ANNEX I – Summary of Truss Types Used in Analysis



**ANNEX – MAJOR CLUSTER CODING IN Grasshopper PYTHON (ghPython)**

**CLUSTER 1 –**

**CLUSTER NAME : MAIN SOLVER**

**import** rhinoscriptsyntax **as** rs

**import** Rhino.Geometry **as** rg

**import** scriptcontext **as** sc

#Creates a copy of an array.

**def copy\_func**(original):

copy = []

**print** original

**for** i **in range**(0,**len**(original)):

copy.**append**(original[i])

return copy

#load\_vec\_copy = copy\_func(lfl)

#MODULE 1

#list the end and start point of lines as per their indices

**def list\_endpoints**(a):

lleng = **len**(x) #get length of list

lfm = [[]] #form diagram line endpoints -2d null matrix

lfs=[] #form diagram slope lines

#Get End points of the lines in form diagram

**for** i **in range**(0, lleng): # LOOP

lfm.**append**([])

ls = rs.**CurveStartPoint**(x[i]) #get start point from force diagram

le = rs.**CurveEndPoint**(x[i]) # get end point from force diagram

lsp = rs.**coerce3dpoint**(ls) #convert guid to point

lep = rs.**coerce3dpoint**(rs.**AddPoint**(le)) #convert guid to point

lfm[i].**append**(lsp) #append to list of points

lfm[i].**append**(lep) #append to list of points

#get the slope of the line

**if abs**(lep[0] - lsp[0]) >= 0.01:

slope **=** (lsp[2]-lep[2])**/**(lsp[0]-lep[0])

lfs.**append**(**round**(slope,2))

**elif abs**(lep[0] - lsp[0]) <= 0.01:

lfs.**append**(321)

#list\_endpoints(1)

# MODULE 2

# Add length of each line in force diagram

#lfl = [] #length of each lines in force diagram

**def force\_list**(a):

rlen = **len**(fd\_line)

**for** i **in range**(0,rlen):

chk=0

**for** j **in range** (0, **len**(y)):

**if float**(i) == y[j]:

lfl.**append**(z[j])

chk =1

**if** chk==0:

lfl.**append**(321)

return lfl

#force\_list(1)

#print lfl

#Create List of points(dummy) for corresponding regions in force diagram. #Points are currently in y plane.

pt\_fd = [] # force diagram region co-ordinates.

**def dummy\_fd\_points**(a):

**for** i **in range**(0,**len**(regd)):

pt\_fd.**append**(rg.**Point3d**(0,1,0))

pt\_fd[start\_reg] =rs.**coerce3dpoint**(fdsp)

return pt\_fd

**dummy\_fd\_points**(1)

#Create a tracker for construction line work done.

tracker\_cline= [321]

# Create a list for lines in force diagram.

tracker\_ptsk = [10]

fdl = []

intersecting\_lines =[]

cline\_all=[[],[]]

#gets the clockwise direction of regions with external loads

**def clock**(r1,r2):

**for** i **in range**(0,**len**(rav)):

**for** j **in range**(1,**len**(rav[i])):

**if** rav[i][j-1] == r1:

**if** rav[i][j]== r2:

return "anti-clock"

**for** j **in range**(1,**len**(rav[i])):

**if** rav[i][j-1] == r2:

**if** rav[i][j]== r1:

return "clock"

#graphic = clock(2,10)

#print graphic

#print rav

# returns vector of intersection of regions provided

**def vbregion**(r1,r2):

curl = rs.**IntersectBreps**(act\_reg[r1],act\_reg[r2])

curlvec = rs.**VectorCreate**(rs.**CurveStartPoint**(curl),rs.**CurveEndPoint**(curl))

return curlvec

fdline\_data = []

# returns line index of intersection of regions provided

**def l\_index**(r1,r2):

curl = rs.**IntersectBreps**(act\_reg[r1],act\_reg[r2])

**for** k **in range**(0,**len**(fd\_line)):

**if** rs.**CurveMidPoint**(curl) == rs.**CurveMidPoint**(fd\_line[k]) :

line\_index = k

return line\_index

#get co-ordinate of lines with known forces & draw line.

