ex1

import pyomo.environ as pyo

EXEC\_PATH = "C:\\glpk-4.65\\w64\\glpsol.exe"

model = pyo.ConcreteModel()

# Variáveis

model.x1 = pyo.Var(within=pyo.NonNegativeReals)  # lingotes

model.x2 = pyo.Var(within=pyo.NonNegativeReals)  # grafite

model.x3 = pyo.Var(within=pyo.NonNegativeReals)  # sucata

# Função objetivo (custos)

model.obj = pyo.Objective(expr=90\*model.x1 + 180\*model.x2 + 25\*model.x3, sense=pyo.minimize)

# Restrição de demanda (10 toneladas de liga)

model.demand = pyo.Constraint(expr=(model.x1 + model.x2 + model.x3) == 10)

# Limites de estoque

model.stock1 = pyo.Constraint(expr=model.x1 <= 5)

model.stock2 = pyo.Constraint(expr=model.x2 <= 5)

model.stock3 = pyo.Constraint(expr=model.x3 <= 12)

# Carbono (0% a 9,5%)

model.carbon\_min = pyo.Constraint(expr=(0.005\*model.x1 + 0.90\*model.x2 + 0.090\*model.x3) >= 0.00\*10)

model.carbon\_max = pyo.Constraint(expr=(0.005\*model.x1 + 0.90\*model.x2 + 0.090\*model.x3) <= 0.095\*10)

# Silício (19% a 20%)

model.silicio\_min = pyo.Constraint(expr=(0.14\*model.x1 + 0.27\*model.x3) >= 0.19\*10)

model.silicio\_max = pyo.Constraint(expr=(0.14\*model.x1 + 0.27\*model.x3) <= 0.20\*10)

# Solver

Otimizador = pyo.SolverFactory('glpk', executable=EXEC\_PATH)

Resultado = Otimizador.solve(model)

# Resultados

print("Status:", Resultado.solver.status)

print("Custo mínimo:", pyo.value(model.obj)) # 603,703

print("x1 (Lingotes) =", pyo.value(model.x1)) # 5

print("x2 (Grafite) =", pyo.value(model.x2)) # 0,1851

print("x3 (Sucata)  =", pyo.value(model.x3)) # 4,8148

print("tonelada total =", pyo.value(model.demand)) # 9,99999999999

Custo mínimo: 603.7037037037037

x1 (Lingotes) = 5.0

x2 (Grafite) = 0.185185185185186

x3 (Sucata) = 4.81481481481481

tonelada total = 9.999999999999996

ex3

import pyomo.environ as pyo

from pyomo.opt import SolverFactory

# Dados

months = [1,2,3,4,5]

demand = {1:30000, 2:20000, 3:40000, 4:10000, 5:50000}

cost = {1:65, 2:100, 3:135, 4:160, 5:190}

# Conjuntos de pares (i,k) válidos

pairs = [(i,k) for i in months for k in cost.keys() if i + k - 1 <= 5]

model = pyo.ConcreteModel()

# Índice para pares

model.P = pyo.Set(initialize=pairs, dimen=2)

# Variáveis y[i,k] >= 0

model.y = pyo.Var(model.P, domain=pyo.NonNegativeReals)

# Objetivo

model.obj = pyo.Objective(

    expr=sum(cost[k] \* model.y[i,k] for (i,k) in model.P),

    sense=pyo.minimize

)

# Restrições de demanda

def demand\_rule(m, t):

    return sum(m.y[i,k] for (i,k) in m.P if i <= t <= i+k-1) >= demand[t]

model.demand\_constr = pyo.Constraint(months, rule=demand\_rule)

