##A) Checking number of segments and no of points in each segment

curve = bpy.context.active\_object.data.splines

No\_of\_segment = len(bpy.context.active\_object.data.splines)

points\_in\_each\_segment = []

for each\_segment in list(range(No\_of\_segment)):

points\_of\_ith\_segment = len(curve[each\_segment].bezier\_points)

points\_in\_each\_segment.append(points\_of\_ith\_segment)

##B) standardization number of points for each segment

standardized\_no\_of\_points = 200

for each\_segment in list(range(No\_of\_segment)):

while (len(curve[each\_segment].bezier\_points)) < standardized\_no\_of\_points:

list\_of\_curvelength=[]

for first\_points in ((list(range(len(curve[each\_segment].bezier\_points))))[:-1]):

the\_distance\_between\_two\_points=(((curve[each\_segment].bezier\_points[(first\_points+1)].co)-(curve[each\_segment].bezier\_points[first\_points].co)).length)

list\_of\_curvelength.append(the\_distance\_between\_two\_points)

longest\_length\_between\_points = max(list\_of\_curvelength)

greatest\_gap\_index=list\_of\_curvelength.index(longest\_length\_between\_points)

curve[each\_segment].bezier\_points[greatest\_gap\_index].select\_control\_point=True

bpy.ops.object.mode\_set(mode = 'EDIT')

bpy.ops.curve.select\_next()

bpy.ops.curve.subdivide()

bpy.ops.curve.select\_all()

bpy.ops.object.mode\_set(mode = 'OBJECT')

##C) calculate length of each segment and save output

curve = bpy.context.active\_object.data.splines

name = bpy.context.active\_object.name

No\_of\_segment = len(bpy.context.active\_object.data.splines)

points\_in\_each\_segment = []

for each\_segment in list(range(No\_of\_segment)):

points\_of\_ith\_segment = len(curve[each\_segment].bezier\_points)

points\_in\_each\_segment.append(points\_of\_ith\_segment)

list\_of\_length\_of\_segments = []

for each\_spline in list(range(No\_of\_segment)):

points\_from\_standardized\_segments = len(curve[each\_spline].bezier\_points)

list\_of\_length\_between\_points = []

for begin\_point in ((list(range(points\_from\_standardized\_segments)))[:-1]):

the\_distance\_between\_points=(((curve[each\_spline].bezier\_points[(begin\_point+1)].co)-(curve[each\_spline].bezier\_points[begin\_point].co)).length)

list\_of\_length\_between\_points.append(the\_distance\_between\_points)

list\_of\_length\_of\_segments.append(sum(list\_of\_length\_between\_points))

output\_temp0 = list\_of\_length\_of\_segments.##reverse()

output\_temp1 = (str(list\_of\_length\_of\_segments)).replace(', ','\n')

output\_temp2 = output\_temp1.replace('[','')

output = output\_temp2.replace(']','')

##Step 12 - write output file in csv format

##Parameters(newfolderpath)

import os

fp=bpy.data.filepath

filepath=os.path.basename(fp)

Blender\_file\_name=filepath[:-6]

####newfolderpath="c:/E\_19072009/Manuscript/PhD thesis/On growth and form of two heteromorphic terrestrial gastropod snails/3D aperture outline analysis/EFA\_blender/"

newfolderpath="c:/E\_19072009/EFA\_blender/"

if not os.path.isdir(newfolderpath):

os.makedirs(newfolderpath)

outputfile1= newfolderpath+" "+Blender\_file\_name+name+"\_analysis"+".csv"

writefile=open(outputfile1, 'w')

writefile.write(output)

writefile.close()

##Script pre-A

##Check broken curve and faciliate broken curve fixing

## Check number of broken segment(s)

import mathutils

CURVE0=bpy.context.active\_object.data.splines

CURVE1= CURVE0[0].bezier\_points

Number\_of\_segment = len(CURVE0)

if Number\_of\_segment >1:

print(Number\_of\_segment)

CURVE2= CURVE0[1].bezier\_points

last\_point\_of\_second\_curve=(len(CURVE2))-1

Local\_position\_of\_1st\_point\_1stCurve= CURVE1[0].select\_control\_point=True

Local\_position\_of\_last\_point\_2ndCurve= CURVE2[last\_point\_of\_second\_curve].select\_control\_point=True

##bpy.ops.curve.make\_segment()

##A) standardizing number of points for outline

##(Parameter= standard\_no\_of\_point)

##Script i - standardization of number of outline points

CURVE0=()

all\_objects=bpy.data.objects

object\_ID\_name=()

total\_number\_of\_objects=(len(all\_objects))

all\_object\_ID=list(range(total\_number\_of\_objects))

for each\_object in all\_object\_ID:

## CURVE0=all\_objects[each\_object]

CURVE0=bpy.context.active\_object

CURVE1= CURVE0.data.splines[0].bezier\_points

points\_of\_curve=CURVE1

Points\_C1=len(CURVE1)

## Output\_raw\_EFA = ''

## Output\_normal\_EFA = ''

no\_of\_points=len(CURVE1)

first\_points=0

list\_of\_curvelength=[]

total\_length\_between\_points=0

standard\_no\_of\_points=800

while first\_points <= (no\_of\_points-2):

the\_distance\_between\_two\_points=(((points\_of\_curve[(first\_points+1)].co)-(points\_of\_curve[first\_points].co)).length)

list\_of\_curvelength.append(the\_distance\_between\_two\_points)

total\_length\_between\_points += the\_distance\_between\_two\_points

first\_points +=1

the\_distance\_between\_1st\_lastpoints=(((points\_of\_curve[0].co)-(points\_of\_curve[-1].co)).length)

total\_length\_between\_points += the\_distance\_between\_1st\_lastpoints

list\_of\_curvelength.append(the\_distance\_between\_1st\_lastpoints)

average\_dis=total\_length\_between\_points/standard\_no\_of\_points

greatest\_gap=max(list\_of\_curvelength)

smallest\_gap=min(list\_of\_curvelength)

while greatest\_gap > (average\_dis \* 1.51) and no\_of\_points <=standard\_no\_of\_points and smallest\_gap > 0:

points\_of\_curve=bpy.context.active\_object.data.splines[0].bezier\_points

greatest\_gap\_index=list\_of\_curvelength.index(greatest\_gap)

if greatest\_gap\_index != ((len(list\_of\_curvelength))-1):

points\_of\_curve[greatest\_gap\_index].select\_control\_point=True

bpy.ops.object.mode\_set(mode = 'EDIT')

bpy.ops.curve.select\_next()

no\_of\_division= round(greatest\_gap/(average\_dis))

bpy.ops.curve.subdivide(number\_cuts=(no\_of\_division))

bpy.ops.curve.select\_all()

bpy.ops.object.mode\_set(mode = 'OBJECT')

points\_of\_curve=bpy.context.active\_object.data.splines[0].bezier\_points

no\_of\_points=len(points\_of\_curve)

first\_points=0

list\_of\_curvelength=[]

total\_length\_between\_points=0

while first\_points <= (no\_of\_points-2):

the\_distance\_between\_two\_points=(((points\_of\_curve[(first\_points+1)].co)-(points\_of\_curve[first\_points].co)).length)

list\_of\_curvelength.append(the\_distance\_between\_two\_points)

total\_length\_between\_points += the\_distance\_between\_two\_points

first\_points +=1

the\_distance\_between\_1st\_lastpoints=(((points\_of\_curve[0].co)-(points\_of\_curve[-1].co)).length)

total\_length\_between\_points += the\_distance\_between\_1st\_lastpoints

list\_of\_curvelength.append(the\_distance\_between\_1st\_lastpoints)

greatest\_gap=max(list\_of\_curvelength)

else:

points\_of\_curve[0].select\_control\_point=True

bpy.ops.object.mode\_set(mode = 'EDIT')

bpy.ops.curve.select\_next()