**def getcord**(r1,r2):

**for** i **in range**(0,1):

curl\_index = **l\_index**(r1,r2)

curl\_vec = **vbregion**(r1,r2)

cur\_cord = pt\_fd[r1]

curl\_mag = lfl[curl\_index]

curl\_clock = **clock**(r1,r2)

**if** curl\_clock == "clock":

v\_clock = -1

**elif** curl\_clock =="anti-clock":

v\_clock= 1 # print curl\_vec

**if** lfl[curl\_index] != 321:

line =rg.**Line**(cur\_cord,curl\_vec \* v\_clock,curl\_mag)

line\_guid = sc.doc.Objects.**AddLine**(line)

cord = rs.**CurveEndPoint**(line\_guid)

pt\_fd[r2]=cord

#tracker\_ptsk.append(r2)

return cord

#getcord(10,2):

# Draw Cline or Fxdline -with coordinate from a point.

**def lines\_and\_pt**(curreg):

neigh\_reg =regd[curreg]

**for** i **in** neigh\_reg:

l\_cmon = **l\_index**(i,curreg)

**if** lfl[l\_cmon] !=321:

neigh\_cord = **getcord**(curreg,i)

pt\_fd[i] = neigh\_cord

**elif** lfl[l\_cmon] == 321:

r1= curreg

r2 = i

**if** pt\_fd[r2][1]==1:

curvec= **vbregion**(r1,r2)

curpt = pt\_fd[curreg]

curl = rg.**Line**(rs.**coerce3dpoint**(curpt),curvec,10000)

cline\_all[0].**append**(curl)

cline\_index = [r1,r2]

cline\_all[1].**append**(cline\_index)

tracker\_cline.**append**(curreg)

return cline\_all

**def knownpts**(zz):

cur\_reglist = []

**for** i **in range**(0,**len**(pt\_fd)):

**if** pt\_fd[i][1] ==0:

a=1

**if** i **in** tracker\_cline:

a=0

**if** a==1:

cur\_reglist.**append**(i)

return cur\_reglist

#cur\_reglist = knownpts(1)

#pick new regions to draw neighbouring or known points/ construction lines

**def draw\_new\_region**():

cur\_reglist = **knownpts**(1) #get known list from list of points

**for** j **in** cur\_reglist:

**lines\_and\_pt**(j)

#draw\_new\_region()

#find list of intersecting regions

**def intersecting\_clinelist**():

int\_index =[]

in\_index =[]

int\_cline =[]

int\_cline\_list =[]

**for** i **in range**(0,**len**(cline\_all[0])):

line1 = cline\_all[1][i][0]

line2 = cline\_all[1][i][1]

**for** j **in range**(i+1,**len**(cline\_all[0])):

**if** line1==cline\_all[1][j][0]:

index = [line2,cline\_all[1][j][1]]

in\_index.**append**(index)

int\_cline.**append**([i,j,line1])

**if** line1==cline\_all[1][j][1]:

index = [line2,cline\_all[1][j][0]]

in\_index.**append**(index)

int\_cline.**append**([i,j,line1])

**if** line2==cline\_all[1][j][0]:

index = [line1,cline\_all[1][j][1]]

in\_index.**append**(index)

int\_cline.**append**([i,j,line2])

**if** line2==cline\_all[1][j][1]:

index = [line1,cline\_all[1][j][0]]

in\_index.**append**(index)

int\_cline.**append**([i,j,line2])

**for** k **in range**(0,**len**(int\_cline)):

**if** rs.**IntersectBreps**(act\_reg[in\_index[k][0]],act\_reg[in\_index[k][1]]):

int\_index.**append**(in\_index[k])

int\_cline\_list.**append**(int\_cline[k])

return int\_cline

#find region point - coordinate by intersection of two cosecutive regions

**def pts\_intersection**(line1,line2,k):

a=cline\_all[0][line1]

b=cline\_all[0][line2]

pt\_intersection = rs.**LineLineIntersection**(a,b)

**if** pt\_intersection:

pt\_fd[k]=pt\_intersection[0]

return pt\_intersection

#find point and add to list after intersection

**def second\_type\_line**():

**for** i **in range** (0,20):

in\_list = **intersecting\_clinelist**()

