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

#ghenv.Component.Params.Output[2].Hidden = True

#função de desenho das cordas
def estendeCorda(linha,ponto,indice):
    pto1 = linha.PointAt(indice)
    return Line(pto1,ponto)
# coloca a força externa na posição (nó) e direção de carregamento
def Linha_force_extena(no, peso, conv):
    lincarg = rs.AddLine(no,gh.Move(no,peso*conv*eY)[0])
    rs.ReverseCurve(lincarg)
    return lincarg
#coloca todas as linahs de carga da viga no respectivo nó e direção
def Linhas_de_Cargas(viga,cargas, nome):
    Cargasvigas=[]
    ptos1 = []
    nos = rs.PolylineVertices(viga[0])
    for i in range(len(cargas)):
        lincarg = Linha_force_extena(nos[i],cargas[i],Escala)
        linCoe = rs.coerceline(lincarg)
        pt1 = linCoe.PointAt(.5)
        pt2 = linCoe.PointAt(0)
        texto1 = str('%.2f' % cargas[i])
        texto2 = 'P'+str( i)+'_'+nome
        Cargasvigas.append(lincarg)
        ptos1 += [pt1,texto1,corcargas]
        ptos1 += [pt2,texto2,corcargas]
    return Cargasvigas, ptos1
#separa uma viga en seus elementos - barras e nós
def Elementos_vigas(viga):
    #banzo superior v3 -- barras
    bs = rs.ExplodeCurves(viga[0])
    #banzo superior v3 -- nos
    nbs = rs.PolylineVertices(viga[0])
    #diagonais v3 -- barras
    dg = rs.ExplodeCurves(viga[1])
    #diagonais v3 -- nos3
    ndg = rs.PolylineVertices(viga[1])
    #banzo inferior v3 -- barras
    bi = rs.ExplodeCurves(viga[2])
    #banzo inferior v3 -- nos
    nbi = rs.PolylineVertices(viga[2])
    return bs, nbs, dg, ndg, bi, nbi
#desenha um circulo no nó com a mesma direção do
#circulo - cirDir - que descreve a direção de seleção das cargas
def SentidoNo(cirDir,no,Plano,cirRaio):
    pOr, eX, eY, eZ = Plano
    planoAux = rs.PlaneFromNormal(no,eZ)
    Circulo = rs.AddCircle(planoAux,cirRaio)
    dir1 =  rs.ClosedCurveOrientation(cirDir,eZ)
    dir2 =  rs.ClosedCurveOrientation(Circulo,eZ)
    if  not dir1 == dir2:
        rs.ReverseCurve(Circulo)
    return Circulo
#retorna 1 para tração e -1 para compressão
def TraComp(no, forcePF, forceFG, nome):
    pontoFG = forceFG.PointAt(.5)
    movevec = rs.AddLine(rs.CurveEndPoint(forcePF),no)
    movevec = rs.coerceline(movevec)
```

```
    testeLin = gh.Move(rs.coerceline(forcePF),movevec)[0]
    dist1 = rs.Distance(testeLin.PointAt(0),pontoFG)
    dist2 = gh.Length(testeLin)
    dist2 += ( gh.Length(forceFG)/2 )
    #testando tração
    if abs(dist1 - dist2) <= Tol:
        #coloca a nomenclattura do elemento na lista de objetos tracionados
        Ltrac.append(nome)
        #retorna 1 para tração
        return 1

    #se compressão
    else:
       #coloca a nomenclattura do elemento na lista de objetos comprimidos
       Lcomp.append(nome)
       #retorna -1 para compressão
       return -1

#testa se o elemento suporta as cargas de tração e compressão aplicadas
def teste_elemento(nome,carga,fglin):
    #banzo ou diagonal
    if nome[0] == 'b':
        bd=0
    else:
        bd= 1

    # se tracionado
    if carga >= 0:
        if abs(carga) < F_adm[bd]:
            #print nome + 'compressão ok'
            return corpass

        else:
            print nome + '  Falhou na tração'
            return corfail



    #se comprimido
    elif carga < 0:
        len1 = rs.CurveLength(fglin)
        rGira = (Areas[1][bd]/Areas[0][bd])**0.5
        esbelt = len1/rGira
        if esbelt >= esb_min:
            T_comp = (3.14**2*Mod_E)/(esbelt**2 * coe_Comp)
            #print nome + '  Formula de Euler'
        else:
            T_comp = (17000 - 0.485 * (esbelt**2))*6894.745
            #print nome + '  Formula do A.I.S.C.'
        if (abs(carga)/Areas[0][bd] < T_comp) and (abs(carga) < F_adm[bd]):
            return corpass
        else:
            print nome + '  Falhou na compressão'
            return corfail



#reotna os elementos de uma lista que se econtram no no em estudo
def elemntos_node(no,eList,nome_lista,nome_viga):
    Nomes = []
    elementos = []
    for K in range(len(eList)) :
        p1 = rs.CurveEndPoint(eList[K])
        p2 = rs.CurveStartPoint(eList[K])
        if rs.PointCompare(no,p1,Tol) or  rs.PointCompare(no,p2,Tol):
            Nomes.append(nome_lista + str(K) + "_" + nome_viga)
            elementos.append(eList[K])
```

