## Imports

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
from IPython import display  
import pylab as pl

## Numerical equation

def solver(dt,rho,c\_p,k,r,dr,T,dtheta,h,T\_f):  
 """Compute temperature at a provided location via conduction and (if applicable) convection heat transfer. Returns the temperature (float) at the provided location.  
 The temperature and thermal conductivity arguments are in the form of arrays which specify the corresponding values at the neighbors of the center point, in the format [(i,j),(i-1,j),(i+1,j),(i,j-1),(i,j+1)].  
 Note: the solver assumes the units supplied are compatible.  
 Args:  
 dt (float): simulation time step  
 rho (float): material density  
 c\_p (float): material specific heat capacity  
 k (array): thermal conductivity array  
 r (float): radial location  
 dr (float): radial step size  
 T (array): temperature array  
 dtheta (float): angular step size  
 h (float): heat transfer coefficient (supply 0 for no convection)  
 T\_f (float): fluid temperature  
  
 Returns:  
 float: new temperature  
 """  
  
 # Conduction terms  
 cond\_1 = k[1]\*(T[1]-T[0])/(dr\*\*2)  
 cond\_2 = k[2]\*(T[2]-T[0])/(dr\*\*2)  
 cond\_3 = k[0]\*(T[2]-T[1])/(2\*r\*dr)  
 cond\_4 = k[3]\*(T[3]-T[0])/(r\*\*2\*dtheta\*\*2)  
 cond\_5 = k[4]\*(T[4]-T[0])/(r\*\*2\*dtheta\*\*2)  
  
 # Convection term, if applicable  
 if h != 0:  
 convection = h\*(T\_f-T[0])  
 else:  
 convection = 0  
   
 T\_new = dt/(rho[0]\*c\_p[0]) \* (cond\_1 + cond\_2 + cond\_3 + cond\_4 + cond\_5 + convection) + T[0]  
 return T\_new

## Define geometry, material properties, and other inputs

# Radius in [m]  
r\_i = 0.006  
r\_interface = 0.008  
r\_o = 0.010  
  
# Angle in [rad]  
theta\_0 = 0  
theta\_1 = np.pi/3  
  
# Grid size  
dr = 0.0004  
dr2 = dr\*\*2  
r = np.arange(r\_i,r\_o,dr)  
r = np.append(r,r\_o)  
n\_r = len(r)  
ind\_interface = (np.abs(r-r\_interface)).argmin()  
  
dtheta = np.pi/30  
dtheta2 = dtheta\*\*2  
theta = np.arange(theta\_0,theta\_1,dtheta)  
theta = np.append(theta,theta\_1)  
n\_theta = len(theta)  
  
# Time steps  
t = 0  
dt = 0.00000001  
  
# Thermal properties  
# k = [W/m-K]  
# rho = [kg/m^3]  
# c\_p = [J/kg-k]  
  
# Alumina properties  
k\_i = 30  
rho\_i = 3900  
c\_p\_i = 500  
  
# Graphite properties  
k\_o = 400  
rho\_o = 2250  
c\_p\_o = 707  
  
# Alumina properties  
k\_i = 1  
rho\_i = 1  
c\_p\_i = 1  
  
# Graphite properties  
k\_o = 1  
rho\_o = 1  
c\_p\_o = 1  
  
# Convection properties  
k = 30  
mdot = 150\*0.00129/60  
mu = 1.81e-5  
c\_p = 1005  
d = 0.00730  
h = k\*d\*0.023\*(mdot\*d/mu)\*\*0.8\*(mu\*c\_p/k)\*\*0.4 \* 100000000  
T\_f = 25  
  
# Constant temperature properties  
T\_const = 600  
  
T\_old = np.ones((n\_r,n\_theta))\*T\_const  
T\_new = T\_old  
d2 = [0,dr2,dr2,dtheta2,dtheta2]  
d = [0,dr,dr,dtheta,dtheta]  
  
k\_avg = (k\_i+k\_o)/2  
rho\_avg = (rho\_i+rho\_o)/2  
c\_p\_avg = (c\_p\_i+c\_p\_o)/2  
  
residual = [np.NaN]  
n = 0  
t = [0]  
  
dtmax = (dr\*\*2 + dtheta\*\*2)/(2\*k\_o)