**for** j **in** in\_list:

#print in\_list

pts\_intersect = **pts\_intersection**(j[0],j[1],j[2])

#in\_list = intersecting\_clinelist()

**draw\_new\_region**()

return pts\_intersect

# Create The force Diagram.

**if** RUN:

**lines\_and\_pt**(start\_reg) #get the system started!!!

**second\_type\_line**()

**CLUSTER 2 : GET LOADS AND OUT REGIONS**

**CLUSTER DETAIL : CREATE REGIONS AND ASSIGN UNIFORMLY DISTRIBUTED LOADS TO BOTTOM CHORD AND REACTION AT END POINTS.**

**import** rhinoscriptsyntax **as** rs

up\_reg =[0]

bot\_reg = []

**def top\_or\_bottom**():

max\_check = 0

a=0

**for** i **in range**(0,**len**(reg)):

z= rs.**SurfaceAreaCentroid**(reg[i])

zco = a\*z[0][0]

check = z[0][2] - zco

**if** check <0:

bot\_reg.**append**(i)

**if** check > 0:

**if** check > max\_check:

max\_check = check

up\_reg[0]=i

return 1

**top\_or\_bottom**()

fdl = []

**for** i **in range**(0,**len**(lines)):

fdl.**append**(321)

uload = span\*udl**/**(span\_no-1)

rxn = span\*udl/2

load\_line = []

**def assign\_load**():

**for** j **in range**(0,**len**(bot\_reg)):

**for** k **in range**(j,**len**(bot\_reg)):

curl = rs.**IntersectBreps**(reg[bot\_reg[j]],reg[bot\_reg[k]])

**if** curl:

**for** i **in range**(0,**len**(lines)):

**if** rs.**CurveMidPoint**(curl) == rs.**CurveMidPoint**(lines[i]):

fdl[i]=uload

load\_line.**append**(i)

**for** m **in range**(0,1):

**for** k **in** bot\_reg:

curl = rs.**IntersectBreps**(reg[k],reg[up\_reg[0]])

**if** curl:

**for** i **in range**(0,**len**(lines)):

**if** rs.**CurveMidPoint**(curl) == rs.**CurveMidPoint**(lines[i]):

fdl[i]= - rxn

load\_line.**append**(i)

**assign\_load**()

CLUSTER 3 : GET FORCES IN FORCE DIAGRAM

**import** rhinoscriptsyntax **as** rs

fdl\_len = []

fdl\_vtx = []

fdl\_line=[]

**for** i **in** tr\_line:

fdl\_len.**append**(0)

fdl\_vtx.**append**(0)

fdl\_line.**append**(0)

#Generate length and vertices of force diagram.

**def fdline\_vertex**() :

**for** j **in range**(0,**len**(reg)):

**for** k **in range**(0,**len**(reg)):

curl = rs.**IntersectBreps**(reg[j],reg[k])

**if** curl:

**for** i **in range**(0,**len**(tr\_line)):

**if** rs.**CurveMidPoint**(curl) == rs.**CurveMidPoint**(tr\_line[i]):

fdl\_len[i] = **round**(rs.**Distance**(pt\_fd[j],pt\_fd[k]))

fdl\_vtx[i]= [j,k]

fdl\_line[i]= rs.**AddLine**(pt\_fd[j],pt\_fd[k])

return fdl\_len

**fdline\_vertex**()

#get volumne of the line = lfd \* ltd

vol = 0

**for** i **in range**(0,**len**(tr\_line)):

vol = vol + rs.**CurveLength**(tr\_line[i])\*fdl\_len [i]

vol = vol/10000

**CLUSTER 4 : BIAS ADDITION**

**CLUSTER DETAIL : ADDITION OT BIAS TO ORIGINAL VOLUME**

**import** rhinoscriptsyntax **as** rs

#get volumne of the line = lfd \* ltd

vol = 0

j\_bias = joint\_bias

**for** i **in range**(0,**len**(tr\_line)):

**if** force[i] == 2:

b\_bias= buckling\_bias

**if** force[i] == 1:

b\_bias = 1

vol = vol + rs.**CurveLength**(tr\_line[i])\*fdl\_len[i]**\*** (1**+**(1-b\_bias)+j\_bias)

vol = vol/10000

**print** vol