```
    return elementos,Nomes
# cálculo grafostático dos elementos do shed
def Grafo_Shed( viga, F_ext, Plano, dicPF, cirDir, nome_viga):
    bs3, nbs3, dg3, ndg3, bi3, nbi3 = Elementos_vigas(viga)
    ptos =[]
    dicUp2 ={}
    count_ext =0
    for i in range(len(ndg3)-1):
        no = ndg3[i]
        nomesNo = []
        elementos =[]
        #se banzo superior
        if (i+len(dg3))%2==0:
           #forças externas
           if i == 0:
               #reação RB
               nomesNo.append('RB')
               elementos.append(F_ext[count_ext])
               count_ext += 1
           else:
               #cargas
               f_ext = F_ext[count_ext]
               p1 = f_ext.PointAt(0)
               p2 = f_ext.PointAt(1)
               f_ext = rs.AddLine(p1,p2)
               nomesNo.append('P'+ str(count_ext)+ '_' + nome_viga)
               elementos.append(f_ext)
               count_ext += 1
           #banzo superior
           bs_no, nome_bs = elemntos_node(no,bs3,'bs','v3')
           elementos += bs_no
           nomesNo += nome_bs
        else:
            #forças externas
            if i == 0:
                #reação RA
                nomesNo.append('RA')
                elementos.append(F_ext[count_ext])
                count_ext += 1
            #Banzo inferior
            bi_no , nome_bi = elemntos_node(no,bi3,'bi','v3')
            elementos += bi_no
            nomesNo += nome_bi
        #diagonais
        dg_no, nome_dg = elemntos_node(no,dg3,'dg','v3')
        elementos += dg_no
        nomesNo += nome_dg
        nomesNo, elementos, countPF, countCalcular = OrdenaLinhasDeAcao(no,
        cirDir,elementos,nomesNo, dicPF, Plano)
        if countCalcular == 2:
            dicUp, ptos1 = Cremona2(no, nomesNo, elementos, countPF, dicPF)
        if countCalcular == 1:
            dicUp, ptos1 = Cremona1(no, nomesNo, elementos, countPF, dicPF)
        ptos += ptos1
        dicPF.update(dicUp)
        dicUp2.update(dicUp)
    return dicUp2, ptos
#retorna linas de ação e parametros
def OrdenaLinhasDeAcao(no,cirDir,Linhas,nomes, dicPF, Plano):
    #serando contadores
    countPF = 0
    countCalcular = 0
    #lista do comprimento das Linhas
    compList = [rs.CurveLength(x) for x in Linhas]
    cRaio = min(compList)/2
    #circulo com direção de desenho na direeção de escolha dos vetores
    Circulo = SentidoNo(cirDir,no,Plano,cRaio)
```

```
    #lista de paramentos paara ordenar
    paramAux=[]
    #para cada linha da entrada 3
    for I in range(len(Linhas)):
        iCoe = rs.coerceline(Linhas[I])
        #testa se a força atuante no elemento já e conehcida
        if nomes[I] in dicPF:
            #contador de forças conecidas no no
            countPF += 1
        else:
            #contador das forças desconhecidas que atuam no no
            countCalcular += 1
        cirCoe = rs.coercecurve(Circulo)
 #lista - inter = [ponto de interseção,parametro no circulo,parametro na linha)]
        inter = gh.CurveXCurve(cirCoe,iCoe)
        # coloca parametro de interceção entre Linha e circulo na lista
        # lista - paramAux - tem a mesma ordem das curvas em Linhas e os nomes
        paramAux.append(inter[1])

    #ordenando linhas e nomes de acordo com os parametros -- paramAux
    Linhas = [x for (y,x) in sorted(zip(paramAux,Linhas), key=lambda pair: pair[0])]
    nomes  = [x for (y,x) in sorted(zip(paramAux, nomes), key=lambda pair: pair[0])]
    if (0 < countPF) :
        while nomes[0] in dicPF:
            nomes = nomes[1:] + nomes[:1]
            Linhas = Linhas[1:] + Linhas[:1]
        while nomes[0] not in dicPF:
            nomes = nomes[1:] + nomes[:1]
            Linhas = Linhas[1:] + Linhas[:1]
    return nomes, Linhas, countPF, countCalcular
#resolve nó com 2 forças desconhecidas
def Cremona2(no, nomes, Linhas, countPF, dicPF):
    ptos1 = []
    dicUp ={}
    for i in range(countPF):
        if i == 0:
            Spt = dicPF[nomes[i]].PointAt(0)
            Ept = dicPF[nomes[i]].PointAt(1)
        else:
            if  i == 1:
                cond1 = rs.PointCompare( dicPF[nomes[i-1]].PointAt(0),
                dicPF[nomes[i]].PointAt(1), Tol)
                cond2 =  rs.PointCompare( dicPF[nomes[i-1]].PointAt(0),
                dicPF[nomes[i]].PointAt(0), Tol)
                if cond1 or cond2:
                    pAux3 = Spt
                    Spt = Ept
                    Ept = pAux3
            if rs.PointCompare( Ept , dicPF[nomes[i]].PointAt(1), Tol):
                ptAux1 = dicPF[nomes[i]].PointAt(1)
                ptAux2 = dicPF[nomes[i]].PointAt(0)
            else:
                ptAux1 = dicPF[nomes[i]].PointAt(0)
                ptAux2 = dicPF[nomes[i]].PointAt(1)
            Ept += (ptAux2 - ptAux1)
    vAux1 = Line( rs.CurveStartPoint(Linhas[-2]), Ept)
    vAux2 = Line( rs.CurveStartPoint(Linhas[-1]), Spt)
    F1 = gh.Move( rs.coerceline(Linhas[-2]), vAux1 )[0]
    F2 = gh.Move( rs.coerceline(Linhas[-1]), vAux2 )[0]
    inter = gh.LineXLine(F1,F2)[2]
    F1 = rs.AddLine(Ept, inter)
    F2= rs.AddLine(inter, Spt)
    dicUp[ nomes[-2] ] = rs.coerceline(F1)
    dicUp[ nomes[-1] ] = rs.coerceline(F2)
    #-cargas e nomenclatura
    #teste de tração e compressão
    sin1 = TraComp(no, F1 , rs.coerceline(Linhas[-2]), nomes[-2])
```