##no\_of\_division=round((greatest\_gap/average\_dis)-1)

bpy.ops.curve.subdivide(number\_cuts=1)

bpy.ops.curve.select\_all()

bpy.ops.object.mode\_set(mode = 'OBJECT')

points\_of\_curve=bpy.context.active\_object.data.splines[0].bezier\_points

no\_of\_points=len(points\_of\_curve)

first\_points=0

list\_of\_curvelength=[]

total\_length\_between\_points=0

while first\_points <= (no\_of\_points-2):

the\_distance\_between\_two\_points=(((points\_of\_curve[(first\_points+1)].co)-(points\_of\_curve[first\_points].co)).length)

list\_of\_curvelength.append(the\_distance\_between\_two\_points)

total\_length\_between\_points += the\_distance\_between\_two\_points

first\_points +=1

the\_distance\_between\_1st\_lastpoints=(((points\_of\_curve[0].co)-(points\_of\_curve[-1].co)).length)

total\_length\_between\_points += the\_distance\_between\_1st\_lastpoints

list\_of\_curvelength.append(the\_distance\_between\_1st\_lastpoints)

greatest\_gap=max(list\_of\_curvelength)

smallest\_gap=min(list\_of\_curvelength)

greatest\_gap=max(list\_of\_curvelength)

smallest\_gap=min(list\_of\_curvelength)

no\_of\_points=len(points\_of\_curve)

while no\_of\_points > standard\_no\_of\_points:

smallest\_gap=min(list\_of\_curvelength)

smallest\_gap\_index=list\_of\_curvelength.index(smallest\_gap)

points\_of\_curve[smallest\_gap\_index].select\_control\_point=True

bpy.ops.object.mode\_set(mode = 'EDIT')

bpy.ops.curve.delete()

bpy.ops.object.mode\_set(mode = 'OBJECT')

points\_of\_curve=bpy.context.active\_object.data.splines[0].bezier\_points

no\_of\_points=len(points\_of\_curve)

no\_of\_points

##B) Reorientation: set the homologous landmark as first point

CURVE0=bpy.context.active\_object

##Step 4 - Reorientation: set the homologous landmark as first point

##(Parameter= X\_reduction\_of\_point, optional function = to reverse points)

CURVE1=CURVE0.data.splines[0].bezier\_points

name=str(CURVE0.name)

Points\_C1=len(CURVE1)

list\_of\_points\_length = []

for each\_point in list(range(Points\_C1)):

point\_length = ((CURVE1[each\_point].co).length)

list\_of\_points\_length.append(point\_length)

position\_of\_homologous\_point = list\_of\_points\_length.index(min(list\_of\_points\_length))

Verts = []

for first\_phase\_point in list(range(position\_of\_homologous\_point,Points\_C1)):#

Verts.append((CURVE1[first\_phase\_point]).co)#

for second\_phase\_point in list(range(0,position\_of\_homologous\_point)):#

Verts.append((CURVE1[second\_phase\_point]).co)#

#list1 = list(range(0,position\_of\_homologous\_point))

#list1.reverse()

#for first\_phase\_point in list1:

# Verts.append((CURVE1[first\_phase\_point]).co)

#list2 = list(range(position\_of\_homologous\_point, Points\_C1))

#list2.reverse()

#for second\_phase\_point in list2:

# Verts.append((CURVE1[second\_phase\_point]).co)

##C) EFA analysis

##Step 7 - calculate EFA analysis from the points of mesh

##(Parameter= no\_of\_harmonics)

x\_coord\_mesh = []

y\_coord\_mesh = []

z\_coord\_mesh = []

no\_of\_points = len(Verts)

for each\_point in range(no\_of\_points):

x\_coord\_mesh.append(Verts[each\_point][0])

y\_coord\_mesh.append(Verts[each\_point][1])

z\_coord\_mesh.append(Verts[each\_point][2])

p=len(x\_coord\_mesh)

no\_of\_harmonics=5

i=0

Dx\_list=[]

Dy\_list=[]

Dz\_list=[]

Dt\_list=[]

firstandlast\_x=x\_coord\_mesh[0]-x\_coord\_mesh[-1]

Dx\_list.append(firstandlast\_x)

firstandlast\_y=y\_coord\_mesh[0]-y\_coord\_mesh[-1]

Dy\_list.append(firstandlast\_y)

firstandlast\_z=z\_coord\_mesh[0]-z\_coord\_mesh[-1]

Dz\_list.append(firstandlast\_z)

firstandlast\_t=sqrt(firstandlast\_x\*\*2 + firstandlast\_y\*\*2 +firstandlast\_z\*\*2)

Dt\_list.append(firstandlast\_t)

while i <= (p-2):

Dx=x\_coord\_mesh[i+1]-x\_coord\_mesh[i]

Dx\_list.append(Dx)

Dy=y\_coord\_mesh[i+1]-y\_coord\_mesh[i]

Dy\_list.append(Dy)

Dz=z\_coord\_mesh[i+1]-z\_coord\_mesh[i]

Dz\_list.append(Dz)

Dt= sqrt(Dx\*\*2 + Dy\*\*2 + Dz\*\*2)

Dt\_list.append(Dt)

i +=1

cumsum\_i = 0

cumsum\_list = []

for each\_Dt in Dt\_list:

if Dt\_list.index((Dt\_list[-1])) != -1:

cumsum\_i += each\_Dt

cumsum\_list.append(cumsum\_i)

t1 = cumsum\_list

t1m1\_temp=[float(0)]

t1m1\_temp1=t1m1\_temp + t1

t1m1=t1m1\_temp1[:-1]

T = sum(Dt\_list)

temp\_output0=[]

list\_of\_points = list(range(p))

list\_of\_harmonic = list(range(1,(no\_of\_harmonics+1)))

harmonics\_an=[]

harmonics\_bn=[]

harmonics\_cn=[]

harmonics\_dn=[]

harmonics\_en=[]

harmonics\_fn=[]

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

a=(Dx\_list[each\_point]/Dt\_list[each\_point]) \* ((cos (2 \* each\_harmonic \* pi \* (t1[each\_point]) / T)) -(cos (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=a

an = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_an.append(an)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

b=(Dx\_list[each\_point]/Dt\_list[each\_point]) \* ((sin (2 \* each\_harmonic \* pi \* (t1[each\_point]) / T)) -(sin (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=b

bn = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_bn.append(bn)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

c=(Dy\_list[each\_point]/Dt\_list[each\_point]) \* ((cos (2 \* each\_harmonic \* pi \* (t1[each\_point]) / T)) -(cos (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=c

cn = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_cn.append(cn)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

d=(Dy\_list[each\_point]/Dt\_list[each\_point]) \* ((sin (2 \* each\_harmonic \* pi \* (t1[each\_point]) / T)) -(sin (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=d

dn = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_dn.append(dn)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

e=(Dz\_list[each\_point]/Dt\_list[each\_point]) \* ((cos (2 \* each\_harmonic \* pi \* (t1[each\_point]) / T)) -(cos (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=e

en = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_en.append(en)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

f=(Dz\_list[each\_point]/Dt\_list[each\_point]) \* ((sin (2 \* each\_harmonic \* pi \* (t1[each\_point]) / T)) -(sin (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=f

fn = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_fn.append(fn)

temp\_ao=0

for each\_point in list\_of\_points:

temp\_output0=(x\_coord\_mesh[each\_point]) \* ((Dt\_list[each\_point])/T)

temp\_ao +=temp\_output0

ao=2 \* temp\_ao

temp\_co=0

for each\_point in list\_of\_points:

temp\_output0=(y\_coord\_mesh[each\_point]) \* ((Dt\_list[each\_point])/T)

temp\_co +=temp\_output0

co=2 \* temp\_co

temp\_eo=0

for each\_point in list\_of\_points:

temp\_output0=(z\_coord\_mesh[each\_point]) \* ((Dt\_list[each\_point])/T)

temp\_eo +=temp\_output0

eo=2 \* temp\_eo

harmonics\_an

harmonics\_bn

harmonics\_cn

harmonics\_dn

harmonics\_en

harmonics\_fn

ao

co

eo

combine\_output=[harmonics\_an, harmonics\_bn,harmonics\_cn,harmonics\_dn,harmonics\_en,harmonics\_fn,ao,co,eo]

combine\_output

##D) Inversion of harmonics from Elliptic fourier Analysis to plot mesh

##Step 9 - Inversion of harmonics from Elliptic fourier Analysis to plot mesh

##(Parameter= n(no\_of\_vertices), k(no\_of\_harmonics),)

####Script K - Inversion of harmonics from Elliptic fourier Analysis to plot outline mesh

import math

n=100

k=no\_of\_harmonics

harmonics\_no\_index=list(range(1,k+1))

harmonics\_value\_index=list(range(k))

theta\_list\_of\_points=[]

theta\_list\_index=list(range(n))

pi\_for\_each\_point=2 \* pi / n

for each\_point in list(range(n)):

theta\_of\_each\_points = pi\_for\_each\_point \* each\_point

theta\_list\_of\_points.append(theta\_of\_each\_points)

x\_EFA=[]

y\_EFA=[]

z\_EFA=[]

for each\_point\_index in theta\_list\_index:

temp\_x\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_x=harmonics\_an[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + harmonics\_bn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_x\_list.append(temp\_x)

x\_coordEFA=ao/2 + sum(temp\_x\_list)

x\_EFA.append(x\_coordEFA)

for each\_point\_index in theta\_list\_index:

temp\_y\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_y=harmonics\_cn[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + harmonics\_dn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_y\_list.append(temp\_y)

y\_coordEFA=co/2 + sum(temp\_y\_list)

y\_EFA.append(y\_coordEFA)

for each\_point\_index in theta\_list\_index:

temp\_z\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_z=harmonics\_en[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + harmonics\_fn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_z\_list.append(temp\_z)

z\_coordEFA=eo/2 + sum(temp\_z\_list)

z\_EFA.append(z\_coordEFA)

