به نام خدا

بخش دوم کدهای پروژه پایانی

استاد راهنما: دکتر صنیعی منفرد

دانشجو: آندیا حاج نوروزی

𝑅𝑠𝑦𝑠(𝑡) = ∏[1 − (1 − 𝑅𝑖(𝑡)) 𝑛𝑖]

𝑀𝑎𝑥 𝑅𝑠𝑦𝑠(𝑡)

𝑠𝑡.

∑𝛼𝑖( −𝑡 𝑙𝑛 𝑅𝑖(𝑡) ) 𝛽𝑖(𝑛𝑖 + 𝑒 0.25𝑛𝑖) ≤ 400

∑𝑣𝑖 ⋅ 𝑛𝑖 2 ≤ 250

∑𝑤𝑖 ⋅ 𝑛𝑖 − 𝑒 0.25𝑛𝑖 ≤ 500

1 ≤ 𝑛𝑖 ≤ 10 ∀ 𝑖 = 1,2,3,4

𝑈𝑛𝑖𝑡 1 ( i = 1 ) (α1 = 1 × 10−5 ) (β1 = 105) (𝑉1 = 1) (𝑊1 = 6) 𝑈𝑛𝑖𝑡 2 ( i = 2 ) (α2 = 2.3 × 10−5 ) (β2 = 105) (𝑉2 = 2) (𝑊2 = 6) 𝑈𝑛𝑖𝑡 3 ( i = 3 ) (α3 = 0.3 × 10−5 ) (β3 = 105) (𝑉3 = 3) (𝑊3 = 8) 𝑈𝑛𝑖𝑡 4 ( i = 4 ) (α4 = 2.3 × 10−5 ) (β4 = 105) (𝑉4 = 2) (𝑊4 = 7)

Model A: Addition to increase the reliability of the problem:

Maximize system reliability by adding redundant units. To improve system reliability. Approach: increasing the number of units

Code and solution:

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants for model A1

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

# Reliability function

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(-alpha\_i \* t \* (np.log(t))\*\*beta\_i)

# System reliability function

def R\_sys(n, t, alpha, beta):

    R = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    return np.prod([1 - (1 - R[i])\*\*n[i] for i in range(4)])

# Objective function to maximize

def objective(n, t, alpha, beta):

    return -R\_sys(n, t, alpha, beta)

# Constraints

def constraint1(n):

    return 400 - np.sum([alpha[i] \* t \* (n[i] + np.exp(0.25 \* n[i])) \* (np.log(t))\*\*beta[i] for i in range(4)])

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(-0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(t, alpha, beta), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Check and adjust if the rounded solution violates any constraints

def is\_feasible(n):

    return (constraint1(n) >= 0 and constraint2(n) >= 0 and constraint3(n) >= 0)

if not is\_feasible(n\_opt):

    n\_opt = solution.x.astype(int)

    n\_opt = np.clip(n\_opt, 1, 10)

    while not is\_feasible(n\_opt):

        for i in range(4):

            if constraint1(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint2(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint3(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

R\_opt = R\_sys(n\_opt, t, alpha, beta)

# Prepare the results table

data = {

    'ni': n\_opt,

    'R(t)': [R\_i(t, alpha[i], beta[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['Rsys(t)'] = R\_opt

# Display results

print("Optimization Results for Model A1 with Integer Constraints:")

print(df)

# Plotting the results

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model A1 with Integer Constraints:

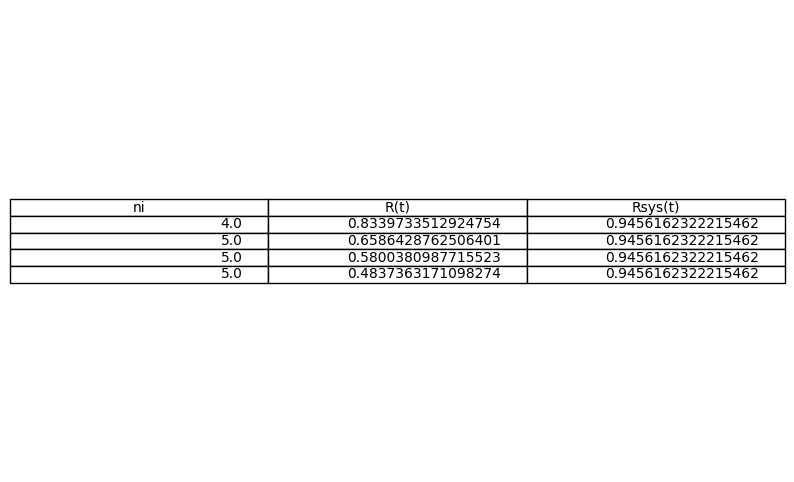
ni R(t) Rsys(t)

0 4 0.833973 0.945616

1 5 0.658643 0.945616

2 5 0.580038 0.945616

3 5 0.483736 0.945616



A2

Model A2 Now just change the 𝛽𝑖s as follows and repeat the calculation results:

β1′=1.28, β2′=4.77, β3′=2.13, β4′=1.75

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants for model A2

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta\_prime = [1.28, 4.77, 2.13, 1.75]  # Changed beta values for A2

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 100 # Assuming a fixed value for t

# Reliability function

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(-alpha\_i \* t \* (np.log(t))\*\*beta\_i)

# System reliability function

def R\_sys(n, t, alpha, beta):

    R = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    return np.prod([1 - (1 - R[i])\*\*n[i] for i in range(4)])

# Objective function to maximize

def objective(n, t, alpha, beta):

    return -R\_sys(n, t, alpha, beta)

# Constraints

def constraint1(n):

    return 400 - np.sum([alpha[i] \* t \* (n[i] + np.exp(0.25 \* n[i])) \* (np.log(t))\*\*beta\_prime[i] for i in range(4)])

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(-0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(t, alpha, beta\_prime), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Check and adjust if the rounded solution violates any constraints

def is\_feasible(n):

    return (constraint1(n) >= 0 and constraint2(n) >= 0 and constraint3(n) >= 0)

if not is\_feasible(n\_opt):

    n\_opt = solution.x.astype(int)

    n\_opt = np.clip(n\_opt, 1, 10)

    while not is\_feasible(n\_opt):

        for i in range(4):

            if constraint1(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint2(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint3(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

R\_opt = R\_sys(n\_opt, t, alpha, beta\_prime)

# Prepare the results table

data = {

    'ni': n\_opt,

    'R(t)': [R\_i(t, alpha[i], beta\_prime[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['Rsys(t)'] = R\_opt

# Display results

print("Optimization Results for Model A2 with Integer Constraints:")

print(df)

# Plotting the results

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model A2 with Integer Constraints:

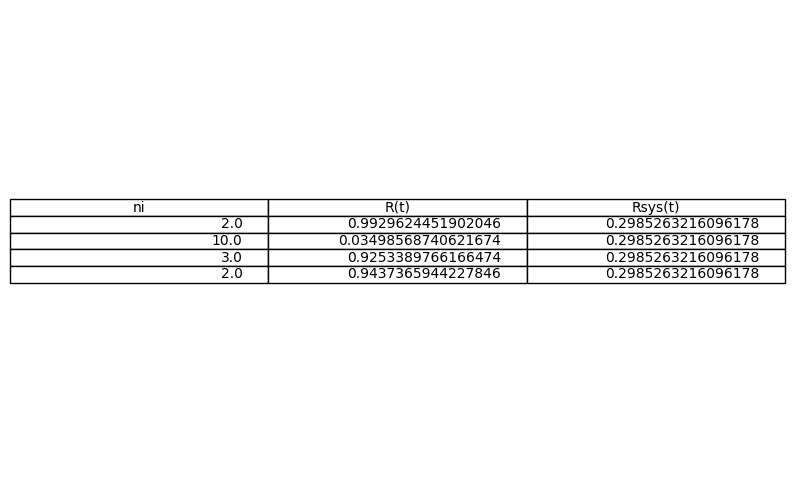
ni R(t) Rsys(t)

0 2 0.992962 0.298526

1 10 0.034986 0.298526

2 3 0.925339 0.298526

3 2 0.943737 0.298526



A3

Define 𝑅𝑖(𝑡)=𝑒−𝜆𝑖∗t and 𝜆𝑖=𝛼𝑖𝛽𝑖, for example for the first unit we have: 1.𝐹𝑜𝑟(α1=10−5),(β1=1.5): λ1=10−51.5=0.66×10− 5 2. 𝐹𝑜𝑟(α2=2.3×10−5),(β2=1.5): λ2=2.3×10−51.5=1.53×10−5] 3.𝐹𝑜𝑟(α3=0.3×10−5),(β3= 1.5): λ3=0.3×10−51.5=0.2×10−5 4.𝐹𝑜𝑟(α4=2.3×10−5),(β4=1.5): λ4=2.3×10−51.5=1.53×10−5

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants for model A3

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta = [1.5, 1.5, 1.5, 1.5]

lambda\_ = [alpha[i] / beta[i] for i in range(4)]  # Calculate lambda\_i

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 1000 # Assuming a fixed value for t

# Reliability function for Exponential distribution

def R\_i\_exp(t, lambda\_i):

    return np.exp(-lambda\_i \* t)

# System reliability function

def R\_sys(n, t, lambda\_):

    R = [R\_i\_exp(t, lambda\_[i]) for i in range(4)]

    return np.prod([1 - (1 - R[i])\*\*n[i] for i in range(4)])

# Objective function to maximize

def objective(n, t, lambda\_):

    return -R\_sys(n, t, lambda\_)

# Constraints

def constraint1(n):

    return 400 - np.sum([lambda\_[i] \* t \* (n[i] + np.exp(0.25 \* n[i])) for i in range(4)])

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(-0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(t, lambda\_), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Check and adjust if the rounded solution violates any constraints

def is\_feasible(n):

    return (constraint1(n) >= 0 and constraint2(n) >= 0 and constraint3(n) >= 0)

if not is\_feasible(n\_opt):

    n\_opt = solution.x.astype(int)

    n\_opt = np.clip(n\_opt, 1, 10)

    while not is\_feasible(n\_opt):

        for i in range(4):

            if constraint1(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint2(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint3(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

R\_opt = R\_sys(n\_opt, t, lambda\_)

# Prepare the results table

data = {

    'ni': n\_opt,

    'R(t)': [R\_i\_exp(t, lambda\_[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['Rsys(t)'] = R\_opt

# Display results

print("Optimization Results for Model A3 with Integer Constraints:")

print(df)

# Plotting the results

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model A3 with Integer Constraints:

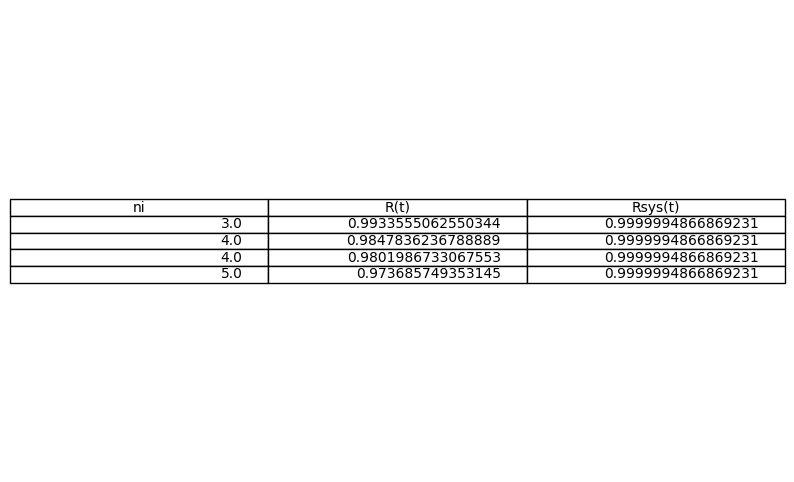
ni R(t) Rsys(t)