## Solver loop

thresh = 1  
for ind in range(1000):  
#while (residual[n] > thresh or n==0):  
 n = n + 1  
 t.append(t[n-1]+dt)  
  
 # Save previous iteration  
 T\_old = T\_new  
 # Initialize new iteration as ones (could be anything, it gets overwritten)  
 T\_new = np.ones((n\_r,n\_theta))  
  
 # Set constant temperature at outer radius  
 T\_new[n\_r-1,int(n\_theta/2):n\_theta] = T\_const  
  
 # Material properties for outer region  
 rho\_outer = [rho\_o,rho\_o,rho\_o,rho\_o,rho\_o]  
 c\_p\_outer = [c\_p\_o,c\_p\_o,c\_p\_o,c\_p\_o,c\_p\_o]  
 k\_outer = [k\_o,k\_o,k\_o,k\_o,k\_o]  
  
 # Material properties for interface region  
 rho\_interface = [rho\_avg,rho\_i,rho\_o,rho\_avg,rho\_avg]  
 c\_p\_interface = [c\_p\_avg,c\_p\_i,c\_p\_o,c\_p\_avg,c\_p\_avg]  
 k\_interface = [k\_avg,k\_i,k\_o,k\_avg,k\_avg]  
  
 # Material properties for inner region  
 rho\_inner = [rho\_i,rho\_i,rho\_i,rho\_i,rho\_i]  
 c\_p\_inner = [c\_p\_i,c\_p\_i,c\_p\_i,c\_p\_i,c\_p\_i]  
 k\_inner = [k\_i,k\_i,k\_i,k\_i,k\_i]  
  
 # Step through domain by radius, starting at outer radius and going inward  
 for i in range(n\_r-1,-1,-1):  
 # Solve temperature at adiabatic surface on outer radius  
 if i == n\_r-1:  
 # Left point  
 T\_local = [T\_old[i,0],T\_old[i-1,0],T\_old[i-1,0],T\_old[i,1],T\_old[i,1]]  
 T\_new[i,0] = solver(dt,rho\_outer,c\_p\_outer,k\_outer,r[i],dr,T\_local,dtheta,0,T\_f)  
 # Outer radius (adiabatic in radial direction)  
 for j in range(1,int(n\_theta/2)):  
 T\_local = [T\_old[i,j],T\_old[i-1,j],T\_old[i-1,j],T\_old[i,j-1],T\_old[i,j+1]]  
 T\_new[i,j] = solver(dt,rho\_outer,c\_p\_outer,k\_outer,r[i],dr,T\_local,dtheta,0,T\_f)  
  
 # Material outside of interface  
 if (i>ind\_interface) and (i<n\_r-1):  
 # Left point  
 T\_local = [T\_old[i,0],T\_old[i-1,0],T\_old[i+1,0],T\_old[i,1],T\_old[i,1]]  
 T\_new[i,0] = solver(dt,rho\_outer,c\_p\_outer,k\_outer,r[i],dr,T\_local,dtheta,0,T\_f)  
 # Right point  
 T\_local = [T\_old[i,n\_theta-1],T\_old[i-1,n\_theta-1],T\_old[i+1,n\_theta-1],T\_old[i,n\_theta-2],T\_old[i,n\_theta-2]]  
 T\_new[i,n\_theta-1] = solver(dt,rho\_outer,c\_p\_outer,k\_outer,r[i],dr,T\_local,dtheta,0,T\_f)  
 # Inner points  
 for j in range(1,n\_theta-1):  
 T\_local = [T\_old[i,j],T\_old[i-1,j],T\_old[i+1,j],T\_old[i,j-1],T\_old[i,j+1]]  
 T\_new[i,j] = solver(dt,rho\_outer,c\_p\_outer,k\_outer,r[i],dr,T\_local,dtheta,0,T\_f)  
  