```
    sin2 = TraComp(no, F2 , rs.coerceline(Linhas[-1]), nomes[-1])
    #valores das cargas
    carga1 = rs.CurveLength(F1)*sin1/Escala
    carga2 = rs.CurveLength(F2)*sin2/Escala
    #teste de tensão admissivel
    cor1 = teste_elemento(nomes[-2],carga1,Linhas[-2])
    cor2 = teste_elemento(nomes[-1],carga2,Linhas[-1])
    #nomenclatura do FG
    txt1 = nomes[-2] + ' = ' + str('%.2f' %carga1)
    txt2 = nomes[-1] + ' = ' + str('%.2f' %carga2)
    pt1 = rs.coerceline(Linhas[-2]).PointAt(.5)
    pt2 = rs.coerceline(Linhas[-1]).PointAt(.5)
    ptos1 += [pt1, txt1,cor1]
    ptos1 += [pt2, txt2,cor2]
    #nimenclatura do PF
    pt1 = rs.coerceline(F1).PointAt(.5)
    pt2 = rs.coerceline(F2).PointAt(.5)
    txt1 = nomes[-2]
    txt2 = nomes[-1]
    ptos1 += [pt1, txt1,cor1]
    ptos1 += [pt2, txt2,cor2]

    return dicUp, ptos1
#resolve nó com 1 forçal desconhecida
def Cremona1(no, nomes, Linhas, countPF, dicPF):
    ptos1 = []
    dicUp ={}
    for i in range(countPF):
        if i == 0:
            Spt = dicPF[nomes[i]].PointAt(0)
            Ept = dicPF[nomes[i]].PointAt(1)
        else:
            if  i == 1:
                cond1 = rs.PointCompare( dicPF[nomes[i-1]].PointAt(0),
                dicPF[nomes[i]].PointAt(1), Tol)
                cond2 =  rs.PointCompare( dicPF[nomes[i-1]].PointAt(0),
                dicPF[nomes[i]].PointAt(0), Tol)
                if cond1 or cond2:
                    pAux3 = Spt
                    Spt = Ept
                    Ept = pAux3

            if rs.PointCompare( Ept , dicPF[nomes[i]].PointAt(1), Tol):
                ptAux1 = dicPF[nomes[i]].PointAt(1)
                ptAux2 = dicPF[nomes[i]].PointAt(0)
            else:
                ptAux1 = dicPF[nomes[i]].PointAt(0)
                ptAux2 = dicPF[nomes[i]].PointAt(1)
            Ept += (ptAux2 - ptAux1)
    F1 = rs.AddLine(Ept,Spt)
    #verificar o paralelismo entre F1 no PF e FG
    vec1 = rs.VectorCreate(rs.CurveEndPoint(Linhas[-1]),
    rs.CurveStartPoint(Linhas[-1]))
    vec2 = rs.VectorCreate(Spt,Ept)
    if rs.IsVectorParallelTo (vec2, vec1):
        print '_____Paralelismo_____'
    #colovcando F1 no dicionario do PF
    dicUp[ nomes[-1] ] = rs.coerceline(F1)
    #-cargas e nomenclatura
    #teste de tração e compressão
    sin1 = TraComp(no, F1 , rs.coerceline(Linhas[-1]), nomes[-1])
    #valores das cargas
    carga1 = rs.CurveLength(F1)*sin1/Escala
    #teste de tensão admissivel
    cor1 = teste_elemento(nomes[-1],carga1,Linhas[-1])
    #nomenclatura do FG
    txt1 = nomes[-1] + ' = ' + str('%.2f' %carga1)
```

```
    pt1 = rs.coerceline(Linhas[-1]).PointAt(.5)
    ptos1 += [pt1, txt1,cor1]
    #nomenclatura do PF
    pt1 = rs.coerceline(F1).PointAt(.5)
    txt1 = nomes[-1]
    ptos1 += [pt1, txt1,cor1]
    return dicUp, ptos1
# PF e funicular
def PF_funic(pto_inicial,polo,carreg, nomes_cargas):
    raios=[]
    ptos=[]
    dicUp={}
    PF =[]
    resultante =[]
    funicular =[]
    ## -- Desenhando o poligono de forças -- ##
    pAux = rs.coerce3dpoint(pto_inicial)
    # polo do PF do Shed
    polo = rs.coerce3dpoint(polo)
    #primeiro raio polar
    raios.append(Line(polo,pAux))
    ptos += [polo, 'polo', cornode]
    #desenhando carregamentos no PF e os raios polares
    for i in range(len(carreg)):
        v = carreg[i]
        #carregamento no FG
        vAux1 = rs.coerceline(v)
        #vetor auxiliar para mover o carregamento para a posição de soma
        vAux2 = pAux - v.PointAt(0)
        # carregamento no PF
        vAux3 = gh.Move(vAux1,vAux2)[0]
        #Nomenclatura - texto da reação Pi
        nome = nomes_cargas[i]
        #Nomenclatua - posição do texto
        txtPt = vAux3.PointAt(.5)
        # Nomenclatura do PF
        ptos += [txtPt, nome, corcargas]
        # colocando carregamento na lista do PF
        PF.append(vAux3)
        # olocando carregamento no dicionario do PF
        dicUp[nome]=vAux3
        # ponto da posição de soma para o proximo carregamento
        pAux = vAux3.PointAt(1)
        #desenhando raio polar
        r = Line(polo,pAux)
        #colocando raio polar na lista de raios
        raios.append(r)
    #desenhando a resultante no PF
    #ponto final da resultante R1
    pto_R1 = pAux
    #resultante r1 no PF
    R1PF =Line(rs.coerce3dpoint(pto_inicial),pto_R1)
    #colocando R1 na lista de resultantes
    resultante.append(R1PF)
    #R1 no dicionario do PF
    dicUp['R1']= R1PF
    #Desenhando o funicular
    for i in range(len(raios)):
        r = rs.coerceline(raios[i])
        #caso da primeira corda
        if i == 0:
            pAux = (rs.coerceline(carreg[0]).PointAt(0))
            vAux1 = Line(r.PointAt(0),pAux)
            corda = gh.Move(r,vAux1)[0]
            corda = rs.coerceline(corda)
        #cordas intermediarias
        elif i < len(raios)-1:
```

