```
#!/usr/bin/env python
# The purpose of this program is to compute the magnetic field at a number of
# positions given some wire pattern

# I would like to do this using an expression for the magnetic field from a
# ring

# I would also like to do this using a list of line-segments where the length
# of each segment is very small in relation to its distance from the field
# point.

# It would be good to couple this program with a program to compute spherical
# harmonic components to the magnetic field.

# It would also be good to compute the field in harmonics using David's
# Spherical Harmonic equation.
#



from math import sqrt,pi,pow
#from numarray import *
from scipy import *
from KStandard import listmult,xyz_from_rtp_vect,xyz_to_rtp_point
from KGeometry import Vector
from constants import mu_0
import cPickle
#from scipy import cross
from TesseralSegs import smoothcoil
from CoilViewer import seglistviewer
from RingArray import MagFieldRingArray
#from sensecoiltypes import smoothcoil

def fieldfind(Rformer,seglist,filename):
    dyz=0.1*Rformer
    Rmesh=Rformer+dyz/2.0
    plotradial=3*Rmesh #meters (a box this size will be plotted in the mid-
dle)
    plotaxial=3*Rmesh  #meters



    rvect=arange(-plotradial,plotradial,dyz)
    zvect=arange(-plotaxial,plotaxial,dyz)

    result=[]
    y=0
    for x in rvect:
        tmp=Biot_Savart_LineSegments(seglist,[Point((x,y,z),type="CARTESIAN")
for z in zvect])
        #print tmp[0]
        #print len(tmp)
        result.append(tmp)
    #print len(result)
    #print len(result[0])
    q=["rvect,zvect,result(x,y,z)",rvect,zvect,result]
    f=file(filename,"w+")
    cPickle.dump(q,f)
    f.close()

def Biot_Savart_LineSegments(seglist,fieldpoints):
    """Given some line segments in the form [[cl,[xs,ys,zs],[xe,ye,ze]],...]
    and some field points in the form [point,...]. This function will
return
    a list of the form [[Bx,By,Bz],...] corresponding to the magnetic
field
    at each field point.

    !!!! The actual magnetic field will be the result returned multiplied
by
    the current !!!! The result will be computed for unity current.

    """
    result=[]
    for point in fieldpoints:
        tmp=_Biot_Savart_LineSegments_at_Point(seglist,point)
        #print tmp
        result.append(tmp)

    return result

def _Biot_Savart_LineSegments_at_Point(seglist,point):
    """Do the calculation for a single field point"""
    Bx=0
    By=0
    Bz=0
    for seg in seglist:
        tBx,tBy,tBz=_Biot_Savart_SingleSeg(seg,point)
        Bx+=tBx
        By+=tBy
        Bz+=tBz
    return [Bx,By,Bz]
```

```
def _cross(a,b):
    """You should really be using numarray or numpy for this stuff"""
    x1=a[0]
    y1=a[1]
    z1=a[2]
    x2=b[0]
    y2=b[1]
    z2=b[2]
    return [(y1*z2-z1*y2),(z1*x2-x1*z2),(x1*y2-y1*x2)]

def _Biot_Savart_SingleSeg(seg,point):
    """Biot Savart Law for a single line segment"""
    start=seg[1]
    xs=start[0]
    ys=start[1]
    zs=start[2]
    end=seg[2]
    xe=end[0]
    ye=end[1]
    ze=end[2]
    midpoint=[(xs+xe)/2.0,(ys+ye)/2.0,(zs+ze)/2.0]
    #print midpoint
    dl=array([(xe-xs),(ye-ys),(ze-zs)])
    r=array([(point.x-midpoint[0]),
             (point.y-midpoint[1]),
             (point.z-midpoint[2])
             ])
    r2=r[0]*r[0]+r[1]*r[1]+r[2]*r[2]
    if r2<1e-13:
        # This gives the wrong answer but we are too close to the wire to care
anyways!
        return [0.0,0.0,0.0]
    r3=pow(r2,3.0/2.0)
    return cross(dl,r)*(mu_0/(4*pi*r3))#listmult(_cross(dl,r),mu_0/(4*pi*r3))

def Biot_Savart_ZRings(ringlist,zvect):
    """Only the z positions of the fieldpoints.
