付録

本研究のプログラム
3 層 3 スパン

```
#必要なライブラリのインストール
import numpy as np
import pandas as pd
import sys
from numpy import zeros
from platypus import NSGAII,Problem,Real,nondominated
from matplotlib import pyplot as plt
#import time
#import tqdm

#for i in tqdm.tqdm(range(int(1e7))):
 #     np.pi*np.pi

#部材の形と梁、柱の数を入力
A = pd.read_excel("l-3-3pydata.xlsx", sheet_name=0)
NNODE = A.iat[0,0]
NMEMB = A.iat[0,1]
COLUMN = A.iat[0,2]
BEAM = A.iat[0,3]
#断面積、荷重、塑性断面係数,全塑性モーメントの計算関数
def mat(H,B,t1,t2,r):
    memb = t1*(H-2*t2)+2*B*t2+0.858*r**2
    w = 0.00785*memb
    Zx = B*t2*(H-t2) + (((H-2*t2)**2)*t1)/4 + 0.4292*(r**2)*(H-2*t2 - 0.4467*r)
    tpm = 235*Zx
    return memb,w,Zx,tpm




#崩壊荷重計算の関数
```

```
def ALM(NNODE, NMEMB, COLUMN, BEAM, H_co, B_co, t1_co, t2_co, r_co, H_be,
B_be, t1_be, t2_be, r_be, H_co1, B_co1, t1_co1, t2_co1, r_co1, H_be1, B_be1, t1_be1, t2_be1,
r_be1, H_co2, B_co2, t1_co2, t2_co2, r_co2, H_be2, B_be2, t1_be2, t2_be2, r_be2):
    np.set_printoptions(precision=3,suppress=True)
    NCOL = 5*NMEMB
    NROW = 3*NNODE + 2*NMEMB
    #各部材の全塑性モーメント
    tpm_column = mat(H_co,B_co,t1_co,t2_co,r_co)[3]
    tpm_beam = mat(H_be,B_be,t1_be,t2_be,r_be)[3]
    tpm_column1 = mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)[3]
    tpm_beam1 = mat(H_be1,B_be1,t1_be1,t2_be1,r_be1)[3]
    tpm_column2 = mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)[3]
    tpm_beam2 = mat(H_be2,B_be2,t1_be2,t2_be2,r_be2)[3]



    I = NROW
    J = NCOL
    H = zeros((I,J))
    P = zeros((I))
    I = NCOL
    R = zeros((I))
    IHING = zeros((I))
    IM = NMEMB
    AMO = zeros((IM,2))
    I = 4
    W = zeros((IM,I))

    AF = []
    AS = ["","FIX","PIN","ROL"]

    #-------sheet2-------

    B = pd.read_excel("l-3-3pydata.xlsx", sheet_name=1)

    J = B["J"]
```

```
#print(J)
#print(J[3])
ISUP = B["ISUP"]
XNODE = B["XNODE"]
YNODE = B["YNODE"]

for i in range(NNODE):
    ISUP1 = ISUP[i]
    AF.append(AS[ISUP1])

#-------sheet3-------sheet4-------sheet5-------
C = pd.read_excel("l-3-3pydata.xlsx", sheet_name=2)
AL = pd.read_excel("l-3-3pydata.xlsx", sheet_name=3)
FYY = pd.read_excel("l-3-3pydata.xlsx", sheet_name=4)

#print(FY)

M = C["M"]
N1 = C["N1"]
N2 = C["N2"]

#print(AL.iat[2,2])
AL2 = AL["AL2"]
AL3 = AL["AL3"]
AL4 = AL["AL4"]
AL5 = AL["AL5"]

FY = zeros((NMEMB,10))
for i in range(NMEMB):
    for j in range(2):
        FY[i,j] = FYY.iat[i,j]
for j in range(2,10):
    for i in range(COLUMN):
        FY[i,j] = tpm_column
    for i in range(COLUMN,COLUMN+BEAM):
        FY[i,j] = tpm_beam
```

```
            for i in range(COLUMN):
                FY[i,j] = tpm_column1
            for i in range(COLUMN,COLUMN+BEAM):
                FY[i,j] = tpm_beam1
            for i in range(COLUMN):
                FY[i,j] = tpm_column2
            for i in range(COLUMN,COLUMN+BEAM):
                FY[i,j] = tpm_beam2


#       Q = pd.read_excel("l-3-3pydata.xlsx", sheet_name=7,nrows=4)
 #     Q=mat(H_co,B_co,t1_co,t2_co,r_co)   #tpm_column
  #    Q1 = pd.read_excel("l-3-3pydata.xlsx", sheet_name=7,skiprows=[1,2,3,4],nrows=4)
   # Q1= mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)
#                                   Q2          =          pd.read_excel("l-3-3pydata.xlsx",
sheet_name=7,skiprows=[1,2,3,4,5,6,7,8],nrows=4)
 #     Q2= mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)
  #                              Q3          =          pd.read_excel("l-3-3pydata.xlsx",
sheet_name=7,skiprows=[1,2,3,4,5,6,7,8,9,10,11,12],nrows=3)
   # Q3=mat(H_be,B_be,t1_be,t2_be,r_be) #tpm_beam
#                                   Q4          =          pd.read_excel("l-3-3pydata.xlsx",
sheet_name=7,skiprows=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],nrows=3)
 #     Q4=mat(H_be1,B_be1,t1_be1,t2_be1,r_be1)
  #                              Q5          =          pd.read_excel("l-3-3pydata.xlsx",
sheet_name=7,skiprows=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18],nrows=3)
   # Q5=mat(H_be2,B_be2,t1_be2,t2_be2,r_be2)
    Q                               =                  pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,2,3,5,6,8,9,11,12,13,14,15,16,17,18,19,20,21])
    Q=mat(H_co,B_co,t1_co,t2_co,r_co)   #tpm_column
    Q1                              =                  pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,3,4,6,7,9,10,12,13,14,15,16,17,18,19,20,21])
    Q1= mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)
    Q2                              =                  pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,4,5,7,8,10,11,13,14,15,16,17,18,19,20,21])
    Q2= mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)
    Q3                              =                  pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,5,6,7,8,9,10,11,12,16,17,18,19,20,21])
```

```
    Q3=mat(H_be,B_be,t1_be,t2_be,r_be) #tpm_beam
    Q4 = pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,19,20,21])
    Q4=mat(H_be1,B_be1,t1_be1,t2_be1,r_be1)
    Q5 = pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18])
    Q5=mat(H_be2,B_be2,t1_be2,t2_be2,r_be2)


    AL = []
    CS = []
    SN = []

    for i in range(NMEMB):
        NL = N1[i]
        NR = N2[i]
        DELX = XNODE[NR] - XNODE[NL]
        DELY = YNODE[NR] - YNODE[NL]

        AL1 = np.sqrt(DELX**2 + DELY**2)
        CS1 = DELX/AL1
        SN1 = DELY/AL1

        AL = np.insert(AL,0,AL1,axis = 0)
        CS.append(CS1)
        SN.append(SN1)

    #print(AL,CS,SN)

    #-------sheet6------
    D = pd.read_excel("l-3-3pydata.xlsx",sheet_name = 5)
    IN = D["IN"]
    X1 = D["X1"]
    X2 = D["X2"]
    X3 = D["X3"]
```

```
for I in IN:
    II = I*3
    P[II] = X1[I]
    P[II+1] = X2[I]
    P[II+2] = X3[I]



#-------sheet7-------
E = pd.read_excel("l-3-3pydata.xlsx",sheet_name = 6)

IM = E["IM"]
X1 = E["X1"]
X2 = E["X2"]
X3 = E["X3"]
X4 = E["X4"]

IM = len(IM)

W = zeros((IM,4))
#print(W)

for I in range(IM):
    W[I,0] = X1[I]
    W[I,1] = X2[I]
    W[I,2] = X3[I]
    W[I,3] = X4[I]

#-------the equiliblium equation for the member in the-------
#-------member coordinate system (H matrix,P matrix)-------
#-------TRANSGORMATION OF THE COORDINATE SYSTEM-------
#-------ASSEMBLING OF THE TOTAL H MATRIX AND FORCE VECTOR-------

def eq():
    for L in range(2):
```

```
            NL = L*3
            for j in range(4):
                X = zeros((3))