0 3 0.993356 0.999999

1 4 0.984784 0.999999

2 4 0.980199 0.999999

3 5 0.973686 0.999999



A4

Run model A4 and model A3 this time with 𝛽i of model A2, that means you have: 1.𝐹𝑜𝑟(α1=10−5),(β1′=1.28): λ1=10−51.28=0.78×10−5 2.𝐹𝑜𝑟(α2 =2.3×10−5),(β2′=4.77): λ2=2.3×10−54.77=0.48×10−5 3.𝐹𝑜𝑟(α3=0.3×10−5),(β3′=2.13): λ3= 0.3×10−52.13=0.14×10−5 4.𝐹𝑜𝑟(α4=2.3×10−5),(β4′=1.75): λ4=2.3×10−51.75=1.31×10−5

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants for model A4

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta\_prime = [1.28, 4.77, 2.13, 1.75]  # Changed beta values for A4

lambda\_ = [alpha[i] / beta\_prime[i] for i in range(4)]  # Calculate lambda\_i for A4

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 1000 # Assuming a fixed value for t

# Reliability function for Exponential distribution

def R\_i\_exp(t, lambda\_i):

    return np.exp(-lambda\_i \* t)

# System reliability function

def R\_sys(n, t, lambda\_):

    R = [R\_i\_exp(t, lambda\_[i]) for i in range(4)]

    return np.prod([1 - (1 - R[i])\*\*n[i] for i in range(4)])

# Objective function to maximize

def objective(n, t, lambda\_):

    return -R\_sys(n, t, lambda\_)

# Constraints

def constraint1(n):

    return 400 - np.sum([lambda\_[i] \* t \* (n[i] + np.exp(0.25 \* n[i])) for i in range(4)])

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(-0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(t, lambda\_), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Check and adjust if the rounded solution violates any constraints

def is\_feasible(n):

    return (constraint1(n) >= 0 and constraint2(n) >= 0 and constraint3(n) >= 0)

if not is\_feasible(n\_opt):

    n\_opt = solution.x.astype(int)

    n\_opt = np.clip(n\_opt, 1, 10)

    while not is\_feasible(n\_opt):

        for i in range(4):

            if constraint1(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint2(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint3(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

R\_opt = R\_sys(n\_opt, t, lambda\_)

# Prepare the results table

data = {

    'ni': n\_opt,

    'R(t)': [R\_i\_exp(t, lambda\_[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['Rsys(t)'] = R\_opt

# Display results

print("Optimization Results for Model A4 with Integer Constraints:")

print(df)

# Plotting the results

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model A4 with Integer Constraints:

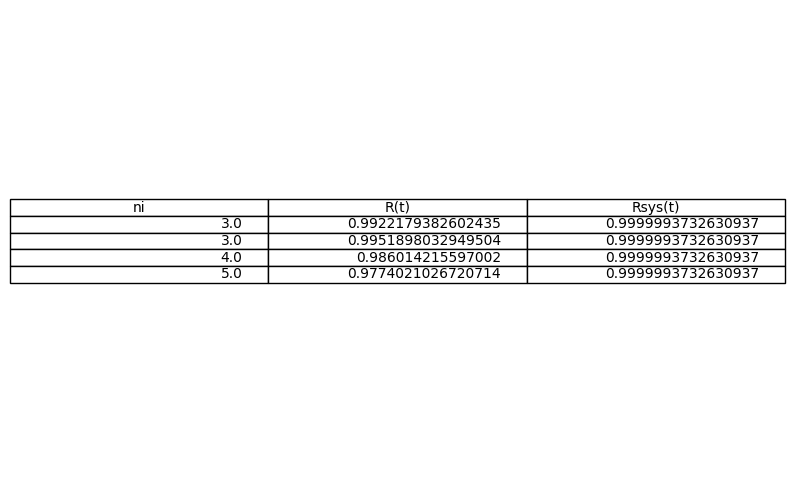
ni R(t) Rsys(t)

0 3 0.992218 0.999999

1 3 0.995190 0.999999

2 4 0.986014 0.999999

3 5 0.977402 0.999999



A5

𝑅standby(𝑡)=𝑒−𝜆𝑖𝑡(1+𝜆𝑖𝑡+⋯)

𝜆1=0.66×10−5, 𝜆2=1.53×10−5, 𝜆3=0.2×10−5, 𝜆4=1.53×10−5

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

from math import factorial

# Constants for model A5

lambda\_ = [0.66e-5, 1.53e-5, 0.2e-5, 1.53e-5]  # Given lambda values

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 1000  # Assuming a fixed value for t

# Reliability function for Cold Standby

def R\_i\_standby(t, lambda\_i, n\_i):

    n\_i = int(n\_i)  # Ensure n\_i is an integer

    if n\_i == 1:

        return np.exp(-lambda\_i \* t)

    else:

        R = np.exp(-lambda\_i \* t)

        summation = sum((lambda\_i \* t)\*\*k / factorial(k) for k in range(n\_i))

        return R \* summation

# System reliability function

def R\_sys(n, t, lambda\_):

    return np.prod([R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)])

# Objective function to maximize

def objective(n, t, lambda\_):

    return -R\_sys(n, t, lambda\_)

# Constraints

def constraint1(n):

    return 400 - np.sum([lambda\_[i] \* t \* (n[i] + np.exp(0.25 \* n[i])) for i in range(4)])

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(-0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(t, lambda\_), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Check and adjust if the rounded solution violates any constraints

def is\_feasible(n):

    return (constraint1(n) >= 0 and constraint2(n) >= 0 and constraint3(n) >= 0)

if not is\_feasible(n\_opt):

    n\_opt = solution.x.astype(int)

    n\_opt = np.clip(n\_opt, 1, 10)

    while not is\_feasible(n\_opt):

        for i in range(4):

            if constraint1(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint2(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint3(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

R\_opt = R\_sys(n\_opt, t, lambda\_)

# Prepare the results table

data = {

    'ni': n\_opt,

    'R(t)': [R\_i\_standby(t, lambda\_[i], n\_opt[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['Rsys(t)'] = R\_opt

# Display results

print("Optimization Results for Model A5 with Integer Constraints:")

print(df)

# Plotting the results

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model A5 with Integer Constraints:

ni R(t) Rsys(t)

0 1 0.993422 0.961558

1 1 0.984816 0.961558

2 1 0.998002 0.961558

3 1 0.984816 0.961558

A6

Run model A6 and model A5 with new 𝜆𝑖s, that is, with 𝜆𝑖s of model A4, which are: λ1=0.78×10−5, λ2=0.48×10−5, λ3=0.14×10−5, λ4=1.31×10 -5

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

from math import factorial

# Constants for model A6

lambda\_ = [0.78e-5, 0.48e-5, 0.14e-5, 1.31e-5]  # New lambda values

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 1000 # Assuming a fixed value for t

# Reliability function for Cold Standby

def R\_i\_standby(t, lambda\_i, n\_i):

    n\_i = int(n\_i)  # Ensure n\_i is an integer

    if n\_i == 1:

        return np.exp(-lambda\_i \* t)

    else:

        R = np.exp(-lambda\_i \* t)

        summation = sum((lambda\_i \* t)\*\*k / factorial(k) for k in range(n\_i))

        return R \* summation

# System reliability function

def R\_sys(n, t, lambda\_):

    return np.prod([R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)])

# Objective function to maximize

def objective(n, t, lambda\_):

    return -R\_sys(n, t, lambda\_)

# Constraints

def constraint1(n):

    return 400 - np.sum([lambda\_[i] \* t \* (n[i] + np.exp(0.25 \* n[i])) for i in range(4)])

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(-0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(t, lambda\_), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Check and adjust if the rounded solution violates any constraints

def is\_feasible(n):

    return (constraint1(n) >= 0 and constraint2(n) >= 0 and constraint3(n) >= 0)

if not is\_feasible(n\_opt):

    n\_opt = solution.x.astype(int)

    n\_opt = np.clip(n\_opt, 1, 10)

    while not is\_feasible(n\_opt):

        for i in range(4):

            if constraint1(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint2(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

            elif constraint3(n\_opt) < 0:

                n\_opt[i] = max(n\_opt[i] - 1, 1)

R\_opt = R\_sys(n\_opt, t, lambda\_)

# Prepare the results table

data = {

    'ni': n\_opt,

    'R(t)': [R\_i\_standby(t, lambda\_[i], n\_opt[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['Rsys(t)'] = R\_opt

# Display results

print("Optimization Results for Model A6 with Integer Constraints:")

print(df)

# Plotting the results

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model A6 with Integer Constraints:

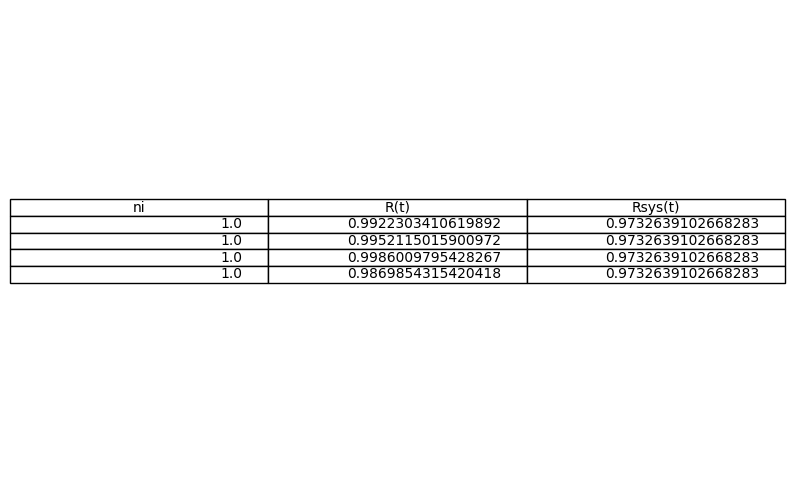
ni R(t) Rsys(t)

0 1 0.992230 0.973264

1 1 0.995212 0.973264

2 1 0.998601 0.973264

3 1 0.986985 0.973264



B

𝑅sys(𝑡)=Π[1−(1−𝑅𝑖(𝑡))𝑛𝑖]4𝑖=1≥0.8⋅𝑅sys∗

It becomes our optimization

Min costsys(𝑡)=Σα𝑖(−𝑡ln(𝑅𝑖(𝑡)))β𝑖4𝑖=1⋅(𝑛𝑖+𝑒0.25𝑛𝑖)

Π[1−(1−𝑅𝑖(𝑡))𝑛𝑖]4𝑖=1≥0.8⋅𝑅sys∗

Σ𝑉𝑖4𝑖=1⋅𝑛𝑖2≤250]