# Solver (ajuste o caminho do executável se necessário)

# Otimizador = SolverFactory('glpk', executable="C:\\caminho\\glpsol.exe")

Otimizador = SolverFactory('glpk')  # assume glpk disponível no PATH

res = Otimizador.solve(model, tee=False)

print("Status:", res.solver.status, res.solver.termination\_condition)

print("Custo mínimo:", pyo.value(model.obj))

for (i,k) in model.P:

    val = pyo.value(model.y[i,k])

    if val is None:

        val = 0.0

    if val > 1e-6:

        print(f"y[{i},{k}] = {val:,.0f} sq.ft.  (inicia no mês {i}, duração {k} meses)")

Custo mínimo: 7650000.0

y[1,5] = 30,000 sq.ft. (inicia no mês 1, duração 5 meses)

y[3,1] = 10,000 sq.ft. (inicia no mês 3, duração 1 meses)

y[5,1] = 20,000 sq.ft. (inicia no mês 5, duração 1 meses)

ex4

import pyomo.environ as pyo

model = pyo.ConcreteModel()

Years = [1,2,3,4,5]

model.A = pyo.Var(years, domain=pyo.NonNegativeReals)

model.B = pyo.Var(years, domain=pyo.NonNegativeReals)

model.C2 = pyo.Var(domain=pyo.NonNegativeReals)  # só t=2

model.D5 = pyo.Var(domain=pyo.NonNegativeReals)  # só t=5

model.R = pyo.Var(years, domain=pyo.NonNegativeReals)  # R1..R5

S1 = 60000.0

model.flow1 = pyo.Constraint(expr= model.A[1] + model.B[1] + model.R[1] == S1)

model.flow2 = pyo.Constraint(expr= model.A[2] + model.B[2] + model.C2 + model.R[2] == model.R[1])

model.flow3 = pyo.Constraint(expr= model.A[3] + model.B[3] + model.R[3] == model.R[2] + 1.4\*model.A[1])

model.flow4 = pyo.Constraint(expr= model.A[4] + model.B[4] + model.R[4] == model.R[3] + 1.4\*model.A[2] + 1.7\*model.B[1])

model.flow5 = pyo.Constraint(expr= model.A[5] + model.B[5] + model.D5 + model.R[5] == model.R[4] + 1.4\*model.A[3] + 1.7\*model.B[2])

# Objetivo: maximizar valor no início do ano 6

model.obj = pyo.Objective(expr= model.R[5] + 1.4\*model.A[4] + 1.7\*model.B[3] + 1.9\*model.C2 + 1.3\*model.D5, sense=pyo.maximize)

solver = pyo.SolverFactory('glpk')

res = solver.solve(model, tee=False)

print("Status:", res.solver.status, res.solver.termination\_condition)

print("Valor acumulado no início do ano 6:", pyo.value(model.obj))

# Mostrar investimentos não nulos

for t in years:

    a = pyo.value(model.A[t])

    b = pyo.value(model.B[t])

    r = pyo.value(model.R[t])

    if a and a > 1e-6:

        print(f"A\_{t} = {a:,.2f}")

    if b and b > 1e-6:

        print(f"B\_{t} = {b:,.2f}")

    if r and r > 1e-6:

        print(f"R\_{t} = {r:,.2f}")

if pyo.value(model.C2) > 1e-6:

    print("C2 =", pyo.value(model.C2))

if pyo.value(model.D5) > 1e-6:

    print("D5 =", pyo.value(model.D5))

Valor acumulado no início do ano 6: 152880.0

A\_1 = 60,000.00

A\_3 = 84,000.00

D5 = 117600.0

Ex5

import pyomo.environ as pyo

from pyomo.opt import SolverFactory

model = pyo.ConcreteModel()

J = [1,2,3,4,5]

# parâmetros

cost = {1:22, 2:20, 3:25, 4:24, 5:27}

tin  = {1:0.60, 2:0.25, 3:0.45, 4:0.20, 5:0.50}

zinc = {1:0.10, 2:0.15, 3:0.45, 4:0.50, 5:0.40}

lead = {1:0.30, 2:0.60, 3:0.10, 4:0.30, 5:0.10}

# variáveis (fração de 1 lb)

model.x = pyo.Var(J, domain=pyo.NonNegativeReals)

