```python
from __future__ import division
#importando bibliotecas do Rhinoceros:
#Rhino Common, rhinoscriptsintax e ghpythonlib
from Rhino.Geometry import Point3d, Line, NurbsCurve
import rhinoscriptsyntax as rs
import ghpythonlib.components as gh

Tr_3D = []
Tr_copias = []
teste=[]
Cargas = []

if not Bz_l:
    Bz_l=Diag_l
if not Peso_esp_Tr:
    Peso_esp_Tr = 7800 #Peso específico do aço Kg/m^3
if not Peso_cobertura:
     Peso_cobertura = 20 #Sobrecarga do telhado Kg/m^2
if not Fator_de_conv:
    Fator_de_conv = 10
if not N_Tr:
    N_Tr = 1

if not Plano:
    Plano = rs.WorldXYPlane()
#decompoe o plano de trabalho nos componentes Origem e os eixos xyz
pOr, eX, eY, eZ = Plano


#Separando os eixos em listas
#conector
Conector = Eixos[-2:]
#vigas
v1 = Eixos[:3]
v2 = Eixos[3:6]
v3 =  Eixos[6:9]
#invertendo sentido da viga v3
rs.ReverseCurve(v3[0])
rs.ReverseCurve(v3[1])
rs.ReverseCurve(v3[2])
#diagonais
Diagonais = [v1[1],v2[1],v3[1]]
#banzos
Banzo_Sup = [v1[0],v2[0],v3[0]]
Banzo_Inf = [v1[2],v2[2],v3[2]]
#apoios
Apoios = [rs.CurveStartPoint(v1[2]),rs.CurveStartPoint(v2[2])]
#Eixo de Simetria do conector
EixoSimetria = Conector[0]
EixoSimetria =  rs.coerceline(EixoSimetria)



def Volumetria(eixos, dist_e, dist_l, pto):

    plAux = rs.CopyObject(eixos,-eZ*(dist_l))

    plAux2 = rs.CopyObject(eixos,eZ*(dist_l))
    pl1 = rs.OffsetCurve(plAux,pto,dist_e,eZ)
    pl1 = rs.coercegeometry(pl1)

    pl2 =  rs.OffsetCurve(plAux,pto,-dist_e,eZ)
    pl2 = rs.coercegeometry(pl2)

    pl3 = rs.AddLine(rs.CurveStartPoint(pl1),rs.CurveStartPoint(pl2))
    pl4 = rs.AddLine(rs.CurveEndPoint(pl1),rs.CurveEndPoint(pl2))
    eixos3D = rs.AddPlanarSrf((pl1,pl3,pl2,pl4))
```

```
    lin = rs.AddLine(rs.CurveStartPoint(plAux),rs.CurveStartPoint(plAux2))

    eixos3D = rs.ExtrudeSurface (eixos3D, lin, True)

    eixos3D = rs.coercegeometry(eixos3D)
    return eixos3D


def Cargas_Vigas(viga, plD, plB , plC , cob, borda1, borda2, conv):
    #
    cob = rs.coercecurve(cob)
    forces = []
    #separando entrada viga em seus elementos (polilinhas)
    #bs = banzo superior, diag = diagonais, bi = banzo inferior
    bs, diag, bi = viga
    #extraindo os nós do banzo superior
    nos = rs.PolylineVertices(bs)
    # explodindo polilinhas
    bs = rs.ExplodeCurves(bs)
    diag = rs.ExplodeCurves(diag)
    bi = rs.ExplodeCurves(bi)
    for i in range(len(nos)):
        P=0
        if 0<i<(len(nos)-1):
            #banzo inferior
            P = rs.CurveLength(bi[i])*plB
            #banzo superior
            P +=(rs.CurveLength(bs[i-1])+rs.CurveLength(bs[i])) /2 *plB
            #diagonais
            P +=( rs.CurveLength(diag[i*2])+rs.CurveLength(diag[(i*2)+1]) )*plD
            #cobertura
            bs1 = bs[i-1]
            bs1 = rs.coerceline(bs1)
            ptAux1 = bs1.PointAt(.5)
            bs2 = bs[i]
            bs2 = rs.coerceline(bs2)
            ptAux2 = bs2.PointAt(.5)
            param1 = rs.CurveClosestPoint(cob,ptAux1)
            param2 = rs.CurveClosestPoint(cob,ptAux2)

        elif i == 0:

            #banzo inferior
            P = rs.CurveLength(bi[i])*plB
            #banzo superior
            P += (rs.CurveLength(bs[i])/2)*plB
            #diagonais