Σ𝑊𝑖4𝑖=1⋅𝑛𝑖⋅𝑒0.25𝑛𝑖≤500

𝑅𝑖(𝑡)=𝑒−(𝑡α𝑖)β

∀𝑖 = 1,2,3,4

10≥𝑛𝑖≥1

∀𝑖 = 1,2,3,4

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

from math import factorial

# Constants for model B1

lambda\_ = [0.78e-5, 0.48e-5, 0.14e-5, 1.31e-5]  # Given lambda values for model A4

alpha = [1e-5, 2.3e-5, 0.3e-5, 2.3e-5]  # Alpha values

beta\_prime = [1.28, 4.77, 2.13, 1.75]  # Beta values for model A4

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 1000  # Assuming a fixed value for t

R\_sys\_opt = 0.95  # Example optimal system reliability from model A (for demonstration)

# Define the cost function

def cost\_function(n, t, alpha, beta):

    return np.sum([alpha[i] \* ((-t / np.log(R\_i\_standby(t, lambda\_[i], n[i]))) \*\* beta[i]) \* (n[i] + np.exp(0.25 \* n[i])) for i in range(4)])

# Reliability function for Cold Standby

def R\_i\_standby(t, lambda\_i, n\_i):

    n\_i = int(n\_i)  # Ensure n\_i is an integer

    if n\_i == 1:

        return np.exp(-lambda\_i \* t)

    else:

        R = np.exp(-lambda\_i \* t)

        summation = sum((lambda\_i \* t)\*\*k / factorial(k) for k in range(n\_i))

        return R \* summation

# System reliability function

def R\_sys(n, t, lambda\_):

    return np.prod([R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)])

# Constraints

def constraint1(n):

    return R\_sys(n, t, lambda\_) - 0.8 \* R\_sys\_opt

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(-0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(cost\_function, n0, args=(t, alpha, beta\_prime), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability and cost

R\_opt = R\_sys(n\_opt, t, lambda\_)

cost\_opt = cost\_function(n\_opt, t, alpha, beta\_prime)

# Display results

print("Optimization Results for Model B1 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability: {R\_opt:.6f}")

print(f"System Cost: {cost\_opt:.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'R(t)': [R\_i\_standby(t, lambda\_[i], n\_opt[i]) for i in range(4)],

    'Cost': [alpha[i] \* ((-t / np.log(R\_i\_standby(t, lambda\_[i], n\_opt[i]))) \*\* beta\_prime[i]) \* (n\_opt[i] + np.exp(0.25 \* n\_opt[i])) for i in range(4)]

}

df = pd.DataFrame(data)

df['Rsys(t)'] = R\_opt

df['Total Cost'] = cost\_opt

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model B1 with Integer Constraints:

Optimized n values: [1 1 1 1]

System Reliability: 0.999729

System Cost: 1.232844e+21

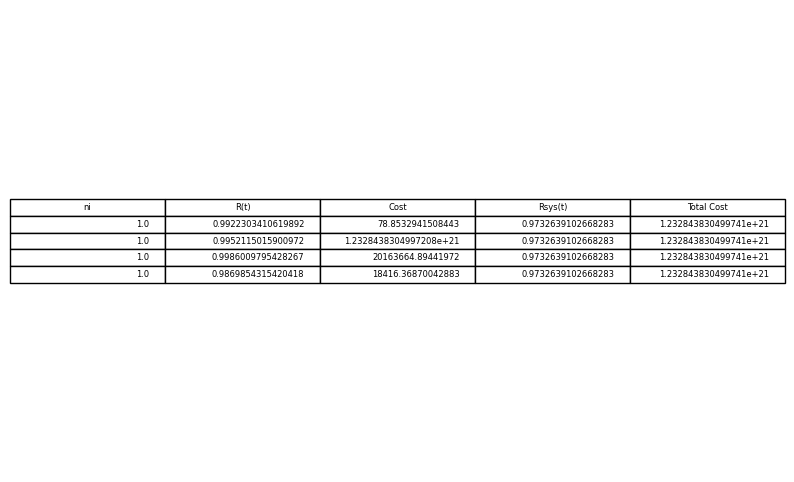
ni R(t) Cost Rsys(t) Total Cost

0 1 0.999922 7.885329e+01 0.999729 1.232844e+21

1 1 0.999952 1.232844e+21 0.999729 1.232844e+21

2 1 0.999986 2.016366e+07 0.999729 1.232844e+21

3 1 0.999869 1.841637e+04 0.999729 1.232844e+21



C

Model C1 to C8 We have assumed in our modeling that the i-th unit follows the Weiball or exponential distribution function with known parameters.

It means that we currently have a 4-unit series system whose reliability is 𝑅sys,now(𝑡)=Π𝑅𝑖(𝑡)4

𝑖=1

This means that the technology is available and we can buy the units from the equipment market. The question we were looking for was how to

In addition, we can convert 𝑅sys,now(𝑡) to 𝑅sys,new(𝑡) and to do this, we consider the minimum cost of Hajj and weight.

We got the answer to our problem was 𝑛1∗ to 𝑛4∗

Another method that is common in the subject literature is that we are basically in the system design phase, our optimal model needs a structure

To determine the optimality of the series-parallel system, including this assignment is to give us 𝑅i(𝑡) along the optimization model.

That is, in addition to the reliability of the units, the modeler will also give us the optimal model. This method may seem strange.

Because the units are already known, but in the sense that in the cost function, we actually have 𝛼𝑖 and 𝛽𝑖, which are in some way the specifications of the length function.

Now we have the reliability according to the reliability because we have 𝛼𝑖 and 𝛽𝑖 of course if we assume Weiball but The reliability of each

We don't have the unit at time t, so we use 𝛼𝑖 and 𝛽𝑖 in the cost limit function, but in the same function and the objective function of our variable 𝑅i(𝑡)

Decision variables are not predetermined numbers. This is the method that is followed in the current literature and many articles

With this method, Hand has been released so far, so models C1 to C8 are the same as models A1 to A8, 8 or B1 to B8, with the only difference

𝑅i(𝑡) are also decision variables, so the optimal solution of such models will be 𝑛1∗

to 𝑛4∗

plus 𝑅i(𝑡)

∀𝑖 = 1,2,3,4

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

from math import factorial

# Constants

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 1000  # Assuming a fixed value for t

# Calculate lambda values based on alpha and beta

lambda\_ = [alpha[i] / beta[i] for i in range(4)]

# Mean Time To Failure (MTTF) function for Cold Standby

def MTTF\_i(lambda\_i, n\_i):

    n\_i = int(n\_i)

    if n\_i == 1:

        return 1 / lambda\_i

    else:

        return (1 / lambda\_i) \* sum((lambda\_i \* t)\*\*k / factorial(k) for k in range(n\_i))

# System MTTF function

def MTTF\_sys(n, lambda\_):

    return np.sum([MTTF\_i(lambda\_[i], n[i]) for i in range(4)])

# Reliability function for Cold Standby

def R\_i\_standby(t, lambda\_i, n\_i):

    n\_i = int(n\_i)  # Ensure n\_i is an integer

    if n\_i == 1:

        return np.exp(-lambda\_i \* t)

    else:

        R = np.exp(-lambda\_i \* t)

        summation = sum((lambda\_i \* t)\*\*k / factorial(k) for k in range(n\_i))

        return R \* summation

# Cost function

def cost\_function(n, alpha, beta):

    total\_cost = 0

    for i in range(4):

        R = R\_i\_standby(t, lambda\_[i], n[i])

        if R > 0:

            total\_cost += alpha[i] \* ((-t / np.log(R)) \*\* beta[i]) \* (n[i] + np.exp(0.25 \* n[i]))

    return total\_cost

# Constraints

def constraint1(n):

    return MTTF\_sys(n, lambda\_) - 0.8 \* MTTF\_sys(np.ones(4), lambda\_)

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [2, 2, 2, 2]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Debugging: Print initial guess and constraint values

print(f"Initial guess: {n0}")

print(f"Initial cost: {cost\_function(n0, alpha, beta)}")

print(f"Initial constraint1 (MTTF constraint): {constraint1(n0)}")

print(f"Initial constraint2 (volume constraint): {constraint2(n0)}")

print(f"Initial constraint3 (weight constraint): {constraint3(n0)}")

# Solving the optimization problem

solution = minimize(cost\_function, n0, args=(alpha, beta), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system MTTF and cost

MTTF\_opt = MTTF\_sys(n\_opt, lambda\_)

cost\_opt = cost\_function(n\_opt, alpha, beta)

# Display results

print("Optimization Results for Model C1 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System MTTF: {MTTF\_opt:.6f}")

print(f"System Cost: {cost\_opt:.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'MTTF': [MTTF\_i(lambda\_[i], n\_opt[i]) for i in range(4)],

    'Cost': [alpha[i] \* ((-t / np.log(R\_i\_standby(t, lambda\_[i], n\_opt[i]))) \*\* beta[i]) \* (n\_opt[i] + np.exp(0.25 \* n\_opt[i])) for i in range(4)]

}

df = pd.DataFrame(data)

df['MTTF\_sys'] = MTTF\_opt

df['Total Cost'] = cost\_opt

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Initial guess: [2, 2, 2, 2]

Initial cost: 15157043.100811373

Initial constraint1 (MTTF constraint): 64543.47826086954

Initial constraint2 (volume constraint): 210

Initial constraint3 (weight constraint): 407.6716088407928

Optimization Results for Model C1 with Integer Constraints:

Optimized n values: [2 2 2 2]

System MTTF: 306717.391304

System Cost: 15157043.100811

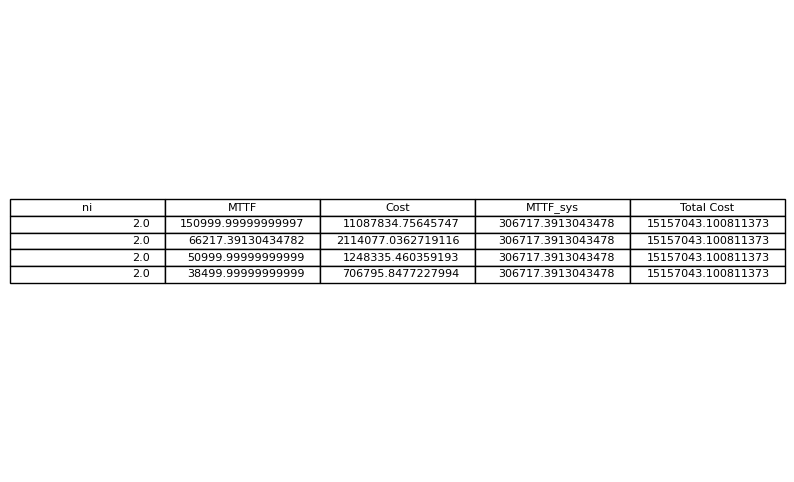
ni MTTF Cost MTTF\_sys Total Cost

0 2 151000.000000 1.108783e+07 306717.391304 1.515704e+07

1 2 66217.391304 2.114077e+06 306717.391304 1.515704e+07

2 2 51000.000000 1.248335e+06 306717.391304 1.515704e+07

3 2 38500.000000 7.067958e+05 306717.391304 1.515704e+07



D1

Min VTTFsys=∑{(∂𝑅sys(𝑡) ∂𝑅𝑖(𝑡) ) 2 ⋅Var(𝑅𝑖(𝑡))}

s𝑡∶All previous constraints

MTTF𝑖=α𝑖 ⋅Γ(1+1 β𝑖 )