# objetivo

model.obj = pyo.Objective(expr=sum(cost[j]\*model.x[j] for j in J), sense=pyo.minimize)

# restrições de composição

model.tin\_constr  = pyo.Constraint(expr=sum(tin[j]\*model.x[j]  for j in J) == 0.40)

model.zinc\_constr = pyo.Constraint(expr=sum(zinc[j]\*model.x[j] for j in J) == 0.35)

model.lead\_constr = pyo.Constraint(expr=sum(lead[j]\*model.x[j] for j in J) == 0.25)

# (opcional) balanço de massa

model.mass = pyo.Constraint(expr=sum(model.x[j] for j in J) == 1.0)

# solver (ajuste se precisar especificar caminho)

solver = SolverFactory('glpk')

res = solver.solve(model, tee=False)

print("Status:", res.solver.status, res.solver.termination\_condition)

print("Custo mínimo por lb: $", pyo.value(model.obj))

for j in J:

    v = pyo.value(model.x[j])

    if v is None: v = 0.0

    if v > 1e-8:

        print(f"x{j} = {v:.9f} lb")

Custo mínimo por lb: $ 23.45652173913044

x1 = 0.043478261 lb

x2 = 0.282608696 lb

x3 = 0.673913043 lb

Ex6

import pyomo.environ as pyo

# Dados do problema

cargas = [1, 2, 3, 4]

compartimentos = [1, 2, 3] # 1=Frontal, 2=Central, 3=Traseiro

# Dicionários com os dados

lucro\_ton = {1: 320, 2: 400, 3: 360, 4: 290}

peso\_disponivel = {1: 20, 2: 16, 3: 25, 4: 13}

volume\_pe\_cubico = {1: 500, 2: 700, 3: 600, 4: 400}

peso\_max\_comp = {1: 12, 2: 18, 3: 10}

volume\_max\_comp = {1: 7000, 2: 9000, 3: 5000}

# Criação do modelo

model = pyo.ConcreteModel()

# Conjuntos

model.cargas = pyo.Set(initialize=cargas)

model.compartimentos = pyo.Set(initialize=compartimentos)

# Variáveis de decisão

model.x = pyo.Var(model.cargas, model.compartimentos, domain=pyo.NonNegativeReals)

# Função Objetivo

def lucro\_total(model):

    return sum(lucro\_ton[i] \* sum(model.x[i, j] for j in model.compartimentos) for i in model.cargas)

model.obj = pyo.Objective(rule=lucro\_total, sense=pyo.maximize)

# Restrições

# 1. Restrições de Peso por compartimento

model.peso\_comp = pyo.ConstraintList()

for j in model.compartimentos:

    model.peso\_comp.add(expr=sum(model.x[i, j] for i in model.cargas) <= peso\_max\_comp[j])

# 2. Restrições de Volume por compartimento

model.volume\_comp = pyo.ConstraintList()

for j in model.compartimentos:

    model.volume\_comp.add(expr=sum(volume\_pe\_cubico[i] \* model.x[i, j] for i in model.cargas) <= volume\_max\_comp[j])

# 3. Restrições de Disponibilidade de Carga

model.carga\_disponivel = pyo.ConstraintList()

for i in model.cargas:

    model.carga\_disponivel.add(expr=sum(model.x[i, j] for j in model.compartimentos) <= peso\_disponivel[i])

# 4. Restrições de Equilíbrio

peso\_frontal = sum(model.x[i, 1] for i in model.cargas)

peso\_central = sum(model.x[i, 2] for i in model.cargas)

peso\_traseiro = sum(model.x[i, 3] for i in model.cargas)

model.equilibrio1 = pyo.Constraint(expr=peso\_frontal / peso\_max\_comp[1] == peso\_central / peso\_max\_comp[2])

model.equilibrio2 = pyo.Constraint(expr=peso\_central / peso\_max\_comp[2] == peso\_traseiro / peso\_max\_comp[3])

