED}[i,j] * x[i,j]$$

This objective function sums up the BED scores for each patient based on the number of proton fractions they receive, as

I have formulated the objective function for the integer linear optimization model. The constraints would still need to

✓ 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 a total of 15 fractions (proton + photon):

For each patient i ($1 \leq i \leq 17$):

$$\sum_{j=0}^{15} x[i,j] = 1$$

This constraint ensures that for each patient i , exactly one decision variable $x[i,j]$ is equal to 1, meaning that the

2. The total number of proton fractions used must not exceed the capacity C (100):

$$\sum_{i=1}^{17} \sum_{j=0}^{15} j * x[i,j] \leq 100$$

This constraint calculates the total number of proton fractions used by summing up the product of the number of proto

3. Binary constraint on the decision variables:

$x[i,j] \in \{0, 1\}$ for all i ($1 \leq i \leq 17$) and j ($0 \leq j \leq 15$)

This constraint ensures that the decision variables $x[i,j]$ can only take the values 0 or 1, indicating whether a pati

These constraints, along with the objective function you provided earlier, complete the formulation of the integer linea

✓ 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 code using Pyomo to formulate and solve the integer linear optimization problem:

```
```python
from pyomo.environ import *

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

Sample BED scores (replace with actual data)
BED = {(i, j): i * j for i in range(1, num_patients + 1) for j in range(max_fractions + 1)}

Create a Pyomo model
model = ConcreteModel()

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

```

Define objective function
def obj_rule(model):
 return sum(BED[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)

Define constraints
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)

def 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)) <= capacity
model.capacity_constraint = Constraint(rule=capacity_constraint_rule)

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

Print the results
print("Objective value:", model.obj())
for i in range(1, num_patients + 1):
 for j in range(max_fractions + 1):
 if model.x[i, j]() == 1:
 print(f"Patient {i} receives {j} proton fractions")
 ...

```

In this code:

1. We import the necessary components from Pyomo.
2. We define the sample data:
  - `num\_patients`: The number of patients (17 in this case).
  - `max\_fractions`: The maximum number of fractions per patient (15 in this case).
  - `capacity`: The total capacity of proton fractions (100 in this case).
3. We create a sample BED scores dictionary using a simple formula `i \* j`. Replace this with the actual BED scores

## 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
capacity = 100

Sample BED scores (replace with actual data)
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)} # ADJUSTED TO CORRECTLY LOAD THE DATA

Create a Pyomo model
model = ConcreteModel()

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

Define objective function
def obj_rule(model):
 return sum(BED[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)

Define constraints
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)

def 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)) <= capacity
model.capacity_constraint = Constraint(rule=capacity_constraint_rule)

```

```
model.capacity_constraint = constraint(name=capacity_constraint_name,
```

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

Print the results
print("Objective value:", model.obj())
for i in range(1, num_patients + 1):
 for j in range(max_fractions + 1):
 if model.x[i, j]() == 1:
 print(f"Patient {i} receives {j} proton fractions")
```

```
↩ Objective value: 8.239999999999998
Patient 1 receives 8 proton fractions
Patient 2 receives 8 proton fractions
Patient 3 receives 3 proton fractions
Patient 4 receives 0 proton fractions
Patient 5 receives 5 proton fractions
Patient 6 receives 0 proton fractions
Patient 7 receives 4 proton fractions
Patient 8 receives 15 proton fractions
Patient 9 receives 4 proton fractions
Patient 10 receives 5 proton fractions
Patient 11 receives 6 proton fractions
Patient 12 receives 0 proton fractions
Patient 13 receives 10 proton fractions
Patient 14 receives 0 proton fractions
Patient 15 receives 10 proton fractions
Patient 16 receives 10 proton fractions
Patient 17 receives 12 proton fractions
```

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