∂𝑅sys(𝑡)/∂𝑅𝑖(𝑡)=∂/∂𝑅𝑖(𝑡)Π[1−(1−𝑅𝑖(𝑡))𝑛𝑖]

import numpy as np

from scipy.optimize import minimize

from scipy.special import gamma

import pandas as pd

import matplotlib.pyplot as plt

alpha = [1e-5, 2.3e-5, 0.3e-5, 2.3e-5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 2]

w = [6, 6, 8, 7]

t = 10

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(-(t / alpha\_i)\*\*beta\_i)

def R\_sys(n, t, alpha, beta):

    R = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    return np.prod([1 - (1 - R[i])\*\*n[i] for i in range(4)])

def safe\_log(x):

    return np.log(np.maximum(x, 1e-10))

def cost\_constraint\_safe(n):

    cost = sum(alpha[i] \* ((-safe\_log(R\_i(t, alpha[i], beta[i])))\*\*beta[i]) \* (n[i] + np.exp(0.25 \* n[i])) for i in range(4))

    return 400 - cost

def volume\_constraint(n):

    volume = sum(v[i] \* n[i]\*\*2 for i in range(4))

    return 250 - volume

def weight\_constraint(n):

    weight = sum(w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4))

    return 500 - weight

bounds = [(1, 10) for \_ in range(4)]

n0 = [1, 1, 1, 1]

cons\_safe = [{'type': 'ineq', 'fun': cost\_constraint\_safe},

             {'type': 'ineq', 'fun': volume\_constraint},

             {'type': 'ineq', 'fun': weight\_constraint}]

solution\_safe = minimize(lambda n: -R\_sys(n, t, alpha, beta), n0, method='SLSQP', bounds=bounds, constraints=cons\_safe)

n\_opt\_safe = np.round(solution\_safe.x).astype(int)

def VTTF\_sys(n, alpha, beta):

    variances = [(alpha[i]\*\*2 \* gamma(1 + 2 / beta[i]) - (alpha[i] \* gamma(1 + 1 / beta[i]))\*\*2) \* n[i] for i in range(4)]

    return sum(variances)

VTTF\_sys\_value = VTTF\_sys(n\_opt\_safe, alpha, beta)

data\_safe = {

    'ni': n\_opt\_safe,

    'MTTF': [alpha[i] \* gamma(1 + 1 / beta[i]) for i in range(4)],

    'VTTF': [alpha[i]\*\*2 \* gamma(1 + 2 / beta[i]) - (alpha[i] \* gamma(1 + 1 / beta[i]))\*\*2 for i in range(4)],

    'VTTF\_sys': [VTTF\_sys\_value] \* 4

}

df\_safe = pd.DataFrame(data\_safe)

print("Optimization Results for Model D1:")

print(df\_safe)

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df\_safe.values, colLabels=df\_safe.columns, loc='center')

ax.axis('off')

plt.show()

ptimization Results for Model D1:

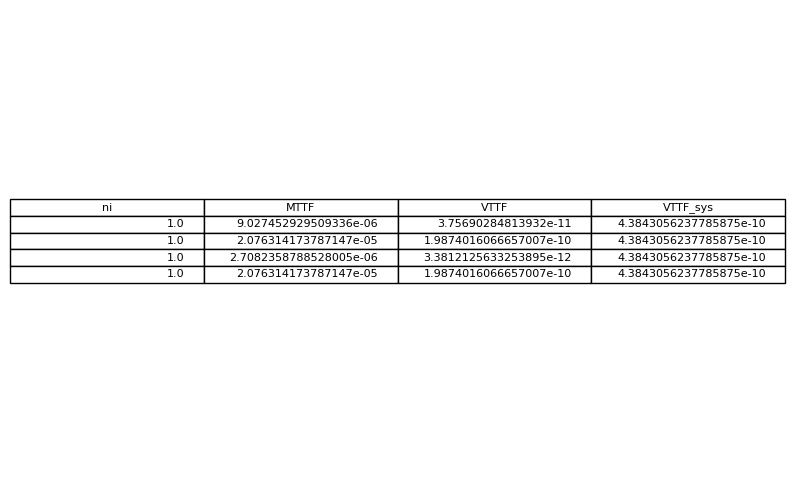
ni MTTF VTTF VTTF\_sys

0 1 0.000009 3.756903e-11 4.384306e-10

1 1 0.000021 1.987402e-10 4.384306e-10

2 1 0.000003 3.381213e-12 4.384306e-10

3 1 0.000021 1.987402e-10 4.384306e-10



D2

MTTFsys=∫𝑅sys(𝑡) 𝑑𝑡=∫Π[1−(1−𝑅𝑖(𝑡))𝑛𝑖]4𝑖 𝑑𝑡

ST. previous constraints

import numpy as np

from scipy.optimize import minimize

from scipy.special import gamma

import pandas as pd

import matplotlib.pyplot as plt

# Constants

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

# Reliability function

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(-(t / alpha\_i)\*\*beta\_i)

# MTTF for cold standby redundancy

def MTTF\_i(alpha\_i, beta\_i, n\_i):

    return n\_i \* alpha\_i \* gamma(1 + 1 / beta\_i)

# System MTTF function

def MTTF\_sys(n, alpha, beta):

    mttf\_values = [MTTF\_i(alpha[i], beta[i], n[i]) for i in range(4)]

    return min(mttf\_values)

# Constraints

def constraint1(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint2(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2}]

# Solving the optimization problem

solution = minimize(lambda n: -MTTF\_sys(n, alpha, beta), n0, method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system MTTF

MTTF\_opt = MTTF\_sys(n\_opt, alpha, beta)

# Display results

print("Optimization Results for Model D2 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System MTTF: {MTTF\_opt:.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'MTTF': [MTTF\_i(alpha[i], beta[i], n\_opt[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['MTTF\_sys'] = MTTF\_opt

# Display the DataFrame

print(df)

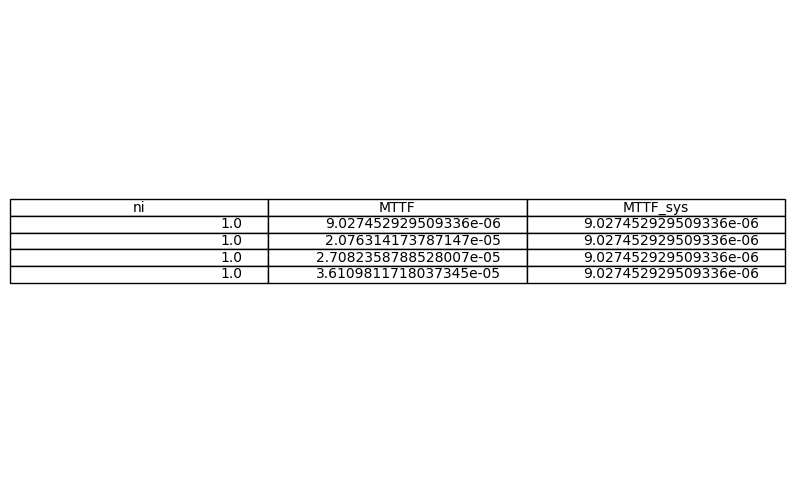
# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()



D3

max{MTTFsys}=max[(Σ{min{λ𝑖𝑗}})−1]=max{Σλ𝑖}

𝑆𝑡.

λ𝑖𝑗=α𝑖𝑗/β𝑖𝑗

𝑅𝑖(𝑡)=𝑒−λ𝑖⋅𝑡

import numpy as np

from scipy.optimize import minimize

from scipy.special import gamma

import pandas as pd

import matplotlib.pyplot as plt

# Constants

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

# Define the initial guess for λ\_{ij}

initial\_guess = np.array([1e-5, 2.3e-5, 3e-5, 4e-5])

# Define the system MTTF function

def MTTF\_sys(lambdas):

    return 1 / np.sum(lambdas)

# Constraints

def constraint1(lambdas):

    return 250 - np.sum([v[i] \* (1/lambdas[i])\*\*2 for i in range(4)])

def constraint2(lambdas):

    return 500 - np.sum([w[i] \* (1/lambdas[i]) \* np.exp(0.25 \* (1/lambdas[i])) for i in range(4)])

# Bounds for λ\_{ij}

bounds = [(1e-7, 1e-3) for \_ in range(4)]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2}]

# Solving the optimization problem

solution = minimize(lambda x: -MTTF\_sys(x), initial\_guess, method='SLSQP', bounds=bounds, constraints=cons)

# Extract results

lambda\_opt = solution.x

# Recalculate system MTTF

MTTF\_opt = MTTF\_sys(lambda\_opt)

# Display results

print("Optimization Results for Model D3 with Integer Constraints:")

print(f"Optimized λ values: {lambda\_opt}")

print(f"System MTTF: {MTTF\_opt:.6f}")

# If desired, visualize the results in a table

data = {

    'λ': lambda\_opt,

    'MTTF': [1/lambda\_opt[i] for i in range(4)]

}

df = pd.DataFrame(data)

df['MTTF\_sys'] = MTTF\_opt

# Display the DataFrame

print(df)

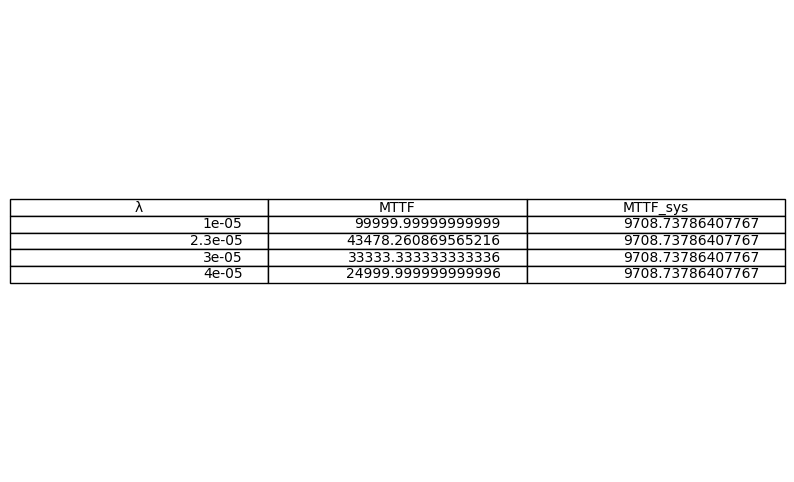
# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()



E1

𝑀inΣ𝐶𝑖̂⋅𝑛𝑖⋅[1−(1−𝑅𝑖(𝑡))𝑛𝑖]

𝑆𝑡.