# Solucionador

solver = pyo.SolverFactory('glpk')

results = solver.solve(model, tee=False)

# Impressão dos resultados

print(f"Status: {results.solver.status}")

print(f"Condição de Término: {results.solver.termination\_condition}")

print(f"\nLucro Total Máximo: ${pyo.value(model.obj):.2f}")

print("\nDistribuição da Carga:")

for i in model.cargas:

    for j in model.compartimentos:

        if pyo.value(model.x[i, j]) > 1e-6:  # Impede valores muito pequenos

            print(f"  - Carga {i} no Compartimento {j}: {pyo.value(model.x[i, j]):.2f} toneladas")

print("\nPesos Totais por Compartimento:")

peso\_total\_frontal = pyo.value(peso\_frontal)

peso\_total\_central = pyo.value(peso\_central)

peso\_total\_traseiro = pyo.value(peso\_traseiro)

print(f"  - Compartimento Frontal: {peso\_total\_frontal:.2f} toneladas (12 max)")

print(f"  - Compartimento Central: {peso\_total\_central:.2f} toneladas (18 max)")

print(f"  - Compartimento Traseiro: {peso\_total\_traseiro:.2f} toneladas (10 max)")

print("\nVerificação do Equilíbrio:")

print(f"  - Frontal/12: {peso\_total\_frontal/12:.4f}")

print(f"  - Central/18: {peso\_total\_central/18:.4f}")

print(f"  - Traseiro/10: {peso\_total\_traseiro/10:.4f}")

Lucro Total Máximo: $13330.00

Distribuição da Carga:

- Carga 1 no Compartimento 1: 2.00 toneladas

- Carga 1 no Compartimento 3: 4.00 toneladas

- Carga 2 no Compartimento 3: 2.00 toneladas

- Carga 3 no Compartimento 1: 10.00 toneladas

- Carga 3 no Compartimento 2: 9.00 toneladas

- Carga 4 no Compartimento 2: 9.00 toneladas

- Carga 4 no Compartimento 3: 4.00 toneladas

Pesos Totais por Compartimento: - Compartimento Frontal: 12.00 toneladas (12 max)

- Compartimento Central: 18.00 toneladas (18 max)

- Compartimento Traseiro: 10.00 toneladas (10 max)

Verificação do Equilíbrio: - Frontal/12: 1.0000 ; - Central/18: 1.0000 ; - Traseiro/10: 1.0000

Ex18

import pyomo.environ as pyo

model = pyo.ConcreteModel()

model.tipos\_caixa = pyo.Set(initialize=['alimento', 'agua', 'municao', 'remedios'])

valores = {'alimento': 1,'agua': 2,'municao': 4,'remedios': 4}

necessidades = {'alimento': 6,'agua': 4,'municao': 2,'remedios': 2}

model.x = pyo.Var(model.tipos\_caixa, domain=pyo.NonNegativeIntegers)

# 4. Definição da Função Objetivo

def objetivo\_regra(model):

    return sum(valores[i] \* model.x[i] for i in model.tipos\_caixa)

model.objetivo = pyo.Objective(rule=objetivo\_regra, sense=pyo.maximize)

# 5. Definição das Restrições

model.restricoes = pyo.ConstraintList()

# Restrição de Capacidade do Helicóptero

# LHS: sum(model.x) <= RHS: 7

model.restricoes.add(expr=sum(model.x[i] for i in model.tipos\_caixa) <= 7)

# Restrições de Necessidade (quantidade máxima útil)

# LHS: model.x[i] <= RHS: necessidades[i]

for i in model.tipos\_caixa:

    model.restricoes.add(expr=model.x[i] <= necessidades[i])

solver = pyo.SolverFactory('glpk')

results = solver.solve(model, tee=False)

print(f"Status do Solucionador: {results.solver.status}")

print(f"Condição de Término: {results.solver.termination\_condition}\n")

print("--- Solução Ótima ---")

print(f"Valor Máximo de Sobrevivência: {pyo.value(model.objetivo)}\n")

print("Quantidade de caixas a serem transportadas:")

for i in model.tipos\_caixa:

    print(f"  - {i.capitalize()}: {int(pyo.value(model.x[i]))} caixas")

print("\n--- Verificação das Restrições ---")