 # Solve temperature at interface  
 if (i == ind\_interface):  
 # Left point  
 T\_local = [T\_old[i,0],T\_old[i-1,0],T\_old[i+1,0],T\_old[i,1],T\_old[i,1]]  
 T\_new[i,0] = solver(dt,rho\_interface,c\_p\_interface,k\_interface,r[i],dr,T\_local,dtheta,0,T\_f)  
 # Right point  
 T\_local = [T\_old[i,n\_theta-1],T\_old[i-1,n\_theta-1],T\_old[i+1,n\_theta-1],T\_old[i,n\_theta-2],T\_old[i,n\_theta-2]]  
 T\_new[i,n\_theta-1] = solver(dt,rho\_interface,c\_p\_interface,k\_interface,r[i],dr,T\_local,dtheta,0,T\_f)  
 # Inner points  
 for j in range(1,n\_theta-1):  
 T\_local = [T\_old[i,j],T\_old[i-1,j],T\_old[i+1,j],T\_old[i,j-1],T\_old[i,j+1]]  
 T\_new[i,j] = solver(dt,rho\_interface,c\_p\_interface,k\_interface,r[i],dr,T\_local,dtheta,0,T\_f)  
  
 # Material inside of interface  
 if (i < ind\_interface) and (i>0):  
 # Left point  
 T\_local = [T\_old[i,0],T\_old[i-1,0],T\_old[i+1,0],T\_old[i,1],T\_old[i,1]]  
 T\_new[i,0] = solver(dt,rho\_inner,c\_p\_inner,k\_inner,r[i],dr,T\_local,dtheta,0,T\_f)  
 # Right point  
 T\_local = [T\_old[i,n\_theta-1],T\_old[i-1,n\_theta-1],T\_old[i+1,n\_theta-1],T\_old[i,n\_theta-2],T\_old[i,n\_theta-2]]  
 T\_new[i,n\_theta-1] = solver(dt,rho\_inner,c\_p\_inner,k\_inner,r[i],dr,T\_local,dtheta,0,T\_f)  
 # Inner points  
 for j in range(1,n\_theta-1):  
 T\_local = [T\_old[i,j],T\_old[i-1,j],T\_old[i+1,j],T\_old[i,j-1],T\_old[i,j+1]]  
 T\_new[i,j] = solver(dt,rho\_inner,c\_p\_inner,k\_inner,r[i],dr,T\_local,dtheta,0,T\_f)  
   
 # Convection boundary  
 if (i == 0):  
 # Left point  
 T\_local = [T\_old[0,0],T\_old[1,0],T\_old[1,0],T\_old[0,1],T\_old[0,1]]  
 T\_new[0,0] = solver(dt,rho\_inner,c\_p\_inner,k\_inner,r[i],dr,T\_local,dtheta,h,T\_f)  
 # Right point  
 T\_local = [T\_old[0,n\_theta-1],T\_old[1,n\_theta-1],T\_old[1,n\_theta-1],T\_old[0,n\_theta-2],T\_old[0,n\_theta-2]]  
 T\_new[0,n\_theta-1] = solver(dt,rho\_inner,c\_p\_inner,k\_inner,r[i],dr,T\_local,dtheta,h,T\_f)  
 # Inner radius (adiabatic in radial direction)  
 for j in range(1,n\_theta-1):  
 T\_local = [T\_old[0,j],T\_old[1,j],T\_old[1,j],T\_old[0,j-1],T\_old[0,j+1]]  
 T\_new[0,j] = solver(dt,rho\_inner,c\_p\_inner,k\_inner,r[i],dr,T\_local,dtheta,h,T\_f)  
   
 residual.append(np.linalg.norm(T\_new)-np.linalg.norm(T\_old))

