

## ✓ 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)
```

 Great! Let's formulate the variables for this mathematical optimization problem. We will use the following notation:

- $t$ : time step ( $t = 1, 2, \dots, 10$ )
- $i, j$ : indices for the connection points (A, B, C, D, E, F, G)
- $p_{ij}$ : index for the pipe connecting points  $i$  and  $j$

Variables:

1.  $x_{ij}(t)$ : the amount of water flowing from point  $i$  to point  $j$  at time step  $t$ , for all connected points  $i$  and  $j$ , and  $t$
2.  $y_{ij}$ : binary variable indicating whether the pipe connecting points  $i$  and  $j$  is activated ( $y_{ij} = 1$ ) or deactivated ( $y_{ij} = 0$ )
3.  $r_i(t)$ : the amount of water released out of the system at point  $i$  and time step  $t$ , for all points  $i$  (except A and G)
4.  $f(t)$ : the amount of water entering point A at time step  $t$ , given by the function  $f(t) = \max(-4/30t^3 + t^2 - 0.234t + 3, 0)$
5.  $C_{ij}$ : the capacity of the pipe connecting points  $i$  and  $j$ , for all available pipes  $p_{ij}$ , as given in the problem state

These variables will be used to formulate the objective function and constraints of the optimization problem, which aims

## ✓ 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)
```

 Thank you for providing the variables. Now, let's formulate the objective function for this mathematical optimization pr

Objective Function:

The goal is to maximize the total amount of water flowing to point G over the 10 time steps. This can be expressed as:

Maximize:  $\sum (x_{FG}(t) + x_{EG}(t))$ , for  $t = 1$  to 10

In other words, the objective is to maximize the sum of the water flowing from point F to point G ( $x_{FG}(t)$ ) and the water

This objective function represents the total amount of water reaching the exit point G throughout the entire time horizon

## ✓ 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)
```

➡ Great! Now let's formulate the constraints for this optimization problem.

Constraints:

1. Water balance at each point (except A and G) for each time step:  
For all points  $i$  (except A and G) and time steps  $t = 1, 2, \dots, 10$ :  
 $\text{Sum}(x_{ji}(t-1), \text{ for all incoming pipes } p_{ji}) - \text{Sum}(x_{ik}(t), \text{ for all outgoing pipes } p_{ik}) - r_i(t) = 0$
2. Water inflow at point A for each time step:  
For time steps  $t = 1, 2, \dots, 10$ :  
 $f(t) - \text{Sum}(x_{Aj}(t), \text{ for all outgoing pipes } p_{Aj}) = 0$
3. Pipe capacity constraints:  
For all activated pipes  $p_{ij}$  and time steps  $t = 1, 2, \dots, 10$ :  
 $x_{ij}(t) \leq C_{ij} * y_{ij}$
4. Activation of exactly 10 pipes:  
 $\text{Sum}(y_{ij}, \text{ for all available pipes } p_{ij}) = 10$
5. Non-negativity constraints:  
For all pipes  $p_{ij}$  and time steps  $t = 1, 2, \dots, 10$ :  
 $x_{ij}(t) \geq 0$   
  
For all points  $i$  (except A and G) and time steps  $t = 1, 2, \dots, 10$ :  
 $r_i(t) \geq 0$
6. Binary constraints:  
For all available pipes  $p_{ij}$ :  
 $y_{ij}$  is binary (0 or 1)

These constraints ensure that:

1. The water balance is maintained at each point (except A and G) for each time step, considering the incoming and outgoing flows.
2. The water inflow at point A follows the given function  $f(t)$  for each time step.
3. The water flow through activated pipes does not exceed their respective capacities.
4. Exactly 10 pipes are activated in the network.
5. The water flows and released water amounts are non-negative.
6. The pipe activation variables are binary, indicating whether a pipe is activated (1) or deactivated (0).

These constraints, along with the objective function, complete the formulation of the mathematical optimization problem

## ✓ 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)

model.x = Var(model.P, model.I, domain=NonNegativeReals) # Water flow
model.y = Var(model.P, domain=Binary) # Pipe activation
model.r = Var(model.I - {'A', 'G'}, model.T, domain=NonNegativeReals) # Water released

# Define the objective function
def obj_rule(model):
    return sum(model.x[('F', 'G'), t] + model.x[('E', 'G'), t] for t in model.T)
model.obj = Objective(rule=obj_rule, sense=maximize)

# Define the constraints
def water_balance_rule(model, i, t):
    if i not in {'A', 'G'}:
        return sum(model.x[j, i, t-1] for j in model.I if (j, i) in model.P) - sum(model.x[i, k, t] for k in model.I if (i, k) in model.P) == 0
    else:
        return Constraint.Skip
model.water_balance = Constraint(model.I, model.T, rule=water_balance_rule)

def water_inflow_rule(model, t):
    return max(-4/30*t**3+t**2-0.234*t+3, 0) - sum(model.x['A', j, t] for j in model.I if ('A', j) in model.P) == 0
model.water_inflow = Constraint(model.T, rule=water_inflow_rule)

def pipe_capacity_rule(model, i, j, t):
    return model.x[i, j, t] <= model.C[i, j] * model.y[i, j]
model.pipe_capacity = Constraint(model.P, model.T, rule=pipe_capacity_rule)

def activate_pipes_rule(model):
    return sum(model.y[i, j] for (i, j) in model.P) == 10
model.activate_pipes = Constraint(rule=activate_pipes_rule)

# Sample data for pipe capacities
pipe_capacities = {
    ('A', 'B'): 5, ('A', 'C'): 7, ('B', 'C'): 4, ('B', 'D'): 6, ('B', 'E'): 5,
    ('C', 'D'): 3, ('C', 'F'): 6, ('D', 'E'): 4, ('D', 'F'): 5, ('E', 'F'): 3,
    ('E', 'G'): 6, ('F', 'G'): 8
}
model.C = Param(model.P, initialize=pipe_capacities)

# Create an instance of the model
instance = model.create_instance()

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

# Print the results
print("Objective value:", value(instance.obj))
for (i, j) in instance.P:
    if value(instance.y[i, j]) == 1:
        print(f"Pipe ({i}, {j}) is activated.")
    else:
        print(f"Pipe ({i}, {j}) is deactivated.")
...