# Verificação da Restrição de Capacidade

lhs\_capacidade = sum(int(pyo.value(model.x[i])) for i in model.tipos\_caixa)

rhs\_capacidade = 7

print(f"Restrição de Capacidade: {lhs\_capacidade} <= {rhs\_capacidade} (LHS <= RHS)")

# Verificação das Restrições de Necessidade

for i in model.tipos\_caixa:

    lhs\_necessidade = int(pyo.value(model.x[i]))

    rhs\_necessidade = necessidades[i]

    print(f"Necessidade de {i.capitalize()}: {lhs\_necessidade} <= {rhs\_necessidade} (LHS <= RHS)")

Valor Máximo de Sobrevivência: 22.0

Quantidade de caixas a serem transportadas: - Alimento: 0 caixas - Agua: 3 caixas - Municao: 2 caixas - Remedios: 2 caixas

--- Verificação das Restrições ---

Restrição de Capacidade: 7 <= 7 (LHS <= RHS)

Necessidade de Alimento: 0 <= 6 (LHS <= RHS)

Necessidade de Agua: 3 <= 4 (LHS <= RHS)

Necessidade de Municao: 2 <= 2 (LHS <= RHS)

Necessidade de Remedios: 2 <= 2 (LHS <= RHS)

Ex19

import pyomo.environ as pyo

# Dados do problema

areas = {1:1500, 2:2000, 3:1200, 4:800}

# Produção anual esperada (m³/ha)

producao = {

    (1,'Pinus'):17, (1,'Carvalho'):14, (1,'Nogueira'):13, (1,'Araucaria'):16,

    (2,'Pinus'):16, (2,'Carvalho'):15, (2,'Nogueira'):11, (2,'Araucaria'):18,

    (3,'Pinus'):12, (3,'Carvalho'):13, (3,'Nogueira'):10, (3,'Araucaria'):17,

    (4,'Pinus'):10, (4,'Carvalho'):11, (4,'Nogueira'): 8, (4,'Araucaria'):15,

}

# Renda anual esperada (R$/ha)

renda = {

    (1,'Pinus'):12, (1,'Carvalho'):20, (1,'Nogueira'):18, (1,'Araucaria'):16,

    (2,'Pinus'):11, (2,'Carvalho'):22, (2,'Nogueira'):20, (2,'Araucaria'):18,

    (3,'Pinus'):10, (3,'Carvalho'):18, (3,'Nogueira'):16, (3,'Araucaria'):17,

    (4,'Pinus'): 9, (4,'Carvalho'):15, (4,'Nogueira'):14, (4,'Araucaria'):13,

}

# Produção mínima exigida (1000 m³)

prod\_min = {'Pinus':9, 'Carvalho':4.8, 'Nogueira':3.6, 'Araucaria':8.5}

cidades = [1,2,3,4]

especies = ['Pinus','Carvalho','Nogueira','Araucaria']

# Modelo

model = pyo.ConcreteModel()

# Variáveis de decisão

model.x = pyo.Var(cidades, especies, domain=pyo.NonNegativeReals)

# Função objetivo

model.obj = pyo.Objective(

    expr=sum(renda[(i,j)] \* model.x[i,j] for i in cidades for j in especies),

    sense=pyo.maximize

)

# Restrições de área

model.area = pyo.ConstraintList()

for i in cidades:

    model.area.add(sum(model.x[i,j] for j in especies) <= areas[i])

# Restrições de produção mínima

model.prod = pyo.ConstraintList()

for j in especies:

    model.prod.add(sum(producao[(i,j)] \* model.x[i,j] for i in cidades) >= prod\_min[j]\*1000)

