```
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

def EixosVigas(BZ,DiagN,d1,od1,d2,od2):
    #----Diagnoais----
    #dividir o banzo superior 1 em 2 x (numero de diagonais de v)
    ptB1 = rs.DivideCurve(BZ,(2*DiagN))
    #a variável EPcurv armazena o start point do eixo do banzo superior
    SPCurv = ptB1[0]
 #exclui o primeiro valor da lista (start point da curva do banzo superior)
    ptB1 = ptB1[1:]
    #pAux é o ponto de partida do eixo do banzo inferior
    pAux = gh.Move(Point3d(SPCurv),-d2*eY)[0]
    #Offsets da curva do banzo sup nas distancias d1 e d2
    cAux1 = rs.OffsetCurve(BZ,od1,d1,eZ)
    cAux2 = rs.OffsetCurve(BZ,od2,d2,eZ)
    #-aplicando mascara binária
    #ptB1 = lista dos nós do banzo sup
    ptB1 = gh.Dispatch(ptB1,[False,True,False,False])[0]
    # iniciando lista dos nós do bazo infeirior
    apoio=pAux
    ptB2=[pAux]
    #-calculando nós do banzo inferior
    #contador FOR do primeiro ao penultimo item de ptb1
    for i in range(len(ptB1)-1):
    #desenha linhas etrne o ponto atual e próximo ponto de ptB1
        linAux = Line(ptB1[i],ptB1[i+1])
    #pAux = ponto médio de linAux
        pAux3 = linAux.PointAt(.5)
    #calcula ponto em cAux1 mais proximo de pAux
    #pAux1 =[(coordenadas do pto),(parametro do pto),(distância pto-crv)]
        pAux1 = gh.CurveClosestPoint(pAux3,rs.coercecurve(cAux1))
        #idem - cAux2
        #pAux2 = idem
        pAux2 = gh.CurveClosestPoint(pAux3,rs.coercecurve(cAux2))
        #linha entre os pontos equivalentes das curvas cAux1 e cAux2
        linAux2 = Line(pAux2[0],pAux1[0])
        #cAux3 = cAux1 cortada (trim) no ponto pAux1
        cAux3= rs.TrimCurve(cAux1,(0,pAux1[1]),False)
    #razão entre o comprimento da curva cAux1 até o ponto pAux1
    #e o compriento total de cAux1
        prmt = rs.CurveLength(cAux3)/rs.CurveLength(cAux1)
        #ponto que interpola as alturas VH1 e VH2
        pAux =linAux2.PointAt(prmt)
        #coloca o pto pAux na lista dos nós
        ptB2.append(pAux)
 #o ponto final do banzo inferior é colocado no final da lista ptB2
    ptB2.append(rs.CurveEndPoint(cAux1))
    #-ajuste do ultimo ponto da treliça
    nosD = gh.Weave([0,1],ptB2,ptB1)

    if DiagN%2 == 0:
        ptB1.append(rs.CurveEndPoint(BZ))
    else:
        del nosD[-1]
    #desenha diagonais
    lDiag = rs.AddPolyline(nosD)

    #desenha banzo inferior
    bzInf = rs.AddPolyline(ptB2)
    #desenha banzo superior
    bzSup=rs.AddPolyline(ptB1)
    return lDiag,bzSup,bzInf,nosD,ptB1,ptB2,apoio
```

```
def EixosShed(BZ,DiagN,pto):

    #----Shed----
    #dividir o banzo superior 1 em 2 x (numero de diagonais de v3)
    ptB1 = rs.DivideCurve(BZ,(DiagN))

    #exclui o primeiro valor da lista (start point da curva do banzo superior)
    if DiagN%2==0:
        mask = [True,False]
    else:
        SPCurv=ptB1[0]
        mask = [False,True]
    #calcula ponto em cAux1 mais proximo de pAux
    #pAux =[(coordenadas do pto),(parametro do pto),(distância pto-crv)]
    pAux = gh.CurveClosestPoint(pto,BZ)
    #Offsets da curva do banzo sup no ponto pto
    cAux1 = rs.OffsetCurve(BZ,pto,pAux[2])

    #c.append(rs.coercecurve(cAux1))