Verts = []

the\_point\_index = 0

while the\_point\_index <= n-1:

unique\_vertex=(x\_EFA[the\_point\_index],y\_EFA[the\_point\_index],z\_EFA[the\_point\_index])

Verts.append(unique\_vertex)

the\_point\_index +=1

numberofedges=len(Verts)

list\_of\_edges=list(range(numberofedges))

Edges=[]

firstedgestart=0

while firstedgestart <= numberofedges-2:

firstpoint=(list\_of\_edges[firstedgestart])

secondpoint=(list\_of\_edges[(firstedgestart+1)])

unique\_edge=(firstpoint,secondpoint)

Edges.append(unique\_edge)

firstedgestart +=1

lastpoint\_and\_firstpoint=((list\_of\_edges[-1]),(list\_of\_edges[0]))

Edges.append(lastpoint\_and\_firstpoint)

the\_temp\_mesh=bpy.data.meshes.new((name+'\_EFA'))

the\_temp\_mesh.from\_pydata(Verts,Edges,[])

the\_temp\_mesh.update()

the\_th\_object=bpy.data.objects.new((name+'\_EFA'), the\_temp\_mesh)

the\_th\_object.data=the\_temp\_mesh

scene=bpy.context.scene

scene.objects.link(the\_th\_object)

the\_th\_object.select = True

##E) calculate perimeter

all\_data = []

total\_length\_between\_points=0

the\_distance\_between\_two\_points=0

points\_of\_curve = bpy.context.active\_object.data.vertices

no\_of\_points = len(points\_of\_curve)

name = bpy.context.active\_object.name

first\_points=0

while first\_points <= (no\_of\_points-2):

the\_distance\_between\_two\_points=(((points\_of\_curve[(first\_points+1)].co)-(points\_of\_curve[first\_points].co)).length)

total\_length\_between\_points += the\_distance\_between\_two\_points

first\_points +=1

the\_distance\_between\_1st\_lastpoints=(((points\_of\_curve[0].co)-(points\_of\_curve[-1].co)).length)

total\_length\_between\_points += the\_distance\_between\_1st\_lastpoints

name

total\_length\_between\_points

data = name,total\_length\_between\_points

all\_data.append(data)

##F) save and export perimeter data for all outline

Output1 = str(all\_data)

Output2 = Output1.replace("), (","\n")

Output3 = Output2.replace("'","")

Output4 = Output3.replace("[(","")

Output = Output4.replace(")]","")

import os

##Step 12 - write output file in csv format

##Parameters(newfolderpath)

fp=bpy.data.filepath

filepath=os.path.basename(fp)

Blender\_file\_name=filepath[:-6]

####newfolderpath="c:/E\_19072009/Manuscript/PhD thesis/On growth and form of two heteromorphic terrestrial gastropod snails/3D aperture outline analysis/EFA\_blender/"

newfolderpath="c:/E\_19072009/EFA\_blender/"

if not os.path.isdir(newfolderpath):

os.makedirs(newfolderpath)

outputfile= newfolderpath+" "+Blender\_file\_name+" perimetere\_new\_18082012"+".csv"

writefile=open(outputfile, 'w')

writefile.write(Output)

writefile.close()

##G) normalized EFA analysis

##Step 1 - import module

import bpy

import math

import os

import array

import mathutils

normalised\_harmonics\_all = []

all\_outline = []

Output\_raw\_EFA = ''

Output\_normal\_EFA = ''

##Step 8 - normalisation of harmonics from EFA analysis

##(Parameter= k as no\_of\_harmonics; choose normalization factor: scale, O\_inverted, and direction\_of\_motion )

####Script L - Normalization of harmonics from Elliptic fourier Analysis

k=no\_of\_harmonics ##see C) EFA analysis

harmonics\_value\_index=list(range(k))

##Scaling

psi = (1/2) \* atan(2 \* ((harmonics\_an[0] \* harmonics\_bn[0]) + (harmonics\_cn[0] \* harmonics\_dn[0]) + (harmonics\_en[0] \* harmonics\_fn[0])) / (harmonics\_bn[0]\*\*2 + harmonics\_dn[0]\*\*2 + harmonics\_fn[0]\*\*2 - harmonics\_an[0]\*\*2 - harmonics\_cn[0]\*\*2 - harmonics\_en[0]\*\*2))

a = sqrt (((harmonics\_an[0]\*\*2 + harmonics\_cn[0]\*\*2 + harmonics\_en[0]\*\*2) \* cos(psi)\*\*2) + ((harmonics\_bn[0]\*\*2 + harmonics\_dn[0]\*\*2 + harmonics\_fn[0]\*\*2) \* sin(psi)\*\*2) - (((harmonics\_an[0] \* harmonics\_bn[0]) + (harmonics\_cn[0] \* harmonics\_dn[0]) + (harmonics\_en[0] \* harmonics\_fn[0])) \* sin (2 \* psi)))

b = sqrt (((harmonics\_an[0]\*\*2 + harmonics\_cn[0]\*\*2 + harmonics\_en[0]\*\*2) \* sin(psi)\*\*2) + ((harmonics\_bn[0]\*\*2 + harmonics\_dn[0]\*\*2 + harmonics\_fn[0]\*\*2) \* cos(psi)\*\*2) + (((harmonics\_an[0] \* harmonics\_bn[0]) + (harmonics\_cn[0] \* harmonics\_dn[0]) + (harmonics\_en[0] \* harmonics\_fn[0])) \* sin (2 \* psi)))

scale = 1/sqrt(pi \* a \* b)

##Rotation - refered to 1st harmonic

w = a\*b/((harmonics\_an[0]\*harmonics\_fn[0]) - (harmonics\_bn[0]\*harmonics\_en[0]))

O21 = ((harmonics\_cn[0]\*cos(psi)) - (harmonics\_dn[0]\*sin(psi)))/a

O31 = ((harmonics\_en[0]\*cos(psi)) - (harmonics\_fn[0]\*sin(psi)))/a

O22 = ((harmonics\_cn[0]\*sin(psi)) + (harmonics\_dn[0]\*cos(psi)))/b

O32 = ((harmonics\_en[0]\*sin(psi)) + (harmonics\_fn[0]\*cos(psi)))/b

alpha = ()

if ((harmonics\_an[0]\*harmonics\_fn[0]) - (harmonics\_bn[0]\*harmonics\_en[0])) > 0:

alpha = atan(((harmonics\_cn[0]\*harmonics\_fn[0])-(harmonics\_dn[0]\*harmonics\_en[0]))/((harmonics\_an[0]\*harmonics\_fn[0]) - (harmonics\_bn[0]\*harmonics\_en[0])))

else:

alpha =(atan(((harmonics\_cn[0]\*harmonics\_fn[0])-(harmonics\_dn[0]\*harmonics\_en[0]))/((harmonics\_an[0]\*harmonics\_fn[0]) - (harmonics\_bn[0]\*harmonics\_en[0])))) + pi

beta = acos(w\*((O21\*O31)+(O22\*O32)))

gamma = ()

if O31 > 0:

gamma = acos(O32/sin(beta))

else:

gamma = -acos(O32/sin(beta))

RX\_alpha = Matrix (((1,0,0),(0, cos(alpha),-sin(alpha)),(0,sin(alpha),cos(alpha))))

RX\_beta = Matrix (((1,0,0),(0, cos(beta),-sin(beta)),(0,sin(beta),cos(beta))))

RX\_gamma = Matrix (((1,0,0),(0, cos(gamma),-sin(gamma)),(0,sin(gamma),cos(gamma))))

RY\_alpha = Matrix (((cos(alpha),0,sin(alpha)),(0,1,0),(-sin(alpha),0,cos(alpha))))

RY\_beta = Matrix (((cos(beta),0,sin(beta)),(0,1,0),(-sin(beta),0,cos(beta))))

RY\_gamma = Matrix (((cos(gamma),0,sin(gamma)),(0,1,0),(-sin(gamma),0,cos(gamma))))

RZ\_alpha = Matrix (((cos(alpha),-sin(alpha),0),(sin(alpha), cos(alpha),0),(0,0,1)))

RZ\_beta = Matrix (((cos(beta),-sin(beta),0),(sin(beta), cos(beta),0),(0,0,1)))

RZ\_gamma = Matrix (((cos(gamma),-sin(gamma),0),(sin(gamma), cos(gamma),0),(0,0,1)))