# Resolução

solver = pyo.SolverFactory('glpk')

solver.solve(model, tee=False)

# Saída organizada

print("=== SOLUÇÃO DO PROBLEMA ===")

print(f"Função objetivo ótima (renda máxima): {pyo.value(model.obj):.2f}\n")

print("Variáveis de decisão (área alocada em hectares):")

for i in cidades:

    for j in especies:

        val = pyo.value(model.x[i,j])

        if val > 1e-6:

            print(f"  Cidade {i}, {j}: {val:.2f}")

print("\nRestrições (LHS ≤ RHS para área, LHS ≥ RHS para produção):")

for i in cidades:

    lhs = sum(pyo.value(model.x[i,j]) for j in especies)

    print(f"  Área cidade {i}: {lhs:.2f} ≤ {areas[i]}")

for j in especies:

    lhs = sum(producao[(i,j)] \* pyo.value(model.x[i,j]) for i in cidades)

    print(f"  Produção {j}: {lhs:.2f} ≥ {prod\_min[j]\*1000}")

Função objetivo ótima (renda máxima): 102414.71

Variáveis de decisão (área alocada em hectares):

Cidade 1, Pinus: 529.41

Cidade 1, Carvalho: 970.59

Cidade 2, Carvalho: 2000.00

Cidade 3, Carvalho: 700.00

Cidade 3, Araucaria: 500.00

Cidade 4, Carvalho: 350.00

Cidade 4, Nogueira: 450.00

Restrições (LHS ≤ RHS para área, LHS ≥ RHS para produção):

Área cidade 1: 1500.00 ≤ 1500

Área cidade 2: 2000.00 ≤ 2000

Área cidade 3: 1200.00 ≤ 1200

Área cidade 4: 800.00 ≤ 800

Produção Pinus: 9000.00 ≥ 9000

Produção Carvalho: 56538.24 ≥ 4800.0

Produção Nogueira: 3600.00 ≥ 3600.0

Produção Araucaria: 8500.00 ≥ 8500.0

Ex23

import pyomo.environ as pyo

# PROBLEMA PRIMAL

print("--- RESOLUÇÃO DO PROBLEMA PRIMAL (Minimizar Custo) ---")

primal\_model = pyo.ConcreteModel()

# Variáveis (toneladas)

primal\_model.A = pyo.Var(domain=pyo.NonNegativeReals)

primal\_model.B = pyo.Var(domain=pyo.NonNegativeReals)

primal\_model.S = pyo.Var(domain=pyo.NonNegativeReals)

# Parâmetros

costA = 60.0

costB = 30.0

costS = 2500.0

Fe\_req = 240.0

# Objetivo (Minimizar)

primal\_model.obj = pyo.Objective(expr = costA\*primal\_model.A + costB\*primal\_model.B + costS\*primal\_model.S, sense=pyo.minimize)

# Restrições

primal\_model.fe = pyo.Constraint(expr = 0.60\*primal\_model.A + 0.40\*primal\_model.B + 1.00\*primal\_model.S == Fe\_req)

primal\_model.si\_fe = pyo.Constraint(expr = 0.30\*primal\_model.A - 0.20\*primal\_model.B + 1.00\*primal\_model.S >= 0)

# Resolver

solver = SolverFactory('glpk')

solver.solve(primal\_model, tee=False)

# Saída organizada do Primal

print("\n--- Solução do Primal ---")

print("Valor ótimo (custo mínimo) = R$ {:.2f}".format(pyo.value(primal\_model.obj)))

print("\nVariáveis de Decisão:")

print("A = {:.4f} t".format(pyo.value(primal\_model.A)))

print("B = {:.4f} t".format(pyo.value(primal\_model.B)))

print("S = {:.4f} t".format(pyo.value(primal\_model.S)))

print("\nRestrições do Primal:")

print("LHS Fe = {:.4f} == RHS 240".format(pyo.value(primal\_model.fe.expr)))

print("LHS Si-Fe = {:.4f} >= RHS 0".format(pyo.value(primal\_model.si\_fe.expr)))

print("\n" + "="\*50 + "\n")