    #-aplicando mascara binária
    #ptB1 = lista dos nós do banzo sup
    ptB1 = gh.Dispatch(ptB1,mask)[0]
    ptB2=[pto]
    #-calculando nós do banzo inferior
    #contador FOR do primeiro ao penultimo item de ptb1
    for i in range(len(ptB1)-1):
    #desenha linhas etrne o ponto atual e próximo ponto de ptB1
        linAux = Line(ptB1[i],ptB1[i+1])
    #pAux = ponto médio de linAux
        pAux3 = linAux.PointAt(.5)
    #calcula ponto em cAux1 mais proximo de pAux
    #pAux1 =[(coordenadas do pto),(parametro do pto),(distância pto-crv)]
        pAux1 = gh.CurveClosestPoint(pAux3,rs.coercecurve(cAux1))
        #idem - cAux2
        #pAux2 = idem
        pAux2 = gh.CurveClosestPoint(pAux3,BZ)
        #linha entre os pontos equivalentes das curvas cAux1 e cAux2
        linAux2 = Line(pAux1[0],pAux2[0])
        #a.append(linAux2)
        #cAux2 = cAux1 cortada (trim) no ponto pAux1
        cAux2= rs.TrimCurve(cAux1,(0,pAux1[1]),False)
    #razão entre o comprimento da curva cAux1 até o ponto pAux1
    #e o compriento total de cAux1
        prmt = rs.CurveLength(cAux2)/rs.CurveLength(cAux1)
        #ponto que interpola as alturas VH1 e VH2
        pAux =linAux2.PointAt(prmt*.75)
        #coloca o pto pAux na lista dos nós
        ptB2.append(pAux)
    #-desenhando diagonais e banzo inf
    #combina lista dos nós, inferiores e superiores
    #ordenados na sequancia das diagonais
    nosD = gh.Weave([0,1],ptB2,ptB1)
    if  DiagN%2==0:
        del nosD[0]
    else:
        ptB1.insert(0,SPCurv)

    #desenha diagonais
    lDiag = rs.AddPolyline(nosD)
    #desenha banzo Superior
    bzSup = rs.AddPolyline(ptB1)
    #desenha banzo inferior
    bzInf = rs.AddPolyline(ptB2)
    return  lDiag,bzSup,bzInf,nosD,ptB1,ptB2,
```

```
#-Listas de saida
#criando listas globais vazias para os outputs
Diagonais=[]
Banzo_Sup=[]
Banzo_Inf=[]
Eixos_do_Conector=[]
Apoios=[]
Cobertura=[]

#variavel de tolerância
tol=.0001

####--MAIN--####


#atribuindo valores as variáveis opcionais
if not HV2:
    HV2=HV1
if not HV3:
    HV3=HV1
if not DiagV2:
    DiagV2=DiagV1
if not DiagV3:
    DiagV3=9
if not No_Shed:
    No_Shed = 2
if not Plano:
    Plano = rs.WorldXYPlane()

#decompoe o plano de trabalho nos componentes Origem e os eixos xyz
pOr, eX, eY, eZ = Plano


#-Copiando Curva inicial
Curva = rs.CopyObjects(Curva,10*eY)
BzSup1 = rs.coercecurve(Curva)

#----Primeiro ponto do Eixo de Simetria----
#A variável Eixo_de_Simetria por entrada um float entre o e 1 ou
#uma reta que intercepta a Curva no ponto do eixo
#caso seja um float:
if type(Eixo_de_Simetria) == float:
    #ptX1 = coordenadas da curva no ponto de interceção com o eixo de simetria
    #vct1 = tangente da curva no ponto
    #prm1 = parametro da curva no ponto
    ptX1, vtc1, prm1 = gh.EvaluateLength(BzSup1,Eixo_de_Simetria,True)
    ptAux = gh.Move(ptX1,-eY)[0]
    #-desenhando linha do Exio de simetria
    Eixo_de_Simetria = rs.AddLine(ptX1,ptAux)
#caso seja uma linha
elif str(type(Eixo_de_Simetria))== "<type 'Guid'>":
    #ptX1 = coordenadas do ponto de interceção da Curva com o eixo de simetria
    #prm1 = parametro da curva no ponto
    #count = conta o número de interceções