##O=RX\_alpha \* RY\_beta \* RZ\_gamma ##x1y2z3

##O=RY\_alpha \* RX\_beta \* RY\_gamma ##y1x2y3

O=RZ\_alpha \* RX\_beta \* RZ\_gamma ##z1x2z3

##O = Matrix(((((cos(alpha)\*cos(gamma)) - (sin(alpha)\*cos(beta)\*sin(gamma))), ((-cos(alpha)\*sin(gamma)) - (sin(alpha)\*cos(beta)\*cos(gamma))), (sin(alpha)\*sin(beta))),(((sin(alpha)\*cos(gamma)) + (cos(alpha)\*cos(beta)\*sin(gamma))), ((-sin(alpha)\*sin(gamma)) - (cos(alpha)\*cos(beta)\*cos(gamma))), (-cos(alpha)\*sin(beta))),((sin(beta)\*sin(gamma)), (sin(beta)\*cos(gamma)),(cos(beta)))))

O\_inverted=O.inverted()

direction\_of\_motion = Matrix(((cos(psi),sin(psi)),((-sin(psi),cos(psi)))))

normalization\_factors = scale\*O\_inverted

##convert each haromonic

normalised\_harmonics\_an = []

normalised\_harmonics\_bn = []

normalised\_harmonics\_cn = []

normalised\_harmonics\_dn = []

normalised\_harmonics\_en = []

normalised\_harmonics\_fn = []

##normalised\_harmonics\_all = []

for each\_harmonic in harmonics\_value\_index:

the\_th\_harmonic\_matrix = Matrix(((harmonics\_an[each\_harmonic],harmonics\_bn[each\_harmonic]),(harmonics\_cn[each\_harmonic],harmonics\_dn[each\_harmonic]),(harmonics\_en[each\_harmonic],harmonics\_fn[each\_harmonic])))

the\_normalised\_th\_harmonic=normalization\_factors\*the\_th\_harmonic\_matrix\*direction\_of\_motion

normalised\_harmonics\_an.append((the\_normalised\_th\_harmonic[0])[0])

normalised\_harmonics\_bn.append((the\_normalised\_th\_harmonic[0])[1])

normalised\_harmonics\_cn.append((the\_normalised\_th\_harmonic[1])[0])

normalised\_harmonics\_dn.append((the\_normalised\_th\_harmonic[1])[1])

normalised\_harmonics\_en.append((the\_normalised\_th\_harmonic[2])[0])

normalised\_harmonics\_fn.append((the\_normalised\_th\_harmonic[2])[1])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[0])[0])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[0])[1])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[1])[0])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[1])[1])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[2])[0])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[2])[1])

combine\_output\_N=[normalised\_harmonics\_an, normalised\_harmonics\_bn, normalised\_harmonics\_cn, normalised\_harmonics\_dn, normalised\_harmonics\_en, normalised\_harmonics\_fn,ao,co,eo]

scale

combine\_output\_N

all\_outline.append(name)

normalised\_harmonics\_all

all\_outline

##H) Inversed Normalized EFA outline

##Step 10 - Inversion of harmonics from Normalised EFA harmonics to plot mesh

##(Parameter= n(no\_of\_vertices), k(no\_of\_harmonics),)

####Script M - Inversion of normalized harmonics from Elliptic fourier Analysis to plot outline mesh

n=500

k=no\_of\_harmonics##see C) EFA analysis & G) normalized EFA

harmonics\_no\_index=list(range(1,k+1))

harmonics\_value\_index=list(range(k))

theta\_list\_of\_points=[]

theta\_list\_index=list(range(n))

pi\_for\_each\_point=2 \* pi / n

for each\_point in list(range(n)):

theta\_of\_each\_points = pi\_for\_each\_point \* each\_point

theta\_list\_of\_points.append(theta\_of\_each\_points)

x\_NEFA=[]

y\_NEFA=[]

z\_NEFA=[]

for each\_point\_index in theta\_list\_index:

temp\_x\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_x=normalised\_harmonics\_an[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + normalised\_harmonics\_bn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_x\_list.append(temp\_x)

x\_coordNEFA=ao/2 + sum(temp\_x\_list)

x\_NEFA.append(x\_coordNEFA)

for each\_point\_index in theta\_list\_index:

temp\_y\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_y=normalised\_harmonics\_cn[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + normalised\_harmonics\_dn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_y\_list.append(temp\_y)

y\_coordNEFA=co/2 + sum(temp\_y\_list)

y\_NEFA.append(y\_coordNEFA)

for each\_point\_index in theta\_list\_index:

temp\_z\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_z=normalised\_harmonics\_en[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + normalised\_harmonics\_fn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_z\_list.append(temp\_z)

z\_coordNEFA=eo/2 + sum(temp\_z\_list)

z\_NEFA.append(z\_coordNEFA)

Verts = []

the\_point\_index = 0

while the\_point\_index <= n-1:

unique\_vertex=(x\_NEFA[the\_point\_index],y\_NEFA[the\_point\_index],z\_NEFA[the\_point\_index])

Verts.append(unique\_vertex)

the\_point\_index +=1

numberofedges=len(Verts)

list\_of\_edges=list(range(numberofedges))

Edges=[]

firstedgestart=0

while firstedgestart <= numberofedges-2:

firstpoint=(list\_of\_edges[firstedgestart])

secondpoint=(list\_of\_edges[(firstedgestart+1)])

unique\_edge=(firstpoint,secondpoint)

Edges.append(unique\_edge)

firstedgestart +=1

lastpoint\_and\_firstpoint=((list\_of\_edges[-1]),(list\_of\_edges[0]))

Edges.append(lastpoint\_and\_firstpoint)

the\_temp\_mesh=bpy.data.meshes.new((name+'\_NEFA'))

the\_temp\_mesh.from\_pydata(Verts,Edges,[])

the\_temp\_mesh.update()

the\_th\_object=bpy.data.objects.new((name+'\_NEFA'), the\_temp\_mesh)

the\_th\_object.data=the\_temp\_mesh

scene=bpy.context.scene

scene.objects.link(the\_th\_object)

the\_th\_object.select = True

##I) save output from EFA and Normalized EFA output – C) & G)

##Step 11 - compiling output data from all harmonics from all outlines of each specimen

str\_raw\_EFA = (((str(combine\_output)).replace('[','')).replace(']','')+"\n")

Output\_raw\_EFA += name + ',' + str\_raw\_EFA

str\_normal\_EFA = (((str(combine\_output\_N)).replace('[','')).replace(']','')+","+str(scale)+"\n")

Output\_normal\_EFA += name + ',' + str\_normal\_EFA

normalised\_harmonics\_all

len(normalised\_harmonics\_all)

all\_outline

##Step 12 - write output file in csv format

##Parameters(newfolderpath)

fp=bpy.data.filepath

filepath=os.path.basename(fp)

Blender\_file\_name=filepath[:-6]

####newfolderpath="c:/E\_19072009/Manuscript/PhD thesis/On growth and form of two heteromorphic terrestrial gastropod snails/3D aperture outline analysis/EFA\_blender/"

newfolderpath="c:/E\_19072009/EFA\_blender/"

if not os.path.isdir(newfolderpath):

os.makedirs(newfolderpath)

outputfile1= newfolderpath+" "+Blender\_file\_name+" raw\_EFA\_04112012"+".csv"

outputfile2= newfolderpath+" "+Blender\_file\_name+" normal\_EFA\_04112012"+".csv"

writefile=open(outputfile1, 'w')

writefile.write("\n")

writefile.write(Output\_raw\_EFA)

writefile.close()

writefile=open(outputfile2, 'w')

writefile.write("\n")

writefile.write(Output\_normal\_EFA)

writefile.close()

