```python
#This is the equation for r, the max height of each y-z parabola
def hzOfx(hx,cx,rx,x):
    hz = -hx/rx**2 * (x-cx)**2 + hx
    return hz


#This is the equation for b, the curvature of each y-z parabola
def bzOfx(hz,cx,rx,rz,x):
    if ((x-cx)/rx)**2 != 1.0:
        bz = -hz/((rz**2)*(1-((x-cx)/rx)**2))
    else:
        bz = 0.0
    return bz


def Parabs(hx,cx,cz,rx,rz,Xraster,Zraster, Nx, Nz):
    import numpy as np
    Parabs_list = np.zeros(Nx*Nz,dtype=float)
    Parabs_array = Parabs_list.reshape([Nx,Nz])
    i = 0
    for x in Xraster:
        d = 0
        hz = hzOfx(hx,cx,rx,x)
        bz = bzOfx(hz,cx,rx,rz,x)
        for z in Zraster:
            if z <= (1-((x-cx)/rx)**2)**0.5*rz+cz and z >= -(1-((x-cx)/rx)**2)**0.5*rz+cz:
                Parabs_array[i][d] =  bz*(z-cz)**2 + hz
            else:
                Parabs_array[i][d] = -1 #0.0

            d = d+1
        i = i+1
    return Parabs_array

def FindBoundaries(Yraster, Nx, Nz):
    import numpy as np
    numbers = np.zeros(Nx, dtype=int)
    zindex0 = np.zeros(Nx, dtype=int)
    zindex1 = np.zeros(Nx, dtype=int)
    for i in range(0, Nx):
        counter1 = 0
        firstindex = -1
        lastindex = -1
        for d in range(0, Nz):
            if Yraster[i][d] > -1:
                counter1 += 1
                if firstindex < 0:
                    firstindex = d
            if lastindex < 0 and firstindex > 0 and Yraster[i][d] < 0:
                lastindex = d-1
        numbers[i] = counter1
        zindex0[i] = firstindex
        if lastindex < 0:
            lastindex = Nz - 1
        zindex1[i] = lastindex

    return numbers, zindex0, zindex1

def MkModel3(Xraster, YrasterT, Zraster, lc, Nx, Nz, file_name="GFG.msh"):
    import gmsh
    import sys
    import numpy as np
    numbersT, zindex0T, zindex1T = FindBoundaries(YrasterT, Nx, Nz)
    gmsh.initialize()
    Points = []
    TopLines = []
    ConnectLines = []
    EdgeLines = []
    TopEdgePoints = []
    TopCrossLines = []

    PointMapTop = np.zeros(Nx*Nz,dtype = int)
    for i in range(0,Nx*Nz):
        PointMapTop[i] = -1
    PointMapTop = PointMapTop.reshape([Nx,Nz])

    #k cycles through points on each parabola and resets when the x value changes
    k = 0
    #for the top parabolas
    for i in range(0,Nx): #for each parabola along x
        if zindex0T[i] > -1 and numbersT[i] > 1:
            TopEdgePoints.append(k)
            for j in range(zindex0T[i],zindex1T[i]+1): #for each z-value of a parabola
                #defines points and adds them to the list Points
                if j == zindex0T[i] or j == zindex1T[i]:
                    Y = 0.0
                else:
                    Y = YrasterT[i][j]
                thisPoint = gmsh.model.geo.add_point(Xraster[i], Y, Zraster[j], lc)
                Points.append(thisPoint)
                PointMapTop[i][j] = k
                k = k+1
            TopEdgePoints.append(k-1)

    #Make Triangles for shell
    for i in range(0,Nx): #for each parabola along x
        if zindex0T[i] > -1: # and numbersT[i] > 1:
            for j in range(zindex0T[i],zindex1T[i]+1):
                if i+1 < Nx and j+1 < Nz:
                    if PointMapTop[i][j] != -1 and PointMapTop[i][j+1] != -1 and
