```
#-------------------------------------------------------------------------------
# Name:     areaquad (similar to areaquad from Matlab)
# Purpose:  Surface area of latitude-longitude spheroid rectangle
#           Produces a world raster in geographic coordinate system
#            with cell size determined by the user. Value stored in the cell
#           is the area of the cell. Useful when you have proportions or
#           density rasters that need to be converted to actual value without
#           projecting the raster (e.g. population density to total population)

#
# Author:      Eugenio Arima, UT Austin
#
#For more details, please check reference:
# Snyder, J. P. (1987). Map projections--A working manual (Vol. 1395):
#          US Government Printing Office.
#
# Created:     06/12/2016
# Copyright:   (c) ea9267 2016
# Licence:     <your licence>
# Modifications:
#
#-------------------------------------------------------------------------------
# Import system modules

import numpy as np
import math

def areaquad(datum, csize, unit='km'):
    '''Creates a GCS raster with cell size csize where the cell value is the
    area of the cell in the unit specified
    Usage: >>> test = areaquad('wgs84, 0.25, 'km')'''

    #common ellipsoid library major, minor axis.
    # can add more ellipsoids to dictionary library as you go
    datum_lib = {'wgs84':  [6378137.0, 6356752.3],
                 'nad83': [6378137.0, 6356752.3],
                 'nad27': [6378206.4, 6356583.8],
                 'sad69': [6378160.0, 6356774.72],
                 'sirgas2000': [6378137.0, 6356752.3]}
    a,b = datum_lib.get(datum) #get ellipsoid parameters
    e2 = (a**2 - b**2)/(a**2)
    e = math.sqrt(e2) #eccentricity
    r_2 =  math.sqrt((a**2/2)+(b**2/2)*(math.atanh(e)/e)) #authalic radius
    # output array dimension
    nrow = int(180/csize)
    ncol = int(360/csize)
    #create arrays of cells of equal size in dd, convert to radians
    lats1 = np.linspace(90,-90+csize, num= nrow)
    lats2 = lats1 - csize
    latin1 = np.radians(lats1)
    latin2 = np.radians(lats2)
    #convert latitudes to authalic latitudes. See Snyder (1987, p.16 eq. 3-18)
    # (want to find beta, not theta, that's why you subtract the series)
    #factor expansion series
```

```
    fact1 = e**2 /3 + 31*e**4 /180 + 517*e**6 /5040 #expansion series 1
    fact2 = 23*e**4 /360 + 251*e**6 /3780 #expansion series 2
    fact3 = 761*e**6 /45360 #expansion series 3
    latout1 = latin1 - fact1*np.sin(2*latin1) + fact2*np.sin(4*latin1) + fact3*
    np.sin(6*latin1)
    latout2 = latin2 - fact1*np.sin(2*latin2) + fact2*np.sin(4*latin2) + fact3*
    np.sin(6*latin2)
    # report value in preferred unit
    if unit == 'm':  #either in meters or km (default)
        r2 = r_2 #radius in meters
    else:
        r2 = r_2/1000.0 # in km
    #calculate area of square on spherical surface
    cst = (np.pi/180)*(r2**2) #just a constant; see Synder 1987.
    area = cst*(np.absolute(np.sin(latout1)-np.sin(latout2)))*np.absolute(csize)
    # replicate column over Earth's extent
    grid = np.tile(area, (ncol,1)).T #replicate lat and transpose because
                #area is stored as a row array, not column
    return grid

################################
import sys, string, os, arcpy
from arcpy import env
from arcpy.sa import *
arcpy.CheckOutExtension("Spatial")
arcpy.env.overwriteOutput = True
import numpy as np
import math

def main():
    csize = 0.25
    test = areaquad('wgs84', csize)
    # export array to raster
    point = arcpy.Point(-180, -90)
    out_r = arcpy.NumPyArrayToRaster(test,point, csize, csize)
    outRaster = 'C:/Users/ea9267/Downloads/test.tif'
    out_r.save(outRaster)
    sr = arcpy.SpatialReference(4326) #GCS WGS84
    arcpy.DefineProjection_management(outRaster, sr)
main()
```