##A) Curvature and torsion estimation

import mathutils

q=200

a1=a2=a3=a4=a5=a6=0

bx1=bx2=bx3=by1=by2=by3=bz1=bz2=bz3=0

l\_list=[]

l\_list\_center =[]

s\_list=[]

wi\_list=[]

list\_of\_curvature = []

list\_of\_torsion = []

list\_of\_binormal\_B = []

list\_of\_normal\_N = []

list\_of\_tangent\_T = []

windows\_frame=[]

li = 0

si = 0

xi = []

yi = []

zi = []

m = 0

ww=1

points\_of\_curve=bpy.context.active\_object.data.vertices

P=len(points\_of\_curve)

name = bpy.context.active\_object.name

for each\_P0 in list(range(q,(P-q))):

windows\_frame=[]

windows\_frame = list(range((each\_P0-q),(each\_P0+q+1)))

for each\_Pi in windows\_frame[:-1]:

si = ((points\_of\_curve[each\_Pi+1].co) - (points\_of\_curve[each\_Pi].co)).length

li += si

l\_list.append(li)

s\_list.append(si)

for each\_Pi in windows\_frame[1:]:

xi.append(points\_of\_curve[each\_Pi].co[0])

yi.append(points\_of\_curve[each\_Pi].co[1])

zi.append(points\_of\_curve[each\_Pi].co[2])

m=l\_list[q]

for each\_li in l\_list:

l\_list\_center.append(each\_li-m)

for each\_Pi in list(range(q\*2)):

wi\_list.append((1/(s\_list[each\_Pi])))##\*((l\_list\_center[each\_Pi])\*\*2))

for each\_i in list(range(2\*q)):

a1 += l\_list\_center[each\_i]\*\*2 \* wi\_list[each\_i]

a2 += (l\_list\_center[each\_i]\*\*3 \* wi\_list[each\_i])/2

a3 += (l\_list\_center[each\_i]\*\*4 \* wi\_list[each\_i])/4

a4 += (l\_list\_center[each\_i]\*\*4 \* wi\_list[each\_i])/6

a5 += (l\_list\_center[each\_i]\*\*5 \* wi\_list[each\_i])/12

a6 += (l\_list\_center[each\_i]\*\*6 \* wi\_list[each\_i])/36

bx1 += wi\_list[each\_i] \* l\_list\_center[each\_i] \* (xi[each\_i]-points\_of\_curve[each\_P0].co[0])

bx2 += (wi\_list[each\_i] \* (l\_list\_center[each\_i]\*\*2) \* (xi[each\_i]-points\_of\_curve[each\_P0].co[0]))/2

bx3 += (wi\_list[each\_i] \* (l\_list\_center[each\_i]\*\*3) \* (xi[each\_i]-points\_of\_curve[each\_P0].co[0]))/6

by1 += wi\_list[each\_i] \* l\_list\_center[each\_i] \* (yi[each\_i]-points\_of\_curve[each\_P0].co[1])

by2 += (wi\_list[each\_i] \* (l\_list\_center[each\_i]\*\*2) \* (yi[each\_i]-points\_of\_curve[each\_P0].co[1]))/2

by3 += (wi\_list[each\_i] \* (l\_list\_center[each\_i]\*\*3) \* (yi[each\_i]-points\_of\_curve[each\_P0].co[1]))/6

bz1 += wi\_list[each\_i] \* l\_list\_center[each\_i] \* (zi[each\_i]-points\_of\_curve[each\_P0].co[2])

bz2 += (wi\_list[each\_i] \* (l\_list\_center[each\_i]\*\*2) \* (zi[each\_i]-points\_of\_curve[each\_P0].co[2]))/2

bz3 += (wi\_list[each\_i] \* (l\_list\_center[each\_i]\*\*3) \* (zi[each\_i]-points\_of\_curve[each\_P0].co[2]))/6

a1\_a6\_M = Matrix (((a1,a2,a4),(a2,a3,a5),(a4,a5,a6)))

bx\_y\_z\_M = Matrix (((bx1,by1,bz1),(bx2,by2,bz2),(bx3,by3,bz3)))

d\_dd\_xyz\_M = (a1\_a6\_M.inverted())\*bx\_y\_z\_M

Curvature = ((d\_dd\_xyz\_M[0].cross(d\_dd\_xyz\_M[1])).magnitude)/(((d\_dd\_xyz\_M[0]).magnitude)\*\*3)

Torsion = ((d\_dd\_xyz\_M[0].cross(d\_dd\_xyz\_M[1])).dot(d\_dd\_xyz\_M[2]))/ ((((d\_dd\_xyz\_M[0].cross(d\_dd\_xyz\_M[1])).magnitude))\*\*2)

tangent\_T = d\_dd\_xyz\_M[0] / d\_dd\_xyz\_M[0].magnitude

normal\_N = (d\_dd\_xyz\_M[1] - ((d\_dd\_xyz\_M[1].dot(tangent\_T))\*tangent\_T))/((d\_dd\_xyz\_M[1] - ((d\_dd\_xyz\_M[1].dot(tangent\_T))\*tangent\_T)).magnitude)

binormal\_B = tangent\_T.cross(normal\_N)

Curvature

Torsion

binormal\_B

normal\_N

tangent\_T

list\_of\_curvature.append(Curvature)

list\_of\_torsion.append(Torsion)

list\_of\_binormal\_B.append(binormal\_B)

list\_of\_normal\_N.append(normal\_N)

list\_of\_tangent\_T.append(tangent\_T)

a1=a2=a3=a4=a5=a6=0

bx1=bx2=bx3=by1=by2=by3=bz1=bz2=bz3=0

l\_list=[]

l\_list\_center =[]

s\_list=[]

wi\_list=[]

windows\_frame=[]

li = 0

si = 0

xi = []

yi = []

zi = []

m = 0

min(list\_of\_curvature)

sum(list\_of\_curvature)/len(list\_of\_curvature)

max(list\_of\_curvature)

len(list\_of\_curvature)

P,q

##B) calculate cumulative length and x,y,z coordinates of spiral

for each\_Pi in list(range(P-1)):

xi.append(points\_of\_curve[each\_Pi].co[0])

yi.append(points\_of\_curve[each\_Pi].co[1])

zi.append(points\_of\_curve[each\_Pi].co[2])

si = (((points\_of\_curve[each\_Pi+1].co)-(points\_of\_curve[each\_Pi].co)).length)

s\_list.append(si)

list\_of\_accumulative\_arclength = []

si=0

s\_list.reverse()

for each\_si in s\_list:

si += each\_si

list\_of\_accumulative\_arclength.append(si)

#list\_of\_curvature = (list\_of\_curvature[::q+1])

#list\_of\_torsion = (list\_of\_torsion[::q+1])

#list\_of\_accumulative\_arclength = (list\_of\_accumulative\_arclength[q+1::q+1])

##C) prepare output for – curvature, torsion, cumulative length, and x,y,z coordinates of spiral

str\_curvature\_list = ''

str\_torsion\_list = ''

Output\_torsion = ''

str\_acculative\_arclength = ''

Output\_arclength = ''

Output\_curvature\_torsion\_arclength = ''

str\_xi = ''

str\_yi = ''

str\_zi = ''

str\_curvature\_list = (((str(list\_of\_curvature)).replace('[','')).replace(']','')+"\n")

Output\_curvature = (',')\*q+str\_curvature\_list

str\_torsion\_list = (((str(list\_of\_torsion)).replace('[','')).replace(']','')+"\n")

Output\_torsion = (',')\*q+str\_torsion\_list

str\_acculative\_arclength = (((str(list\_of\_accumulative\_arclength)).replace('[','')).replace(']','')+"\n")

Output\_arclength = str\_acculative\_arclength

str\_xi = (str(xi)).replace('[','').replace(']','')+"\n"

Output\_x = str\_xi

str\_yi = (str(yi)).replace('[','').replace(']','')+"\n"

Output\_y = str\_yi

str\_zi = (str(zi)).replace('[','').replace(']','')+"\n"

Output\_z = str\_zi

Output\_curvature\_torsion\_arclength = Output\_curvature+Output\_torsion+Output\_arclength+Output\_x+Output\_y+Output\_z

##c) save all data – curvature, torsion, cumulative length, and x,y,z coordinates of spiral

import os

fp=bpy.data.filepath

filepath=os.path.basename(fp)

Blender\_file\_name=filepath[:-6]

####newfolderpath="c:/E\_19072009/Manuscript/PhD thesis/On growth and form of two heteromorphic terrestrial gastropod snails/3D aperture outline analysis/EFA\_blender/"

newfolderpath="c:/E\_19072009/EFA\_blender/"

if not os.path.isdir(newfolderpath):

os.makedirs(newfolderpath)

outputfile1= newfolderpath+" "+Blender\_file\_name+ name + "\_q" + (str(q))+".csv"

outputfile2= newfolderpath+" "+Blender\_file\_name+ name + "torsion\_q" + (str(q)) +".csv"

outputfile3= newfolderpath+" "+Blender\_file\_name+" arclength\_w\_q" + (str(q)) +".csv"

writefile=open(outputfile1, 'w')

writefile.write("\n")

writefile.write(Output\_curvature\_torsion\_arclength)

writefile.close()

**##Working script 04112012 after standardized number of point**

**##Step 1 of 3 steps**

##B) Reorientation: set the homologous landmark as first point

CURVE0=bpy.context.active\_object

##Step 4 - Reorientation: set the homologous landmark as first point

##(Parameter= X\_reduction\_of\_point, optional function = to reverse points)

CURVE1=CURVE0.data.splines[0].bezier\_points

name=str(CURVE0.name)

Points\_C1=len(CURVE1)

list\_of\_points\_length = []

for each\_point in list(range(Points\_C1)):

point\_length = ((CURVE1[each\_point].co).length)

list\_of\_points\_length.append(point\_length)

position\_of\_homologous\_point = list\_of\_points\_length.index(min(list\_of\_points\_length))

Verts = []

for first\_phase\_point in list(range(position\_of\_homologous\_point,Points\_C1)):