                for i in range(3):
                    A = 0
                    for k in range(3):
                        A = A+T[i,k]*AM[k+NL,j]
                    X[i] = A
                for i in range(3):
                    AM[i+NL,j] = X[i]

        for x in range(8):
            II = N1[im]*3+x
            if x >= 3 and x < 6:
                II = (N2[im]-1)*3+x
            if x >= 6:
                II = NNODE*3+(im-3)*2+x
            for y in range(4):
                if y == 3:
                    P[II] = P[II]+AM[x,y]
                    break
                JJ = im*3+y
                H[II,JJ] = AM[x,y]

for im in range(NMEMB):
    T = zeros((3,3))
    T[0,0] = CS[im]
    T[1,0] = SN[im]
    T[0,1] = -SN[im]
    T[1,1] = CS[im]
    T[2,2] = 1
    #print(T)

    AM = zeros((8,4))
    A = 1 / (AL[im] - AL2[im] - AL3[im])
    #print(A)
```

```
B1 = AL2[im]*A
B2 = AL3[im]*A
AM[0,0] = -1
AM[1,1] = A
AM[1,2] = -A
AM[2,1] = 1+B1
AM[2,2] = -B1
AM[3,0] = 1
AM[4,1] = -A
AM[4,2] = A
AM[5,1] = B2
AM[5,2] = -(1+B2)
#print(AM)

AA = AL4[im] + AL5[im]
if AA == 0:
    eq()
    continue

C11 = AL[im] - AL3[im] - AL4[im]
C21 = AL4[im] - AL2[im]
C12 = AL[im] - AL3[im] - AL5[im]
C22 = AL5[im] - AL2[im]
#print(C11)

AM[6,1] = -A*C11
AM[6,2] = -A*C21
AM[7,1] = -A*C12
AM[7,2] = -A*C22

WX1 = W[im,0]
WY1 = W[im,1]
WX2 = W[im,2]
WY2 = W[im,3]

AM[0,3] = ((C11+B2/A)*WX1+(C12+B2/A)*WX2)/AL[im]
```

```
        AA = C11*WY1+C12*WY2



        AM[1,3] = A*AA
        AM[2,3] = B1*AA

        AM[3,3] = ((C21+B1/A)*WX1 + (C22+B1/A)*WX2)/AL[im]
        AA = C21*WY1 + C22*WY2
        AM[4,3] = A*AA
        AM[5,3] = -B2*AA
        AM[6,3] = C21*AM[1,3]
        AM[7,3] = C22*AM[1,3]
        AA = AL5[im] - AL4[im]
        if AA < 0:
            AM[6,3] = AM[6,3] + AA*WY2
        if AA > 0:
            AM[7,3] = AM[7,3] - AA*WY1
        AMO[im,0] = AM[6,3]
        AMO[im,1] = AM[7,3]
        eq()#つり合い方程式の作成
N = NMEMB*2
for v in range(N):
        II = NNODE*3+v
        JJ = NMEMB*3+v
        H[II,JJ] = 1
ALM = 0
#ジョルダン
def jordan(I,JC):
        PIV = H[I,JC]
        if PIV == 0:
            print(I,JC,"-------STOP AT JORDAN-------")
            sys.exit()
        for J in range(NCOL):
            H[I,J] = H[I,J]/PIV
        P[I] = P[I]/PIV
        for i in range(NROW):
```

```
            if i == I:
                continue
            else:
                PIV = H[i,JC]
                for J in range(NCOL):
                    H[i,J] = H[i,J] - H[I,J]*PIV
                P[i] = P[i] - P[I]*PIV

#何回も出てくるところを変数化
def gj_510():
    PIV = abs(H[I,0])
    #print(PIV)
    JCOL[I] = 0
    #JCOL は H 座標 I 行の列番号を示すため、0 はじまりにする
    for J in range(1,NCOL):
        AA = abs(H[I,J])
        if AA > PIV:
            PIV = AA
            JCOL[I] = J
            continue
        else:
            continue
    JC = JCOL[I]
    #print(JC)
    JC = int(JC)
    return JC

JCOL = np.ones((NROW))
JCOL = -1*JCOL
#print(type(H))

for I in range(NROW):
    #JCOL[I] = 0
    if I > 3*NNODE-1:
        IM = int((I-3*NNODE)/2)
        J = int((I-3*NNODE-2*IM)*2)
```

```
            A = 0
            for jj in range(2):
                A = A + abs(W[IM,J+jj])
            if A == 0:
                continue
            else:
                JC = gj_510()
                jordan(I,JC)
                #JCOL[I] = JCOL[I] + 1
                continue
        else:
            IN = I/3
            IN = int(IN)
            ISUP1 = ISUP[IN] + 1
            #ISUP 内の変数ＩＮは部材番号であり、ISUP 自体は支持状態
            #Fortran においても 0，1，2，3 で表されるのでこのままでいい
            #また、ISUP1 に関しても、次に行う処理を指示するための変数なので、原文と変える必要はない

            if ISUP1 == 1:
                JC = gj_510()
                jordan(I,JC)
                #JCOL[I] = JCOL[I]+1
                continue
            elif ISUP1 == 2:
                continue
            elif ISUP1 == 3:
                aa = I - 3*IN
                if aa == 0:
                    JC = gj_510()
                    jordan(I,JC)
                    #JCOL[I] = JCOL[I] + 1
                    continue
                else:
                    continue
            elif ISUP1 == 4:
```

```
                bb = I - 3*IN + 1
                if bb == 0:
                    continue
                else:
                    JC = gj_510()
                    jordan(I,JC)
                    #JCOL[I] = JCOL[I] + 1
                    continue

    #------STEP ONE-------

    #以下、JC を数値として扱う場合は＋ 1 された状態、番号の場合 JCOL の値そのままにする
    NC = 0
    while NC <= NROW:
        # if NC >= NROW:
        #         print("MECHANISM WAS NOT REACHED WITHIN NROW CYCLES,TWO PROCEDURE INTERRUPTED")
        #     print("STOP")
        #     break

        DLMK = 10000 #1000 ~ 100000
        NC = NC + 1
        for I in range(NROW):
            JC = int(JCOL[I]+1)
            if JC == 0:
                continue
            elif JC > 3*NMEMB:
                IM = int((JC-3*NMEMB-1)/2 + 1)
                JJ = int(JC-3*NMEMB-2*(IM-1)+3)
            else:
                IM = int((JC-1)/3 + 1)
                JJ = int(JC-3*(IM-1))
            PP = P[I]
            if PP > 0:
                DLM = ((FY[IM-1,2*JJ-2])-R[JC-1])/P[I]
```

```
            elif PP == 0:
                continue
            elif PP < 0:
                DLM = ((-FY[IM-1,2*JJ-1])-R[JC-1])/P[I]
            if DLM >= DLMK:
                continue
            DLMK = DLM
            K = I

#-------K COMPONENT NO YATU-------

        JC = int(JCOL[K]+1)
        IHING[JC-1] = 1
        if P[K] < 0:
            IHING[JC-1] = -1
        ALM = ALM + DLMK
        for I in range(NROW):
            JC = int(JCOL[I]+1)
            if JC == 0:
                continue
            R[JC-1] = R[JC-1]+DLMK*P[I]
            continue

        #------STEP TWO-------

        ZMAX = -10000 #-1000 ~ -100000
        for J in range(NCOL):
            for I in range(NROW):
                JC = JCOL[I]+1
                if J == JC:
                    break
            if J == JC-1:
                continue
            else:
                Z = H[K,J]*P[K]/abs(P[K])
                if R[J] == 0:
```

```
                Z = abs(Z)
            elif R[J] > 0:
                Z = -Z
            if Z > ZMAX:
                ZMAX = Z
                L = J
                L = int(L)
            else:
                continue
        if ZMAX <= 0:
            return ALM
        if NC == NROW:
            return ALM
        else:
            #print(H)
            IHING[L] = 0
            JC = int(JC-1)
            jordan(K,L)
            continue

#最適化の準備と実行__崩壊荷重、柱と梁の断面積
def objective(vars):
    H_co = vars[0]
    B_co = vars[1]
    t1_co = vars[2]
    t2_co = vars[3]
    r_co = vars[4]
    H_co1 = vars[5]
    B_co1 = vars[6]
    t1_co1= vars[7]
    t2_co1 = vars[8]
    r_co1 = vars[9]
    H_co2 = vars[10]
    B_co2 = vars[11]
    t1_co2 = vars[12]
    t2_co2 = vars[13]
```

```
    r_co2 = vars[14]
    H_be =vars[15]
    B_be = vars[16]
    t1_be = vars[17]
    t2_be = vars[18]
    r_be = vars[19]
    H_be1 = vars[20]
    B_be1 = vars[21]
    t1_be1 = vars[22]
    t2_be1 = vars[23]
    r_be1 = vars[24]
    H_be2 = vars[25]
    B_be2 = vars[26]
    t1_be2 = vars[27]
    t2_be2 = vars[28]
    r_be2 = vars[29]

    f1 = 1/ALM(NNODE, NMEMB, COLUMN, BEAM, H_co, B_co, t1_co, t2_co, r_co,
H_be, B_be, t1_be, t2_be, r_be, H_co1, B_co1, t1_co1, t2_co1, r_co1, H_be1, B_be1, t1_be1,
t2_be1, r_be1, H_co2, B_co2, t1_co2, t2_co2, r_co2, H_be2, B_be2, t1_be2, t2_be2, r_be2)
    f2 = mat(H_co,B_co,t1_co,t2_co,r_co)[0]
    f3 = mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)[0]
    f4 = mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)[0]
    f5 = mat(H_be,B_be,t1_be,t2_be,r_be)[0]
    f6 = mat(H_be1,B_be1,t1_be1,t2_be1,r_be1)[0]
    f7 = mat(H_be2,B_be2,t1_be2,t2_be2,r_be2)[0]

   # v1=-f3+f5
    #v2=-f5+f7
    #v3=-f3+f7

    return[f1,f2,f3,f4,f5,f6,f7]#,[v1,v2,v3]



def optimization(n_var,n_obj,vars,n_run):
```

```
    #設計変数と目的関数の値を設定
    problem = Problem(n_var,n_obj)
    #最適化問題に最小化を設定
    problem.directions[:] = Problem.MINIMIZE
   # problem.constraints[:] = "<=0"
    #設計変数と目的関数を設定
    problem.types[:] = vars
    problem.function = objective
    #最適化アルゴリズムを設定して計算する
    algorithm = NSGAII(problem,population_size=3)
    algorithm.run(n_run)

    return algorithm

#変数を設定する
vars1 = Real(100.,1000.)
vars2 = Real(50.,500.)
vars3 = Real(5.,50.)
vars4 = Real(5.,80.)
vars5 = Real(5.,25.)
vars6 = Real(100.,1000.)
vars7 = Real(50.,500)
vars8 = Real(5.,50.)
vars9 = Real(5.,80.)
vars10 = Real(5.,25.)
vars11 = Real(100.,1000.)
vars12 = Real(50.,500.)
vars13 = Real(5.,50.)
vars14 = Real(5.,80.)
vars15 = Real(5.,25.)
vars16 = Real(100.,1000.)
vars17 = Real(50.,500.)
vars18 = Real(5.,50.)
vars19 = Real(5.,80.)
vars20 = Real(5.,25.)
vars21 = Real(100.,1000.)
```

```
vars22 = Real(50.,500.)
vars23 = Real(5.,50.)
vars24 = Real(5.,80.)
vars25 = Real(5.,25.)
vars26 = Real(100.,1000.)
vars27 = Real(50.,500.)
vars28 = Real(5.,50.)
vars29 = Real(5.,80.)
vars30 = Real(5.,25.)
vars =
[vars1,vars2,vars3,vars4,vars5,vars6,vars7,vars8,vars9,vars10,vars11,vars12,vars13,vars14,va
rs15,vars16,vars17,vars18,vars19,vars20,vars21,vars22,vars23,vars24,vars25,vars26,vars27,v
ars28,vars29,vars30]
#最適化計算を実行する
algorithm = optimization(n_var=len(vars),n_obj=7,vars=vars,n_run=2000)

#グラフの設定
plt.rcParams['font.size'] = 14
plt.rcParams['font.family'] = 'Times New Roman'
fig = plt.figure(figsize=(6, 6))
ax1 = fig.add_subplot(111,projection="3d")

# 軸のラベルを設定する。
ax1.set_xlabel('load')
ax1.set_ylabel('column')
ax1.set_zlabel('beam')

#解の抽出
obj1 = []
obj2 = []
obj3 = []
obj4 = []
obj5 = []
obj6 = []
obj7 = []
```

```
for solution in algorithm.result:
    obj1.append(solution.objectives[0])
    obj2.append(solution.objectives[1])
    obj3.append(solution.objectives[2])
    obj4.append(solution.objectives[3])
    obj5.append(solution.objectives[4])
    obj6.append(solution.objectives[5])
    obj7.append(solution.objectives[6])

ax1.scatter(obj1, obj2, obj5, color='gray', edgecolor='gray', label='Trial', alpha=0.5)
ax1.scatter(obj1,obj3,obj6, color='gray', edgecolor='gray', label='Trial', alpha=0.5)
ax1.scatter(obj1,obj4,obj7, color='gray', edgecolor='gray', label='Trial', alpha=0.5)

# パレート解をプロットする。
n_dominated = nondominated(algorithm.result)      # 非劣解を抽出する
obj1 = []
obj2 = []
obj3 = []
obj4 = []
obj5 = []
obj6 = []
obj7 = []
for solution in n_dominated:
    obj1.append(solution.objectives[0])
    obj2.append(solution.objectives[1])
    obj3.append(solution.objectives[2])
    obj4.append(solution.objectives[3])
    obj5.append(solution.objectives[4])
    obj6.append(solution.objectives[5])
    obj7.append(solution.objectives[6])
ax1.scatter(obj1, obj2, obj5, color='yellow', edgecolor='black', label='Pareto front')
ax1.scatter(obj1,obj3,obj6, color='red', edgecolor='black', label='Pareto front')
ax1.scatter(obj1,obj4,obj7, color='blue', edgecolor='black', label='Pareto front')
ax1.legend()

# グラフを表示する。
```

```
plt.show()
plt.close()



#部材選択プログラム
n_d = len(n_dominated)
result_array = np.zeros((n_d,37))
for i in range(len(n_dominated)):
    for j in range(30):
        result_array[i,j] = n_dominated[i].variables[j]
    for j in range(30,37):
        result_array[i,j] = n_dominated[i].objectives[j-30]


result_column = result_array[:,0:5]
result_column1 = result_array[:,5:10]
result_column2= result_array[:,10:15]
result_beam = result_array[:,15:20]
result_beam1 = result_array[:,20:25]
result_beam2 = result_array[:,25:30]
result_purpus = result_array[:,30:37]
mat_data = pd.read_excel("mat_data_mm.xlsx")
mat_data = mat_data.to_numpy()
r_c = pd.DataFrame(result_column)
r_b = pd.DataFrame(result_beam)
r_c1 = pd.DataFrame(result_column1)
r_b1 = pd.DataFrame(result_beam1)
r_c2 = pd.DataFrame(result_column2)
r_b2 = pd.DataFrame(result_beam2)
#print("result column")
#print(r_c)
#print(r_c1)
#print(r_c2)
#print("result beam")
#print(r_b)
```

```
#print(r_b1)
#print(r_b2)

#誤差を格納する行列作成
err_mat_c = zeros((len(mat_data),5))
err_mat_b = zeros((len(mat_data),5))
err_mat_c1 = zeros((len(mat_data),5))
err_mat_b1 = zeros((len(mat_data),5))
err_mat_c2 = zeros((len(mat_data),5))
err_mat_b2 = zeros((len(mat_data),5))
#最適部材の要素を格納する行列
fin_c = []
fin_b = []
fin_c1 = []
fin_b1 = []
fin_c2 = []
fin_b2 = []
#パレート解の取り出し　ひとつずつ行うため
for m in range(n_d):
    for i in range(len(mat_data)):
        for j in range(5):
            err_mat_c[i,j] = (mat_data[i,j]/result_column[m,j])-1
            err_mat_b[i,j] = (mat_data[i,j]/result_beam[m,j])-1
            err_mat_c1[i,j] = (mat_data[i,j]/result_column1[m,j])-1
            err_mat_b1[i,j] = (mat_data[i,j]/result_beam1[m,j])-1
            err_mat_c2[i,j] = (mat_data[i,j]/result_column2[m,j])-1
