```
# Created by Darren Holland
# Modified by Darren Holland 2020-11-02
#*********************************************************************
# File for creating the 2D mask matrix used to create the geometry
#*********************************************************************

# Make all numpy available via shorter 'num' prefix
import numpy as np
# Make some matlib functions accessible directly at the top level via, e.g.
rand(3,3)
from numpy.matlib import rand,zeros,ones,empty,eye

# Function to create matrix
def
CreateMat(Outputfile,finthick,wallthick,deltatheta,deltaphi,finwidth,wallwidth,p
histart,phiend,MaskMinThick):
    # Set discretization
    # NOTE: if 360 is not evenly divisible by deltatheta, there will be a gap in
the
    # geometry (unless CreateGeo.py is changed)
    ntheta = int(np.floor(360 / deltatheta))
    nphi = int(np.floor((phiend-phistart) / deltaphi))

    # Initialize design matrix and fin/wall valley centers
    mat = np.ones((ntheta, nphi))*MaskMinThick
    finpos = np.zeros((1, nphi))
    wallpos = np.ones((1, nphi))*float(wallwidth)/2

    # Create wall
    mat[0:wallwidth,:] = wallthick

    # Add fin (middle of fin opposite wall)
    bad_geo = 0
    rpl_wall = float(ntheta - wallwidth - nphi)
    # Don't run geometries which should be avoided
    # Wall width spans more than # theta (no wall evident)
    if wallwidth > 360/deltatheta:
        bad_geo = 1
    # Fin width spans more than # phi (duplicates case where fin width = # phi)
    if finwidth > (180-phistart)/deltaphi-1:
        bad_geo = 1
    #Case where fin would disappear (have fin replace part of wall)
    if -rpl_wall > wallwidth - 1:
        startpos = int(wallwidth)
        bad_geo = 1
        # print("Running this geometry would make the wall disappear.  Skipping
this design...\n")
    else:
        # If portions of wall will be replaced, change starting position
        if rpl_wall >= 0:
            if finwidth <= rpl_wall:
                # Overlap multiple wall pixels
                startpos = int(np.ceil(int(wallwidth) + rpl_wall/2 -
finwidth/2))
            else:
                # Overlap one wall pixel
                startpos = int(wallwidth)
        else:
            # If wall is not overlapped, start near middle of mask
            startpos = ntheta - nphi
    for jj in np.arange(nphi):
        # Create fin
        mat[startpos+jj:startpos+finwidth+jj,jj] = finthick
        # To calculate fin position in degrees, first cell only counts as 0.5
```

```
since start at middle of voxel
        if startpos+jj+finwidth > ntheta:
            finend = startpos + jj + finwidth - ntheta
        else:
            finend = 0
        # Calculate middle of fin (aka location of fin response valley minimum)
        finpos[0,jj] = (startpos+jj+(finwidth-finend)/2)
        # Calculate middle of wall for overlapped pixels (aka location of wall
response valley minimum)
        if startpos + jj <= wallwidth:
            wallpos[0,jj] = (startpos - 1 + jj)/2

    mat=np.matrix(mat)
    # Save to output file
    # Design matrix
    with open(Outputfile, 'wb') as fname:
       for line in mat:
           np.savetxt(fname, line, fmt='%.15f')

    # Fin middle location (response valley minimum)
    with open('FinPos.inp', 'wb') as fname:
       for line in finpos:
           np.savetxt(fname, line, fmt='%.8f')

    # Wall middle location (response valley minimum)
    with open('WallPos.inp', 'wb') as fname:
       for line in wallpos:
           np.savetxt(fname, line, fmt='%.8f')

    # Return if valid/invalid design
    return bad_geo
```