```
            vAux1 = Line(r.PointAt(1),pAux)
            crdAux = gh.Move(r,vAux1)[0]
            crdAux = rs.coerceline(crdAux)
            pAux2 = pAux
            pAux = gh.LineXLine(crdAux,carreg[i])[-1]
            corda = Line(pAux2,pAux)
        #caso da ultima corda
        else:
            vAux1 = Line(r.PointAt(1),pAux)
            corda = gh.Move(r,vAux1)[0]
            corda = rs.coerceline(corda)
        #adicionando corda na lista do funicular
        funicular.append(corda)
    #resesenhando as cordas extremas
    return dicUp, raios, funicular, resultante, ptos
def Grafo_Viga(viga, F_ext, Plano,dicPF,cirDir,Bconect,reac,nome_viga):
    bs, nbs, dg, ndg, bi, nbi = Elementos_vigas(viga)
    Bconect = rs.ExplodeCurves(Bconect)
    ptos =[]
    dicUp2 ={}
    for i in range(len(ndg)-1):
        no = ndg[i]
        nomesNo = []
        elementos =[]
        j = i//2
        #se banzo superior
        if i%2 == 1:
            #Força extena
            f_ext = F_ext[j]
            p1 = f_ext.PointAt(0)
            p2 = f_ext.PointAt(1)
            f_ext = rs.AddLine(p1,p2)
            nomesNo.append('P'+ str(j)+ '_' + nome_viga)
            elementos.append(f_ext)
            #banzo superior
            nomesNo.append('bs'+ str(j)+ '_' + nome_viga)
            elementos.append(bs[j])
            if not j==0:
                nomesNo.append('bs'+ str(j-1)+ '_' + nome_viga)
                elementos.append(bs[j-1])
            #diagonais
            nomesNo.append('dg'+ str(i-1)+ '_' + nome_viga)
            elementos.append(dg[i-1])
            nomesNo.append('dg'+ str(i)+ '_' + nome_viga)
            elementos.append(dg[i])
        #se banzo inferior
        elif i%2 == 0:
            if i == 0:
                #reação
                p1 = reac.PointAt(0)
                p2 = reac.PointAt(1)
                reac = rs.AddLine(p1,p2)
                nome_reac = 'R'+ '_' + nome_viga
                nomesNo.append(nome_reac)
                elementos.append(reac)
            if not j == 0:
                #diagonal anterior
                nomesNo.append('dg'+ str(i-1)+ '_' + nome_viga)
                elementos.append(dg[i-1])
                #banzo inferior anterior
                nomesNo.append('bi'+ str(j-1)+ '_' + nome_viga)
                elementos.append(bi[j-1])
            #diagonal posterior
            nomesNo.append('dg'+ str(i)+ '_' + nome_viga)
            elementos.append(dg[i])
            #banzo inferior posterior
            nomesNo.append('bi'+ str(j)+ '_' + nome_viga)
```

```
            elementos.append(bi[j])
        #ordenar linhas de ação em torno do nó
        nomesNo, elementos, countPF, countCalcular = OrdenaLinhasDeAcao(no,
        cirDir,elementos,nomesNo, dicPF, Plano)
        if countCalcular == 2:
            dicUp, ptos1 = Cremona2(no, nomesNo, elementos, countPF, dicPF)
        if countCalcular == 1:
            dicUp, ptos1 = Cremona1(no, nomesNo, elementos, countPF, dicPF)
        ptos += ptos1
        dicPF.update(dicUp)
        dicUp2.update(dicUp)
    ####------conector------####
    # ùltimo no do banzo superior
    no = nbs[-1]
    #elementos do conector
    elementos, nomesNo = elemntos_node(no,Bconect,'bc','c')
    #ùltimo elemento do banzo superior
    elementos.append(bs[-1])
    nomesNo.append('bs'+str(len(bs)-1)+'_'+nome_viga)
    #Força externa
    f_ext = F_ext[len(F_ext)-1]
    p1 = f_ext.PointAt(0)
    p2 = f_ext.PointAt(1)
    f_ext = rs.AddLine(p1,p2)
    nomesNo.append('P'+ str(len(F_ext)-1)+ '_' + nome_viga)
    elementos.append(f_ext)
    #diagonal
    if len(dg)%2 == 1:
        elementos.append(dg[-1])
        nomesNo.append('dg'+str(len(dg)-1)+'_'+nome_viga)
    nomesNo, elementos, countPF, countCalcular = OrdenaLinhasDeAcao(no,cirDir,
    elementos,nomesNo, dicPF, Plano)
    dicUp, ptos1 = Cremona2(no, nomesNo, elementos, countPF, dicPF)
    ptos += ptos1
    dicPF.update(dicUp)
    dicUp2.update(dicUp)
    return dicUp2, ptos

