```
from __future__ import division

from Rhino.Geometry import Point3d, Line
import ghpythonlib.components as gh
import rhinoscriptsyntax as rs

#Cirando Listas vazias para os outputs
funicular=[]
PF=[]
raios=[]
resultante=[]
construc_aux=[]

#função de desenho das cordas
def estendeCorda(linha,ponto,indice):
    pto1 = gh.EndPoints(linha)[indice]
    return Line(pto1,ponto)

#função de redesenho das  linhas de ação para melhor visualisação
def estendeFG(linha,corda,ponto):
    linnha = rs.coerceline(linha)
    ponto = rs.coerce3dpoint(ponto)
    if not gh.CurveXCurve(linha,corda)[0]:
        pt =gh.EndPoints(linha)[0]
        construc_aux.append(Line(pt,ponto))
        return True
    else:
        return False

#função que testa se o funicular é fechado ou aberto
def testeFunicular(vetores,funicular,tolerancia):
    prm1 = gh.LineXLine(rs.coerceline(vetores[0]),funicular[0])[0]
    prm2 = gh.LineXLine(rs.coerceline(vetores[0]),funicular[-1])[0]
    if abs(prm1 - prm2) <= tolerancia :
        return True
    else:
        return False

#Calcula a distãncia entre duas retas paralelas
def distParalela(reta1,reta2):
    pt1 = gh.EndPoints(reta1)[0]
    pt2 = reta2.ClosestPoint(pt1,False)
    return gh.Length(Line(pt1,pt2))


# Desenhando o poligono de forças
pAux = rs.coerce3dpoint(pto_inicial)


polo = rs.coerce3dpoint(polo)

#primeiro raio polar
raios.append(Line(polo,pAux))

for v in vetores:
    vAux1 = rs.coerceline(v)
    vAux2 = pAux - v.PointAt(0)
    vAux3 = gh.Move(vAux1,vAux2)[0]
    pAux = vAux3.PointAt(1)
    PF.append(vAux3)
    r=Line(polo,pAux)
    raios.append(r)

#desenhando a resultante
result=Line(rs.coerce3dpoint(pto_inicial),pAux)
```

```
#Desenhando o funicular

j = len(raios)

for i in range(j):
    r = rs.coerceline(raios[i])
    if i == 0:
        pAux = (rs.coerceline(vetores[0]).PointAt(.5))
        vAux1 = Line(r.PointAt(0),pAux)
        corda = gh.Move(r,vAux1)[0]
        corda = rs.coerceline(corda)
        funicular.append(corda)

    elif i < j-1:
        vAux1 = Line(r.PointAt(1),pAux)
        crdAux = gh.Move(r,vAux1)[0]
        crdAux = rs.coerceline(crdAux)
        pAux2 = pAux
        pAux = gh.LineXLine(crdAux,vetores[i])[-1]
        corda = Line(pAux2,pAux)
        funicular.append(corda)
        estendeFG(vetores[i],corda,pAux)


    else:
        vAux1 = Line(r.PointAt(1),pAux)
        corda = gh.Move(r,vAux1)[0]
        corda = rs.coerceline(corda)
        funicular.append(corda)

#testando PF ppara equilíbrio de translação
testePF=gh.Length(result)
if testePF <= 0.0001:
    eqtrans = True
    print "Equilíbrio de Translação"

    #Testando o Funicular para equilibrio de Rotação

    #Testando paralelismo das cordas extremas do funicular
    if testeFunicular(vetores,funicular,0.0001):
        print "Equilíbrio de Rotação"
        pAux = gh.LineXLine(funicular[0],vetores[0])[-1]
        pAux = rs.coerce3dpoint(pAux)
        funicular[0] = estendeCorda(funicular[0],pAux,0)
        funicular[-1] = estendeCorda(funicular[-1],pAux,1)
    else:
        #Cálculo do Momento
        braco = distParalela(funicular[0],funicular[-1])
        force = gh.Length(Line(polo,rs.coerce3dpoint(pto_inicial)))
        binario = braco * force
        print "Momento ", binario
#Caso o PF seja Aberto
#Posicionar a linha de ação da resultante na FG
else:
    print "Módulo da resultante: ", testePF
    resultante.append(result)
    pAux = gh.LineXLine(funicular[0],funicular[-1])[-1]
    funicular[0] = estendeCorda(funicular[0],pAux,0)
    funicular[-1] = estendeCorda(funicular[-1],pAux,1)
    vAux1 = Line(result.PointAt(.5),pAux)
    result2 = gh.Move(result,vAux1)[0]
    resultante.append(result2)
    eqtrans = False
```