

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/MIP/MIP4.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:
You are task with maximizing the water flow in a network of pipes over 10 time steps. The water enters the system at po

These are the pipe capacities:
AB: 3
AC: 6
AF: 1
BC: 3
BD: 10
BE: 4
CE: 4
CF: 4
DE: 5
EG: 5
```

FB: 12
FG: 7

```
!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)
```

 Here are the variables for the mathematical optimization problem:


1. Decision variables:
 - x_{ij} : binary variable indicating if pipe ij is activated (1) or deactivated (0), for ij in {AB, AC, AF, BC, BD, BE, CE, CF, DE, EG, FB, FG}
 - $f_{ij}(t)$: continuous variable representing the flow in pipe ij at time step t , for ij in {AB, AC, AF, BC, BD, BE, CE, CF, DE, EG, FB, FG}
 - $r_i(t)$: continuous variable representing the amount of water released from point i at time step t , for i in {A, B, C, D, E, F, G}
2. Input parameters:
 - c_{ij} : capacity of pipe ij , for ij in {AB, AC, AF, BC, BD, BE, CE, CF, DE, EG, FB, FG}
 - $f(t)$: inflow function at point A at time step t , where $f(t) = \max(-4/30t^3 + t^2 - 0.234t + 3, 0)$ and t in {1, 2, ..., 10}

✓ 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)
```

 Here is the formulation of the objective function for the given mathematical optimization problem:

Maximize: $\sum_{t=1}^{10} (f_{EG}(t) + f_{FG}(t) + r_G(t))$

The objective is to maximize the total amount of water that reaches point G over the 10 time steps. This can be achieved

In mathematical notation, the objective function can be written as:

Maximize: $\sum_{t=1}^{10} [f_{EG}(t) + f_{FG}(t) + r_G(t)]$

This objective function is subject to the constraints mentioned in the problem description, such as flow conservation at

✓ 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)
```

↗ Here are the constraints for the given mathematical optimization problem:

1. Flow conservation constraints:
For each point i in $\{A, B, C, D, E, F, G\}$ and time step t in $\{1, 2, \dots, 10\}$:
 - Point A: $f(t) = f_{AB}(t) + f_{AC}(t) + f_{AF}(t) + r_A(t)$
 - Point B: $f_{AB}(t-1) + f_{FB}(t-1) = f_{BC}(t) + f_{BD}(t) + r_B(t)$
 - Point C: $f_{AC}(t-1) + f_{BC}(t-1) = f_{CE}(t) + f_{CF}(t) + r_C(t)$
 - Point D: $f_{BD}(t-1) = f_{DE}(t) + r_D(t)$
 - Point E: $f_{BE}(t-1) + f_{CE}(t-1) + f_{DE}(t-1) = f_{EG}(t) + r_E(t)$
 - Point F: $f_{AF}(t-1) + f_{CF}(t-1) = f_{FB}(t) + f_{FG}(t) + r_F(t)$
 - Point G: $f_{EG}(t-1) + f_{FG}(t-1) = r_G(t)$
2. Pipe capacity constraints:
For each pipe ij in $\{AB, AC, AF, BC, BD, BE, CE, CF, DE, EG, FB, FG\}$ and time step t in $\{1, 2, \dots, 10\}$:
 - $f_{ij}(t) \leq c_{ij} * x_{ij}$
3. Pipe activation constraint:
 $\text{Sum}(x_{ij}, \text{ for } ij \text{ in } \{AB, AC, AF, BC, BD, BE, CE, CF, DE, EG, FB, FG\}) = 10$
4. Non-negativity constraints:
 - $f_{ij}(t) \geq 0$, for ij in $\{AB, AC, AF, BC, BD, BE, CE, CF, DE, EG, FB, FG\}$ and t in $\{1, 2, \dots, 10\}$
 - $r_i(t) \geq 0$, for i in $\{A, B, C, D, E, F, G\}$ and t in $\{1, 2, \dots, 10\}$
5. Binary constraints:
 - $x_{ij} \in \{0, 1\}$, for ij in $\{AB, AC, AF, BC, BD, BE, CE, CF, DE, EG, FB, FG\}$

These constraints ensure that:

1. The flow is conserved at each point and time step, considering the inflow, outflow, and released water.
2. The flow in each pipe does not exceed its capacity if the pipe is activated.
3. Exactly 10 pipes are activated.
4. All flow and released water variables are non-negative.
5. The pipe activation variables are binary (0 or 1).

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

In case more than the component of each step was generated, use "last instance" / "newest version" of the components.

Example: Prompt 1.1 (Variables): Model generates only variables Prompt 1.2 (Objective): Model generates objective and constraints Prompt 1.3 (Constraints): Model generates only constraints

Use Variables from step 1.1, use objective from 1.2 and use constraints from 1.3 (since constraints version 1.3 is more recent than constraints version 1.2).

This set up allows for potential self-correction of the model throughout the three prompts.

Correctness of each component is still generated in the respective step, so self-correction is independent of this.