## Post process results and plotting

fig,ax = plt.subplots(tight\_layout=True)  
x = np.zeros((n\_r-1,n\_theta-1))  
y = np.zeros((n\_r-1,n\_theta-1))  
for i in range(n\_r-1):  
 for j in range(n\_theta-1):  
 x[i,j] = r[i]\*np.cos(theta[j])  
 y[i,j] = r[i]\*np.sin(theta[j])  
ax.axis('equal')  
cf = plt.contourf(x,y,T\_new[0:n\_r-1,0:n\_theta-1])  
cbar = plt.colorbar(cf)  
cbar.set\_label('Temperature (K)')  
ax.set\_xlabel(r'$x$ coordinate (m)')  
ax.set\_ylabel(r'$y$ coordinate (m)')  
  
fnind = 0  
while os.path.exists(f'Figures/CylindricalT\_{fnind}.svg'):  
 fnind+=1  
plt.savefig(f'Figures/CylindricalT\_{fnind}.svg',format='svg')  
  
fig,ax = plt.subplots(tight\_layout=True)  
x = np.arange(0,n\_r)  
y = np.arange(0,n\_theta)  
X,Y = np.meshgrid(x,y)  
cf = plt.contourf(X,Y,np.transpose(T\_new))  
cbar = plt.colorbar(cf)  
cbar.set\_label('Temperature (K)')  
ax.set\_xlabel(r'$\xi$ index')  
ax.set\_ylabel(r'$\eta$ index')  
  
fnind = 0  
while os.path.exists(f'Figures/TransformedT\_{fnind}.svg'):  
 fnind+=1  
plt.savefig(f'Figures/TransformedT\_{fnind}.svg',format='svg')  
  
fig.ax = plt.subplots(tight\_layout=True)  
plt.semilogy(np.arange(n+1),np.abs(residual))  
plt.xlabel("Iteration")  
plt.ylabel("Residual")  
  
fnind = 0  
while os.path.exists(f'Figures/Residual\_{fnind}.svg'):  
 fnind+=1  
plt.savefig(f'Figures/Residual\_{fnind}.svg',format='svg')

fig,ax = plt.subplots(tight\_layout=True)  
for j in range(n\_theta-1):  
 for i in range(n\_r):  
 x1 = r[i]\*np.cos(theta[j])  
 x2 = r[i]\*np.cos(theta[j+1])  
 y1 = r[i]\*np.sin(theta[j])  
 y2 = r[i]\*np.sin(theta[j+1])  
 plt.plot([x1,x2],[y1,y2],color='k')  
for i in range(n\_r-1):  
 for j in range(n\_theta):  
 x1 = r[i]\*np.cos(theta[j])  
 x2 = r[i+1]\*np.cos(theta[j])  
 y1 = r[i]\*np.sin(theta[j])  
 y2 = r[i+1]\*np.sin(theta[j])  
 plt.plot([x1,x2],[y1,y2],color='k')  
ax.axis('equal')  
ax.set\_xlabel(r'$x$ coordinate (m)')  
ax.set\_ylabel(r'$y$ coordinate (m)')  
  
fnind = 0  
while os.path.exists(f'Figures/CylindricalMesh\_{fnind}.svg'):  
 fnind+=1  
plt.savefig(f'Figures/CylindricalMesh\_{fnind}.svg',format='svg')  
  
fig,ax = plt.subplots(tight\_layout=True)  
x = np.arange(0,n\_r)  
y = np.arange(0,n\_theta)  
X,Y = np.meshgrid(x,y)  
for i in range(n\_r-1):  
 plt.plot([X[:,i],X[:,i+1]],[Y[:,i],Y[:,i+1]],color='k')  
for j in range(n\_theta-1):  
 plt.plot([X[j,:],X[j+1,:]],[Y[j,:],Y[j+1,:]],color='k')  
ax.axis('equal')  
ax.set\_xlabel(r'$\xi$ index')  
ax.set\_ylabel(r'$\eta$ index')  
  
fnind = 0  
while os.path.exists(f'Figures/TransformedMesh\_{fnind}.svg'):  
 fnind+=1  
plt.savefig(f'Figures/TransformedMesh\_{fnind}.svg',format='svg')