Π[1−(1−𝑅𝑖(𝑡))𝑛𝑖]4≥0.8⋅𝑅sys

Σ𝑉𝑖4𝑖=1⋅𝑛𝑖2≤250

Σ𝑊𝑖⋅𝑛𝑖⋅𝑒\*\*0.25𝑛𝑖≤500

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

R\_sys\_opt = 0.95  # Example optimal system reliability from model A (for demonstration)

# Reliability function

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(-(t / alpha\_i)\*\*beta\_i)

# Partial derivative of R\_sys with respect to R\_i

def partial\_R\_sys(n, alpha, beta, t):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    partials = []

    for i in range(4):

        partial = np.prod([(1 - (1 - Rs[j])\*\*n[j]) for j in range(4) if j != i])

        partial \*= n[i] \* (1 - Rs[i])\*\*(n[i] - 1)

        partials.append(partial)

    return partials

# Criticality index

def criticality\_index(n, alpha, beta, t):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    R\_sys\_value = R\_sys(n, t, alpha, beta)

    partials = partial\_R\_sys(n, alpha, beta, t)

    criticalities = [partials[i] \* Rs[i] / R\_sys\_value for i in range(4)]

    return criticalities

# Objective function to maximize

def objective(n, alpha, beta, t):

    criticalities = criticality\_index(n, alpha, beta, t)

    return -np.sum(criticalities)

# System reliability function

def R\_sys(n, t, alpha, beta):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Constraints

def constraint1(n):

    return R\_sys(n, t, alpha, beta) - 0.8 \* R\_sys\_opt

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(alpha, beta, t), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability

R\_sys\_optimal = R\_sys(n\_opt, t, alpha, beta)

# Calculate criticality indices for the optimal solution

criticalities\_opt = criticality\_index(n\_opt, alpha, beta, t)

# Display results

print("Optimization Results for Model E3 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

print(f"Sum of Criticality Indices: {np.sum(criticalities\_opt):.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'Reliability': [R\_i(t, alpha[i], beta[i]) for i in range(4)],

    'Criticality Index': criticalities\_opt

}

df = pd.DataFrame(data)

df['System Reliability'] = R\_sys\_optimal

df['Sum of Criticality Indices'] = np.sum(criticalities\_opt)

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model E1 with Integer Constraints:

Optimized n values: [1 1 1 1]

System Reliability: 0.000001

E2

MaxΣ∂𝑅sys(𝑡)/∂𝑅𝑖(𝑡)=∂𝑅sys(𝑡)/∂𝑅1(𝑡)+⋯+∂𝑅sys(𝑡)/∂𝑅4(𝑡)

𝑆𝑡.∶Previous constraints

import numpy as np

from scipy.optimize import minimize

# Constants

alpha = [1e+5, 2.3e+5, 3e+6, 2.3e+5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 2]

w = [6, 6, 8, 7]

t = 10  # Assuming a fixed value for t

R\_sys\_opt = 0.95  # Example optimal system reliability from model A (for demonstration)

# Reliability function

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(-(t / alpha\_i)\*\*beta\_i)

# System reliability function

def R\_sys(n, t, alpha, beta):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Derivative of system reliability with respect to R\_i

def dR\_sys\_dR\_i(n, t, alpha, beta, i):

    Rs = [R\_i(t, alpha[j], beta[j]) for j in range(4)]

    product\_terms = [1 - (1 - Rs[j])\*\*n[j] for j in range(4) if j != i]

    partial\_derivative = np.prod(product\_terms) \* n[i] \* (1 - Rs[i])\*\*(n[i] - 1)

    return partial\_derivative

# Objective function to maximize

def objective(n, alpha, beta, t):

    sum\_burnhaven = 0

    for i in range(4):

        sum\_burnhaven += dR\_sys\_dR\_i(n, t, alpha, beta, i)

    return -sum\_burnhaven  # Minimize the negative to maximize the positive

# Constraints

def constraint1(n):

    return R\_sys(n, t, alpha, beta) - 0.8 \* R\_sys\_opt

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(alpha, beta, t), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability and cost

R\_sys\_optimal = R\_sys(n\_opt, t, alpha, beta)

total\_cost\_opt = objective(n\_opt, alpha, beta, t)

# Display results

print("Optimization Results for Model E2 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

print(f"Total Cost: {-total\_cost\_opt:.6f}")  # Negate to show the positive value

Optimization Results for Model E2 with Integer Constraints:

Optimized n values: [1 1 1 1]

System Reliability: 0.998422

Total Cost: 3.995265

E3

MaxΣ𝐼𝑖CS

St. :Previous constraints

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants

alpha = [1e+5, 2.3e+5, 3e+5, 4e+5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 1000  # Assuming a fixed value for t

R\_sys\_opt = 0.95  # Example optimal system reliability from model A (for demonstration)

# Reliability function

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(-(t / alpha\_i)\*\*beta\_i)

# Partial derivative of R\_sys with respect to R\_i

def partial\_R\_sys(n, alpha, beta, t):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    partials = []

    for i in range(4):

        partial = np.prod([(1 - (1 - Rs[j])\*\*n[j]) for j in range(4) if j != i])

        partial \*= n[i] \* (1 - Rs[i])\*\*(n[i] - 1)

        partials.append(partial)

    return partials

# Criticality index

def criticality\_index(n, alpha, beta, t):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    R\_sys\_value = R\_sys(n, t, alpha, beta)

    partials = partial\_R\_sys(n, alpha, beta, t)

    criticalities = [partials[i] \* Rs[i] / R\_sys\_value for i in range(4)]

    return criticalities

# Objective function to maximize

def objective(n, alpha, beta, t):

    criticalities = criticality\_index(n, alpha, beta, t)

    return -np.sum(criticalities)

# System reliability function

def R\_sys(n, t, alpha, beta):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Constraints

def constraint1(n):

    return R\_sys(n, t, alpha, beta) - 0.8 \* R\_sys\_opt

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective, n0, args=(alpha, beta, t), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability

R\_sys\_optimal = R\_sys(n\_opt, t, alpha, beta)

# Calculate criticality indices for the optimal solution

criticalities\_opt = criticality\_index(n\_opt, alpha, beta, t)

# Display results

print("Optimization Results for Model E3 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

print(f"Sum of Criticality Indices: {np.sum(criticalities\_opt):.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'Reliability': [R\_i(t, alpha[i], beta[i]) for i in range(4)],

    'Criticality Index': criticalities\_opt

}

df = pd.DataFrame(data)

df['System Reliability'] = R\_sys\_optimal

df['Sum of Criticality Indices'] = np.sum(criticalities\_opt)

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model E3 with Integer Constraints:

Optimized n values: [1 1 1 1]

System Reliability: 0.998397

Sum of Criticality Indices: 4.000000

ni Reliability Criticality Index System Reliability \

0 1 0.999000 1.0 0.998397

1 1 0.999713 1.0 0.998397

2 1 0.999808 1.0 0.998397

3 1 0.999875 1.0 0.998397

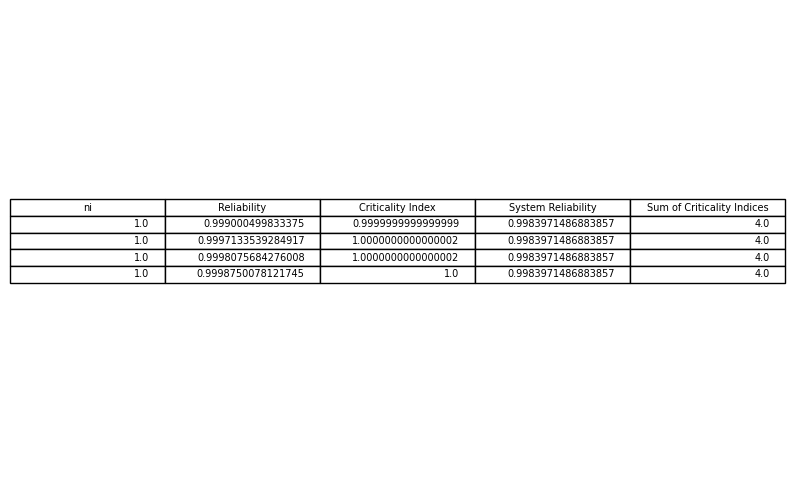
Sum of Criticality Indices

0 4.0

1 4.0

2 4.0

3 4.0



E4

Run the E2 model assuming that all units are exponential

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants

lambda\_ = [1e-5, 2.3e-5, 3e-5, 4e-5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

R\_sys\_opt = 0.95  # Example optimal system reliability from model A (for demonstration)

# Reliability function for exponential distribution

def R\_i\_exp(t, lambda\_i):

    return np.exp(-lambda\_i \* t)

# Partial derivative of R\_sys with respect to R\_i for exponential distribution

def partial\_R\_sys\_exp(n, lambda\_, t):

    Rs = [R\_i\_exp(t, lambda\_[i]) for i in range(4)]

    partials = []

    for i in range(4):

        partial = np.prod([(1 - (1 - Rs[j])\*\*n[j]) for j in range(4) if j != i])

        partial \*= n[i] \* (1 - Rs[i])\*\*(n[i] - 1)

        partials.append(partial)

    return partials

# Objective function to maximize

def objective\_E4(n, lambda\_, t):

    partials = partial\_R\_sys\_exp(n, lambda\_, t)

    return -np.sum(partials)

# System reliability function for exponential distribution

def R\_sys\_exp(n, t, lambda\_):

    Rs = [R\_i\_exp(t, lambda\_[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Constraints

def constraint1(n):

    return R\_sys\_exp(n, t, lambda\_) - 0.8 \* R\_sys\_opt

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective\_E4, n0, args=(lambda\_, t), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability

R\_sys\_optimal = R\_sys\_exp(n\_opt, t, lambda\_)

# Calculate Birnbaum importance indices for the optimal solution

partials\_opt = partial\_R\_sys\_exp(n\_opt, lambda\_, t)

# Display results

print("Optimization Results for Model E4 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

print(f"Sum of Birnbaum Importance Indices: {np.sum(partials\_opt):.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'Reliability': [R\_i\_exp(t, lambda\_[i]) for i in range(4)],

    'Birnbaum Importance': partials\_opt

}

df = pd.DataFrame(data)

df['System Reliability'] = R\_sys\_optimal

df['Sum of Birnbaum Importance Indices'] = np.sum(partials\_opt)

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model E4 with Integer Constraints:

Optimized n values: [1 1 1 1]

System Reliability: 0.998971

Sum of Birnbaum Importance Indices: 3.996911

ni Reliability Birnbaum Importance System Reliability \

0 1 0.99990 0.99907 0.998971

1 1 0.99977 0.99920 0.998971

2 1 0.99970 0.99927 0.998971

3 1 0.99960 0.99937 0.998971

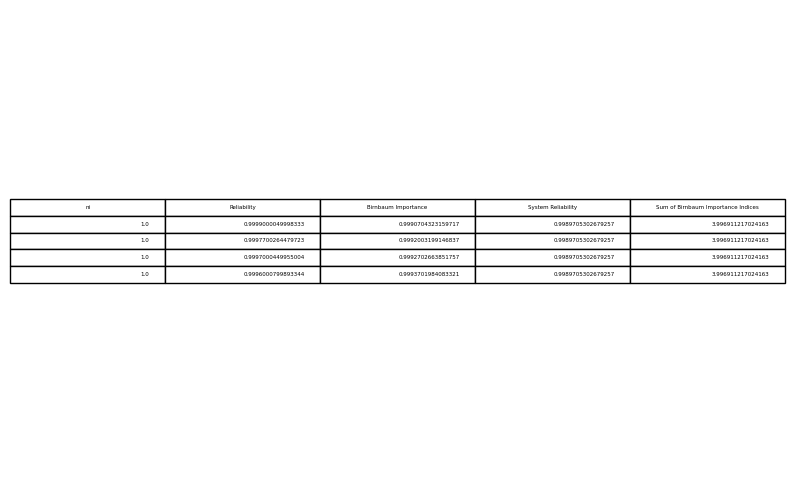
Sum of Birnbaum Importance Indices

0 3.996911

1 3.996911

2 3.996911

3 3.996911



E5

import numpy as np

from scipy.optimize import minimize

from math import factorial

import pandas as pd

import matplotlib.pyplot as plt

# Constants

lambda\_ = [1e-5, 2.3e-5, 3e-5, 4e-5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