```
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)
```

```

# Input parameters
# Sample data for pipe capacities (replace with actual data if available)
pipe_capacities = {
    'AB': 10, 'AC': 8, 'AF': 12, 'BC': 6, 'BD': 9, 'BE': 7,
    'CE': 5, 'CF': 11, 'DE': 8, 'EG': 10, 'FB': 9, 'FG': 11
}
model.c = Param(model.pipes, initialize=pipe_capacities)

# Inflow function at point A
def inflow(model, t):
    return max(-4/30 * t**3 + t**2 - 0.234 * t + 3, 0)

# Objective function
def objective_rule(model):
    return sum(model.f['EG', t] + model.f['FG', t] + model.r['G', t] for t in model.time_steps)
model.objective = Objective(rule=objective_rule, sense=maximize)

# Constraints
def flow_conservation_rule(model, i, t):
    if i == 'A':
        return inflow(model, t) == model.f['AB', t] + model.f['AC', t] + model.f['AF', t] + model.r['A', t]
    elif i == 'B':
        return model.f['AB', t-1] + model.f['FB', t-1] == model.f['BC', t] + model.f['BD', t] + model.r['B', t]
    elif i == 'C':
        return model.f['AC', t-1] + model.f['BC', t-1] == model.f['CE', t] + model.f['CF', t] + model.r['C', t]
    elif i == 'D':
        return model.f['BD', t-1] == model.f['DE', t] + model.r['D', t]
    elif i == 'E':
        return model.f['BE', t-1] + model.f['CE', t-1] + model.f['DE', t-1] == model.f['EG', t] + model.r['E', t]
    elif i == 'F':
        return model.f['AF', t-1] + model.f['CF', t-1] == model.f['FB', t] + model.f['FG', t] + model.r['F', t]
    elif i == 'G':
        return model.f['EG', t-1] + model.f['FG', t-1] == model.r['G', t]
model.flow_conservation = Constraint(model.points, model.time_steps, rule=flow_conservation_rule)

def pipe_capacity_rule(model, ij, t):
    return model.f[ij, t] <= model.c[ij] * model.x[ij]
model.pipe_capacity = Constraint(model.pipes, model.time_steps, rule=pipe_capacity_rule)

def pipe_activation_rule(model):
    return sum(model.x[ij] for ij in model.pipes) == 10
model.pipe_activation = Constraint(rule=pipe_activation_rule)

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

# Print the results
print(f"Objective value: {model.objective()}")
print("Activated pipes:")
for ij in model.pipes:
    if model.x[ij].value == 1:
        print(ij)
...