###---MAIN---###
#listas vazias para as saidas

txt_pontos = [] # listas com pontos de inseção, texto e cores para vizualiação
Linhas_de_Carga=[] # o carregamento na vifa
raios_1=[] # raios polares do Shed
carreg_1 = [] # carregamento e reações do Shed
FG1=[] # forma greométria do shed
PF1 = [] # Polígono de forças  do Shed
funicular_1 =[] # funicilar do Shed
resultante_1 =[] #resultante do Shed
dic_1 ={} #dicionário do PF do Shed
raios_2=[] # raios polares das vigas v1 e v2
carreg_2 = [] # carregamento das vigas v1 e v2
carreg_v1 = []
carreg_v2 = []
FG2=[] # forma greométria das vigas v1 e v2
PF2 = [] # Polígono de forças  das vigas v1 e v2
funicular_2 =[] # funicilar das vigas v1 e v2
resultante_2 =[] #resultante das vigas v1 e v2
dic_2 ={} #dicionário do PF das vigas v1 e v2
Ltrac = [] #lista de elementos tracionados
Lcomp = [] #lista de elementos comprimidos

#variáveis globais com as cores de vizualização dos textos

cornode = '0,0,255' #cor para pontos e nos
corcargas = '0,255,255'# cor para o textos das cargas
corpass = '0,255,100' #cor para o textos das barras que resitem à tensão admissivel
```

```
corfail = '255,0,255' #cor para o textos das barras que NÂO resitem à tensão admissive

#tolerancia
Tol = 0.0001
#caso nãp seja definida uma escala
if not Escala:
    Escala = .0001 #assume-se a escala de 1/10000
Escala
#caso não seja fornecido um Plano de trabalho
if not Plano:
    Plano = rs.WorldXYPlane()# assume-se o palno xy
#decompoe o plano de trabalho nos componentes Origem e os eixos xyz
pOr, eX, eY, eZ = Plano
#separando cargas por vigas
P_v1, P_v2, P_v3 = Cargas

# se a tensão admissivel não é definida
if not T_adm:
    T_adm = 150*10**6 #assume-se 150 MPa
if not coe_Trac: # se o coeficiente de segurança para tração não é definido
    coe_Trac = 2 # assume-se 2
if not coe_Comp: # se o coeficiente de segurança para compressão não é definido
    coe_Comp = 2 # assume-se 2
if not Mod_E: # se o modulo de elasticidade não foi informado
    Mod_E = 200*10**9 #assume-se 200 GPa
if not esb_min: # se a esbeltez minima não foi informada
    esb_min = 105 # assume-se 105

# lista com as forças limites para banzos e diagonais
F_adm = [(x*T_adm)/coe_Trac for x in Areas[0]]
#Separando os eixos em listas
#conector
Conector = Eixos[-2:]
#vigas
v1 = Eixos[:3]
v2 = Eixos[3:6]
v3 =  Eixos[6:9]
#apoios
Apoios = [rs.CurveStartPoint(v1[2]),rs.CurveStartPoint(v2[2])]
#Eixo de Simetria do conector
EixoSimetria = Conector[0]
Bconect = Conector[1]
EixoSimetria =  rs.coerceline(EixoSimetria)
ptX = gh.EndPoints(EixoSimetria)
# vareável booleana para iniciar análise
if Iniciar_Analise == None:
    Iniciar_Analise = True # calcula tensões para True ou none
if Iniciar_Analise: #não calcula para False
    #Cargas na viga v1
    C_v1, ptC_v1 = Linhas_de_Cargas(v1,P_v1,'V1')
    Linhas_de_Carga += C_v1
    txt_pontos += ptC_v1
    #Cargas na viga v2
    C_v2, ptC_v2 = Linhas_de_Cargas(v2,P_v2, 'V2')
    Linhas_de_Carga += C_v2
    txt_pontos += ptC_v2
    #Cargas na viga v3
    C_v3, ptC_v3 = Linhas_de_Cargas(v3,P_v3, 'V3')
    Linhas_de_Carga += C_v3
    txt_pontos += ptC_v3
    #determinando sendido de seleção dos elementos
    cirDir = rs.AddCircle3Pt(Apoios[0],ptX[0],Apoios[1])
    #### --- Cálculo do Shed --- ####
    print '####------------------Shed----------------------####'
    bi3 = v3[2]
    pto = rs.CurveStartPoint(bi3)
    if not pto_base_FG1:
```