Verts.append((CURVE1[first\_phase\_point]).co)

for second\_phase\_point in list(range(0,position\_of\_homologous\_point)):

Verts.append((CURVE1[second\_phase\_point]).co)

## reverse orientation

#list1 = list(range(0,position\_of\_homologous\_point))

#list1.reverse()

#for first\_phase\_point in list1:

# Verts.append((CURVE1[first\_phase\_point]).co)

#list2 = list(range(position\_of\_homologous\_point, Points\_C1))

#list2.reverse()

#for second\_phase\_point in list2:

# Verts.append((CURVE1[second\_phase\_point]).co)

**##Step 4 - Reorientation: set the homologous landmark as first point (FOR MESH)**

**##(Parameter= X\_reduction\_of\_point, optional function = to reverse points)**

**Aperture\_outline = bpy.context.active\_object.data**

**Aperture\_outline\_vertices = Aperture\_outline.vertices**

**name=str(Aperture\_outline.name)**

**number\_of\_vertices= len(Aperture\_outline\_vertices)**

**list\_of\_point\_length = []**

**for each\_vertices in list(range(number\_of\_vertices)):**

**edge\_point\_length = ((Aperture\_outline\_vertices [each\_vertices].co).length)**

**list\_of\_point\_length.append(edge\_point\_length)**

**position\_of\_homologous\_point = list\_of\_points\_length.index(min(list\_of\_points\_length))**

**CURVE1=CURVE0.data.splines[0].bezier\_points**

**name=str(CURVE0.name)**

**Points\_C1=len(CURVE1)**

**list\_of\_points\_length = []**

**for each\_point in list(range(Points\_C1)):**

**point\_length = ((CURVE1[each\_point].co).length)**

**list\_of\_points\_length.append(point\_length)**

**position\_of\_homologous\_point = list\_of\_points\_length.index(min(list\_of\_points\_length))**

**Verts = []**

**for first\_phase\_point in list(range(position\_of\_homologous\_point,Points\_C1)):**

**Verts.append((CURVE1[first\_phase\_point]).co)**

**for second\_phase\_point in list(range(0,position\_of\_homologous\_point)):**

**Verts.append((CURVE1[second\_phase\_point]).co)**

**## reverse orientation**

**#list1 = list(range(0,position\_of\_homologous\_point))**

**#list1.reverse()**

**#for first\_phase\_point in list1:**

**# Verts.append((CURVE1[first\_phase\_point]).co)**

**#list2 = list(range(position\_of\_homologous\_point, Points\_C1))**

**#list2.reverse()**

**#for second\_phase\_point in list2:**

**# Verts.append((CURVE1[second\_phase\_point]).co)**

##C) EFA analysis

##Step 7 - calculate EFA analysis from the points of mesh

##(Parameter= no\_of\_harmonics)

x\_coord\_mesh = []

y\_coord\_mesh = []

z\_coord\_mesh = []

no\_of\_points = len(Verts)

for each\_point in range(no\_of\_points):

x\_coord\_mesh.append(Verts[each\_point][0])

y\_coord\_mesh.append(Verts[each\_point][1])

z\_coord\_mesh.append(Verts[each\_point][2])

p=len(x\_coord\_mesh)

no\_of\_harmonics=5

i=0

Dx\_list=[]

Dy\_list=[]

Dz\_list=[]

Dt\_list=[]

firstandlast\_x=x\_coord\_mesh[0]-x\_coord\_mesh[-1]

Dx\_list.append(firstandlast\_x)

firstandlast\_y=y\_coord\_mesh[0]-y\_coord\_mesh[-1]

Dy\_list.append(firstandlast\_y)

firstandlast\_z=z\_coord\_mesh[0]-z\_coord\_mesh[-1]

Dz\_list.append(firstandlast\_z)

firstandlast\_t=sqrt(firstandlast\_x\*\*2 + firstandlast\_y\*\*2 +firstandlast\_z\*\*2)

Dt\_list.append(firstandlast\_t)

while i <= (p-2):

Dx=x\_coord\_mesh[i+1]-x\_coord\_mesh[i]

Dx\_list.append(Dx)

Dy=y\_coord\_mesh[i+1]-y\_coord\_mesh[i]

Dy\_list.append(Dy)

Dz=z\_coord\_mesh[i+1]-z\_coord\_mesh[i]

Dz\_list.append(Dz)

Dt= sqrt(Dx\*\*2 + Dy\*\*2 + Dz\*\*2)

Dt\_list.append(Dt)

i +=1

cumsum\_i = 0

cumsum\_list = []

for each\_Dt in Dt\_list:

if Dt\_list.index((Dt\_list[-1])) != -1:

cumsum\_i += each\_Dt

cumsum\_list.append(cumsum\_i)

t1 = cumsum\_list

t1m1\_temp=[float(0)]

t1m1\_temp1=t1m1\_temp + t1

t1m1=t1m1\_temp1[:-1]

T = sum(Dt\_list)

temp\_output0=[]

list\_of\_points = list(range(p))

list\_of\_harmonic = list(range(1,(no\_of\_harmonics+1)))

harmonics\_an=[]

harmonics\_bn=[]

harmonics\_cn=[]

harmonics\_dn=[]

harmonics\_en=[]

harmonics\_fn=[]

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

a=(Dx\_list[each\_point]/Dt\_list[each\_point]) \* ((cos (2 \* each\_harmonic \* pi \*

(t1[each\_point]) / T)) -(cos (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=a

an = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_an.append(an)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

b=(Dx\_list[each\_point]/Dt\_list[each\_point]) \* ((sin (2 \* each\_harmonic \* pi \*

(t1[each\_point]) / T)) -(sin (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=b

bn = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_bn.append(bn)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

c=(Dy\_list[each\_point]/Dt\_list[each\_point]) \* ((cos (2 \* each\_harmonic \* pi \*

(t1[each\_point]) / T)) -(cos (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=c

cn = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_cn.append(cn)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

d=(Dy\_list[each\_point]/Dt\_list[each\_point]) \* ((sin (2 \* each\_harmonic \* pi \*

(t1[each\_point]) / T)) -(sin (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=d

dn = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_dn.append(dn)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

e=(Dz\_list[each\_point]/Dt\_list[each\_point]) \* ((cos (2 \* each\_harmonic \* pi \*

(t1[each\_point]) / T)) -(cos (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=e

en = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_en.append(en)

for each\_harmonic in list\_of\_harmonic:

temp\_output0=0

for each\_point in list\_of\_points:

f=(Dz\_list[each\_point]/Dt\_list[each\_point]) \* ((sin (2 \* each\_harmonic \* pi \*

(t1[each\_point]) / T)) -(sin (2 \* pi \* each\_harmonic \* (t1m1[each\_point]) / T)))

temp\_output0 +=f

fn = (T/(2 \* pi\*\*2 \* each\_harmonic\*\*2)) \* temp\_output0

harmonics\_fn.append(fn)

temp\_ao=0

for each\_point in list\_of\_points:

temp\_output0=(x\_coord\_mesh[each\_point]) \* ((Dt\_list[each\_point])/T)

temp\_ao +=temp\_output0

ao=2 \* temp\_ao

temp\_co=0

for each\_point in list\_of\_points:

temp\_output0=(y\_coord\_mesh[each\_point]) \* ((Dt\_list[each\_point])/T)

temp\_co +=temp\_output0

co=2 \* temp\_co

temp\_eo=0

for each\_point in list\_of\_points:

temp\_output0=(z\_coord\_mesh[each\_point]) \* ((Dt\_list[each\_point])/T)

temp\_eo +=temp\_output0

eo=2 \* temp\_eo

harmonics\_an

harmonics\_bn

harmonics\_cn

harmonics\_dn

harmonics\_en

harmonics\_fn

ao

co

eo

combine\_output=[harmonics\_an,

harmonics\_bn,harmonics\_cn,harmonics\_dn,harmonics\_en,harmonics\_fn,ao,co,eo]

combine\_output

##D) Inversion of harmonics from Elliptic fourier Analysis to plot mesh

##Step 9 - Inversion of harmonics from Elliptic fourier Analysis to plot mesh

##(Parameter= n(no\_of\_vertices), k(no\_of\_harmonics),)