PointMapTop[i+1][j] != -1:
                        thisLine1 = gmsh.model.geo.add_line(Points[PointMapTop[i][j]],
Points[PointMapTop[i][j+1]])
                        thisLine2 = gmsh.model.geo.add_line(Points[PointMapTop[i]
[j+1]],Points[PointMapTop[i+1][j]])
                        thisLine3 = gmsh.model.geo.add_line(Points[PointMapTop[i+1]
[j]],Points[PointMapTop[i][j]])
                        face = gmsh.model.geo.add_curve_loop([thisLine1,thisLine2,thisLine3])
                        gmsh.model.geo.add_plane_surface([face])
                    if PointMapTop[i][j+1] != -1 and PointMapTop[i+1][j+1] != -1 and
PointMapTop[i+1][j] != -1:
                        thisLine1 = gmsh.model.geo.add_line(Points[PointMapTop[i][j+1]],
Points[PointMapTop[i+1][j+1]])
                        thisLine2 = gmsh.model.geo.add_line(Points[PointMapTop[i+1]
[j+1]],Points[PointMapTop[i+1][j]])
                        thisLine3 = gmsh.model.geo.add_line(Points[PointMapTop[i+1]
[j]],Points[PointMapTop[i][j+1]])
                        face = gmsh.model.geo.add_curve_loop([thisLine1,thisLine2,thisLine3])
                        gmsh.model.geo.add_plane_surface([face])

    #Make drum surface
    DrumPointMap = np.zeros(Nx*Nz,dtype = int)
    for i in range(0,Nx*Nz):
        DrumPointMap[i] = -1
    DrumPointMap = DrumPointMap.reshape([Nx,Nz])

    DrumEdgePoints = []
    #k = 0
    Y = 0.0
    for i in range(0,Nx):
        if zindex0T[i] > -1 and numbersT[i] > 1:
            DrumEdgePoints.append(k)
            for j in range(zindex0T[i],zindex1T[i]+1):
                thisPoint = gmsh.model.geo.add_point(Xraster[i], Y, Zraster[j], lc)
                Points.append(thisPoint)
                DrumPointMap[i][j] = k
                k = k+1
            DrumEdgePoints.append(k-1)
    #print(DrumEdgePoints)

    #Make Triangles for drum
    for i in range(0,Nx):
        if zindex0T[i] > -1:
            for j in range(zindex0T[i],zindex1T[i]+1):
                if i+1 < Nx and j+1 < Nz:
                    if DrumPointMap[i][j] != -1 and DrumPointMap[i][j+1] != -1 and 
DrumPointMap[i+1][j] != -1:
                        thisLine1 = gmsh.model.geo.add_line(Points[DrumPointMap[i][j]], 
Points[DrumPointMap[i+1][j]])
                        thisLine2 = gmsh.model.geo.add_line(Points[DrumPointMap[i+1]
[j]],Points[DrumPointMap[i][j+1]])
                        thisLine3 = gmsh.model.geo.add_line(Points[DrumPointMap[i]
[j+1]],Points[DrumPointMap[i][j]])
                        face = gmsh.model.geo.add_curve_loop([thisLine1,thisLine2,thisLine3])
                        gmsh.model.geo.add_plane_surface([face])
                    if DrumPointMap[i][j+1] != -1 and DrumPointMap[i+1][j+1] != -1 and 
DrumPointMap[i+1][j] != -1:
                        thisLine1 = gmsh.model.geo.add_line(Points[DrumPointMap[i][j+1]], 
Points[DrumPointMap[i+1][j]])
                        thisLine2 = gmsh.model.geo.add_line(Points[DrumPointMap[i+1]
[j]],Points[DrumPointMap[i+1][j+1]])
                        thisLine3 = gmsh.model.geo.add_line(Points[DrumPointMap[i+1]
[j+1]],Points[DrumPointMap[i][j+1]])
                        face = gmsh.model.geo.add_curve_loop([thisLine1,thisLine2,thisLine3])
                        gmsh.model.geo.add_plane_surface([face])



