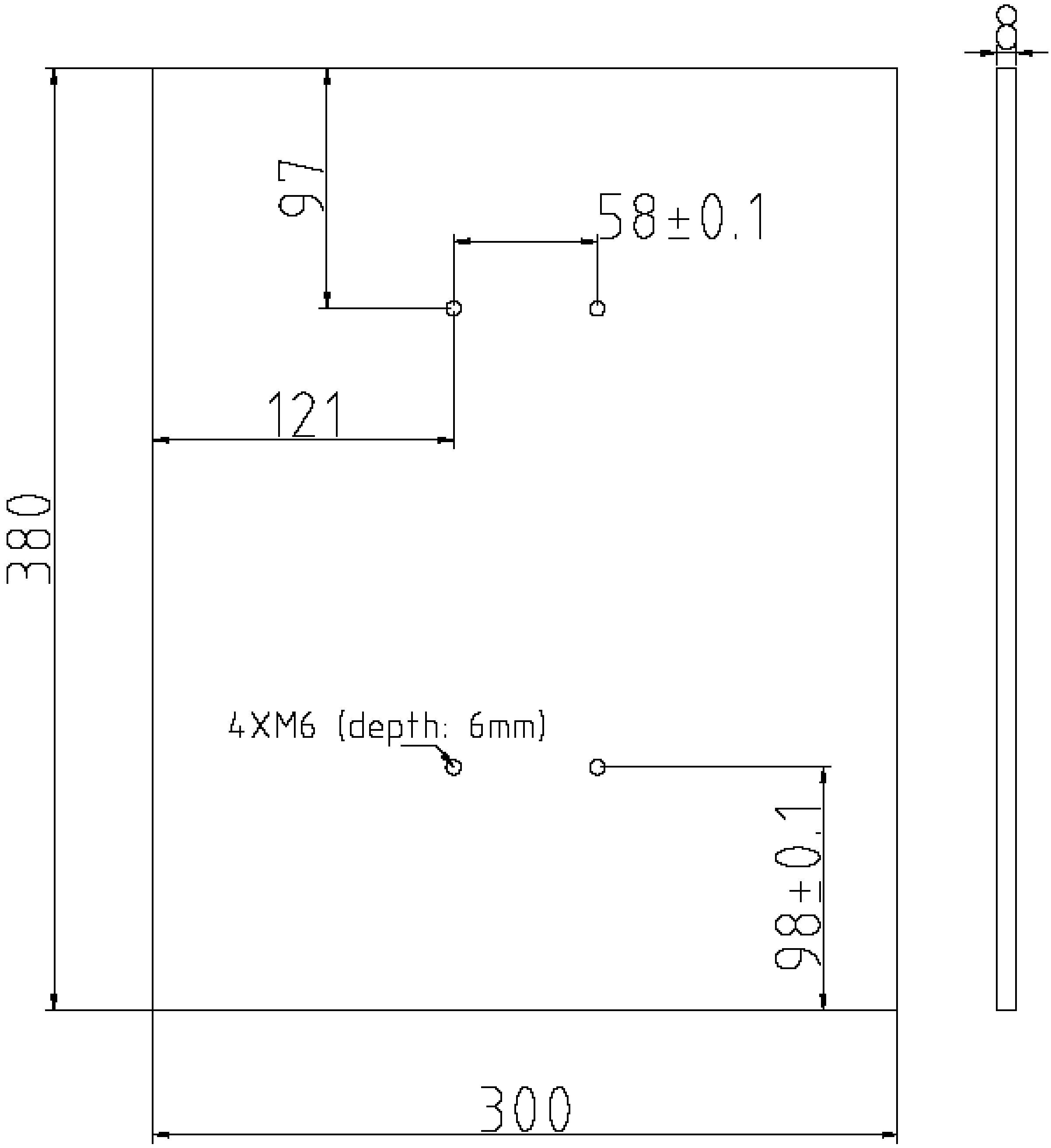
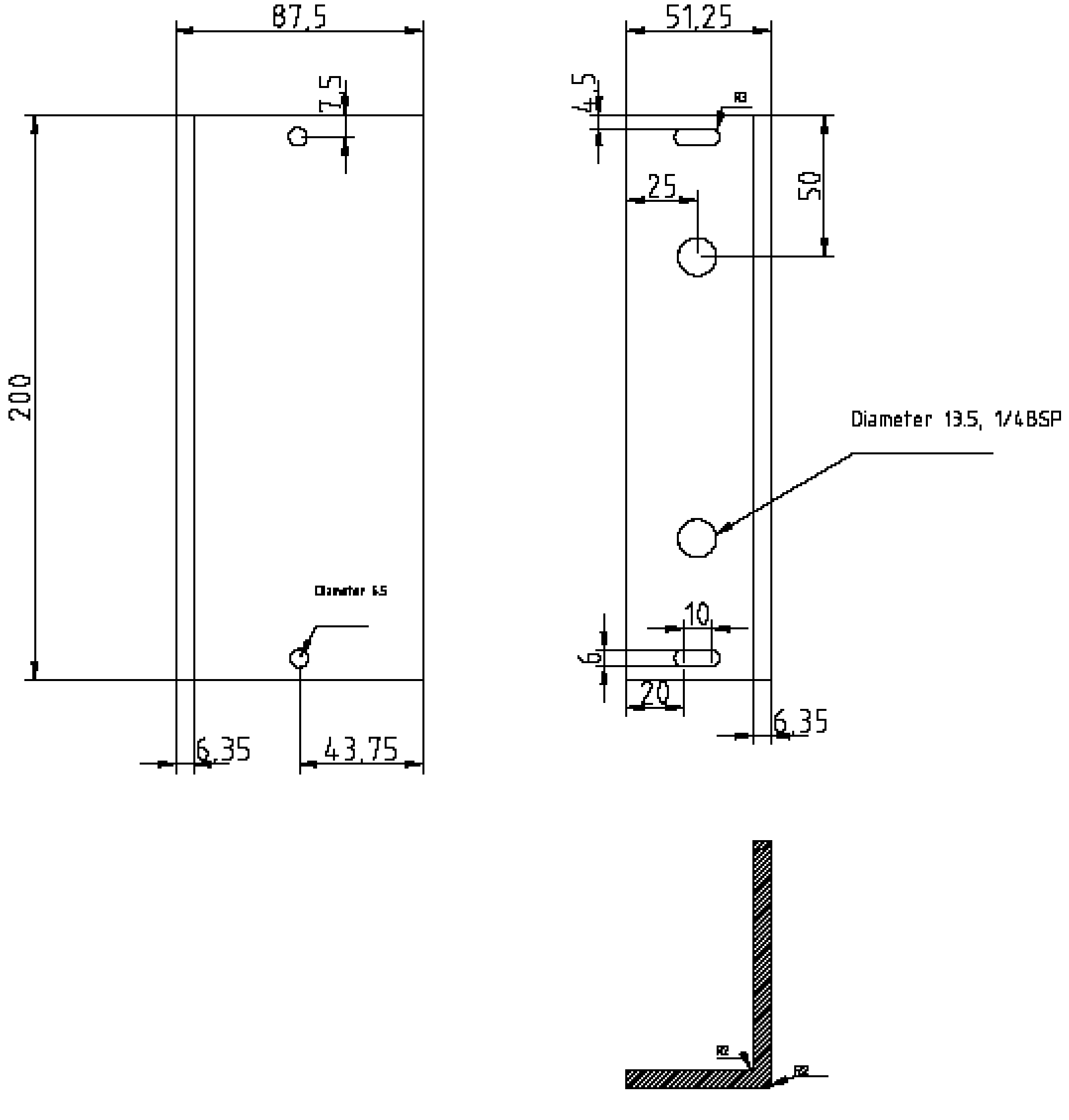
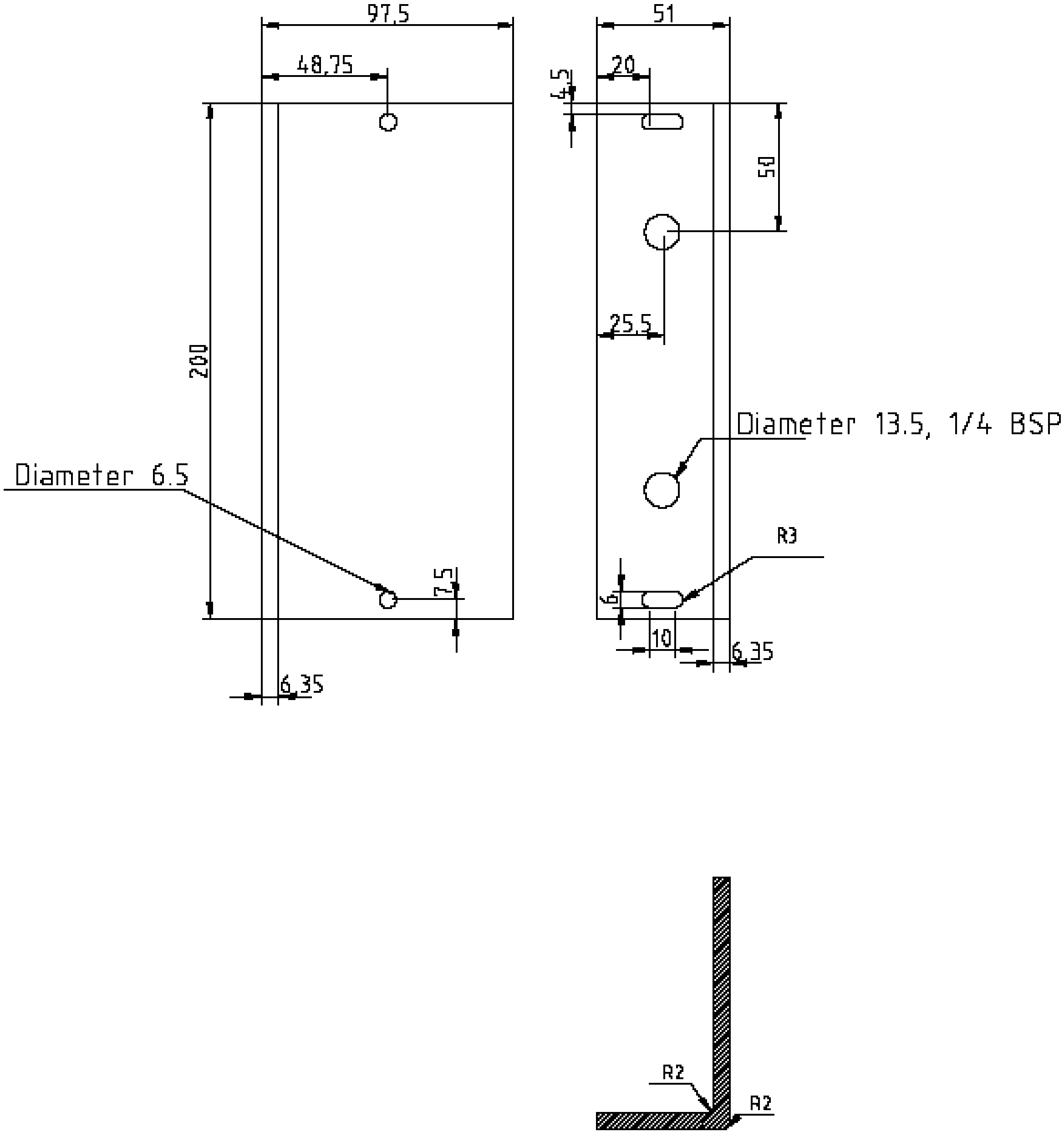
Appendix A

Containing drawings for T-Joint mould tool. In order: Base plate, L plate 1, L plate 2, base spacer 1, base spacer 2, spacer 1, spacer 2 and cap.