            P +=( rs.CurveLength(diag[i*2])+rs.CurveLength(diag[(i*2)+1]) )*plD
            #cobertura
            param1 = 0
            bs1 = bs[i]
            bs1 = rs.coerceline(bs1)
            ptAux = bs1.PointAt(.5)
            #teste.append(ptAux)
            param2 = rs.CurveClosestPoint(cob,ptAux)

        elif i == (len(nos)-1):
            #banzo superior e conector
            P = (rs.CurveLength(bs[i-1])+rs.CurveLength(borda1)) /2 *plB
            #conector
            P += rs.CurveLength(borda2) *plB
            #cobertura
            bs1 = bs[i-1]
            bs1 = rs.coerceline(bs1)
            ptAux1 = bs1.PointAt(.5)
            borda1 = rs.coerceline(borda1)
            ptAux2  = borda1.PointAt(.5)
```

```
            param1 = rs.CurveClosestPoint(cob,ptAux1)
            param2 = rs.CurveClosestPoint(cob,ptAux2)
            if not len(diag)%2 == 0:
                #banzo inferior
                P += rs.CurveLength(bi[i])*plB
                #diagonal
                P += ( rs.CurveLength(diag[i*2]) )*plD
        P_viga = P
        if not param1 == param2:
            cobAux = rs.TrimCurve(cob,[param1,param2],False)
            P_cob = rs.CurveLength(cobAux)*plC
            P += P_cob
        P *= conv
        forces.append(P)
    return forces, P_viga, P_cob

def Cargas_Shed(viga, plD, plB , plC , cob, borda1, borda2, conv):
    rs.ReverseCurve(cob)
    cob = rs.coercecurve(cob)

    forces = []
    #separando entrada viga em seus elementos (polilinhas)
    #bs = banzo superior, diag = diagonais, bi = banzo inferior
    bs, diag, bi = viga
    #extraindo os nós do banzo superior
    nos = rs.PolylineVertices(bs)
    # explodindo polilinhas
    bs = rs.ExplodeCurves(bs)
    diag = rs.ExplodeCurves(diag)
    bi = rs.ExplodeCurves(bi)
    for i in range(len(nos)):
        P=0
        if 0<i<(len(nos)-1):
            #banzo inferior
            P = rs.CurveLength(bi[i-1])*plB
            #banzo superior
            P += (rs.CurveLength(bs[i-1])+rs.CurveLength(bs[i])) /2 *plB
            #diagonais
            P +=( rs.CurveLength(diag[i*2])+rs.CurveLength(diag[(i*2)-1]) )*plD
            #cobertura
            bs1 = bs[i-1]
            bs1 = rs.coerceline(bs1)
            ptAux1 = bs1.PointAt(.5)
            bs2 = bs[i]
            bs2 = rs.coerceline(bs2)
            ptAux2 = bs2.PointAt(.5)
            param1 = rs.CurveClosestPoint(cob,ptAux1)
            param2 = rs.CurveClosestPoint(cob,ptAux2)

        elif i == 0:

            #banzo superior
            P += (rs.CurveLength(bs[i])/2)*plB
            #diagonal
            P += ( rs.CurveLength(diag[i*2])) *plD
            #cobertura
            param1 = 0
            bs1 = bs[i]
            bs1 = rs.coerceline(bs1)
            ptAux = bs1.PointAt(.5)

            param2 = rs.CurveClosestPoint(cob,ptAux)

        elif i == (len(nos)-1):
            #banzo superior e conector
            P = ( rs.CurveLength(bs[-1])/2 +rs.CurveLength(borda1) )  *plB
            #eixo do conector
```

```
            P += rs.CurveLength(borda2) *plB
            #cobertura
            bs1 = bs[i-1]
            bs1 = rs.coerceline(bs1)
            ptAux1 = bs1.PointAt(.5)