            err_mat_b2[i,j] = (mat_data[i,j]/result_beam2[m,j])-1
    # print("column's err")
    # print(err_mat_c)
    # print("beam's err")
    # print(err_mat_b)
    #変数の誤差合計を格納する行列
    err_tot_c = []
    err_tot_b = []
    err_tot_c1 = []
    err_tot_b1 = []
```

```
    err_tot_c2 = []
    err_tot_b2 = []
    #誤差の大きさを行列にまとめる
    for i in range(len(mat_data)):
        err_tot_c.append(sum(abs(err_mat_c[i])))
        err_tot_b.append(sum(abs(err_mat_b[i])))
        err_tot_c1.append(sum(abs(err_mat_c1[i])))
        err_tot_b1.append(sum(abs(err_mat_b1[i])))
        err_tot_c2.append(sum(abs(err_mat_c2[i])))
        err_tot_b2.append(sum(abs(err_mat_b2[i])))
    #誤差の合計値が最も小さいものを選択,部材の選択
    t_c = err_tot_c.index(min(err_tot_c))
    t_b = err_tot_b.index(min(err_tot_b))
    t_c1 = err_tot_c1.index(min(err_tot_c1))
    t_b1 = err_tot_b1.index(min(err_tot_b1))
    t_c2 = err_tot_c2.index(min(err_tot_c2))
    t_b2 = err_tot_b2.index(min(err_tot_b2))
    fin_c.append(mat_data[t_c])
    fin_b.append(mat_data[t_b])
    fin_c1.append(mat_data[t_c1])
    fin_b1.append(mat_data[t_b1])
    fin_c2.append(mat_data[t_c2])
    fin_b2.append(mat_data[t_b2])


#最適部材を見やすくしてる
fin_c = pd.DataFrame(fin_c)
fin_b = pd.DataFrame(fin_b)
fin_c1 = pd.DataFrame(fin_c1)
fin_b1 = pd.DataFrame(fin_b1)
fin_c2 = pd.DataFrame(fin_c2)
fin_b2 = pd.DataFrame(fin_b2)

print("Column's optimal solution")
print(fin_c)
print(fin_c1)
```

```
print(fin_c2)
print("Beam's optimal solution")
print(fin_b)
print(fin_b1)
print(fin_b2)


#df = pd.DataFrame([[fin_c],[fin_c1],[fin_c2]],
 #                        index=['1F','2F','3F'], columns=['H','B','t1','t2','r'])
#df1 =pd.DataFrame([[fin_b],[fin_b1],[fin_b2]],
 #                        index=['1F','2F','3F'], columns=['H','B','t1','t2','r'])
#with pd.ExcelWriter('spyder-python/l-3-3-cpm-size3-run1000.xlsx', mode='a') as writer:
 #      df.to_excel(writer, sheet_name='sheet1')
  #     df1.to_excel(writer, sheet_name='sheet2')


sys.exit()


#         fin_result.append(["-----","-----","-----","-----","-----"])
#         fin_result.append(["pareto",j,"-----","-----","-----"])
#         fin_result.append(["purpus","---","---","---","---"])
#         fin_result.append(result_purpus[j])
#         fin_result.append(["col","---","---","---","---"])
#         fin_result.append(res_col)
#         fin_result.append(res_col_err)
#         fin_result.append(["beam","---","---","---","---"])
#         fin_result.append(res_beam)
#         fin_result.append(res_beam_err)

# #選択結果の印刷
# print("closest value")
# print("pareto=",n_d,"¥n")
# column_name = ["H","B","t1","t2","r"]
# fin_result = pd.DataFrame(fin_result,columns = column_name)
# print(fin_result)
```

5 層 3 スパン

```
#必要なライブラリのインストール
import numpy as np
import pandas as pd
import sys
from numpy import zeros
from platypus import NSGAII,Problem,Real,nondominated
from matplotlib import pyplot as plt


#部材の形と梁、柱の数を入力
A = pd.read_excel("l-3-5pydata_400mm.xlsx", sheet_name=0)
NNODE = A.iat[0,0]
NMEMB = A.iat[0,1]
COLUMN = A.iat[0,2]
BEAM = A.iat[0,3]
#断面積、荷重、塑性断面係数,全塑性モーメントの計算関数
def mat(H,B,t1,t2,r):
    memb = t1*(H-2*t2)+2*B*t2+0.858*r**2
    w = 0.00785*memb
    Zx = B*t2*(H-t2) + (((H-2*t2)**2)*t1)/4 + 0.4292*(r**2)*(H-2*t2 - 0.4467*r)
    tpm = 235*Zx
    return memb,w,Zx,tpm

#崩壊荷重計算の関数
def ALM(NNODE, NMEMB, COLUMN, BEAM,¥
        H_co, B_co, t1_co, t2_co, r_co, H_be, B_be, t1_be, t2_be, r_be,¥
        H_co1, B_co1, t1_co1, t2_co1, r_co1, H_be1, B_be1, t1_be1, t2_be1, r_be1,¥
        H_co2, B_co2, t1_co2, t2_co2, r_co2, H_be2, B_be2, t1_be2, t2_be2, r_be2,¥
        H_co3, B_co3, t1_co3, t2_co3, r_co3, H_be3, B_be3, t1_be3, t2_be3, r_be3,¥
        H_co4, B_co4, t1_co4, t2_co4, r_co4, H_be4, B_be4, t1_be4, t2_be4, r_be4):
    np.set_printoptions(precision=3,suppress=True)
    NCOL = 5*NMEMB
    NROW = 3*NNODE + 2*NMEMB
```

```
#各部材の全塑性モーメント
tpm_column = mat(H_co,B_co,t1_co,t2_co,r_co)[3]
tpm_beam = mat(H_be,B_be,t1_be,t2_be,r_be)[3]
tpm_column1 = mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)[3]
tpm_beam1 = mat(H_be1,B_be1,t1_be1,t2_be1,r_be1)[3]
tpm_column2 = mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)[3]
tpm_beam2 = mat(H_be2,B_be2,t1_be2,t2_be2,r_be2)[3]
tpm_column3 = mat(H_co3,B_co3,t1_co3,t2_co3,r_co3)[3]
tpm_beam3 = mat(H_be3,B_be3,t1_be3,t2_be3,r_be3)[3]
tpm_column4 = mat(H_co4,B_co4,t1_co4,t2_co4,r_co4)[3]
tpm_beam4 = mat(H_be4,B_be4,t1_be4,t2_be4,r_be4)[3]



I = NROW
J = NCOL
H = zeros((I,J))
P = zeros((I))
I = NCOL
R = zeros((I))
IHING = zeros((I))
IM = NMEMB
AMO = zeros((IM,2))
I = 4
W = zeros((IM,I))

AF = []
AS = ["","FIX","PIN","ROL"]

#-------sheet2-------

B = pd.read_excel("l-3-5pydata_400mm.xlsx", sheet_name=1)

J = B["J"]
#print(J)
```

```
#print(J[3])
ISUP = B["ISUP"]
XNODE = B["XNODE"]
YNODE = B["YNODE"]

for i in range(NNODE):
    ISUP1 = ISUP[i]
    AF.append(AS[ISUP1])

#-------sheet3-------sheet4-------sheet5-------
C = pd.read_excel("l-3-5pydata_400mm.xlsx", sheet_name=2)
AL = pd.read_excel("l-3-5pydata_400mm.xlsx", sheet_name=3)
FYY = pd.read_excel("l-3-5pydata_400mm.xlsx", sheet_name=4)

#print(FY)

M = C["M"]
N1 = C["N1"]
N2 = C["N2"]

#print(AL.iat[2,2])
AL2 = AL["AL2"]
AL3 = AL["AL3"]
AL4 = AL["AL4"]
AL5 = AL["AL5"]

FY = zeros((NMEMB,10))
for i in range(NMEMB):
    for j in range(2):
        FY[i,j] = FYY.iat[i,j]
for j in range(2,10):
    for i in range(COLUMN):
        FY[i,j] = tpm_column
    for i in range(COLUMN,COLUMN+BEAM):
        FY[i,j] = tpm_beam
    for i in range(COLUMN):
```

```
            FY[i,j] = tpm_column1
        for i in range(COLUMN,COLUMN+BEAM):
            FY[i,j] = tpm_beam1
        for i in range(COLUMN):
            FY[i,j] = tpm_column2
        for i in range(COLUMN,COLUMN+BEAM):
            FY[i,j] = tpm_beam2
        for i in range(COLUMN):
            FY[i,j] = tpm_column3
        for i in range(COLUMN,COLUMN+BEAM):
            FY[i,j] = tpm_beam3
        for i in range(COLUMN):
            FY[i,j] = tpm_column4
        for i in range(COLUMN,COLUMN+BEAM):
            FY[i,j] = tpm_beam4

    Q=                                          pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,2,3,4,5,7,8,9,10,12,13,14,15,17,18,19,20,21,22,23,24,25,26,27,2
8,29,30,31,32,33,34,35])
    Q=mat(H_co,B_co,t1_co,t2_co,r_co)   #tpm_column