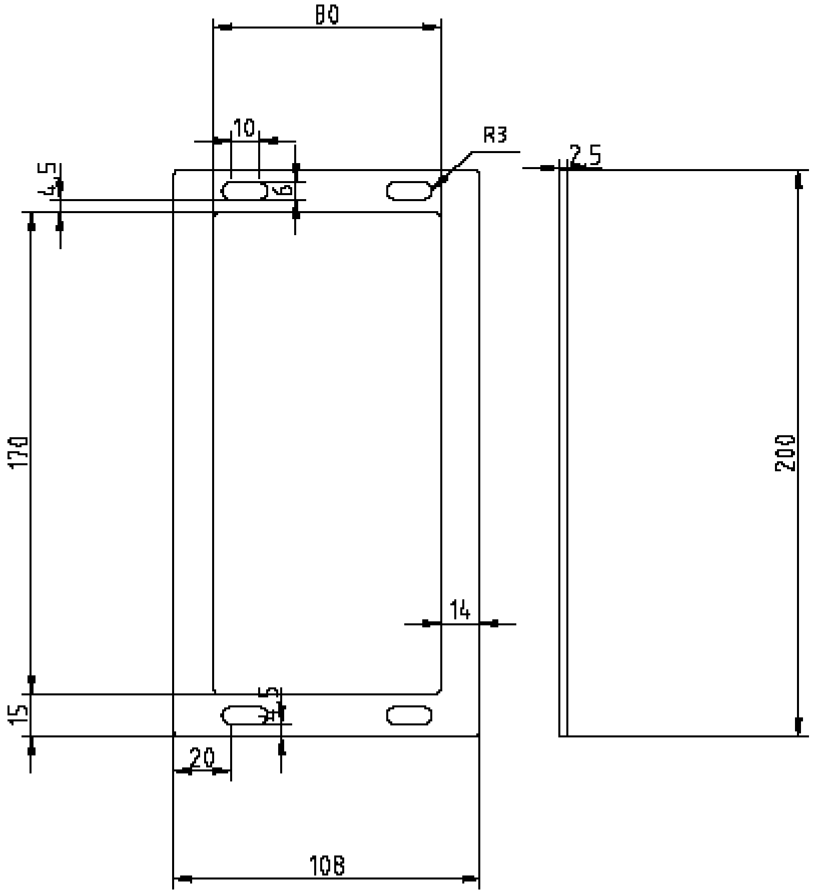


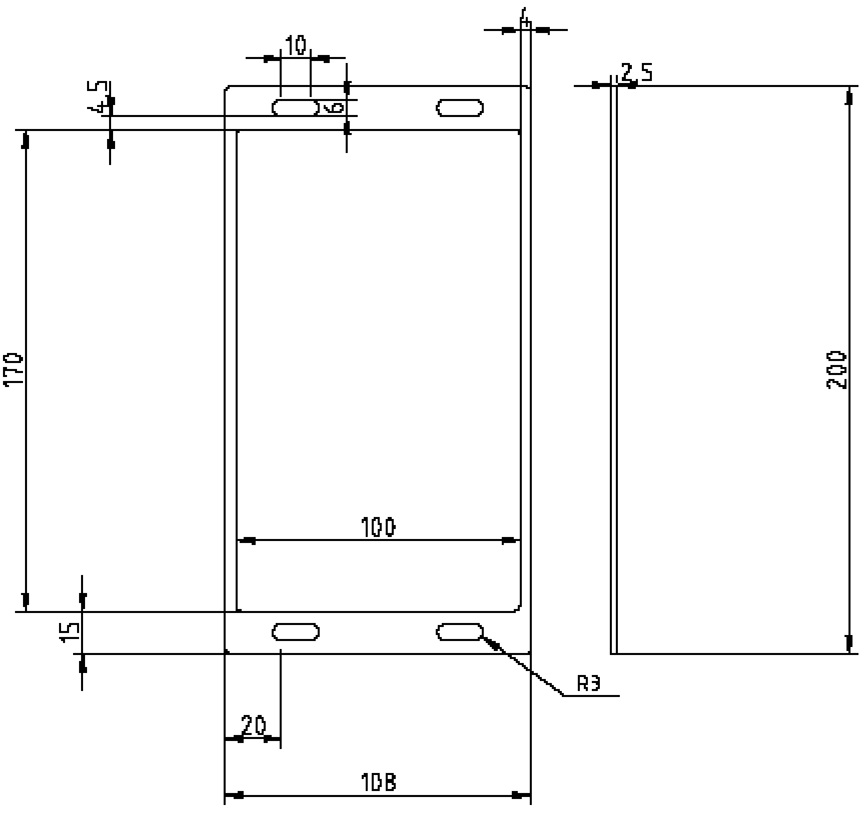


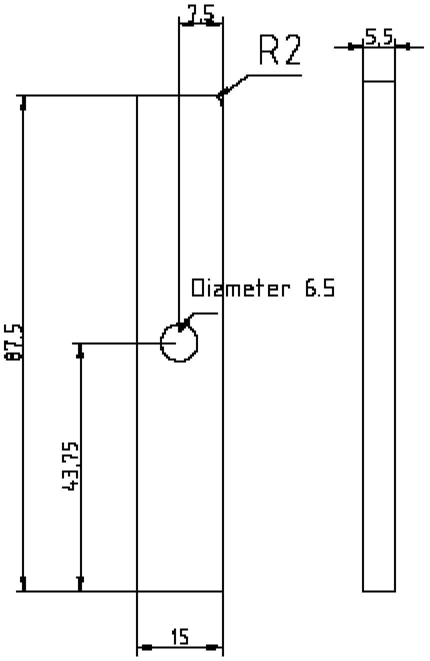
87,5
7,5
200
Diameter 6.5
6,35
43,75
51,25
4,5
R3
50
25
Diameter 13.5, 1/4BSP
10
6
20
6,35
R2
R2

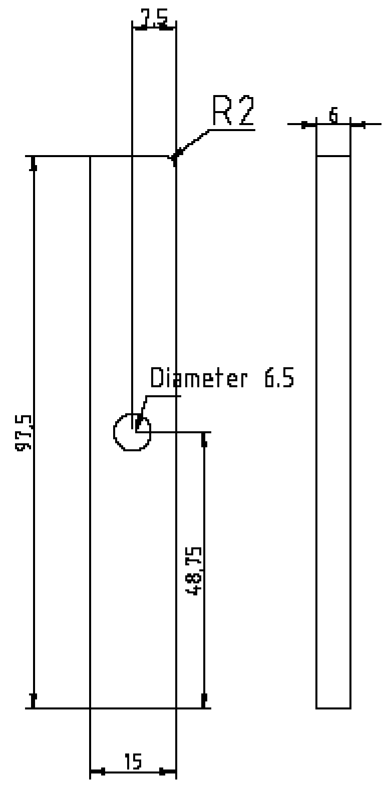


97,5
48,75
200
Diameter 6.5
7.5
6,35
51
20
4.5
50
25.5
Diameter 13.5, 1/4 BSP
R3
6
10
6,35
R2
R2


80
10
R3
2.5
4.5
6
170
200
14
15
4.5
20
108


4
10
4.5
6
2.5
170
200
100
15
R3
20
108


7.5
R2
5.5
Diameter 6.5
87.5
43.75
15


7,5
R2
6
Diameter 6.5
97,5
48,75
15

Appendix B

This appendix contains the Python script to generate the flat woven models in Chapter 4.