R\_sys\_opt = 0.95  # Example optimal system reliability from model A (for demonstration)

# Reliability function for Cold Standby

def R\_i\_standby(t, lambda\_i, n\_i):

    n\_i = int(n\_i)  # Ensure n\_i is an integer

    if n\_i == 1:

        return np.exp(-lambda\_i \* t)

    else:

        R = np.exp(-lambda\_i \* t)

        summation = sum((lambda\_i \* t)\*\*k / factorial(k) for k in range(n\_i))

        return R \* summation

# Partial derivative of R\_sys with respect to R\_i for Cold Standby

def partial\_R\_sys\_standby(n, lambda\_, t):

    Rs = [R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)]

    partials = []

    for i in range(4):

        partial = np.prod([(1 - (1 - Rs[j])\*\*n[j]) for j in range(4) if j != i])

        partial \*= n[i] \* (1 - Rs[i])\*\*(n[i] - 1)

        partials.append(partial)

    return partials

# Criticality index for Cold Standby

def criticality\_index\_standby(n, lambda\_, t):

    Rs = [R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)]

    R\_sys\_value = R\_sys\_standby(n, t, lambda\_)

    partials = partial\_R\_sys\_standby(n, lambda\_, t)

    criticalities = [partials[i] \* Rs[i] / R\_sys\_value for i in range(4)]

    return criticalities

# Objective function to maximize for Model E7

def objective\_E7(n, lambda\_, t):

    criticalities = criticality\_index\_standby(n, lambda\_, t)

    return -np.sum(criticalities)

# System reliability function for Cold Standby

def R\_sys\_standby(n, t, lambda\_):

    Rs = [R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Constraints

def constraint1(n):

    return R\_sys\_standby(n, t, lambda\_) - 0.8 \* R\_sys\_opt

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective\_E7, n0, args=(lambda\_, t), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability

R\_sys\_optimal = R\_sys\_standby(n\_opt, t, lambda\_)

# Calculate criticality indices for the optimal solution

criticalities\_opt = criticality\_index\_standby(n\_opt, lambda\_, t)

# Display results

print("Optimization Results for Model E7 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

print(f"Sum of Criticality Indices: {np.sum(criticalities\_opt):.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'Reliability': [R\_i\_standby(t, lambda\_[i], n\_opt[i]) for i in range(4)],

    'Criticality Index': criticalities\_opt

}

df = pd.DataFrame(data)

df['System Reliability'] = R\_sys\_optimal

df['Sum of Criticality Indices'] = np.sum(criticalities\_opt)

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model E5 with Integer Constraints:

Optimized n values: [1 1 1 1]

System Reliability: 0.998971

Sum of Criticality Indices: 4.000000

ni Reliability Criticality Index System Reliability \

0 1 0.99990 1.0 0.998971

1 1 0.99977 1.0 0.998971

2 1 0.99970 1.0 0.998971

3 1 0.99960 1.0 0.998971

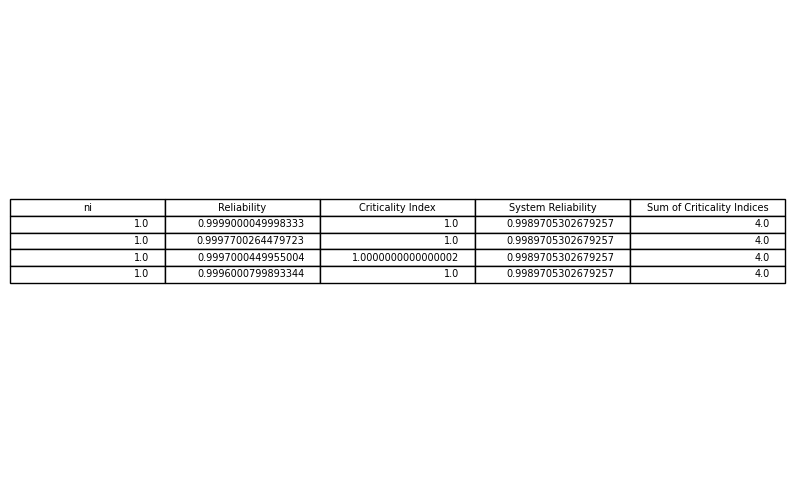
Sum of Criticality Indices

0 4.0

1 4.0

2 4.0

3 4.0



E6

E2 model assuming that the extensions are cold standby

import numpy as np

from scipy.optimize import minimize

from math import factorial

import pandas as pd

import matplotlib.pyplot as plt

# Constants

lambda\_ = [1e-5, 2.3e-5, 3e-5, 4e-5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

R\_sys\_opt = 0.95  # Example optimal system reliability from model A (for demonstration)

# Reliability function for Cold Standby

def R\_i\_standby(t, lambda\_i, n\_i):

    n\_i = int(n\_i)  # Ensure n\_i is an integer

    if n\_i == 1:

        return np.exp(-lambda\_i \* t)

    else:

        R = np.exp(-lambda\_i \* t)

        summation = sum((lambda\_i \* t)\*\*k / factorial(k) for k in range(n\_i))

        return R \* summation

# Partial derivative of R\_sys with respect to R\_i for Cold Standby

def partial\_R\_sys\_standby(n, lambda\_, t):

    Rs = [R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)]

    partials = []

    for i in range(4):

        partial = np.prod([(1 - (1 - Rs[j])\*\*n[j]) for j in range(4) if j != i])

        partial \*= n[i] \* (1 - Rs[i])\*\*(n[i] - 1)

        partials.append(partial)

    return partials

# Objective function to maximize for Model E6

def objective\_E6(n, lambda\_, t):

    partials = partial\_R\_sys\_standby(n, lambda\_, t)

    return -np.sum(partials)

# System reliability function for Cold Standby

def R\_sys\_standby(n, t, lambda\_):

    Rs = [R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Constraints

def constraint1(n):

    return R\_sys\_standby(n, t, lambda\_) - 0.8 \* R\_sys\_opt

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective\_E6, n0, args=(lambda\_, t), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability

R\_sys\_optimal = R\_sys\_standby(n\_opt, t, lambda\_)

# Calculate Birnbaum importance indices for the optimal solution

partials\_opt = partial\_R\_sys\_standby(n\_opt, lambda\_, t)

# Display results

print("Optimization Results for Model E6 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

print(f"Sum of Birnbaum Importance Indices: {np.sum(partials\_opt):.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'Reliability': [R\_i\_standby(t, lambda\_[i], n\_opt[i]) for i in range(4)],

    'Birnbaum Importance': partials\_opt

}

df = pd.DataFrame(data)

df['System Reliability'] = R\_sys\_optimal

df['Sum of Birnbaum Importance Indices'] = np.sum(partials\_opt)

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model E6 with Integer Constraints:

Optimized n values: [1 1 1 1]

System Reliability: 0.998971

Sum of Birnbaum Importance Indices: 3.996911

ni Reliability Birnbaum Importance System Reliability \

0 1 0.99990 0.99907 0.998971

1 1 0.99977 0.99920 0.998971

2 1 0.99970 0.99927 0.998971

3 1 0.99960 0.99937 0.998971

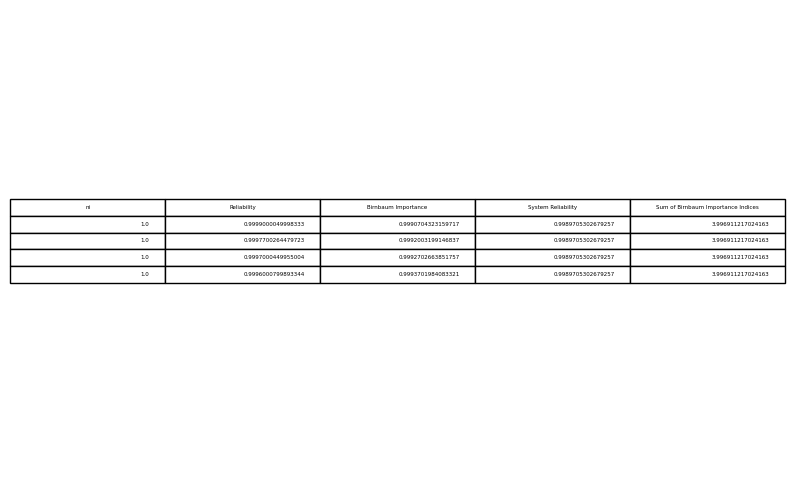
Sum of Birnbaum Importance Indices

0 3.996911

1 3.996911

2 3.996911

3 3.996911



E7

E3 model assuming that the extensions are cold standby

import numpy as np

from scipy.optimize import minimize

from math import factorial

import pandas as pd

import matplotlib.pyplot as plt

# Constants

lambda\_ = [1e-5, 2.3e-5, 3e-5, 4e-5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

R\_sys\_opt = 0.95  # Example optimal system reliability from model A (for demonstration)

# Reliability function for Cold Standby

def R\_i\_standby(t, lambda\_i, n\_i):

    n\_i = int(n\_i)  # Ensure n\_i is an integer

    if n\_i == 1:

        return np.exp(-lambda\_i \* t)

    else:

        R = np.exp(-lambda\_i \* t)

        summation = sum((lambda\_i \* t)\*\*k / factorial(k) for k in range(n\_i))

        return R \* summation

# Partial derivative of R\_sys with respect to R\_i for Cold Standby

def partial\_R\_sys\_standby(n, lambda\_, t):

    Rs = [R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)]

    partials = []

    for i in range(4):

        partial = np.prod([(1 - (1 - Rs[j])\*\*n[j]) for j in range(4) if j != i])

        partial \*= n[i] \* (1 - Rs[i])\*\*(n[i] - 1)

        partials.append(partial)

    return partials

# Criticality index for Cold Standby

def criticality\_index\_standby(n, lambda\_, t):

    Rs = [R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)]

    R\_sys\_value = R\_sys\_standby(n, t, lambda\_)

    partials = partial\_R\_sys\_standby(n, lambda\_, t)

    criticalities = [partials[i] \* Rs[i] / R\_sys\_value for i in range(4)]

    return criticalities

# Objective function to maximize for Model E7

def objective\_E7(n, lambda\_, t):

    criticalities = criticality\_index\_standby(n, lambda\_, t)

    return -np.sum(criticalities)

# System reliability function for Cold Standby

def R\_sys\_standby(n, t, lambda\_):

    Rs = [R\_i\_standby(t, lambda\_[i], n[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Constraints

def constraint1(n):

    return R\_sys\_standby(n, t, lambda\_) - 0.8 \* R\_sys\_opt

def constraint2(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def constraint3(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': constraint1},