```
    pto_base_FG1 = pto
pto_base_FG1 = rs.coerce3dpoint(pto_base_FG1)
vAux1 = Line(pto,pto_base_FG1)
for i in range(len(v3)):
    v3[i] = gh.Move(rs.coercegeometry(v3[i]),vAux1)[0]
for i in range(len(C_v3)):
   C_aux = gh.Move(rs.coerceline(C_v3[i]),vAux1)[0]
   carreg_1.append(C_aux)
FG1 = v3 + carreg_1
carreg_1.pop(0)
nomes_cargas =[]
for i in range(len(carreg_1)):
    nome = "P" + str(i+1) + '_v3'
    nomes_cargas.append(nome)
dic_1, raios_1, funicular_1, resultante_1, ptos = PF_funic(pto_inicial_1,
polo_1,carreg_1, nomes_cargas)
for i in dic_1.keys():
    PF1.append(dic_1[i])
txt_pontos += ptos
# - desenhando a resultante no FG
# ponto de intecessão entre a primeira e ultuma corda
pAux = gh.LineXLine(funicular_1[0],funicular_1[-1])[-1]
#resesenhando as cordas extremas
funicular_1[0] = estendeCorda(funicular_1[0],pAux,0)
funicular_1[-1] = estendeCorda(funicular_1[-1],pAux,1)
#vetor auxiliar para mover R1PF para o FG
vAux1 = Line(dic_1['R1'].PointAt(.5),pAux)
#R1FG R1 no FG
R1FG = gh.Move(dic_1['R1'],vAux1)[0]
#coloca R1FG na lista de resultates_1
resultante_1.append(R1FG)
# separando a viga v3 em seus elementos -
#bs3 = lista de barras do banzo supeiro
#nbs3 = lista de nós do bazo superior
#dg3 e ndg3 para diagoains e bi3 e nbi3 para banzo inferior
bs3, nbs3, dg3, ndg3, bi3, nbi3 = Elementos_vigas(v3)
raAux = bi3[0]
rbAux = bs3[0]
pAux1 = nbi3[0]
pAux2 = nbs3[0]
#caso o numero de diagonais seja par
if len(dg3)%2 == 0:
    raAux = rs.coerceline(raAux)
    #ponto de interseção das resultantes
    pt_inter = gh.LineXLine(R1FG,raAux)[-1]
    resultante_1.append(pt_inter)
    #resultante entre banzo e diagonal
    rbAux = rs.AddLine(pAux2,pt_inter)
    rbAux = rs.coerceline(rbAux)
    #vetor auxiliar para deslocamento gh.Move
    pto_R1 = dic_1['R1'].PointAt(1)
    vAux1 = rs.AddLine(pAux1,pto_R1)
    vAux1 = rs.coerceline(vAux1)
    #vetor auxiliar para deslocamento gh.Move
    vAux2 = rs.AddLine(pAux2,pto_inicial_1)
    vAux2 = rs.coerceline(vAux2)
    #movendo linhas do FG para o PF
    raAux = gh.Move(raAux,vAux1)[0]
    rbAux = gh.Move(rbAux,vAux2)[0]
    #ponto de interseção das reações no PF
    pto_PF = gh.LineXLine(raAux,rbAux)[-1]
    # - desenhando reação RA no PF
    RA = rs.AddLine(pto_R1,pto_PF)

    PF1.append(rs.coerceline(RA))
    #dicionario do PF
    dic_1['RA'] = rs.coerceline(RA)
```

```
        #texto - ponto de inserção
        p_txt = rs.coerceline(RA).PointAt(.5)
        #texto
        txt_pontos += [p_txt,'RA',corcargas]
        # - desenhando  reaçãao RB no PF
        RB = rs.AddLine(pto_PF, pto_inicial_1)
        PF1.append(rs.coerceline(RB))
        #dicionario do PF
        dic_1['RB'] = rs.coerceline(RB)
        #texto
        p_txt = rs.coerceline(RB).PointAt(.5)
        txt_pontos += [p_txt,'RB',corcargas]
        # - desenhando reação RA no FG
        vAux1 = rs.AddLine(rs.CurveEndPoint(RA),nbi3[0])
        RAFG = gh.Move(rs.coerceline(RA),rs.coerceline(vAux1))[0]
        p_txt = RAFG.PointAt(.5)
        carga1 = rs.CurveLength(RA)*1/Escala
        texto = 'RA = ' + str('%.2f' % carga1)
        txt_pontos += [p_txt,texto,corcargas]
        FG1.append(rs.coerceline(RAFG))
        # - desenhando reação RB no FG
        vAux2 = rs.AddLine(rs.CurveStartPoint(RB),nbs3[0])
        RBFG = gh.Move(rs.coerceline(RB),rs.coerceline(vAux2))[0]
        p_txt = RBFG.PointAt(.5)
        carga2 = rs.CurveLength(RB)-1/Escala
        texto = 'RB = ' + str('%.2f' % carga2)
        txt_pontos += [p_txt,texto,corcargas]
        FG1.append(rs.coerceline(RBFG))
        # - tensão em bi3[0]
        bi0PF = rs.CopyObjects(RA)
        rs.ReverseCurve(bi0PF)
        bi0PF = rs.coerceline(bi0PF)
        #dicionario
        dic_1['bi0_v3'] = bi0PF
        #textos
        carga1=-1*carga1
        cor1 = teste_elemento(texto,carga1,bi3[0])
        p_txt = rs.coerceline(bi0PF).PointAt(.75)
        txt_pontos += [p_txt,'bi0_v3',cor1]
        p_txt = rs.coerceline(bi3[0]).PointAt(.5)
        texto = 'bi0_v3 = ' + str('%.2f' % (carga1))
        txt_pontos += [p_txt,texto,cor1]
        #carregamento no nó[0]
        carreg_1.insert(0,RB)
        #coloca nomenclatura do elemento na lista de elementos comprimidos
        Lcomp.append('bi0_v3')
    # caso o numero de diagonais seja impar
    else:
        pto_R1 = dic_1['R1'].PointAt(1)
        rbAux = rs.coerceline(rbAux)
        pt_inter = gh.LineXLine(R1FG,rbAux)[-1]
        resultante_1.append(pt_inter)
        #resultante entre banzo e diagonal
        raAux = rs.AddLine(pAux1,pt_inter)
        raAux = rs.coerceline(raAux)
        #vetor auxiliar para deslocamento gh.Move
        vAux1 = rs.AddLine(pAux1,pto_R1)
        vAux1 = rs.coerceline(vAux1)
        #vetor auxiliar para deslocamento gh.Move
        vAux2 = rs.AddLine(pAux2,pto_inicial_1)
        vAux2 = rs.coerceline(vAux2)
        #movendo linhas do FG para o PF
        raAux = gh.Move(raAux,vAux1)[0]
        rbAux = gh.Move(rbAux,vAux2)[0]
        #ponto de interseção das reações no PF
        pto_PF = gh.LineXLine(raAux,rbAux)[-1]
        # - desenhando reação RA no PF
```