    Q1=                                         pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,3,4,5,6,8,9,10,11,13,14,15,16,18,19,20,21,22,23,24,25,26,27,2
8,29,30,31,32,33,34,35])
    Q1= mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)
    Q2=                                         pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,4,5,6,7,9,10,11,12,14,15,16,17,19,20,21,22,23,24,25,26,27,2
8,29,30,31,32,33,34,35])
    Q2= mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)
    Q3=pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,5,6,7,8,10,11,12,13,15,16,17,18,20,21,22,23,24,25,26,27,2
8,29,30,31,32,33,34,35])
    Q3=mat(H_co3,B_co3,t1_co3,t2_co3,r_co3)
    Q4=pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,6,7,8,9,11,12,13,14,16,17,18,19,21,22,23,24,25,26,27,28
,29,30,31,32,33,34,35])
    Q4=mat(H_co4,B_co4,t1_co4,t2_co4,r_co4)
```

```
    Q5=                                                    pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,22,23,24,25,
27,28,29,30,32,33,34,35])
    Q5=mat(H_be,B_be,t1_be,t2_be,r_be) #tpm_beam
    Q6                    =                    pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,23,24,25,
26,28,29,30,31,33,34,35])
    Q6=mat(H_be1,B_be1,t1_be1,t2_be1,r_be1)
    Q7                    =                    pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,24,25,
26,27,29,30,31,32,34,35])
    Q7=mat(H_be2,B_be2,t1_be2,t2_be2,r_be2)
    Q8=pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,25,
26,27,28,30,31,32,33,35])
    Q8=mat(H_be3,B_be3,t1_be3,t2_be3,r_be3)
    Q9=pd.read_excel("l-3-3pydata.xlsx",
sheet_name=2,skiprows=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
26,27,28,29,31,32,33,34])
    Q9=mat(H_be4,B_be4,t1_be4,t2_be4,r_be4)
```

```
    AL = []
    CS = []
    SN = []

    for i in range(NMEMB):
        NL = N1[i]
        NR = N2[i]
        DELX = XNODE[NR] - XNODE[NL]
```

```
    DELY = YNODE[NR] - YNODE[NL]

    AL1 = np.sqrt(DELX**2 + DELY**2)
    CS1 = DELX/AL1
    SN1 = DELY/AL1

    AL = np.insert(AL,0,AL1,axis = 0)
    CS.append(CS1)
    SN.append(SN1)

#print(AL,CS,SN)

#-------sheet6------
D = pd.read_excel("l-3-5pydata_400mm.xlsx",sheet_name = 5)
IN = D["IN"]
X1 = D["X1"]
X2 = D["X2"]
X3 = D["X3"]


for I in IN:
    II = I*3
    P[II] = X1[I]
    P[II+1] = X2[I]
    P[II+2] = X3[I]



#-------sheet7-------
E = pd.read_excel("l-3-5pydata_400mm.xlsx",sheet_name = 6)

IM = E["IM"]
X1 = E["X1"]
X2 = E["X2"]
X3 = E["X3"]
X4 = E["X4"]
```

```
IM = len(IM)

W = zeros((IM,4))
#print(W)

for I in range(IM):
    W[I,0] = X1[I]
    W[I,1] = X2[I]
    W[I,2] = X3[I]
    W[I,3] = X4[I]

#-------the equiliblium equation for the member in the-------
#-------member coordinate system (H matrix,P matrix)-------
#-------TRANSGORMATION OF THE COORDINATE SYSTEM-------
#-------ASSEMBLING OF THE TOTAL H MATRIX AND FORCE VECTOR-------

def eq():
    for L in range(2):
        NL = L*3
        for j in range(4):
            X = zeros((3))
            for i in range(3):
                A = 0
                for k in range(3):
                    A = A+T[i,k]*AM[k+NL,j]
                X[i] = A
            for i in range(3):
                AM[i+NL,j] = X[i]

    for x in range(8):
        II = N1[im]*3+x
        if x >= 3 and x < 6:
            II = (N2[im]-1)*3+x
        if x >= 6:
            II = NNODE*3+(im-3)*2+x
```

```
            for y in range(4):
                if y == 3:
                    P[II] = P[II]+AM[x,y]
                    break
                JJ = im*3+y
                H[II,JJ] = AM[x,y]

for im in range(NMEMB):
    T = zeros((3,3))
    T[0,0] = CS[im]
    T[1,0] = SN[im]
    T[0,1] = -SN[im]
    T[1,1] = CS[im]
    T[2,2] = 1
    #print(T)

    AM = zeros((8,4))
    A = 1 / (AL[im] - AL2[im] - AL3[im])
    #print(A)
    B1 = AL2[im]*A
    B2 = AL3[im]*A
    AM[0,0] = -1
    AM[1,1] = A
    AM[1,2] = -A
    AM[2,1] = 1+B1
    AM[2,2] = -B1
    AM[3,0] = 1
    AM[4,1] = -A
    AM[4,2] = A
    AM[5,1] = B2
    AM[5,2] = -(1+B2)
    #print(AM)

    AA = AL4[im] + AL5[im]
    if AA == 0:
        eq()
```

```
        continue

    C11 = AL[im] - AL3[im] - AL4[im]
    C21 = AL4[im] - AL2[im]
    C12 = AL[im] - AL3[im] - AL5[im]
    C22 = AL5[im] - AL2[im]
    #print(C11)

    AM[6,1] = -A*C11
    AM[6,2] = -A*C21
    AM[7,1] = -A*C12
    AM[7,2] = -A*C22

    WX1 = W[im,0]
    WY1 = W[im,1]
    WX2 = W[im,2]
    WY2 = W[im,3]

    AM[0,3] = ((C11+B2/A)*WX1+(C12+B2/A)*WX2)/AL[im]
    AA = C11*WY1+C12*WY2


    AM[1,3] = A*AA
    AM[2,3] = B1*AA

    AM[3,3] = ((C21+B1/A)*WX1 + (C22+B1/A)*WX2)/AL[im]
    AA = C21*WY1 + C22*WY2
    AM[4,3] = A*AA
    AM[5,3] = -B2*AA
    AM[6,3] = C21*AM[1,3]
    AM[7,3] = C22*AM[1,3]
    AA = AL5[im] - AL4[im]
    if AA < 0:
        AM[6,3] = AM[6,3] + AA*WY2
    if AA > 0:
        AM[7,3] = AM[7,3] - AA*WY1
```

```
        AMO[im,0] = AM[6,3]
        AMO[im,1] = AM[7,3]
        eq()#つり合い方程式の作成
N = NMEMB*2
for v in range(N):
        II = NNODE*3+v
        JJ = NMEMB*3+v
        H[II,JJ] = 1
ALM = 0
#ジョルダン
def jordan(I,JC):
        PIV = H[I,JC]
        if PIV == 0:
                print(I,JC,"-------STOP AT JORDAN-------")
                sys.exit()
        for J in range(NCOL):
                H[I,J] = H[I,J]/PIV
        P[I] = P[I]/PIV
        for i in range(NROW):
                if i == I:
                        continue
                else:
                        PIV = H[i,JC]
                        for J in range(NCOL):
                                H[i,J] = H[i,J] - H[I,J]*PIV
                        P[i] = P[i] - P[I]*PIV

#何回も出てくるところを変数化
def gj_510():
        PIV = abs(H[I,0])
        #print(PIV)
        JCOL[I] = 0
        #JCOL は H 座標 I 行の列番号を示すため、0 はじまりにする
        for J in range(1,NCOL):
                AA = abs(H[I,J])
                if AA > PIV:
```

```
                PIV = AA
                JCOL[I] = J
                continue
            else:
                continue
        JC = JCOL[I]
        #print(JC)
        JC = int(JC)
        return JC

JCOL = np.ones((NROW))
JCOL = -1*JCOL
#print(type(H))

for I in range(NROW):
        #JCOL[I] = 0
        if I > 3*NNODE-1:
            IM = int((I-3*NNODE)/2)
            J = int((I-3*NNODE-2*IM)*2)
            A = 0
            for jj in range(2):
                A = A + abs(W[IM,J+jj])
            if A == 0:
                continue
            else:
                JC = gj_510()
                jordan(I,JC)
                #JCOL[I] = JCOL[I] + 1
                continue
        else:
            IN = I/3
            IN = int(IN)
            ISUP1 = ISUP[IN] + 1
            #ISUP 内の変数ＩＮは部材番号であり、ISUP 自体は支持状態
            #Fortran においても 0，1，2，3 で表されるのでこのままでいい
            #また、ISUP1 に関しても、次に行う処理を指示するための変数なので、原文
```

と変える必要はない

```
            if ISUP1 == 1:
                JC = gj_510()
                jordan(I,JC)
                #JCOL[I] = JCOL[I]+1
                continue
            elif ISUP1 == 2:
                continue
            elif ISUP1 == 3:
                aa = I - 3*IN
                if aa == 0:
                    JC = gj_510()
                    jordan(I,JC)
                    #JCOL[I] = JCOL[I] + 1
                    continue
                else:
                    continue
            elif ISUP1 == 4:
                bb = I - 3*IN + 1
                if bb == 0:
                    continue
                else:
                    JC = gj_510()
                    jordan(I,JC)
                    #JCOL[I] = JCOL[I] + 1
                    continue

    #------STEP ONE-------

    #以下、JC を数値として扱う場合は＋ 1 された状態、番号の場合 JCOL の値そのままにする
    NC = 0
    while NC <= NROW:
        # if NC >= NROW:
        #         print("MECHANISM WAS NOT REACHED WITHIN NROW
```

```
CYCLES,TWO PROCEDURE INTERRUPTED")
        #       print("STOP")
        #       break

        DLMK = 10000 #1000 ~ -100000
        NC = NC + 1
        for I in range(NROW):
            JC = int(JCOL[I]+1)
            if JC == 0:
                continue
            elif JC > 3*NMEMB:
                IM = int((JC-3*NMEMB-1)/2 + 1)
                JJ = int(JC-3*NMEMB-2*(IM-1)+3)
            else:
                IM = int((JC-1)/3 + 1)
                JJ = int(JC-3*(IM-1))
            PP = P[I]
            if PP > 0:
                DLM = ((FY[IM-1,2*JJ-2])-R[JC-1])/P[I]
            elif PP == 0:
                continue
            elif PP < 0:
                DLM = ((-FY[IM-1,2*JJ-1])-R[JC-1])/P[I]
            if DLM >= DLMK:
                continue
            DLMK = DLM
            K = I

    #-------K COMPONENT NO YATU-------

        JC = int(JCOL[K]+1)
        IHING[JC-1] = 1
        if P[K] < 0:
            IHING[JC-1] = -1
        ALM = ALM + DLMK
        for I in range(NROW):
```

```
        JC = int(JCOL[I]+1)
        if JC == 0:
            continue
        R[JC-1] = R[JC-1]+DLMK*P[I]
        continue

#------STEP TWO-------

ZMAX = -10000 #-1000 ~ -100000
for J in range(NCOL):
        for I in range(NROW):
            JC = JCOL[I]+1
            if J == JC:
                break
        if J == JC-1:
            continue
        else:
            Z = H[K,J]*P[K]/abs(P[K])
            if R[J] == 0:
                Z = abs(Z)
            elif R[J] > 0:
                Z = -Z
            if Z > ZMAX:
                ZMAX = Z
                L = J
                L = int(L)
            else:
                continue
if ZMAX <= 0:
        return ALM
if NC == NROW:
        return ALM
else:
        #print(H)
        IHING[L] = 0
        JC = int(JC-1)
```

```
            jordan(K,L)
            continue

#最適化の準備と実行__崩壊荷重、柱と梁の断面積
def objective(vars):
    H_co = vars[0]
    B_co = vars[1]
    t1_co = vars[2]
    t2_co = vars[3]
    r_co = vars[4]
    H_co1 = vars[5]
    B_co1 = vars[6]
    t1_co1 = vars[7]
    t2_co1 = vars[8]
    r_co1 = vars[9]
    H_co2 = vars[10]
    B_co2 = vars[11]
    t1_co2 = vars[12]
    t2_co2 = vars[13]
    r_co2 = vars[14]
    H_co3 =vars[15]
    B_co3 = vars[16]
    t1_co3 = vars[17]
    t2_co3 = vars[18]
    r_co3 = vars[19]
    H_co4 = vars[20]
    B_co4 = vars[21]
    t1_co4 = vars[22]
    t2_co4 = vars[23]
    r_co4 = vars[24]
    H_be = vars[25]
    B_be = vars[26]
    t1_be = vars[27]
    t2_be = vars[28]
    r_be = vars[29]
    H_be1 = vars[30]
```

```
    B_be1 = vars[31]
    t1_be1 = vars[32]
    t2_be1 = vars[33]
    r_be1 = vars[34]
    H_be2 = vars[35]
    B_be2 = vars[36]
    t1_be2 = vars[37]
    t2_be2 = vars[38]
    r_be2 = vars[39]
    H_be3 = vars[40]
    B_be3 = vars[41]
    t1_be3 = vars[42]
    t2_be3 = vars[43]
    r_be3 = vars[44]
    H_be4 = vars[45]
    B_be4 = vars[46]
    t1_be4 = vars[47]
    t2_be4 = vars[48]
    r_be4 = vars[49]



    f1 = 1/ALM(NNODE, NMEMB, COLUMN, BEAM,¥
                H_co, B_co, t1_co, t2_co, r_co, H_be, B_be, t1_be, t2_be, r_be,¥
                H_co1, B_co1, t1_co1, t2_co1, r_co1, H_be1, B_be1, t1_be1, t2_be1,
r_be1,¥
                H_co2, B_co2, t1_co2, t2_co2, r_co2, H_be2, B_be2, t1_be2, t2_be2,
r_be2,¥
                H_co3, B_co3, t1_co3, t2_co3, r_co3, H_be3, B_be3, t1_be3, t2_be3,
r_be3,¥
                H_co4, B_co4, t1_co4, t2_co4, r_co4, H_be4, B_be4, t1_be4, t2_be4, r_be4)
    f2 = mat(H_co,B_co,t1_co,t2_co,r_co)[0]
    f3 = mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)[0]
    f4 = mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)[0]
    f5 = mat(H_co3,B_co3,t1_co3,t2_co3,r_co3)[0]
    f6 = mat(H_co4,B_co4,t1_co4,t2_co4,r_co4)[0]
```

```
    f7 = mat(H_be,B_be,t1_be,t2_be,r_be)[0]
    f8 = mat(H_be1,B_be1,t1_be1,t2_be1,r_be1)[0]
    f9 = mat(H_be2,B_be2,t1_be2,t2_be2,r_be2)[0]
    f10 = mat(H_be3,B_be3,t1_be3,t2_be3,r_be3)[0]
    f11 = mat(H_be4,B_be4,t1_be4,t2_be4,r_be4)[0]


   # v1=-mat(H_co,B_co,t1_co,t2_co,r_co)[0]+mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)[0]
    #v2=-
mat(H_co1,B_co1,t1_co1,t2_co1,r_co1)[0]+mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)[0]
    #v3=-
mat(H_co2,B_co2,t1_co2,t2_co2,r_co2)[0]+mat(H_co3,B_co3,t1_co3,t2_co3,r_co3)[0]
    #v4=-
mat(H_co3,B_co3,t1_co3,t2_co3,r_co3)[0]+mat(H_co4,B_co4,t1_co4,t2_co4,r_co4)[0]


    return[f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11] #,[v1,v2,v3,v4]

def optimization(n_var,n_obj,vars,n_run):
    #設計変数と目的関数の値を設定
    problem = Problem(n_var,n_obj)
    #最適化問題に最小化を設定
    problem.directions[:] = Problem.MINIMIZE
    #problem.constraints[:] = "<=0"
    #設計変数と目的関数を設定
    problem.types[:] = vars
    problem.function = objective
    #最適化アルゴリズムを設定して計算する
    algorithm = NSGAII(problem,population_size=3)
    algorithm.run(n_run)

    return algorithm

#変数を設定する
vars1 = Real(100.,400.) #100-350
vars2 = Real(50.,350.) #50-350
```

vars3 = Real(5.,50.)