```
import os
import sys
sys.path.append('F:\\')
sys.path.append('C:\\SIMULIA\\CAE\2018\\win_b64\\tools\\SMApy\\python2.7\\Lib\\site-packages')
from TexGen.Core import *
import imp
import math

NumXYarns = 6
NumYYarns = 6
NumWeftLayers = 4
NumWarpLayers = NumWeftLayers - 1
XSpacing = 3.8
YSpacing = 3.8
WarpHeight = 0.5
WeftHeight = 0.5
BinderHeight = 0.4
WarpRatio = 3
BinderRatio = 2
WarpWidth = 3
WeftWidth = 3
BinderWidth = 1.5
WarpYarnPower = 0.6
```

```
WeftYarnPower = 0.6
BinderYarnPower = 0.8


def CopyBinderYarns():
    Textile=GetTextile()
    weave3D=Textile.Get3DWeave()
    NumXYarns=weave3D.GetNumXYarns()
    NumYarns=Textile.GetNumYarns()
    ##zero index below for Textiles with binder not at edge of Textile
    NumWarpLayers=weave3D.GetNumXLayers(0)
    NumBindersCounted=0
    for i in range(NumXYarns):
        Binder=weave3D.IsBinderYarn(i)
        if Binder:
            YarnIndex=((i-NumBindersCounted)*NumWarpLayers)+ NumBindersCounted
            NumBindersCounted += 1
            BinderYarn=Textile.GetYarn(YarnIndex)
            Nodes=BinderYarn.GetMasterNodes()
            NumNodes=BinderYarn.GetNumNodes()
            CopiedYarn=CYarn()
            ZPos=[]
            for i in range(NumNodes):
                NodePosition=Nodes[i].GetPosition()
                ZPos.append(NodePosition.z)
            Average=(max(ZPos)+min(ZPos))/2
            #copy and transform the nodes
            for i in range(NumNodes):
                NodePosition=Nodes[i].GetPosition()
                UpVector=Nodes[i].GetUp()
                Thickness = WarpHeight*NumWeftLayers + WeftHeight*NumWeftLayers

                NewNodeZPos= (2*Average) - NodePosition.z - (Thickness*0.5)

                NewNode=CNode(XYZ(NodePosition.x, NodePosition.y, NewNodeZPos))

                NewNode.SetUp(UpVector)

                CopiedYarn.AddNode(NewNode)




            CopiedYarnIndex=NumXYarns + NumYYarns + NumBindersCounted



            #add sections and interpolation

            Section=CYarnSectionConstant(CSectionPowerEllipse(BinderWidth, BinderHeight, BinderYarnPower))

            CopiedYarn.AssignSection( Section )

            CopiedYarn.AssignInterpolation( CInterpolationBezier() )

            CopiedYarn.SetResolution(30)

            CopiedYarn.AddRepeat(XYZ(12, 0, 0 ))

            CopiedYarn.AddRepeat(XYZ(0, 20, 0 ))

            Textile.AddYarn(CopiedYarn)


    AddTextile(Textile)


    return



def CrossProduct(u,v):

    dim = len(u)

    s = []
    for i in range(dim):
        if i == 0:

            s.append(u.y*v.z - u.z*v.y)
        elif i == 1:

            s.append(-u.x*v.z + u.z*v.x)
        else:

            s.append(u.x*v.y - u.y*v.x)
    return s




def AbsoluteMagnitude(u):
    var= u.x**2 + u.y**2 + u.z**2
    mag=math.sqrt(var)

    return mag




#open file to read in paramters
cwd=os.getcwd()
file = open('f:\\parameter.dat', 'r')

#open file to read in paramters
allLines=file.readlines()
```

```
print(allLines)
lastLine=allLines[-1]



#read last line, splitting the string based on whitespace delimiter
x = lastLine
parameter = x.split()
nbl=int(parameter[0])

path1cell_offset=IntVector()
for i in range(1,7):
    if nbl == 2:
        if int(parameter[i]) == 4:
            parameter[i] = int(parameter[i]) - 1
    path1cell_offset.push_back( int(parameter[i]) )

path2cell_offset = IntVector()
for i in range(7,13):
    if nbl == 2:
        if int(parameter[i]) == 4:
            parameter[i] = int(parameter[i]) - 1
    path2cell_offset.push_back( int(parameter[i]) )

path3cell_offset = IntVector()
for i in range(13,19):
    if nbl == 2:
        if int(parameter[i]) == 4:
            parameter[i] = int(parameter[i]) - 1
    path3cell_offset.push_back( int(parameter[i]) )
```

```
#create orthogonal 3D weave using generic base class, 6 x yarns, 4 wefts
#spacing of 3.2 and heights of 0.35 and 0.25 for warp and weft respectively
Textile = CTextileLayerToLayer(NumXYarns, NumYYarns, XSpacing, YSpacing, WarpHeight, WeftHeight, nbl)