####Script K - Inversion of harmonics from Elliptic fourier Analysis to plot outline mesh

import math

n=100

k=no\_of\_harmonics

harmonics\_no\_index=list(range(1,k+1))

harmonics\_value\_index=list(range(k))

theta\_list\_of\_points=[]

theta\_list\_index=list(range(n))

pi\_for\_each\_point=2 \* pi / n

for each\_point in list(range(n)):

theta\_of\_each\_points = pi\_for\_each\_point \* each\_point

theta\_list\_of\_points.append(theta\_of\_each\_points)

x\_EFA=[]

y\_EFA=[]

z\_EFA=[]

for each\_point\_index in theta\_list\_index:

temp\_x\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_x=harmonics\_an[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + harmonics\_bn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_x\_list.append(temp\_x)

x\_coordEFA=ao/2 + sum(temp\_x\_list)

x\_EFA.append(x\_coordEFA)

for each\_point\_index in theta\_list\_index:

temp\_y\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_y=harmonics\_cn[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + harmonics\_dn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \*theta\_list\_of\_points[each\_point\_index])

temp\_y\_list.append(temp\_y)

y\_coordEFA=co/2 + sum(temp\_y\_list)

y\_EFA.append(y\_coordEFA)

for each\_point\_index in theta\_list\_index:

temp\_z\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_z=harmonics\_en[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) + harmonics\_fn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_z\_list.append(temp\_z)

z\_coordEFA=eo/2 + sum(temp\_z\_list)

z\_EFA.append(z\_coordEFA)

Verts = []

the\_point\_index = 0

while the\_point\_index <= n-1:

unique\_vertex=(x\_EFA[the\_point\_index],y\_EFA[the\_point\_index],z\_EFA[the\_point\_index])

Verts.append(unique\_vertex)

the\_point\_index +=1

numberofedges=len(Verts)

list\_of\_edges=list(range(numberofedges))

Edges=[]

firstedgestart=0

while firstedgestart <= numberofedges-2:

firstpoint=(list\_of\_edges[firstedgestart])

secondpoint=(list\_of\_edges[(firstedgestart+1)])

unique\_edge=(firstpoint,secondpoint)

Edges.append(unique\_edge)

firstedgestart +=1

lastpoint\_and\_firstpoint=((list\_of\_edges[-1]),(list\_of\_edges[0]))

Edges.append(lastpoint\_and\_firstpoint)

the\_temp\_mesh=bpy.data.meshes.new((name+'\_EFA'))

the\_temp\_mesh.from\_pydata(Verts,Edges,[])

the\_temp\_mesh.update()

the\_th\_object=bpy.data.objects.new((name+'\_EFA'), the\_temp\_mesh)

the\_th\_object.data=the\_temp\_mesh

scene=bpy.context.scene

scene.objects.link(the\_th\_object)

the\_th\_object.select = True

**##Step 2 of 3 steps**

##E) calculate perimeter

#all\_data = [] #silent this after used for the first time

total\_length\_between\_points=0

the\_distance\_between\_two\_points=0

points\_of\_curve = bpy.context.active\_object.data.vertices

no\_of\_points = len(points\_of\_curve)

name = bpy.context.active\_object.name

first\_points=0

while first\_points <= (no\_of\_points-2):

the\_distance\_between\_two\_points=(((points\_of\_curve[(first\_points+1)].co)-

(points\_of\_curve[first\_points].co)).length)

total\_length\_between\_points += the\_distance\_between\_two\_points

first\_points +=1

the\_distance\_between\_1st\_lastpoints=(((points\_of\_curve[0].co)-(points\_of\_curve[-

1].co)).length)

total\_length\_between\_points += the\_distance\_between\_1st\_lastpoints

name

total\_length\_between\_points

data = name,total\_length\_between\_points

all\_data.append(data)

##G) normalized EFA analysis

##Step 1 - import module

import bpy

import math

import os

import array

import mathutils

normalised\_harmonics\_all = []

all\_outline = []

#Output\_raw\_EFA = '' #silent this after used for the first time

#Output\_normal\_EFA = '' #silent this after used for the first time

##Step 8 - normalisation of harmonics from EFA analysis

##(Parameter= k as no\_of\_harmonics; choose normalization factor: scale, O\_inverted, and direction\_of\_motion)

####Script L - Normalization of harmonics from Elliptic fourier Analysis

k=no\_of\_harmonics ##see C) EFA analysis

harmonics\_value\_index=list(range(k))

##Scaling

psi = (1/2) \* atan(2 \* ((harmonics\_an[0] \* harmonics\_bn[0]) + (harmonics\_cn[0] \* harmonics\_dn[0]) + (harmonics\_en[0] \* harmonics\_fn[0])) / (harmonics\_bn[0]\*\*2 + harmonics\_dn[0]\*\*2 + harmonics\_fn[0]\*\*2 - harmonics\_an[0]\*\*2 - harmonics\_cn[0]\*\*2 - harmonics\_en[0]\*\*2))

a = sqrt (((harmonics\_an[0]\*\*2 + harmonics\_cn[0]\*\*2 + harmonics\_en[0]\*\*2) \* cos(psi)\*\*2) + ((harmonics\_bn[0]\*\*2 + harmonics\_dn[0]\*\*2 + harmonics\_fn[0]\*\*2) \* sin(psi)\*\*2) - (((harmonics\_an[0] \* harmonics\_bn[0]) + (harmonics\_cn[0] \* harmonics\_dn[0]) + (harmonics\_en[0] \* harmonics\_fn[0])) \* sin (2 \* psi)))

b = sqrt (((harmonics\_an[0]\*\*2 + harmonics\_cn[0]\*\*2 + harmonics\_en[0]\*\*2) \* sin(psi)\*\*2) + ((harmonics\_bn[0]\*\*2 + harmonics\_dn[0]\*\*2 + harmonics\_fn[0]\*\*2) \* cos(psi)\*\*2) + (((harmonics\_an[0] \* harmonics\_bn[0]) + (harmonics\_cn[0] \* harmonics\_dn[0]) + (harmonics\_en[0] \* harmonics\_fn[0])) \* sin (2 \* psi)))

scale = 1/sqrt(pi \* a \* b)

##Rotation - refered to 1st harmonic

w = a\*b/((harmonics\_an[0]\*harmonics\_fn[0]) - (harmonics\_bn[0]\*harmonics\_en[0]))

O21 = ((harmonics\_cn[0]\*cos(psi)) - (harmonics\_dn[0]\*sin(psi)))/a

O31 = ((harmonics\_en[0]\*cos(psi)) - (harmonics\_fn[0]\*sin(psi)))/a

O22 = ((harmonics\_cn[0]\*sin(psi)) + (harmonics\_dn[0]\*cos(psi)))/b

O32 = ((harmonics\_en[0]\*sin(psi)) + (harmonics\_fn[0]\*cos(psi)))/b

alpha = ()

if ((harmonics\_an[0]\*harmonics\_fn[0]) - (harmonics\_bn[0]\*harmonics\_en[0])) > 0:

alpha = atan(((harmonics\_cn[0]\*harmonics\_fn[0])-(harmonics\_dn[0]\*harmonics\_en[0]))/

((harmonics\_an[0]\*harmonics\_fn[0]) - (harmonics\_bn[0]\*harmonics\_en[0])))

else:

alpha =(atan(((harmonics\_cn[0]\*harmonics\_fn[0])-(harmonics\_dn[0]\*harmonics\_en[0]))/((harmonics\_an[0]\*harmonics\_fn[0]) - (harmonics\_bn[0]\*harmonics\_en[0])))) + pi

beta = acos(w\*((O21\*O31)+(O22\*O32)))

gamma = ()

if O31 > 0:

gamma = acos(O32/sin(beta))

else:

gamma = -acos(O32/sin(beta))

RX\_alpha = Matrix (((1,0,0),(0, cos(alpha),-sin(alpha)),(0,sin(alpha),cos(alpha))))

RX\_beta = Matrix (((1,0,0),(0, cos(beta),-sin(beta)),(0,sin(beta),cos(beta))))

RX\_gamma = Matrix (((1,0,0),(0, cos(gamma),-sin(gamma)),(0,sin(gamma),cos(gamma))))

RY\_alpha = Matrix (((cos(alpha),0,sin(alpha)),(0,1,0),(-sin(alpha),0,cos(alpha))))

RY\_beta = Matrix (((cos(beta),0,sin(beta)),(0,1,0),(-sin(beta),0,cos(beta))))

RY\_gamma = Matrix (((cos(gamma),0,sin(gamma)),(0,1,0),(-sin(gamma),0,cos(gamma))))

RZ\_alpha = Matrix (((cos(alpha),-sin(alpha),0),(sin(alpha), cos(alpha),0),(0,0,1)))

RZ\_beta = Matrix (((cos(beta),-sin(beta),0),(sin(beta), cos(beta),0),(0,0,1)))

RZ\_gamma = Matrix (((cos(gamma),-sin(gamma),0),(sin(gamma), cos(gamma),0),(0,0,1)))