    gmsh.model.geo.synchronize()
    gmsh.model.mesh.generate()
    gmsh.write(file_name)
    #if 'close' not in sys.argv:
    #    gmsh.fltk.run()
    gmsh.finalize()




def StlConverter(froot):
    import os
    cmd = 'gmsh '+froot+'.msh -1 -o '+froot+'.stl'
    os.system(cmd)




def CleanUp(froot):
    import pymeshlab
    ms = pymeshlab.MeshSet()
    ms.load_new_mesh(froot+'.stl',unify_vertices=True)
    ms.apply_filter('meshing_close_holes', maxholesize=300)
    #ms.save_current_mesh(pathroot + froot+'.stl')
    ms.save_current_mesh(froot+'.stl')

def InpConverter(froot):
    import os
    cmd = 'gmsh '+froot+'.stl -1 -o '+froot+'.inp'
    os.system(cmd)

def ParseInput(fname):
    infile = open(fname, 'r')
    node_flag=False
    element_flag=False
    Nodes = []
    Elements = []
    for line in infile:
        if element_flag:
            eindex, nindex1, nindex2, nindex3 = line.split(',')
            eindex_i = int(eindex)
            nindex1_i = int(nindex1)
            nindex2_i = int(nindex2)
            nindex3_i = int(nindex3)
            Elements.append([eindex_i, nindex1_i, nindex2_i, nindex3_i])

        if node_flag:
            if '***' in line:
                node_flag=False
            else:
                nindex, nindex1, nindex2, nindex3 = line.split(',')
                nindex_i = int(nindex)
                nindex1_i = float(nindex1)
                nindex2_i = float(nindex2)
                nindex3_i = float(nindex3)
                Nodes.append([nindex_i, nindex1_i, nindex2_i, nindex3_i])

        if '*NODE' in line:
            node_flag=True

        if '*ELEMENT' in line:
            element_flag=True

    infile.close()
    return Nodes, Elements

def MaterialSorter(Nodes, Elements, tol):
    Leather = []
    Shell = []
    counter = 0
    for element in Elements:
        if (Nodes[element[1]-1][2] < tol) and (Nodes[element[2]-1][2] < tol) and
(Nodes[element[3]-1][2] < tol):
            This_e = [-1, element[1], element[2], element[3]]
            Leather.append(This_e)
        else:
            This_e = [-1, element[1], element[2], element[3]]
            Shell.append(This_e)
    for el in Shell:
        counter = counter+1
        el[0] = counter
    for el in Leather:
        counter = counter+1
        el[0] = counter

    return Leather, Shell
```

```
def LowerBoundaryFinder(Nodes, Shell, Leather, xmin, thickness):
    Boundaries = []
    for element in Shell:
        if (Nodes[element[1]-1][1] < xmin+thickness) or (Nodes[element[2]-1][1] <
xmin+thickness) or (Nodes[element[3]-1][1] < xmin+thickness):
            Boundaries.append(Nodes[element[1]-1][0])
            Boundaries.append(Nodes[element[2]-1][0])
            Boundaries.append(Nodes[element[3]-1][0])
    for element in Leather:
        if (Nodes[element[1]-1][1] < xmin+thickness) or (Nodes[element[2]-1][1] <
xmin+thickness) or (Nodes[element[3]-1][1] < xmin+thickness):
            Boundaries.append(Nodes[element[1]-1][0])
            Boundaries.append(Nodes[element[2]-1][0])
            Boundaries.append(Nodes[element[3]-1][0])
    return Boundaries


def WriteCleanMesh(froot, Nodes, Shell, Leather):
    fname = froot+'_2Mat.inp'
    f = open(fname,'w')