NumBinders = 2
bpattern = BoolVector([False, True, False, True, False, True]) ###, False, True, False, True, False, False])
Textile.SetBinderPattern(bpattern)


#need to set yarn widths, height and spacing for Textile
#need to set up binder pattern before widths etc. otherwise will assign incorrect dimensions to yarns
Textile.SetWarpYarnWidths(WarpWidth)
Textile.SetBinderYarnWidths(BinderWidth)
Textile.SetupLayers(NumWarpLayers, NumWeftLayers, 1)


Textile.SetYYarnWidths(WeftWidth)


Textile.SetWarpYarnHeights(WarpHeight)
Textile.SetYYarnHeights(WeftHeight)
Textile.SetBinderYarnHeights(BinderHeight)
Textile.SetXYarnSpacings(XSpacing)
Textile.SetYYarnSpacings(YSpacing)
Textile.SetWarpYarnPower(WarpYarnPower)
Textile.SetWeftYarnPower(WeftYarnPower)
Textile.SetBinderYarnPower(BinderYarnPower)



#y positions based on above vector, for binder yarns after bifurcation add a reflected version
for i in range(6):
    Textile.SetBinderPosition(i, 1, path1cell_offset[i])

for i in range(6):
    Textile.SetBinderPosition(i, 3, path2cell_offset[i])

for i in range(6):
    Textile.SetBinderPosition(i, 5, path3cell_offset[i])




#set the material properties

Yarns=Textile.GetYarns()

for index in range(len(Yarns)):
    Yarns[index].SetYoungsModulusX(174.4, 'GPa')
    Yarns[index].SetYoungsModulusY(8.9, 'GPa')
    Yarns[index].SetYoungsModulusZ(8.9, 'GPa')
    Yarns[index].SetShearModulusXY(4.2, 'GPa')
    Yarns[index].SetShearModulusXZ(4.2, 'GPa')
    Yarns[index].SetShearModulusYZ(3, 'GPa')
    Yarns[index].SetPoissonsRatioX(0.3)
    Yarns[index].SetPoissonsRatioY(0.3)
    Yarns[index].SetPoissonsRatioZ(0.3)
    Yarns[index].SetAlphaX(5.4)
    Yarns[index].SetAlphaY(5.4)
    Yarns[index].SetAlphaZ(5.4)
    print("material props set")
```

```
# Matrix material properties
Textile.SetMatrixYoungsModulus(3.5, 'GPa')
Textile.SetMatrixPoissonsRatio(0.35)
Textile.SetMatrixAlpha(52.7e-6)

Textile.SetFibreDiameter(WARP, 0.007, "mm")
Textile.SetFibreDiameter(WEFT, 0.007, "mm")
Textile.SetFibreDiameter(BINDER, 0.007, "mm")
Textile.SetFibresPerYarn(WARP, 5000)
Textile.SetFibresPerYarn(WEFT, 8000)
Textile.SetFibresPerYarn(BINDER, 3500)

# Textile.BuildTextile()

# Textile.SetMaxVolFraction(0.78)



Thickness=NumWarpLayers*WarpHeight + NumWeftLayers*WeftHeight + BinderHeight

#create custom domain planes
domain = CDomainPlanes()
domain.AddPlane(PLANE(XYZ(-1, 0, 0), -NumYYarns*YSpacing))
domain.AddPlane(PLANE(XYZ(1, 0, 0), -0.1*WeftWidth))
domain.AddPlane(PLANE(XYZ(0, 1, 0), -0.1*WarpWidth))
domain.AddPlane(PLANE(XYZ(0, -1, 0), -NumXYarns*WarpWidth - 0.5*NumXYarns*BinderWidth))
domain.AddPlane(PLANE(XYZ(0, 0, 1), -BinderHeight - 0.1*BinderHeight))
domain.AddPlane(PLANE(XYZ(0, 0, -1), -(Thickness + BinderHeight) + 0.9*BinderHeight))


Textile.AssignDomain( domain )
#Textile.AssignDefaultDomain()



AddTextile(Textile)

from CheckBinderPaths import *
a=CheckBinderPaths3NoBifurcation(planepos=3, nbl=1)
b=CheckBinderPaths1NoBifurcation(nly=4, nbl=1)
if (a != 0 or b != 0):
    file=open('f:\\fitfun.dat', 'a')
    sum=((a*100) + (b*100))
    file.write(str(sum) + ' \n')
    file.close()

else:

    width = -(-0.1*WarpWidth + (-NumXYarns*WarpWidth - 0.5*NumXYarns*BinderWidth))
    length = -(-NumYYarns*YSpacing + (-0.1*WeftWidth))
    height = -(-BinderHeight - 0.1*BinderHeight + (-(Thickness + BinderHeight) + 0.9*BinderHeight))



    NumXVoxels=140
```

```
VoxelSize = length/NumXVoxels
NumYVoxels = int(width/(VoxelSize))
NumZVoxels = int(height/(VoxelSize))
volume=length*height*width
ModelName= "weave_" + "_" + str(NumXVoxels)

#SaveToXML(ModelName+".tg3", "Textile", OUTPUT_STANDARD)

cwd = os.getcwd()
FileName=ModelName + '.inp'


t=GetTextile()
rv=CRectangularVoxelMesh("CPeriodicBoundaries")
rv.SaveVoxelMesh(t, FileName, NumXVoxels, NumYVoxels, NumZVoxels*2, True, True, MATERIAL_CONTINUUM, 1)

from SubmitJobElastic import *
SubmitJob(4,4, str(ModelName))
```

Appendix C

Python script to rule out unfeasible models in the optimisation and apply a penalty value.