```

In this code, we define the sets, variables, objective function, and constraints using Pyomo. We use sample data for the pipe capacities. The code solves the optimization problem using the GLPK solver and prints the objective value (total water flow to point G). Please note that this code assumes you have Pyomo and the GLPK solver installed. You can install them using `pip install pyomo glpk-utils`.

## 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 *

# Define the model
model = AbstractModel()

# Define the sets
model.T = RangeSet(1, 10) # Time steps
model.I = Set(initialize=['A', 'B', 'C', 'D', 'E', 'F', 'G']) # Connection points
model.P = Set(initialize=[('A', 'B'), ('A', 'C'), ('A', 'F'), ('B', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'E'), ('C', 'F'), ('C', 'G'), ('D', 'E'), ('D', 'F'), ('D', 'G'), ('E', 'F'), ('E', 'G'), ('F', 'G')])

# Define the variables
model.x = Var(model.P, model.T, domain=NonNegativeReals) # Water flow
model.y = Var(model.P, domain=Binary) # Pipe activation
model.r = Var(model.I - {'A', 'G'}, model.T, domain=NonNegativeReals) # Water released

# Define the objective function
def obj_rule(model):
    return sum(model.x[('F', 'G'), t] + model.x[('E', 'G'), t] for t in model.T)
model.obj = Objective(rule=obj_rule, sense=maximize)

# Define the constraints
def water_balance_rule(model, i, t):
    if i not in {'A', 'G'}:
        return sum(model.x[j, i, t-1] for j in model.I if (j, i) in model.P) - sum(model.x[i, k, t] for k in model.I if (i, k) in model.P) == 0
    else:
        return Constraint.Skip
model.water_balance = Constraint(model.I, model.T, rule=water_balance_rule)

def water_inflow_rule(model, t):
    return max(-4/30*t**3+t**2-0.234*t+3, 0) - sum(model.x['A', j, t] for j in model.I if ('A', j) in model.P) == 0
model.water_inflow = Constraint(model.T, rule=water_inflow_rule)

def pipe_capacity_rule(model, i, j, t):
    return model.x[i, j, t] <= model.C[i, j] * model.y[i, j]
model.pipe_capacity = Constraint(model.P, model.T, rule=pipe_capacity_rule)

def activate_pipes_rule(model):
    return sum(model.y[i, j] for (i, j) in model.P) == 10
model.activate_pipes = Constraint(rule=activate_pipes_rule)

# Sample data for pipe capacities
pipe_capacities = {
    ('A', 'B'): 3, ('A', 'C'): 6, ('A', 'F'): 1, ('B', 'C'): 3, ('B', 'D'): 10, ('B', 'E'): 4,
    ('C', 'E'): 4, ('C', 'F'): 4, ('D', 'E'): 5, ('E', 'G'): 5, ('F', 'B'): 12,
    ('F', 'G'): 7
}
model.C = Param(model.P, initialize=pipe_capacities)

# Create an instance of the model
instance = model.create_instance()

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

# Print the results
print("Objective value:", value(instance.obj))
for (i, j) in instance.P:
    if value(instance.y[i, j]) == 1:
        print(f"Pipe ({i}, {j}) is activated.")
    else:
        print(f"Pipe ({i}, {j}) is deactivated.")

```