    ############################
    #Write the Nodes Section
    ############################
    ln  = '*Heading\n'
    f.write(ln)
    ln = froot+'\n'
    f.write(ln)
    ln = '*NODE\n'
    f.write(ln)
    for nd in Nodes:
        ln = str(nd[0])+','+str(nd[1])+','+str(nd[2])+','+str(nd[3])+'\n'
        f.write(ln)

    ############################
    #Write the Elements Section
    ############################
    el_header1 = '******* E L E M E N T S *************\n'
    f.write(el_header1)

    el_header2 = '*ELEMENT, type=S3, ELSET=Shell\n'
    f.write(el_header2)
    for el in Shell:
        ln = str(el[0])+','+str(el[1])+','+str(el[2])+','+str(el[3])+'\n'
        f.write(ln)

    el_header3 = '*ELEMENT, type=S3, ELSET=Leather\n'
    f.write(el_header3)
    for el in Leather:
        ln = str(el[0])+','+str(el[1])+','+str(el[2])+','+str(el[3])+'\n'
        f.write(ln)

    f.close()



def WriteBoundaries(froot, Boundaries):
    fname = froot+'.bounds'
    f = open(fname,'w')
    N = len(Boundaries)
    for b in range(0,N-1):
        f.write(str(Boundaries[b])+',\n')
    f.write(str(Boundaries[N-1])+'\n')
    f.close()
```

```python
def WriteFinalInp(froot, ShellYoungs, ShellPoisson, ShellDensity, ShellThickness,
                 LeatherYoungs, LeatherPoisson, LeatherDensity, LeatherThickness, Nfreq):

    #Import the mesh
    newinp = froot+'_ccx.inp'
    f = open(newinp,'w')
    ln = '*INCLUDE,INPUT =  '+froot+'_2Mat.inp\n'
    f.write(ln)

    #Create the boundary conditions
    ln = '*NSET, NSET=LEFTSIDE\n'
    f.write(ln)
    ln = '*INCLUDE,INPUT =  '+froot+'.bounds\n'
    f.write(ln)
    ln = '*BOUNDARY\n'
    f.write(ln)
    ln = 'LEFTSIDE, 1\n'
    f.write(ln)
    ln = '*BOUNDARY\n'
    f.write(ln)#*BOUNDARY
    ln = 'LEFTSIDE, 2\n'
    f.write(ln)#LEFTSIDE, 2
    ln = '*BOUNDARY\n'
    f.write(ln)
    ln = 'LEFTSIDE, 3\n'
    f.write(ln)

    #Define Shell Properties
    ln = '*MATERIAL, NAME=ELShell\n'
    f.write(ln)
    ln = '*ELASTIC\n'
    f.write(ln)
    ln = str(ShellYoungs)+',        '+str(ShellPoisson)+'\n'
    f.write(ln)
    ln = '*DENSITY\n'
    f.write(ln)
    ln = str(ShellDensity)+'\n'
    f.write(ln)
    ln = '*SHELL SECTION,MATERIAL=ELShell,ELSET=Shell\n'
    f.write(ln)
    ln = str(ShellThickness)+'\n'
    f.write(ln)


    #Define Leather Properties
    ln = '*MATERIAL, NAME=ELLeather\n'
    f.write(ln)
    ln = '*ELASTIC\n'
    f.write(ln)
    ln = str(LeatherYoungs)+',        '+str(LeatherPoisson)+'\n'
    f.write(ln)
    ln = '*DENSITY\n'
    f.write(ln)
    ln = str(LeatherDensity)+'\n'
    f.write(ln)
    ln = '*SHELL SECTION,MATERIAL=ELLeather,ELSET=Leather\n'
    f.write(ln)
    ln = str(LeatherThickness)+'\n'
    f.write(ln)

    #Calculate the frequencies
    ln = '*STEP,PERTURBATION\n'
    f.write(ln)
    ln = '*FREQUENCY\n'
    f.write(ln)
    ln = str(Nfreq)+'\n'
    f.write(ln)
    ln = '*ENDSTEP\n'
    f.write(ln)
    f.close()

def GetFreq(froot):
    fname = froot+'_ccx.dat'
    f = open(fname)
    Res = []
    for i in range(0,7):
        junk = f.readline()
    for i in range(0,Nfreq):
        ln = f.readline()
        ln = ln[7:]
        j1, j2, j3, freq, j5 = ln.split('   ')
        Res.append(float(freq))
    f.close()
    return Res
```