    ptX1,prm1,count = gh.CurveXLine(BzSup1,rs.coerceline(Eixo_de_Simetria))
    ptAux = gh.Move(ptX1,-eY)[0]
#caso o vão da viga seja definido pela variavel VigaDist
if VigaDist:
    ptobase = rs.CurveStartPoint(Curva)
    ptoAux = rs.LineClosestPoint(Eixo_de_Simetria, ptobase)
    dist1 = rs.Distance(ptobase,ptoAux)
    escala = (VigaDist/2)/dist1
    escala=[escala,escala,escala]
    Curva,Eixo_de_Simetria = rs.ScaleObjects((Curva,Eixo_de_Simetria),ptobase,escala)
    #ptX1 = coordenadas do ponto de interceção da Curva com o eixo de simetria
    #prm1 = parametro da curva no ponto
    #count = conta o número de interceções
```

```
    BzSup1 = rs.coercecurve(Curva)
    ptX1,prm1,count = gh.CurveXLine(BzSup1,rs.coerceline(Eixo_de_Simetria))
    ptAux = gh.Move(ptX1,-eY)[0]
#desenha um seguimento no eixo de simetria da treliça
lAux=rs.coerceline(Eixo_de_Simetria)
cAux = rs.OffsetCurve(BzSup1,ptAux,HV1,eZ)
ptX2 = gh.CurveXLine(rs.coercecurve(cAux),lAux)[0]
#eixo do conector
EixoSimetria = rs.AddLine(ptX1,ptX2)
Eixos_do_Conector.append(EixoSimetria)
#----Linhas auxiliares dos Banzos Superiores----
#Ponto de corte (trim) do banzo sup. de V1
#ptY1 = [ponto,parametro,dist]
ptY1,paramY1,distY1 = gh.CurveClosestPoint(ptX2,BzSup1)
#corte (trim) do banzo sup v1
BS1 = rs.TrimCurve(BzSup1,[0,paramY1],False)
BS1 = rs.coercecurve(BS1)
#-espelhando (mirror) eixo sup de v1
#plano de espelhamento
mirplano = rs.PlaneFromNormal(ptX1, eX,eZ)
#BS2 = eixo banzo sup de v2
BS2 = gh.Mirror(rs.coercecurve(BS1),mirplano)[0]
#cortando a curva incicial no eixo de simetria
BS3 = rs.TrimCurve(BzSup1,[prm1,0],False)
BS3 = rs.coercecurve(BS3)
#bordas do conector
ptY2 = (rs.CurveEndPoint(BS2))
Nos_do_Conector = [ptX1,ptY1,ptX2,ptY2]
conect = rs.AddPolyline(Nos_do_Conector+Nos_do_Conector[:1])
Eixos_do_Conector.append(conect)
#eixos da viga v1
Diag1, BzS1, BzI1, nDiag1, nBs1, nBi1, apoio1= EixosVigas(BS1,DiagV1,HV1,ptX2,HV2,ptX2
V1 =[BzS1,Diag1,BzI1]
Apoios.append(apoio1)
#eixos da viga v2
Diag2, BzS2, BzI2, nDiag2, nBs2, nBi2, apoio2 = EixosVigas(BS2,DiagV2,HV1,ptX2,HV3,ptX
V2 =[BzS2,Diag2,BzI2]
Apoios.append(apoio2)
#eixos do shed (viga v3)
Diag3, BzS3, BzI3, nDiag3, nBs3, nBi3 = EixosShed(BS3,DiagV3,ptY2,)
V3 =[BzS3,Diag3,BzI3]
#saida Eixos
Eixos = V1 + V2 + V3 + Eixos_do_Conector
#---checando se a treliça e estaticamente estavel
#numero de barras das vigas + 5 barras do conetor
nBarras = len(rs.ExplodeCurves((BzI1,BzI2,BzI3,BzS1,BzS2,BzS3,Diag1,Diag2,Diag3)))+5
#número de nos na treliça - 2 nós pertencententes a 2 banzos
nNos = len(nBs1+nBs2+nBs3+nBi1+nBi2+nBi3)-2
#testando determinação estática
if (nNos*2)-3 == nBarras:
    print 'Treliça Estaticamente Determinada'
#Mostrando númerod e nós e de barras
print 'Número de Nos = ', nNos
print 'Número de Barras = ', nBarras
#--- desenhando cobertura
cob1 = rs.OffsetCurve(BzSup1,ptX2,-.2,eZ)
cob1 = rs.coercecurve(cob1)
Cobertura.append(cob1)
#abertura do shed
if No_Shed >= len(nBs2):
    No_Shed = len(nBs2)
cob2 = rs.OffsetCurve(BS2,ptX2,-.2,eZ)
ptShed = rs.CurveClosestPoint(cob2,nBs2[-No_Shed])
cob2 = rs.TrimCurve(cob2,[0,ptShed],False)
Cobertura.append(cob2)
```