vars4 = Real(5.,80.)
vars5 = Real(5.,25.)
vars6 = Real(100.,400.)
vars7 = Real(50.,350)
vars8 = Real(5.,50.)
vars9 = Real(5.,80.)
vars10 = Real(5.,25.)
vars11 = Real(100.,400.)
vars12 = Real(50.,350.)
vars13 = Real(5.,50.)
vars14 = Real(5.,80.)
vars15 = Real(5.,25.)
vars16 = Real(100.,400.)
vars17 = Real(50.,350.)
vars18 = Real(5.,50.)
vars19 = Real(5.,80.)
vars20 = Real(5.,25.)
vars21 = Real(100.,400.)
vars22 = Real(50.,350.)
vars23 = Real(5.,50.)
vars24 = Real(5.,80.)
vars25 = Real(5.,25.)
vars26 = Real(100.,400.) #100-400
vars27 = Real(50.,300.) #50-300
vars28 = Real(5.,50.)
vars29 = Real(5.,80.)
vars30 = Real(5.,25.)
vars31 = Real(100.,400.) #100-400
vars32 = Real(50.,300.)
vars33 = Real(5.,50.)
vars34 = Real(5.,80.)
vars35 = Real(5.,25.)
vars36 = Real(100.,400.) #100-400
vars37 = Real(50.,300)
vars38 = Real(5.,50.)

```
vars39 = Real(5.,80.)
vars40 = Real(5.,25.)
vars41 = Real(100.,400.) #100-400
vars42 = Real(50.,300.)
vars43 = Real(5.,50.)
vars44 = Real(5.,80.)
vars45 = Real(5.,25.)
vars46 = Real(100.,400.) #100-400
vars47 = Real(50.,300.)
vars48 = Real(5.,50.)
vars49 = Real(5.,80.)
vars50 = Real(5.,25.)


vars = [vars1,vars2,vars3,vars4,vars5,vars6,vars7,vars8,vars9,vars10,¥
        vars11,vars12,vars13,vars14,vars15,vars16,vars17,vars18,vars19,vars20,¥
        vars21,vars22,vars23,vars24,vars25,vars26,vars27,vars28,vars29,vars30,¥
        vars31,vars32,vars33,vars34,vars35,vars36,vars37,vars38,vars39,vars40,¥
        vars41,vars42,vars43,vars44,vars45,vars46,vars47,vars48,vars49,vars50]

#最適化計算を実行する
algorithm = optimization(n_var=len(vars),n_obj=11,vars=vars,n_run=3000)

#グラフの設定
plt.rcParams['font.size'] = 14
plt.rcParams['font.family'] = 'Times New Roman'
fig = plt.figure(figsize=(6, 6))
ax1 = fig.add_subplot(111,projection="3d")

# 軸のラベルを設定する。
ax1.set_xlabel('load')
ax1.set_ylabel('column')
ax1.set_zlabel('beam')

#解の抽出
obj1 = []
```

```
obj2 = []
obj3 = []
obj4 = []
obj5 = []
obj6 = []
obj7 = []
obj8 = []
obj9 = []
obj10 = []
obj11 = []



for solution in algorithm.result:
    obj1.append(solution.objectives[0])
    obj2.append(solution.objectives[1])
    obj3.append(solution.objectives[2])
    obj4.append(solution.objectives[3])
    obj5.append(solution.objectives[4])
    obj6.append(solution.objectives[5])
    obj7.append(solution.objectives[6])
    obj8.append(solution.objectives[7])
    obj9.append(solution.objectives[8])
    obj10.append(solution.objectives[9])
    obj11.append(solution.objectives[10])



ax1.scatter(obj1, obj2, obj7, color='gray', edgecolor='gray', label='Trial', alpha=0.5)
ax1.scatter(obj1, obj3, obj8, color='gray', edgecolor='gray', label='Trial', alpha=0.5)
ax1.scatter(obj1, obj4, obj9, color='gray', edgecolor='gray', label='Trial', alpha=0.5)
ax1.scatter(obj1, obj5, obj10, color='gray', edgecolor='gray', label='Trial', alpha=0.5)
ax1.scatter(obj1, obj6, obj11, color='gray', edgecolor='gray', label='Trial', alpha=0.5)
```

```
# パレート解をプロットする。
n_dominated = nondominated(algorithm.result)      # 非劣解を抽出する
obj1 = []
obj2 = []
obj3 = []
obj4 = []
obj5 = []
obj6 = []
obj7 = []
obj8 = []
obj9 = []
obj10 = []
obj11 = []

for solution in n_dominated:
    obj1.append(solution.objectives[0])
    obj2.append(solution.objectives[1])
    obj3.append(solution.objectives[2])
    obj4.append(solution.objectives[3])
    obj5.append(solution.objectives[4])
    obj6.append(solution.objectives[5])
    obj7.append(solution.objectives[6])
    obj8.append(solution.objectives[7])
    obj9.append(solution.objectives[8])
    obj10.append(solution.objectives[9])
    obj11.append(solution.objectives[10])


ax1.scatter(obj1, obj2, obj7, color='yellow', edgecolor='black', label='Pareto front')
ax1.scatter(obj1, obj3, obj8, color='red', edgecolor='black', label='Pareto front')
ax1.scatter(obj1, obj4, obj9, color='blue', edgecolor='black', label='Pareto front')
ax1.scatter(obj1, obj5, obj10, color='green', edgecolor='black', label='Pareto front')
ax1.scatter(obj1, obj6, obj11, color='orange', edgecolor='black', label='Pareto front')
ax1.legend()
```

```
# グラフを表示する。
plt.show()
plt.close()



#部材選択プログラム
n_d = len(n_dominated)
result_array = np.zeros((n_d,61))
for i in range(len(n_dominated)):
    for j in range(50):
        result_array[i,j] = n_dominated[i].variables[j]
    for j in range(50,61):
        result_array[i,j] = n_dominated[i].objectives[j-50]


result_column = result_array[:,0:5]
result_column1 = result_array[:,5:10]
result_column2= result_array[:,10:15]
result_column3 = result_array[:,15:20]
result_column4 = result_array[:,20:25]
result_beam = result_array[:,25:30]
result_beam1 = result_array[:,30:35]
result_beam2 = result_array[:,35:40]
result_beam3 = result_array[:,40:45]
result_beam4 = result_array[:,45:50]
result_purpus = result_array[:,50:61]
mat_data = pd.read_excel("mat_data_mm.xlsx")
mat_data = mat_data.to_numpy()
r_c = pd.DataFrame(result_column)
r_b = pd.DataFrame(result_beam)
r_c1 = pd.DataFrame(result_column1)
r_b1 = pd.DataFrame(result_beam1)
r_c2 = pd.DataFrame(result_column2)