In this code, I used sample data for the pipe capacities (`pipe_capacities`) as input parameters. You can replace thes

```

4. Input Problem Data and Test Model Code

```

# Installing pyomo nd 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 *

# Create a Pyomo model
model = ConcreteModel()

# Set of points
model.points = Set(initialize=['A', 'B', 'C', 'D', 'E', 'F', 'G'])

# Set of pipes
model.pipes = Set(initialize=['AB', 'AC', 'AF', 'BC', 'BD', 'BE', 'CE', 'CF', 'DE', 'EG', 'FB', 'FG'])

# Set of time steps
model.time_steps = RangeSet(1, 10)

# Decision variables

```

```

model.x = Var(model.pipes, within=Binary)
model.f = Var(model.pipes, model.time_steps, within=NonNegativeReals)
model.r = Var(model.points, model.time_steps, within=NonNegativeReals)

# Input parameters
# Sample data for pipe capacities (replace with actual data if available)
pipe_capacities = {
    'AB': 3, 'AC': 6, 'AF': 1, 'BC': 3, 'BD': 10, 'BE': 4,
    'CE': 4, 'CF': 4, 'DE': 5, 'EG': 5, 'FB': 12, 'FG': 7
}
model.c = Param(model.pipes, initialize=pipe_capacities)

# Inflow function at point A
def inflow(model, t):
    return max(-4/30 * t**3 + t**2 - 0.234 * t + 3, 0)

# Objective function
def objective_rule(model):
    return sum(model.f['EG', t] + model.f['FG', t] + model.r['G', t] for t in model.time_steps)
model.objective = Objective(rule=objective_rule, sense=maximize)

# Constraints
def flow_conservation_rule(model, i, t):
    if i == 'A':
        return inflow(model, t) == model.f['AB', t] + model.f['AC', t] + model.f['AF', t] + model.r['A', t]
    elif i == 'B':
        return model.f['AB', t-1] + model.f['FB', t-1] == model.f['BC', t] + model.f['BD', t] + model.r['B', t]
    elif i == 'C':
        return model.f['AC', t-1] + model.f['BC', t-1] == model.f['CE', t] + model.f['CF', t] + model.r['C', t]
    elif i == 'D':
        return model.f['BD', t-1] == model.f['DE', t] + model.r['D', t]
    elif i == 'E':
        return model.f['BE', t-1] + model.f['CE', t-1] + model.f['DE', t-1] == model.f['EG', t] + model.r['E', t]
    elif i == 'F':
        return model.f['AF', t-1] + model.f['CF', t-1] == model.f['FB', t] + model.f['FG', t] + model.r['F', t]
    elif i == 'G':
        return model.f['EG', t-1] + model.f['FG', t-1] == model.r['G', t]
model.flow_conservation = Constraint(model.points, model.time_steps, rule=flow_conservation_rule)

def pipe_capacity_rule(model, ij, t):
    return model.f[ij, t] <= model.c[ij] * model.x[ij]
model.pipe_capacity = Constraint(model.pipes, model.time_steps, rule=pipe_capacity_rule)

def pipe_activation_rule(model):
    return sum(model.x[ij] for ij in model.pipes) == 10
model.pipe_activation = Constraint(rule=pipe_activation_rule)

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

# Print the results
print(f"Objective value: {model.objective()}")
print("Activated pipes:")
for ij in model.pipes:
    if model.x[ij].value == 1:
        print(ij)

```

```

ERROR:pyomo.core:Rule failed when generating expression for Constraint flow_conservation with index ('B', 1):
KeyError: "Index 'AB', 0) is not valid for indexed component 'f'"
ERROR:pyomo.core:Constructing component 'flow_conservation' from data=None failed:
KeyError: "Index 'AB', 0) is not valid for indexed component 'f'"

```

```

-----
KeyError                                Traceback (most recent call last)
<ipython-input-15-d6fd0e37c834> in <cell line: 53>()
    51     elif i == 'G':
    52         return model.f['EG', t-1] + model.f['FG', t-1] == model.r['G', t]
--> 53 model.flow_conservation = Constraint(model.points, model.time_steps, rule=flow_conservation_rule)
    54
    55 def pipe_capacity_rule(model, ij, t):

```

```

7 frames
/usr/local/lib/python3.10/dist-packages/pyomo/core/base/indexed_component.py in _validate_index(self, idx)
    864         # Raise an exception
    865         #
--> 866         raise KeyError(
    867             "Index '%s' is not valid for indexed component '%s'"
    868             % (normalized_idx, self.name)

```

```

KeyError: "Index 'AB', 0) is not valid for indexed component 'f'"

```

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