```
        RA = rs.AddLine(pto_R1,pto_PF)
        PF1.append(rs.coerceline(RA))
        #dicionario do PF
        dic_1['RA'] = rs.coerceline(RA)
        #texto - ponto de inserção
        p_txt = rs.coerceline(RA).PointAt(.5)
        #texto
        txt_pontos += [p_txt,'RA',corcargas]
        # - desenhando  reaçãao RB no PF
        RB = rs.AddLine(pto_PF, pto_inicial_1)
        PF1.append(rs.coerceline(RB))
        #dicionario do PF
        dic_1['RB'] = rs.coerceline(RB)
        #texto
        p_txt = rs.coerceline(RB).PointAt(.5)
        txt_pontos += [p_txt,'RB',corcargas]
        # - desenhando reação RA no FG
        vAux1 = rs.AddLine(rs.CurveEndPoint(RA),nbi3[0])
        RAFG = gh.Move(rs.coerceline(RA),rs.coerceline(vAux1))[0]
        p_txt = RAFG.PointAt(.5)
        carga1 = rs.CurveLength(RA)*1/Escala
        texto = 'RA = ' + str('%.2f' % carga1)
        txt_pontos += [p_txt,texto,corcargas]
        FG1.append(rs.coerceline(RAFG))
        # - desenhando reação RB no FG
        vAux2 = rs.AddLine(rs.CurveStartPoint(RB),nbs3[0])
        RBFG = gh.Move(rs.coerceline(RB),rs.coerceline(vAux2))[0]
        p_txt = RBFG.PointAt(.5)
        carga2 = rs.CurveLength(RB)*-1/Escala
        texto = 'RB = ' + str('%.2f' % carga2)
        txt_pontos += [p_txt,texto,corcargas]
        FG1.append(rs.coerceline(RBFG))
        # - tensão em bs3[0]
        bs0PF = rs.CopyObjects(RB)
        rs.ReverseCurve(bs0PF)
        bs0PF = rs.coerceline(bs0PF)
        #dicionario
        dic_1['bs0_v3'] = bs0PF
        #textos
        carga2 = -1*carga2
        cor1 = teste_elemento('bs0_v3',carga2,bs3[0])
        p_txt = rs.coerceline(bs0PF).PointAt(.75)
        txt_pontos += [p_txt,'bs0_v3',cor1]
        p_txt = rs.coerceline(bs3[0]).PointAt(.5)
        texto = 'bs0_v3 = ' + str('%.2f' % (carga2))
        txt_pontos += [p_txt,texto,cor1]
        #carregamento no nó[0]
        carreg_1.insert(0,RA)
        #coloca nomenclatura do elemento na lista de elementos tracionados
        Ltrac.append('bi0_v3')
    #calcula as forças nas barras da treliça do shed
    dicUp, ptos = Grafo_Shed( v3, carreg_1, Plano, dic_1, cirDir, 'v3')
    #atualizando nomenclatura
    txt_pontos += ptos
    #atualizando dicionário
    dic_1.update(dicUp)
    #Atualizando elementos na saida PF1
    for i in dicUp.keys():
        PF1.append(dicUp[i])
#####---------- vigas v1 e v2 ----------####
    # força F0 -- P0_v3 + RB
    fAux = gh.FlipCurve( dic_1['RB'])[0]
    p1 = fAux.PointAt(1)
    p2 = rs.CurveStartPoint(C_v3[0])
    vAux = Line(p1,p2)
    fAux =  gh.Move(fAux,vAux)[0]
    p1 = fAux.PointAt(0)
```

```
p2= rs.CurveEndPoint(C_v3[0])
F0 = rs.AddLine(p1,p2)
F0 = rs.CopyObjects(F0,(p2-p1))
# força C_v2[-1] -- C_v2[-1] + RA
fAux = gh.FlipCurve( dic_1['RA'])[0]
p1 = fAux.PointAt(1)
p2 = rs.CurveStartPoint(C_v2[-1])
bi1 = v1[2]
p2 = rs.CurveStartPoint(C_v2[-1])
vAux = Line(p1,p2)
fAux =  gh.Move(fAux,vAux)[0]
p1 = fAux.PointAt(0)
p2= rs.CurveEndPoint(C_v2[-1])
C_v2[-1] = rs.AddLine(p1,p2)
#Movendo FG2
pto = rs.CurveStartPoint(bi1)
if not pto_base_FG2:
    pto_base_FG2 = pto
ptoMove = rs.coerce3dpoint(pto_base_FG2)
vAux1 = Line(pto,ptoMove)
for i in range(len(v1)):
    v1[i] = gh.Move(rs.coercegeometry(v1[i]),vAux1)[0]
for i in range(len(v2)):
    v2[i] = gh.Move(rs.coercegeometry(v2[i]),vAux1)[0]
for i in range(len(C_v1)):
   C_aux = gh.Move(rs.coerceline(C_v1[i]),vAux1)[0]
   carreg_v1.append(C_aux)
for i in range(len(C_v2)):
   C_aux = gh.Move(rs.coerceline(C_v2[i]),vAux1)[0]
   carreg_v2.append(C_aux)
for i in range(len(Conector)):
    Conector[i] = gh.Move(rs.coercegeometry(Conector[i]),vAux1)[0]
F0 = gh.Move(rs.coerceline(F0), vAux1)[0]
#carregamento na ordem de seleção das forças
carreg_2 = carreg_v1 + [F0] + carreg_v2[::-1] #L[::-1] ==inverso da lista L
FG2 = v1 + v2 + Conector + carreg_2
nomes_cargas_v1 =[]
for i in range(len(carreg_v1)):
    nome = "P" + str(i) + '_v1'
    nomes_cargas_v1.append(nome)
nomes_cargas_v2 =[]
for i in range(len(carreg_v2)):
    nome = "P" + str(i) + '_v2'
    nomes_cargas_v2.append(nome)
nomes_cargas_2 = nomes_cargas_v1 + ['F0'] + nomes_cargas_v2[::-1]
dic_2, raios_2, funicular_2, resultante_2, ptos = PF_funic(pto_inicial_2
,polo_2,carreg_2, nomes_cargas_2)
for i in dic_2.keys():
    PF2.append(dic_2[i])
txt_pontos += ptos
#calculo das reações
bi1 = v1[2]
pto1 = rs.CurveStartPoint(bi1)
bi2 = v2[2]
pto2 = rs.CurveStartPoint(bi2)
#Eixo reação R_v1
L1 = Linha_force_extena(pto1, 1,1)
#Eixo reação R_v2
L2 = Linha_force_extena(pto2, 1,1)
pAux1 = gh.LineXLine(funicular_2[0],rs.coerceline(L1))[-1]
pAux2 = gh.LineXLine(funicular_2[-1],rs.coerceline(L2))[-1]
#resesenhando as cordas extremas
funicular_2[0] = estendeCorda(funicular_2[0],pAux1,0)
funicular_2[-1] = estendeCorda(funicular_2[-1],pAux2,1)
funicAux = rs.AddLine(pAux1,pAux2)
#fechando o funicular
funicular_2.append(funicAux)
```