```
import sys
sys.path.append("C:\SIMULIA\CAE\2018\win_b64\tools\SMApy\python2.7\Lib\site-packages")
from TexGen.Core import *

class BinderFunctions:

    """
    Binder functions class, based off TexGen code by Louise Brown
    """

    def GetXYarnIndex(self, iIndex):
        textile=GetTextile()
        weave3D=textile.Get3DWeave()
        NumXYarns=weave3D.GetNumXYarns()
        for k in range(NumXYarns):
            if k==iIndex:
                return k
        return -1

    def GetBinderOffsets(self, x, y):
        textile=GetTextile()
        weave3D=textile.Get3DWeave()
        vector1=weave3D.GetCell(x, y)
        #print('vector 1', vector1)
        TopBinder=self.FindTopBinderYarns(vector1)
        offset=((len(vector1)-1)-TopBinder)/2
        return offset


    def FindTopBinderYarns(self, vector1):
        #print('vector1 here', vector1)
        i=len(vector1)-1
        while i>0:
            if vector1[i]==1:
                return i
            i=i-1
        return i




def CheckBinderPaths3NoBifurcation(planepos, nbl):
## This function checks that binder yarns cross every internal plane between weft layers
##
    #get the textile and relevant types with their methods
    textile=GetTextile()
    weave3D=textile.Get3DWeave()
    layertolayer=textile.GetLayerToLayerWeave()
    #to get number of x and y yarns use weave3D
    NumXYarns=weave3D.GetNumXYarns()
    NumWeftStacks=weave3D.GetNumYYarns()
    binders=[]
    IY=[]



    #look at XYarns only
    for i in range(NumXYarns):
```

```
    binder=weave3D.IsBinderYarn(i)
    if binder:
        binderyarn=textile.GetYarn(i)
        nodes=binderyarn.GetMasterNodes()
        for node in nodes:
            nodepos=node.GetPosition()
            nodey=nodepos.y
            print nodey
        #check this gets the correct yarn by printing out the nodes
        binders.append(binderyarn)
        #instantiate binder functions object
        BF=BinderFunctions()
        YPosition=BF.GetXYarnIndex(i)
        IY.append(YPosition)
c=0
#get the binder offsets
#below would iterate through list of all the iy positions generated above
for iy in IY:
    offsets=[]
    for ix in range(NumWeftStacks):
        offset=BF.GetBinderOffsets(ix, iy)
        offsets.append(offset)
        if nbl>=1:
            offsets.append(offset+(nbl-1)) #if 3 binders needs to be +2 etc.
    a=0
    b=0
    for value in offsets:
        #remember 0 at top of weft stack so may seem backwards
        if value > planepos:
            a=a+1
            elif value < planepos:
                b=b+1
        if a>=1 and b>=1: #one of offsets for yarn above and another below plane
            c=c+1
        else:
            c=c
    z=planepos-1
    if z>=1:
        if c>=1:
            #Recursively call the function, raising the plane position until the algorithm reaches the top of the stack
            return CheckBinderPaths3NoBifurcation(planepos-1, nbl)
        #if not satisfied for plane, exit the recursion and apply penalty
        else:
            return 1
    #check last plane
    else:
        if c>=1:
            return 0
        else:
            return 1
```

```
def CheckBinderPaths3WarpBifurcation(planepos, bifstart, bifplane):
    #bifurcation along the warp direction, want to penalise weaves that don't satisfy con 3 before bifstart and those that do after for a certain plane
```

```
textile=GetTextile()
weave3D=textile.Get3DWeave()
layertolayer=textile.GetLayerToLayerWeave()
#to get number of x and y yarns use weave3D
NumXYarns=weave3D.GetNumXYarns()
NumWeftStacks=weave3D.GetNumYYarns()
binders=[]
IY=[]


for i in range(NumXYarns):
    binder=weave3D.IsBinderYarn(i)
    if binder:
        binderyarn=textile.GetYarn(i)
        binders.append(binderyarn)
        BF=BinderFunctions()
        YPosition=BF.GetXYarnIndex(i)
        IY.append(YPosition)
j=0
k=0   #j, k after bifurcation
c=0   #before bifurcation
#get the binder offsets
#below would iterate through list of all the iy positions generated above
for iy in IY:
    offsets=[]



    #perform checks that all planes are crossed before bifurcation
    for ix in range(bifstart):
        offset=BF.GetBinderOffsets(ix, iy)
        offsets.append(offset)
    a=0
    b=0
    for value in offsets:
        #remember 0 at top of weft stack so may seem backwards
        if value > planepos:
            a=a+1
        elif value < planepos:
            b=b+1
    if a>=1 and b>=1: #yarn crosses plane
        c=c+1
    else: #yarn does not cross plane
        c=c


    #perform check that a yarn does not cross plane after bif, if planepos==bifplane
    bifoffsets=[]
    for ix in range(bifstart, NumWeftStacks, 1):
        offset=BF.GetBinderOffsets(ix, iy)
        bifoffsets.append(offset)
    g=0
    h=0
    for value in bifoffsets:
        if value > planepos:
            g=g+1
        elif value < planepos:
            h=h+1
    if planepos==bifplane:
        if g<1 and h<1: #means binder does not cross bifurcation plane - good
            j=j+1
        else:           #means binder does cross bifplane - raise penalty value
            j=j
    else:
        if g>=1 and h>=1: #binder must cross plane - raise penalty value
            k=k+1
        else:
            k=k




# should have increases or decreases in c, j, k if constraint 3 violated or if yarn crosses at bifurcation
print('The outcome of CBP3 is:')