# PROBLEMA DUAL

print("--- RESOLUÇÃO DO PROBLEMA DUAL (Maximizar Valor das Restrições) ---")

dual\_model = pyo.ConcreteModel()

# Variáveis duais

dual\_model.pi = pyo.Var(domain=pyo.Reals)

dual\_model.mu = pyo.Var(domain=pyo.Reals)

# Correção na definição do domínio da variável mu

# A restrição do primal `si\_fe` (>= 0) corresponde a uma variável dual `mu` com domínio `NonNegativeReals`.

# O erro no código original é que `mu` é definido como `Reals`, o que é incorreto.

# Corrigindo para o código mesclado.

dual\_model.pi = pyo.Var(domain=pyo.Reals)

dual\_model.mu = pyo.Var(domain=pyo.NonNegativeReals)

# Objetivo (Maximizar)

dual\_model.obj = pyo.Objective(expr = 240\*dual\_model.pi + 0\*dual\_model.mu, sense=pyo.maximize)

# Restrições do dual (ligadas a A, B, S)

dual\_model.consA = pyo.Constraint(expr = 0.60\*dual\_model.pi + 0.30\*dual\_model.mu <= 60)

dual\_model.consB = pyo.Constraint(expr = 0.40\*dual\_model.pi - 0.20\*dual\_model.mu <= 30)

dual\_model.consS = pyo.Constraint(expr = 1.00\*dual\_model.pi + 1.00\*dual\_model.mu <= 2500)

# Resolver

solver = pyo.SolverFactory('glpk')

solver.solve(dual\_model, tee=False)

# Saída organizada do Dual

print("\n--- Solução do Dual ---")

print("Valor ótimo (receita máxima) = R$ {:.2f}".format(pyo.value(dual\_model.obj)))

print("\nVariáveis de Decisão:")

print("pi = {:.4f}".format(pyo.value(dual\_model.pi)))

print("mu = {:.4f}".format(pyo.value(dual\_model.mu)))

print("\nRestrições do Dual:")

print("LHS consA = {:.4f} <= RHS 60".format(pyo.value(dual\_model.consA.expr)))

print("LHS consB = {:.4f} <= RHS 30".format(pyo.value(dual\_model.consB.expr)))

print("LHS consS = {:.4f} <= RHS 2500".format(pyo.value(dual\_model.consS.expr)))

print("\n" + "="\*50 + "\n")

print("--- VERIFICAÇÃO DA TEORIA DA DUALIDADE ---")

if abs(pyo.value(primal\_model.obj) - pyo.value(dual\_model.obj)) < 1e-6:

    print("Os valores da função objetivo dos modelos Primal e Dual são iguais.")

    print("Isso confirma a Teoria da Dualidade Forte.")

else:

    print("Os valores da função objetivo dos modelos Primal e Dual são diferentes.")

    print("Isso pode indicar um erro de formulação ou na solução.")

Valor ótimo (custo mínimo) = R$ 21000.00

Variáveis de Decisão:

A = 200.0000 t

B = 300.0000 t

S = 0.0000 t

Restrições do Primal:

LHS Fe = 1.0000 == RHS 240

LHS Si-Fe = 1.0000 >= RHS 0

--- Solução do Dual ---

Valor ótimo (receita máxima) = R$ 21000.00

Variáveis de Decisão:

pi = 87.5000

mu = 25.0000

Restrições do Dual:

LHS consA = 1.0000 <= RHS 60

LHS consB = 1.0000 <= RHS 30

LHS consS = 1.0000 <= RHS 2500

--- VERIFICAÇÃO DA TEORIA DA DUALIDADE ---

Os valores da função objetivo dos modelos Primal e Dual são iguais.

Isso confirma a Teoria da Dualidade Forte.