```
vAux = rs.AddLine(pAux1,polo_2)
#Movendo corda que fecha o funicular para o PF
raio_aux = gh.Move(rs.coerceline(funicAux),rs.coerceline(vAux))[0]
#dividindo a resultante nas reações
pAux = gh.LineXLine(raio_aux,resultante_2[0])[-1]
resultante_2.append(pAux)
#desenhando o raio polar
raio_aux = rs.AddLine(polo_2,pAux)
#colocando o raio polar na lista de sarida raios_2
raios_2.append(raio_aux)
#reação R_V1 no PF
R_v1 = rs.AddLine(pAux,pto_inicial_2)
#reação R_V2 no PF
pAux3 = dic_2['P0_v2'].PointAt(1)
R_v2 = rs.AddLine( pAux3, pAux )
#Colocando reações no dicionário do PF
dic_2['R_v1'] = rs.coerceline(R_v1)
dic_2['R_v2'] = rs.coerceline(R_v2)
#Colocando reações na lista de saida resultante_2
resultante_2 += [pAux3,R_v1,R_v2]
#movendo reações para o FG
vAux1 = rs.AddLine(pto_inicial_2,pto1)
vAux2 = rs.AddLine(pAux,pto2)
fgR_v1 = gh.Move(rs.coerceline(R_v1),rs.coerceline(vAux1))[0]
fgR_v2 = gh.Move(rs.coerceline(R_v2),rs.coerceline(vAux2))[0]
#Nomenclatura R_v1
carga1 = gh.Length(fgR_v1)*1/Escala
p_txt = fgR_v1.PointAt(.5)
texto = 'R_v1 = ' + str('%.2f' % carga1)
txt_pontos += [p_txt,texto,corcargas]
#Nomenclatura R_v2
carga2 = gh.Length(fgR_v2)*1/Escala
p_txt = fgR_v2.PointAt(.5)
texto = 'R_v2 = ' + str('%.2f' % carga2)
txt_pontos += [p_txt,texto,corcargas]
#Colocando reações na lista de saida do FG
FG2.append(fgR_v1)
FG2.append(fgR_v2)
print '####--------------------V1-----------------------####'
dicUp, ptos = Grafo_Viga( v1, carreg_v1, Plano, dic_2, cirDir,
Conector[1], fgR_v1, 'v1')
txt_pontos += ptos
#atualizando dicionário
dic_2.update(dicUp)
#Atualizando elementos na saida PF2
for i in dicUp.keys():
    PF2.append(dicUp[i])
print '####--------------------V2-----------------------####'
dicUp, ptos = Grafo_Viga( v2, carreg_v2, Plano, dic_2, cirDir,
Conector[1], fgR_v2, 'v2')
txt_pontos += ptos
#atualizando dicionário
dic_2.update(dicUp)
#Atualizando elementos na saida PF2
for i in dicUp.keys():
    PF2.append(dicUp[i])
print '####--------------Eixo do Conector--------------####'
nos = rs.PolylineVertices(Conector[1])
no = nos[0]
conector = rs.ExplodeCurves(Conector[1])
conector.append(rs.AddLine(no,nos[2]))
#elementos do conector
elementos, nomesNo = elemntos_node(no,conector,'bc','c')
#Força externa
fgF0 = (rs.AddLine(F0.PointAt(0),F0.PointAt(1)))
nomesNo, elementos, countPF, countCalcular = OrdenaLinhasDeAcao(no,cirDir,
elementos,nomesNo, dic_2, Plano)
```

```
nomesNo.insert(1,'F0')
elementos.insert(1,fgF0)
dicUp, ptos1 = Cremona1(no, nomesNo, elementos, countPF+1, dic_2)
txt_pontos += ptos1
dic_2.update(dicUp)
#Atualizando elementos na saida PF2
for i in dicUp.keys():
    PF2.append(dicUp[i])
print 'elementos tracionados:', len(Ltrac)
print 'elementos comprimidos:', len(Lcomp)
```