#check this tomorrow
z=planepos-1
if z>=1:
    if c>=1: #textile before bifplane is fully bound
        if 'k' in locals() and k>=1:
            #Recursively call the function, raising the plane position until the algorithm reaches the top of the stack
            return CheckBinderPaths3(planepos-1, bifstart, bifplane, bifurcation=True)
        elif 'j' in locals() and j>=1:
            return CheckBinderPaths3(planepos-1, bifstart, bifplane, bifurcation=True)
        elif 'k' in locals() and k<1:
            return 1
        elif 'j' in locals() and k<1:
            return 5
    #if not satisfied for plane, move to next plane for now
    elif c<1: #textile before plane not fully bound
        if 'k' in locals() and k>=1:
            #Recursively call the function, raising the plane position until the algorithm reaches the top of the stack
            return 1
        elif 'j' in locals() and j>=1:
            return 1
        elif 'k' in locals() and k<1:
            return 3
        elif 'j' in locals() and k<1:
            return 8


#check last plane
else:
    if c>=1:
        if 'k' in locals() and k>=1:
            #Recursively call the function, raising the plane position until the algorithm reaches the top of the stack
            return 0
        elif 'j' in locals() and j>=1:
            return 0
        elif 'k' in locals() and k<=1:
            return 1
        elif 'j' in locals() and k<1:
            return 5
        #if not satisfied for plane, move to next plane for now
    elif c<1:
        if 'k' in locals() and k>=1:
            #Recursively call the function, raising the plane position until the
algorithm reaches the top of the stack

            return 1

        elif 'j' in locals() and j>=1:

            return 1

        elif 'k' in locals() and k<1:

            return 2

        elif 'j' in locals() and k<1:

            return 6





def CheckBinderPaths1NoBifurcation(nly, nbl):

#making a list of a list of all the zpos offsets ie. the yarns, make a list of

#offsets in a weft stack in Matrix and a list of all the z offsets in a yarn in
znode



#check CBP3 works first before imposing bifurcation in here

maxOffset = nly - (nbl-1)

textile=GetTextile()

weave3D=textile.Get3DWeave()

layertolayer=textile.GetLayerToLayerWeave()

#to get number of x and y yarns use weave3D

NumXYarns=weave3D.GetNumXYarns()

NumWeftStacks=weave3D.GetNumYYarns()

binders=[]

IY=[]


for i in range(NumXYarns):
```

```
    binder=weave3D.IsBinderYarn(i)
    if binder:
        binderyarn=textile.GetYarn(i)
        binders.append(binderyarn)
        BF=BinderFunctions()
        YPosition=BF.GetXYarnIndex(i)
        IY.append(YPosition)

offsets=[]
for iy in IY:
    for ix in range(NumWeftStacks):
        offset=BF.GetBinderOffsets(ix, iy)
        offsets.append(offset)

WeftStacks=[]
for i in range(NumWeftStacks):
    WeftStacks.append(offsets[i::NumWeftStacks])

b=0
for Stack in WeftStacks:
    stack=Stack
    #print stack
    if all(x in stack for x in [0, maxOffset]):
        b=b+0
        print 'first constraint not violated, all yarns bound in stack'
    else:
        b=b+1
        print 'first constraint violated, not all yarns are bound'

print 'the value of b is', b
```

```
    return b

def CheckBinderPaths1WarpBifurcation(nly, bifstart, bifplane):
#Check the 1st constraint before and after the bifurcation in the warp direction



    textile=GetTextile()
    weave3D=textile.Get3DWeave()
    layertolayer=textile.GetLayerToLayerWeave()
    NumYarns=textile.GetNumYarns()
    NumXYarns=weave3D.GetNumXYarns()
    NumWeftStacks=weave3D.GetNumYYarns()
    binders=[]
    IY=[]

    for i in range(NumXYarns):
        binder=weave3D.IsBinderYarn(i)
        if binder:
            binderyarn=textile.GetYarn(i)
            binders.append(binderyarn)
            BF=BinderFunctions()
            YPosition=BF.GetXYarnIndex(i)
            IY.append(YPosition)

    offsets=[]
    for iy in IY:
        for ix in range(NumWeftStacks):
            offset=BF.GetBinderOffsets(ix, iy)
        offsets.append(offset)



StacksBefore=[]
StacksAfter=[]
#this is where to modify the code
for i in range(bifstart):
    StacksBefore.append(offsets[i::bifstart])

for i in range(bifstart, NumWeftStacks):
    StacksAfter.append(offsets[i::NumWeftStacks])
a=0
b=0
c=0

for Stack in StacksBefore:
    stack=Stack
    #print stack
    if all(x in stack for x in [0, nly]):
        a=a+0
        print 'first constraint not violated, all yarns bound in stack'
    else:
        a=a+1
        print 'first constraint violated, not all yarns are bound'




for Stack in StacksAfter:
    stack=Stack
    seen=[]
```

```
        #need 2 in list at bifplane so that both strands after bifurcation are fully bound

        for x in stack:

            if x==bifplane:

                seen.append(x)

        if len(seen)>1:

            c=c

        else:

            c=c+1

        #print stack

        if all(x in stack for x in [0, nly, bifplane]):

            b=b+0

            print 'first constraint not violated, all yarns bound in stack'

        else:

            b=b+1

            print 'first constraint violated, not all yarns are bound'



    violation = a + b + c

    print 'the value of constraint violation is', violation


    return violation
```