```

ERROR:pyomo.core:Rule failed when generating expression for Constraint water_l
KeyError: "Index>('A', 'B', 0)' is not valid for indexed component 'x'"
ERROR:pyomo.core:Constructing component 'water_balance' from data=None failed
KeyError: "Index>('A', 'B', 0)' is not valid for indexed component 'x'"
-----
KeyError                                Traceback (most recent call last)
<ipython-input-15-acd466e2007b> in <cell line: 50>()
    48
    49 # Create an instance of the model
--> 50 instance = model.create_instance()
    51
    52 # Solve the optimization problem

-----
10 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>('A', 'B', 0)' is not valid for indexed component 'x'"

```

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

```

from pyomo.environ import *

# Define the model
model = AbstractModel()

# Define the sets
model.T = RangeSet(1, 10) # Time steps
model.I = Set(initialize=['A', 'B', 'C', 'D', 'E', 'F', 'G']) # Connection points
model.P = Set(initialize=[('A', 'B'), ('A', 'C'), ('A', 'F'), ('B', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'E'), ('C', 'F'), ('D', 'E'), ('E', 'G')])

# Define the variables
model.x = Var(model.P, model.T, domain=NonNegativeReals) # Water flow
model.y = Var(model.P, domain=Binary) # Pipe activation
model.r = Var(model.I - {'A', 'G'}, model.T, domain=NonNegativeReals) # Water released

# Sample data for pipe capacities
pipe_capacities = {
    ('A', 'B'): 3, ('A', 'C'): 6, ('A', 'F'): 1, ('B', 'C'): 3, ('B', 'D'): 10, ('B', 'E'): 4,
    ('C', 'E'): 4, ('C', 'F'): 4, ('D', 'E'): 5, ('E', 'G'): 5, ('F', 'B'): 12,
    ('F', 'G'): 7
}
model.C = Param(model.P, initialize=pipe_capacities)

# Define the objective function
def obj_rule(model):
    return sum(model.x[('F', 'G'), t] + model.x[('E', 'G'), t] for t in model.T)
model.obj = Objective(rule=obj_rule, sense=maximize)

# Define the constraints
def water_balance_rule(model, i, t):
    if i not in {'A', 'G'} and t != 1:
        return sum(model.x[j, i, t-1] for j in model.I if (j, i) in model.P) - sum(model.x[i, k, t] for k in model.I if (i, k) in model.P)
    else:
        return Constraint.Skip
model.water_balance = Constraint(model.I, model.T, rule=water_balance_rule)

def water_inflow_rule(model, t):
    return max(-4/30*t**3+t**2-0.234*t+3, 0) - sum(model.x['A', j, t] for j in model.I if ('A', j) in model.P) == 0
model.water_inflow = Constraint(model.T, rule=water_inflow_rule)

def pipe_capacity_rule(model, i, j, t):
    return model.x[i, j, t] <= model.C[i, j] * model.y[i, j]
model.pipe_capacity = Constraint(model.P, model.T, rule=pipe_capacity_rule)

def activate_pipes_rule(model):
    return sum(model.y[i, j] for (i, j) in model.P) == 10
model.activate_pipes = Constraint(rule=activate_pipes_rule)

# Create an instance of the model
instance = model.create_instance()

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

```

```
# Print the results
print("Objective value:", value(instance.obj))
for (i, j) in instance.P:
    if value(instance.y[i, j]) == 1:
        print(f"Pipe ({i}, {j}) is activated.")
    else:
        print(f"Pipe ({i}, {j}) is deactivated.")
```

ERROR:pyomo.common.numeric\_types:evaluating object as numeric value: x[F,G,1]  
(object: <class 'pyomo.core.base.var.VarData'>)  
No value for uninitialized NumericValue object x[F,G,1]  
ERROR:pyomo.common.numeric\_types:evaluating object as numeric value: obj  
(object: <class 'pyomo.core.base.objective.ScalarObjective'>)  
No value for uninitialized NumericValue object x[F,G,1]

-----  
**ValueError** Traceback (most recent call last)  
<ipython-input-17-455ab34d65cb> in <cell line: 57>()

```
55
56 # Print the results
--> 57 print("Objective value:", value(instance.obj))
58 for (i, j) in instance.P:
59     if value(instance.y[i, j]) == 1:
```

⏏ 7 frames

```
/usr/local/lib/python3.10/dist-packages/pyomo/common/numeric_types.py in value(obj, exception)
382         tmp = obj(exception=True)
383         if tmp is None:
--> 384             raise ValueError(
385                 "No value for uninitialized NumericValue object %s" % (obj.name,)
386             )
```

**ValueError:** No value for uninitialized NumericValue object x[F,G,1]

```
print(results)
```

Problem:

- Name: unknown
- Lower bound: -inf
- Upper bound: inf
- Number of objectives: 1
- Number of constraints: 176
- Number of variables: 177
- Number of nonzeros: 498
- Sense: maximize

Solver:

- Status: ok
- Termination condition: infeasible
- Statistics:
  - Branch and bound:
    - Number of bounded subproblems: 0
    - Number of created subproblems: 0
- Error rc: 0
- Time: 0.006079673767089844