            borda1 = rs.coerceline(borda1)
            ptAux2  = borda1.PointAt(.5)
            param1 = rs.CurveClosestPoint(cob,ptAux1)
            param2 = rs.CurveClosestPoint(cob,ptAux2)
            if  len(diag)%2 == 0:
                #banzo inferior
                P += rs.CurveLength(bi[i-1])*plB
                #diagonal
                P += ( rs.CurveLength(diag[(i*2)-1]) )*plD
        P_viga = P
        if not param1 == param2:
            cobAux = rs.TrimCurve(cob,[param1,param2],False)
            P_cob = rs.CurveLength(cobAux)*plC
            P += P_cob
        P *= conv
        forces.append(P)
    return forces, P_viga, P_cob
def MomInerMinQuad(b,h):
    m1 = (b*(h**3))/12
    m2 = (h*(b**3))/12
    if m1 < m2:
        return m1
    else:
        return m2

### - treliça 3D - ###
if Mostrar_Volume == None:
    Mostrar_Volume = True
if Mostrar_Volume:
# ponto de orientação dos offsets
    ptX = EixoSimetria.PointAt(.5)
# 3D das diagonais e eixo do conector
    DiagEixo = Diagonais
    for i in DiagEixo:
        vol = Volumetria(i,Diag_e/2,Diag_l/2,ptX)
        Tr_3D.append(vol)
# 3D dos banzos e bordas do conector
    BzEixo = Banzo_Sup + Banzo_Inf + Conector
    for i in BzEixo:
        vol = Volumetria(i,Bz_e/2,Bz_l/2,ptX)
        Tr_3D.append(vol)
# 3D da cobertura
    for i in Cobertura:
        vol = Volumetria(i,.01,Dist_entre_eixos/2,ptX)
        Tr_3D.append(vol)
# copiando treliças de acordo com a distancia entre os eixos
    for i in range(1,N_Tr,1):
        Tr_copias += gh.Move(Tr_3D,i*Dist_entre_eixos*eZ)[0]

### - cargas - ###
#area das diagonais
AreaD = Diag_e * Diag_l
### menor momento de inercia das diagonais
ID = MomInerMinQuad(Diag_l,Diag_e)
#area do banzo
AreaB = Bz_e * Bz_l
### menor momento de inercia dos Banzos
IB = MomInerMinQuad(Bz_l,Bz_e)
#peso linear das diagonais
plDiag =  AreaD * Peso_esp_Tr
#peso linear do banzo
plBz = AreaB * Peso_esp_Tr
```

```
#peso linear da cobertura
plCob = Peso_cobertura * Dist_entre_eixos

ConectBorda = Conector[1:]
ConectBorda = rs.ExplodeCurves(ConectBorda)

#calculando as cargas na viga v1
# P_v1 = lista de cargas nos nós superiores da viga v1
# PP_v1 = peso próprio da viga v1
# P_cob1 = peso da cobertura sobe da viga v1
P_v1, PP_v1, P_cob1 = Cargas_Vigas(v1, plDiag, plBz, plCob, Cobertura[0], ConectBorda[
#calculando as cargas na viga v2
# P_v2 = lista de cargas nos nós superiores da viga v2
# PP_v2 = peso próprio da viga v2
# P_cob2 = peso da cobertura sobe da viga v2
P_v2, PP_v2, P_cob2 = Cargas_Vigas(v2, plDiag, plBz, plCob, Cobertura[1], ConectBorda[
#calculando as cargas no shed
# P_v1 = lista de cargas nos nós superiores do shed
# PP_v1 = peso próprio do shed
# P_cob1 = peso da cobertura sobe do shed
P_v3, PP_v3, P_cob3 = Cargas_Shed(v3, plDiag, plBz, plCob, Cobertura[0], ConectBorda[0
P_v3.reverse()
# lista dos modulos das cargas nos nós das vigas e shed
Cargas = [P_v1, P_v2, P_v3]
# peso próprio total
PP_total = PP_v1 + PP_v2 + PP_v3
print 'Peso próprio da viga = ', PP_total
# sobrecarga da cobertura total
Pcob_total = P_cob1 + P_cob2 + P_cob3
print 'Sobrecarca da cobertura = ', Pcob_total
# saida com as áreas das seções do banzoe  das diagonais
Areas = [[AreaB, AreaD],[IB,ID]]
```