##O=RX\_alpha \* RY\_beta \* RZ\_gamma ##x1y2z3

##O=RY\_alpha \* RX\_beta \* RY\_gamma ##y1x2y3

O=RZ\_alpha \* RX\_beta \* RZ\_gamma ##z1x2z3

##O = Matrix(((((cos(alpha)\*cos(gamma)) - (sin(alpha)\*cos(beta)\*sin(gamma))), ((-cos(alpha)\*sin(gamma)) - (sin(alpha)\*cos(beta)\*cos(gamma))), (sin(alpha)\*sin(beta))),(((sin(alpha)\*cos(gamma)) + (cos(alpha)\*cos(beta)\*sin(gamma))), ((-sin(alpha)\*sin(gamma)) - (cos(alpha)\*cos(beta)\*cos(gamma))), (-cos(alpha)\*sin(beta))),((sin(beta)\*sin(gamma)), (sin(beta)\*cos(gamma)),(cos(beta)))))

O\_inverted=O.inverted()

direction\_of\_motion = Matrix(((cos(psi),sin(psi)),((-sin(psi),cos(psi)))))

normalization\_factors = scale\*O\_inverted

##convert each haromonic

normalised\_harmonics\_an = []

normalised\_harmonics\_bn = []

normalised\_harmonics\_cn = []

normalised\_harmonics\_dn = []

normalised\_harmonics\_en = []

normalised\_harmonics\_fn = []

##normalised\_harmonics\_all = []

for each\_harmonic in harmonics\_value\_index:

the\_th\_harmonic\_matrix = Matrix(((harmonics\_an[each\_harmonic],harmonics\_bn[each\_harmonic]),(harmonics\_cn[each\_harmonic],harmonics\_dn[each\_harmonic]),(harmonics\_en[each\_harmonic],harmonics\_fn[each\_harmonic])))

the\_normalised\_th\_harmonic=normalization\_factors\*the\_th\_harmonic\_matrix\*direction\_of\_motion

normalised\_harmonics\_an.append((the\_normalised\_th\_harmonic[0])[0])

normalised\_harmonics\_bn.append((the\_normalised\_th\_harmonic[0])[1])

normalised\_harmonics\_cn.append((the\_normalised\_th\_harmonic[1])[0])

normalised\_harmonics\_dn.append((the\_normalised\_th\_harmonic[1])[1])

normalised\_harmonics\_en.append((the\_normalised\_th\_harmonic[2])[0])

normalised\_harmonics\_fn.append((the\_normalised\_th\_harmonic[2])[1])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[0])[0])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[0])[1])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[1])[0])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[1])[1])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[2])[0])

normalised\_harmonics\_all.append((the\_normalised\_th\_harmonic[2])[1])

combine\_output\_N=[normalised\_harmonics\_an, normalised\_harmonics\_bn,

normalised\_harmonics\_cn, normalised\_harmonics\_dn, normalised\_harmonics\_en,

normalised\_harmonics\_fn,ao,co,eo]

scale

combine\_output\_N

all\_outline.append(name)

normalised\_harmonics\_all

all\_outline

##H) Inversed Normalized EFA outline

##Step 10 - Inversion of harmonics from Normalised EFA harmonics to plot mesh

##(Parameter= n(no\_of\_vertices), k(no\_of\_harmonics),)

####Script M - Inversion of normalized harmonics from Elliptic fourier Analysis to plot

outline mesh

n=500

k=no\_of\_harmonics##see C) EFA analysis & G) normalized EFA

harmonics\_no\_index=list(range(1,k+1))

harmonics\_value\_index=list(range(k))

theta\_list\_of\_points=[]

theta\_list\_index=list(range(n))

pi\_for\_each\_point=2 \* pi / n

for each\_point in list(range(n)):

theta\_of\_each\_points = pi\_for\_each\_point \* each\_point

theta\_list\_of\_points.append(theta\_of\_each\_points)

x\_NEFA=[]

y\_NEFA=[]

z\_NEFA=[]

for each\_point\_index in theta\_list\_index:

temp\_x\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_x=normalised\_harmonics\_an[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) +normalised\_harmonics\_bn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_x\_list.append(temp\_x)

x\_coordNEFA=ao/2 + sum(temp\_x\_list)

x\_NEFA.append(x\_coordNEFA)

for each\_point\_index in theta\_list\_index:

temp\_y\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_y=normalised\_harmonics\_cn[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) +normalised\_harmonics\_dn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_y\_list.append(temp\_y)

y\_coordNEFA=co/2 + sum(temp\_y\_list)

y\_NEFA.append(y\_coordNEFA)

for each\_point\_index in theta\_list\_index:

temp\_z\_list=[]

for each\_harmonics\_index in harmonics\_value\_index:

temp\_z=normalised\_harmonics\_en[each\_harmonics\_index] \* cos(harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index]) +normalised\_harmonics\_fn[each\_harmonics\_index] \* sin (harmonics\_no\_index[each\_harmonics\_index] \* theta\_list\_of\_points[each\_point\_index])

temp\_z\_list.append(temp\_z)

z\_coordNEFA=eo/2 + sum(temp\_z\_list)

z\_NEFA.append(z\_coordNEFA)

Verts = []

the\_point\_index = 0

while the\_point\_index <= n-1:

unique\_vertex=(x\_NEFA[the\_point\_index],y\_NEFA[the\_point\_index],z\_NEFA

[the\_point\_index])

Verts.append(unique\_vertex)

the\_point\_index +=1

numberofedges=len(Verts)

list\_of\_edges=list(range(numberofedges))

Edges=[]

firstedgestart=0

while firstedgestart <= numberofedges-2:

firstpoint=(list\_of\_edges[firstedgestart])

secondpoint=(list\_of\_edges[(firstedgestart+1)])

unique\_edge=(firstpoint,secondpoint)

Edges.append(unique\_edge)

firstedgestart +=1

lastpoint\_and\_firstpoint=((list\_of\_edges[-1]),(list\_of\_edges[0]))

Edges.append(lastpoint\_and\_firstpoint)

the\_temp\_mesh=bpy.data.meshes.new((name+'\_NEFA'))

the\_temp\_mesh.from\_pydata(Verts,Edges,[])

the\_temp\_mesh.update()

the\_th\_object=bpy.data.objects.new((name+'\_NEFA'), the\_temp\_mesh)

the\_th\_object.data=the\_temp\_mesh

scene=bpy.context.scene

scene.objects.link(the\_th\_object)

the\_th\_object.select = True

##I) save output from EFA and Normalized EFA output – C) & G)

##Step 11 - compiling output data from all harmonics from all outlines of each specimen

str\_raw\_EFA = (((str(combine\_output)).replace('[','')).replace(']','')+"\n")

Output\_raw\_EFA += name + ',' + str\_raw\_EFA

str\_normal\_EFA = (((str(combine\_output\_N)).replace('[','')).replace(']','')+","+str

(scale)+"\n")

Output\_normal\_EFA += name + ',' + str\_normal\_EFA

normalised\_harmonics\_all

len(normalised\_harmonics\_all)

all\_outline

**##Step 3 of 3 steps after finished step 1 and 2 for all outlines**

##F) save and export perimeter data for all outline

Output1 = str(all\_data)

Output2 = Output1.replace("), (","\n")

Output3 = Output2.replace("'","")

Output4 = Output3.replace("[(","")

Output = Output4.replace(")]","")

import os

##Step 12 - write output file in csv format

##Parameters(newfolderpath)

fp=bpy.data.filepath

filepath=os.path.basename(fp)

Blender\_file\_name=filepath[:-6]

####newfolderpath="c:/E\_19072009/Manuscript/PhD thesis/On growth and form of two heteromorphic terrestrial gastropod snails/3D aperture outline analysis/EFA\_blender/"

newfolderpath="c:/E\_19072009/abc/"

if not os.path.isdir(newfolderpath):

os.makedirs(newfolderpath)

outputfile= newfolderpath+" "+Blender\_file\_name+" abc\_new\_04112012"+".csv"

writefile=open(outputfile, 'w')

writefile.write(Output)

writefile.close()

##Step 12 - write output file in csv format

##Parameters(newfolderpath)

fp=bpy.data.filepath

filepath=os.path.basename(fp)

Blender\_file\_name=filepath[:-6]

####newfolderpath="c:/E\_19072009/Manuscript/PhD thesis/On growth and form of two heteromorphic terrestrial gastropod snails/3D aperture outline analysis/EFA\_blender/"

newfolderpath="c:/E\_19072009/abc/"

if not os.path.isdir(newfolderpath):

os.makedirs(newfolderpath)

outputfile1= newfolderpath+" "+Blender\_file\_name+" raw\_EFA\_abc04112012"+".csv"

outputfile2= newfolderpath+" "+Blender\_file\_name+" normal\_EFA\_abc04112012"+".csv"

writefile=open(outputfile1, 'w')

writefile.write("\n")

writefile.write(Output\_raw\_EFA)

writefile.close()

writefile=open(outputfile2, 'w')

writefile.write("\n")

writefile.write(Output\_normal\_EFA)

writefile.close()