r_b2 = pd.DataFrame(result_beam2)
r_c3 = pd.DataFrame(result_column3)
```

```
r_b3 = pd.DataFrame(result_beam3)
r_c4 = pd.DataFrame(result_column4)
r_b4 = pd.DataFrame(result_beam4)


#print("result column")
#print(r_c)
#print(r_c1)
#print(r_c2)
#print("result beam")
#print(r_b)
#print(r_b1)
#print(r_b2)

#誤差を格納する行列作成
err_mat_c = zeros((len(mat_data),5))
err_mat_b = zeros((len(mat_data),5))
err_mat_c1 = zeros((len(mat_data),5))
err_mat_b1 = zeros((len(mat_data),5))
err_mat_c2 = zeros((len(mat_data),5))
err_mat_b2 = zeros((len(mat_data),5))
err_mat_c3 = zeros((len(mat_data),5))
err_mat_b3 = zeros((len(mat_data),5))
err_mat_c4 = zeros((len(mat_data),5))
err_mat_b4 = zeros((len(mat_data),5))


#最適部材の要素を格納する行列
fin_c = []
fin_b = []
fin_c1 = []
fin_b1 = []
fin_c2 = []
fin_b2 = []
fin_c3 = []
```

```
fin_b3 = []
fin_c4 = []
fin_b4 = []


#パレート解の取り出し　ひとつずつ行うため
for m in range(n_d):
    for i in range(len(mat_data)):
        for j in range(5):
            err_mat_c[i,j] = (mat_data[i,j]/result_column[m,j])-1
            err_mat_b[i,j] = (mat_data[i,j]/result_beam[m,j])-1
            err_mat_c1[i,j] = (mat_data[i,j]/result_column1[m,j])-1
            err_mat_b1[i,j] = (mat_data[i,j]/result_beam1[m,j])-1
            err_mat_c2[i,j] = (mat_data[i,j]/result_column2[m,j])-1
            err_mat_b2[i,j] = (mat_data[i,j]/result_beam2[m,j])-1
            err_mat_c3[i,j] = (mat_data[i,j]/result_column3[m,j])-1
            err_mat_b3[i,j] = (mat_data[i,j]/result_beam3[m,j])-1
            err_mat_c4[i,j] = (mat_data[i,j]/result_column4[m,j])-1
            err_mat_b4[i,j] = (mat_data[i,j]/result_beam4[m,j])-1



    # print("column's err")
    # print(err_mat_c)
    # print("beam's err")
    # print(err_mat_b)
    #変数の誤差合計を格納する行列
    err_tot_c = []
    err_tot_b = []
    err_tot_c1 = []
    err_tot_b1 = []
    err_tot_c2 = []
    err_tot_b2 = []
    err_tot_c3 = []
    err_tot_b3 = []
    err_tot_c4 = []
```

```
err_tot_b4 = []


#誤差の大きさを行列にまとめる
for i in range(len(mat_data)):
    err_tot_c.append(sum(abs(err_mat_c[i])))
    err_tot_b.append(sum(abs(err_mat_b[i])))
    err_tot_c1.append(sum(abs(err_mat_c1[i])))
    err_tot_b1.append(sum(abs(err_mat_b1[i])))
    err_tot_c2.append(sum(abs(err_mat_c2[i])))
    err_tot_b2.append(sum(abs(err_mat_b2[i])))
    err_tot_c3.append(sum(abs(err_mat_c3[i])))
    err_tot_b3.append(sum(abs(err_mat_b3[i])))
    err_tot_c4.append(sum(abs(err_mat_c4[i])))
    err_tot_b4.append(sum(abs(err_mat_b4[i])))


#誤差の合計値が最も小さいものを選択,部材の選択
t_c = err_tot_c.index(min(err_tot_c))
t_b = err_tot_b.index(min(err_tot_b))
t_c1 = err_tot_c1.index(min(err_tot_c1))
t_b1 = err_tot_b1.index(min(err_tot_b1))
t_c2 = err_tot_c2.index(min(err_tot_c2))
t_b2 = err_tot_b2.index(min(err_tot_b2))
t_c3 = err_tot_c3.index(min(err_tot_c3))
t_b3 = err_tot_b3.index(min(err_tot_b3))
t_c4 = err_tot_c4.index(min(err_tot_c4))
t_b4 = err_tot_b4.index(min(err_tot_b4))


fin_c.append(mat_data[t_c])
fin_b.append(mat_data[t_b])
fin_c1.append(mat_data[t_c1])
fin_b1.append(mat_data[t_b1])
```

```
        fin_c2.append(mat_data[t_c2])
        fin_b2.append(mat_data[t_b2])
        fin_c3.append(mat_data[t_c3])
        fin_b3.append(mat_data[t_b3])
        fin_c4.append(mat_data[t_c4])
        fin_b4.append(mat_data[t_b4])



#最適部材を見やすくしてる
fin_c = pd.DataFrame(fin_c)
fin_b = pd.DataFrame(fin_b)
fin_c1 = pd.DataFrame(fin_c1)
fin_b1 = pd.DataFrame(fin_b1)
fin_c2 = pd.DataFrame(fin_c2)
fin_b2 = pd.DataFrame(fin_b2)
fin_c3 = pd.DataFrame(fin_c3)
fin_b3 = pd.DataFrame(fin_b3)
fin_c4 = pd.DataFrame(fin_c4)
fin_b4 = pd.DataFrame(fin_b4)



print("Column's optimal solution")
print(fin_c)
print(fin_c1)
print(fin_c2)
print(fin_c3)
print(fin_c4)

print("Beam's optimal solution")
print(fin_b)
print(fin_b1)
print(fin_b2)
print(fin_b3)
```

```
print(fin_b4)


#df = pd.DataFrame([[fin_c],[fin_c1],[fin_c2],[fin_c3],[fin_c4],[fin_c5]],
 #                        index=['1F','2F','3F','4F','5F','6F'], columns=['H','B','t1','t2','r'])
#df1 =pd.DataFrame([[fin_b],[fin_b1],[fin_b2],[fin_b3],[fin_b4],[fin_b5]],
 #                        index=['1F','2F','3F','4F','5F','6F'], columns=['H','B','t1','t2','r'])
#with pd.ExcelWriter('spyder-python/l-3-6-cpm-size-run.xlsx',1 mode='a') as writer:
 #      df.to_excel(writer, sheet_name='sheet1')
  #     df1.to_excel(writer, sheet_name='sheet2')



sys.exit()
#       fin_result.append(["-----","-----","-----","-----","-----"])
#       fin_result.append(["pareto",j,"-----","-----","-----"])
#       fin_result.append(["purpus","---","---","---","---"])
#       fin_result.append(result_purpus[j])
#       fin_result.append(["col","---","---","---","---"])
#       fin_result.append(res_col)
#       fin_result.append(res_col_err)
#       fin_result.append(["beam","---","---","---","---"])
#       fin_result.append(res_beam)
#       fin_result.append(res_beam_err)

# #選択結果の印刷
# print("closest value")
# print("pareto=",n_d,"¥n")
# column_name = ["H","B","t1","t2","r"]
# fin_result = pd.DataFrame(fin_result,columns = column_name)
# print(fin_result)
```