        {'type': 'ineq', 'fun': constraint2},

        {'type': 'ineq', 'fun': constraint3}]

# Solving the optimization problem

solution = minimize(objective\_E7, n0, args=(lambda\_, t), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability

R\_sys\_optimal = R\_sys\_standby(n\_opt, t, lambda\_)

# Calculate criticality indices for the optimal solution

criticalities\_opt = criticality\_index\_standby(n\_opt, lambda\_, t)

# Display results

print("Optimization Results for Model E7 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

print(f"Sum of Criticality Indices: {np.sum(criticalities\_opt):.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'Reliability': [R\_i\_standby(t, lambda\_[i], n\_opt[i]) for i in range(4)],

    'Criticality Index': criticalities\_opt

}

df = pd.DataFrame(data)

df['System Reliability'] = R\_sys\_optimal

df['Sum of Criticality Indices'] = np.sum(criticalities\_opt)

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model E7 with Integer Constraints:

Optimized n values: [1 1 1 1]

System Reliability: 0.998971

Sum of Criticality Indices: 4.000000

ni Reliability Criticality Index System Reliability \

0 1 0.99990 1.0 0.998971

1 1 0.99977 1.0 0.998971

2 1 0.99970 1.0 0.998971

3 1 0.99960 1.0 0.998971

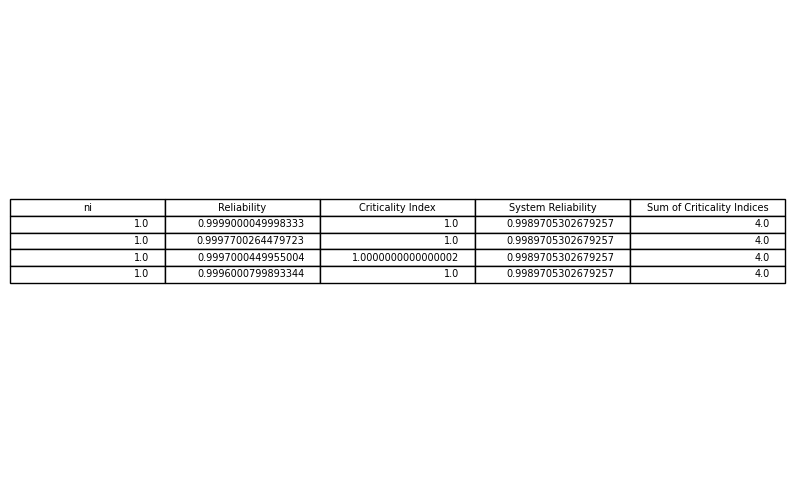
Sum of Criticality Indices

0 4.0

1 4.0

2 4.0

3 4.0



E8

Ri, ni=1(t) = 1 − (1 − e−λ t)2 = 2e−λ t − e−2λ t

𝐸𝑖(𝑡)= 1−𝑒−λ𝑡

𝐸𝑖,𝑛𝑖(𝑡)=1−(1−𝑒−λ𝑡)\*\*𝑛/1−(1−𝑒2λ𝑡)\*\*𝑛−1=1−[𝐸𝑖(𝑡)]\*\*𝑛/1−[𝐸𝑖(𝑡)]\*\*𝑛−1

Max𝑅sys,new(𝑡)−𝑅sys,now(𝑡/)𝑅sys,now(𝑡)

𝑆𝑡. Σ(𝛼𝑖(−𝑡/ln(𝑅𝑖(𝑡)))\*\*𝛽𝑖⋅(𝑛𝑖+𝑒\*\*0.25𝑛𝑖))≤400

Σ𝑉𝑖𝑛𝑖\*\*2≤250

Σ𝑊𝑖𝑛𝑖⋅𝑒\*\*0.25𝑛𝑖≤500

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants

lambda\_ =  [1e-5, 2.3e-5, 3e-6, 2.3e-5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

# Reliability function for exponential distribution

def R\_i\_exp(t, lambda\_i):

    return np.exp(-lambda\_i \* t)

# Reliability function for n\_i parallel units (Cold Standby)

def R\_i\_parallel(t, lambda\_i, n\_i):

    return 1 - (1 - R\_i\_exp(t, lambda\_i))\*\*n\_i

# Entropy index for a single unit

def entropy\_index(t, lambda\_i, n\_i):

    R\_single = R\_i\_exp(t, lambda\_i)

    R\_parallel = R\_i\_parallel(t, lambda\_i, n\_i)

    return (R\_parallel - R\_single) / R\_single

# System reliability function for current and new configurations

def R\_sys\_exp(n, t, lambda\_):

    Rs = [R\_i\_exp(t, lambda\_[i]) for i in range(4)]

    R\_sys\_now = np.prod(Rs)

    R\_sys\_new = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_now, R\_sys\_new

# Objective function to maximize

def objective\_E8(n, lambda\_, t):

    R\_sys\_now, R\_sys\_new = R\_sys\_exp(n, t, lambda\_)

    entropy\_sys = (R\_sys\_new - R\_sys\_now) / R\_sys\_now

    return -entropy\_sys

# Constraints

def cost\_constraint(n, alpha, beta, t):

    return 400 - np.sum([alpha[i] \* ((-t / np.log(R\_i\_exp(t, lambda\_[i])))\*\*beta[i]) \* (n[i] + np.exp(0.25 \* n[i])) for i in range(4)])

def volume\_constraint(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def weight\_constraint(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Parameters for cost calculation (alpha and beta are assumed for demonstration)

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta = [1.5, 1.5, 1.5, 1.5]

# Bounds for n\_i

bounds = [(1, 10) for \_ in range(4)]

# Initial guess

n0 = [1, 1, 1, 1]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': cost\_constraint, 'args': (alpha, beta, t)},

        {'type': 'ineq', 'fun': volume\_constraint},

        {'type': 'ineq', 'fun': weight\_constraint}]

# Solving the optimization problem

solution = minimize(objective\_E8, n0, args=(lambda\_, t), method='SLSQP', bounds=bounds, constraints=cons)

# Extract results and round to nearest integers

n\_opt = np.round(solution.x).astype(int)

# Recalculate system reliability and entropy

R\_sys\_now, R\_sys\_new = R\_sys\_exp(n\_opt, t, lambda\_)

entropy\_sys\_optimal = (R\_sys\_new - R\_sys\_now) / R\_sys\_now

# Display results

print("Optimization Results for Model E8 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"System Reliability (current): {R\_sys\_now:.6f}")

print(f"System Reliability (new): {R\_sys\_new:.6f}")

print(f"System Entropy: {entropy\_sys\_optimal:.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'Reliability (new)': [R\_i\_parallel(t, lambda\_[i], n\_opt[i]) for i in range(4)],

    'Entropy Index': [entropy\_index(t, lambda\_[i], n\_opt[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['System Reliability (current)'] = R\_sys\_now

df['System Entropy'] = entropy\_sys\_optimal

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

System Reliability (current): 0.999410

System Reliability (new): 0.999410

System Entropy: 0.000000

ni Reliability (new) Entropy Index System Reliability (current) \

0 1 0.99990 0.0 0.99941

1 1 0.99977 0.0 0.99941

2 1 0.99997 0.0 0.99941

3 1 0.99977 0.0 0.99941

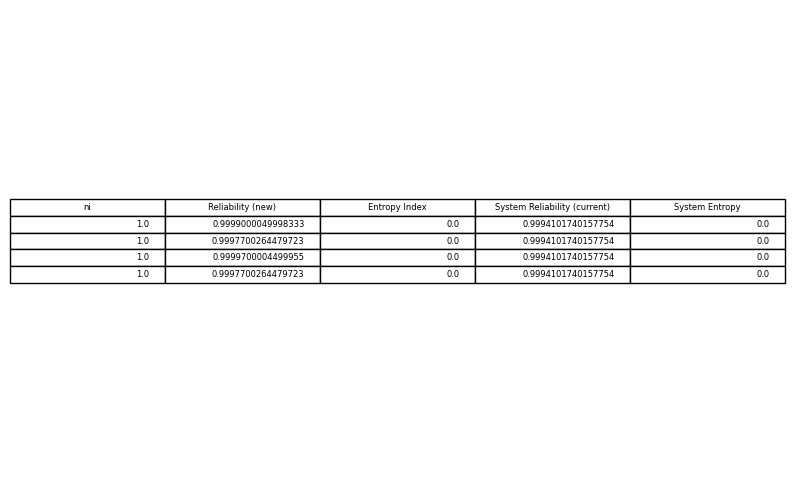
System Entropy

0 0.0

1 0.0

2 0.0

3 0.0



E9

Min Entropy=0.5772(1−1/𝛼sys+ln(𝛽sy\s𝛼sys)+1)

𝑆𝑡. previous constraints

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants

gamma = 0.5772

alpha = [1e-5, 2.3e-5, 3e-5, 4e-5]

beta = [1.5, 1.5, 1.5, 1.5]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

t = 10  # Assuming a fixed value for t

# Reliability function for Weibull distribution

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(- (t / alpha\_i)\*\*beta\_i)

# System reliability function

def R\_sys(n, t, alpha, beta):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - Rs[i])\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Objective function to minimize entropy

def objective\_entropy(x, n, t, alpha, beta):

    alpha\_sys, beta\_sys = x

    entropy = gamma \* (1 - 1/alpha\_sys) + np.log(beta\_sys / alpha\_sys) + 1

    return entropy

# Constraint for matching the system reliability derivative

def reliability\_constraint(x, n, t, alpha, beta):

    alpha\_sys, beta\_sys = x

    R\_sys\_value = R\_sys(n, t, alpha, beta)

    dR\_sys\_dt = -np.exp(-(t / alpha\_sys)\*\*beta\_sys)

    return R\_sys\_value - dR\_sys\_dt

# Additional constraints

def cost\_constraint(n, alpha, beta, t):

    return 400 - np.sum([alpha[i] \* ((-t / np.log(R\_i(t, alpha[i], beta[i])))\*\*beta[i]) \* (n[i] + np.exp(0.25 \* n[i])) for i in range(4)])

def volume\_constraint(n):

    return 250 - np.sum([v[i] \* n[i]\*\*2 for i in range(4)])

def weight\_constraint(n):

    return 500 - np.sum([w[i] \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i and initial guess for alpha\_sys, beta\_sys

bounds\_n = [(1, 10) for \_ in range(4)]

bounds\_ab = [(1e-5, None), (1, None)]

n0 = [1, 1, 1, 1]

x0 = [1e-5, 1.5]

# Constraints in dictionary form for n optimization

cons\_n = [{'type': 'ineq', 'fun': cost\_constraint, 'args': (alpha, beta, t)},

          {'type': 'ineq', 'fun': volume\_constraint},

          {'type': 'ineq', 'fun': weight\_constraint}]

# Optimization function for n

def optimize\_n(n0, alpha, beta, t):

    solution\_n = minimize(lambda n: 0, n0, method='SLSQP', bounds=bounds\_n, constraints=cons\_n)

    return np.round(solution\_n.x).astype(int)

# Optimize n values

n\_opt = optimize\_n(n0, alpha, beta, t)

# Constraints in dictionary form for alpha\_sys and beta\_sys optimization

cons\_ab = [{'type': 'eq', 'fun': reliability\_constraint, 'args': (n\_opt, t, alpha, beta)}]

# Solving the optimization problem for alpha\_sys and beta\_sys

solution\_ab = minimize(objective\_entropy, x0, args=(n\_opt, t, alpha, beta), method='SLSQP', bounds=bounds\_ab, constraints=cons\_ab)

# Extract results

alpha\_sys\_opt, beta\_sys\_opt = solution\_ab.x

# Recalculate system reliability

R\_sys\_optimal = R\_sys(n\_opt, t, alpha, beta)

entropy\_sys\_optimal = objective\_entropy([alpha\_sys\_opt, beta\_sys\_opt], n\_opt, t, alpha, beta)

# Display results

print("Optimization Results for Model E9 with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"Optimized alpha\_sys: {alpha\_sys\_opt:.6f}")

print(f"Optimized beta\_sys: {beta\_sys\_opt:.6f}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

print(f"System Entropy: {entropy\_sys\_optimal:.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'Reliability': [R\_i(t, alpha[i], beta[i]) for i in range(4)],

    'alpha\_sys': [alpha\_sys\_opt]\*4,

    'beta\_sys': [beta\_sys\_opt]\*4,

    'System Reliability': [R\_sys\_optimal]\*4,

    'System Entropy': [entropy\_sys\_optimal]\*4

}

df = pd.DataFrame(data)

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model E9 with Integer Constraints:

Optimized n values: [1 1 1 1]

Optimized alpha\_sys: 0.000010

Optimized beta\_sys: 1.500000

System Reliability: 0.000000

System Entropy: -57706.504409

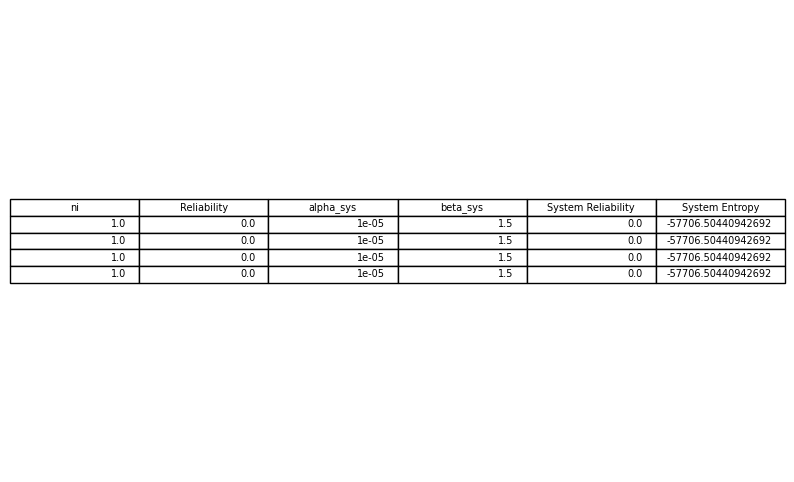
ni Reliability alpha\_sys beta\_sys System Reliability System Entropy

0 1 0.0 0.00001 1.5 0.0 -57706.504409

1 1 0.0 0.00001 1.5 0.0 -57706.504409

2 1 0.0 0.00001 1.5 0.0 -57706.504409

3 1 0.0 0.00001 1.5 0.0 -57706.504409



F

MaxΠ[1−(1−𝑅𝑖(𝑡)∗𝑦𝑖−𝑅𝑖′(𝑡)(1−𝑦𝑖))𝑛𝑖]

𝑆𝑡.

Σ{α𝑖(−𝑡/ln(𝑅𝑖(𝑡)))\*\*β𝑖 𝑦𝑖+α𝑖′(−𝑡/ln(𝑅𝑖′(𝑡)))β𝑖′(1−𝑦𝑖)}(𝑛𝑖+𝑒\*\*0.25𝑛𝑖)≤400

Σ(𝑉𝑖𝑦𝑖+𝑉𝑖′(1−𝑦𝑖))𝑛𝑖2≤250

Σ(𝑊𝑖𝑦𝑖+𝑊𝑖′(1−𝑦𝑖))𝑛𝑖\*\*𝑒0.25𝑛𝑖≤500

α1=α1′ , α2=α2′ , α3=α3′ , α4=α4′ β1=1.5 , β2=1.5 , β3=1.5 , β4=1.5 β1′=1.28 , β2′=4.77 , β3′=2.13 , β4′=1.75 𝑉𝑖=𝑉𝑖′ , 𝑊𝑖=𝑊𝑖′ , ∀𝑖=1,2,3,4

#F

import numpy as np

from scipy.optimize import minimize

import pandas as pd

import matplotlib.pyplot as plt

# Constants

alpha =  [1e+5, 2.3e+5, 3e+6, 2.3e+5]

alpha\_prime = alpha  # Since alpha\_i = alpha\_i\_prime

beta = [1.5, 1.5, 1.5, 1.5]

beta\_prime = [1.28, 4.77, 2.13, 1.75]

v = [1, 2, 3, 4]

w = [6, 6, 8, 8]

V\_prime = v  # Since V\_i = V\_i\_prime

W\_prime = w  # Since W\_i = W\_i\_prime

t = 1000  # Assuming a fixed value for t

# Reliability functions for Weibull distribution

def R\_i(t, alpha\_i, beta\_i):

    return np.exp(- (t / alpha\_i)\*\*beta\_i)

def R\_i\_prime(t, alpha\_i\_prime, beta\_i\_prime):

    return np.exp(- (t / alpha\_i\_prime)\*\*beta\_i\_prime)

# Combined system reliability function

def R\_sys\_combined(n, y, t, alpha, beta, alpha\_prime, beta\_prime):

    Rs = [R\_i(t, alpha[i], beta[i]) for i in range(4)]

    Rs\_prime = [R\_i\_prime(t, alpha\_prime[i], beta\_prime[i]) for i in range(4)]

    R\_sys\_value = np.prod([1 - (1 - (Rs[i] \* y[i] + Rs\_prime[i] \* (1 - y[i])))\*\*n[i] for i in range(4)])

    return R\_sys\_value

# Objective function to maximize system reliability

def objective\_system\_reliability(n\_y, t, alpha, beta, alpha\_prime, beta\_prime):

    n = n\_y[:4]

    y = n\_y[4:]

    return -R\_sys\_combined(n, y, t, alpha, beta, alpha\_prime, beta\_prime)

# Constraints

def cost\_constraint(n\_y, alpha, beta, alpha\_prime, beta\_prime, t):

    n = n\_y[:4]

    y = n\_y[4:]

    return 400 - np.sum([(alpha[i] \* ((-t / np.log(R\_i(t, alpha[i], beta[i])))\*\*beta[i]) \* y[i] +

                          alpha\_prime[i] \* ((-t / np.log(R\_i\_prime(t, alpha\_prime[i], beta\_prime[i])))\*\*beta\_prime[i]) \* (1 - y[i])) \*

                         (n[i] + np.exp(0.25 \* n[i])) for i in range(4)])

def volume\_constraint(n\_y):

    n = n\_y[:4]

    y = n\_y[4:]

    return 250 - np.sum([(v[i] \* y[i] + V\_prime[i] \* (1 - y[i])) \* n[i]\*\*2 for i in range(4)])

def weight\_constraint(n\_y):

    n = n\_y[:4]

    y = n\_y[4:]

    return 500 - np.sum([(w[i] \* y[i] + W\_prime[i] \* (1 - y[i])) \* n[i] \* np.exp(0.25 \* n[i]) for i in range(4)])

# Bounds for n\_i and y\_i

bounds\_n\_y = [(1, 10) for \_ in range(4)] + [(0, 1) for \_ in range(4)]

# Initial guess for n\_i and y\_i

n\_y0 = [1, 1, 1, 1, 0.5, 0.5, 0.5, 0.5]

# Constraints in dictionary form

cons = [{'type': 'ineq', 'fun': cost\_constraint, 'args': (alpha, beta, alpha\_prime, beta\_prime, t)},

        {'type': 'ineq', 'fun': volume\_constraint},

        {'type': 'ineq', 'fun': weight\_constraint}]

# Solving the optimization problem

solution = minimize(objective\_system\_reliability, n\_y0, args=(t, alpha, beta, alpha\_prime, beta\_prime), method='SLSQP', bounds=bounds\_n\_y, constraints=cons)

# Extract results

n\_opt = np.round(solution.x[:4]).astype(int)

y\_opt = np.round(solution.x[4:]).astype(int)

# Recalculate system reliability

R\_sys\_optimal = R\_sys\_combined(n\_opt, y\_opt, t, alpha, beta, alpha\_prime, beta\_prime)

# Display results

print("Optimization Results for Model F with Integer Constraints:")

print(f"Optimized n values: {n\_opt}")

print(f"Optimized y values: {y\_opt}")

print(f"System Reliability: {R\_sys\_optimal:.6f}")

# If desired, visualize the results in a table

data = {

    'ni': n\_opt,

    'yi': y\_opt,

    'Reliability (new)': [R\_i(t, alpha[i], beta[i]) \* y\_opt[i] + R\_i\_prime(t, alpha\_prime[i], beta\_prime[i]) \* (1 - y\_opt[i]) for i in range(4)]

}

df = pd.DataFrame(data)

df['System Reliability'] = R\_sys\_optimal

# Display the DataFrame

print(df)

# Optionally, plot the table

fig, ax = plt.subplots(figsize=(10, 6))

ax.table(cellText=df.values, colLabels=df.columns, loc='center')

ax.axis('off')

plt.show()

Optimization Results for Model F with Integer Constraints:

Optimized n values: [1 1 1 1]

Optimized y values: [0 0 0 0]

System Reliability: 0.999992

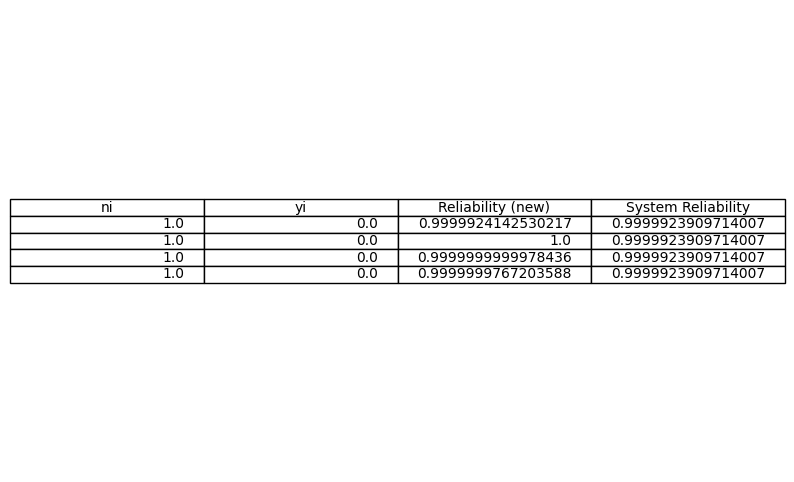
ni yi Reliability (new) System Reliability

0 1 0 0.999992 0.999992

1 1 0 1.000000 0.999992

2 1 0 1.000000 0.999992

3 1 0 1.000000 0.999992



**با تشکر از توجه شما**