    The Dimension inputs should all be in meters.
    The result is in
    Teslas/Amp"""
    result=[]
    for z in zvect:
        bz=0
        for ring in ringlist:
            zR=ring[1]
            w=ring[0]
            R=ring[2]
            dz=z-zR
            bz+=w*(mu_0/2.0)*(R*R)/pow(R*R+dz*dz,1.5)
        result.append(bz)
    return result


def Biot_Savart_Rings(ringlist,fieldpoints_xyz):
    """the ringlist is [[wraps,zpos,rpos],[wraps,zpos,rpos],etc]. The
fieldpoints_xyz is
    [point,point,...] of the desired fieldpoints in type 'point'. The
result needs to be multiplied by
    the current. It is in teslas,amp [[Bx,By,Bz],[Bx,By,Bz],...]

    The result is in Teslas/amp and the inputs should all be in meters
or
    number of wraps
    """
    result=[]
    for p in fieldpoints_xyz:
        fieldpoint=Vector(p,type="CARTESIAN")
        result.append(MagFieldRingArray(ringlist,fieldpoint))
                        #_Biot_Savart_Rings_singlepoint(ringlist,point))
    return result

def _Biot_Savart_Rings_singlepoint(ringlist,fieldpoint_xyz):
    Bx=0
    By=0
    Bz=0
    for ring in ringlist:
        tBx,tBy,tBz=_Biot_Savart_SingleRing(ring,fieldpoint_xyz)
        Bx+=tBx
        By+=tBy
        Bz+=tBz
    return [Bx,By,Bz]

def _Biot_Savart_SingleRing(ring_wzr,fieldpoint_xyz):
    """Uses Jackson's approximation described in SimpleRing.lyx"""

    # Jackson's approximation only works far from the RING!!
    # See RingArray.py for a better way.
    raise Exception
    x=fieldpoint_xyz[0]
    y=fieldpoint_xyz[1]
    z=fieldpoint_xyz[2]-ring_wzr[1]
    R=ring_wzr[2]
    R2=R*R
    r=sqrt(x*x+y*y+z*z)
    r2=r*r
    R2pr2=R2+r2
    #print "z: ",z," r: ",r
    if z==0.0 and r==0.0:
        costheta=1.0
        sintheta=0.0
    else:
        costheta=z/float(r)
        sintheta=sqrt(1-costheta*costheta)
    sin2theta=sintheta*sintheta
    Br=(mu_0*R2*costheta/(2*pow(R2pr2,1.5)))*(1+15*R2*r2*sin2theta/(4*R2pr2*R2pr2))
    Btheta=-((mu_0*R2*sintheta)/(4*pow(R2pr2,2.5)))*(2*R2-r2+((15*R2*r2*sin2theta)/(8*R2pr2*
    Bphi=0
    fact=ring_wzr[0]
    #print Br
    Bx,By,Bz=xyz_from_rtp_vect([fact*Br,fact*Btheta,fact*Bphi],xyz_to_rtp_point([x,y,z]))
    return [Bx,By,Bz]

def testring():
    from sensecoiltypes import HelixSeg
    from CoilViewer import seglistviewer
    from RingArray import fieldplot
    numsegs=30
    Rring=1.0

    dphi=2*pi/float(numsegs)
    seglist=[]
    phistart=0
    phiend=dphi
    for i in range(numsegs):
        seglist.append(HelixSeg(2,phistart,-Rring,phiend,-Rring,Rring).xyzSeg())
        seglist.append(HelixSeg(3,phiend,Rring,phistart,Rring,Rring).xyzSeg())
        phistart=phistart+dphi
        phiend=phiend+dphi

    seglistviewer(seglist)
    filename="/tmp/tmp.txt"
    fieldfind(Rring,seglist,filename)
    fieldplot(filename,XYZS=[False,True,True,True])
```

```
def fieldonline():
    Rring=1.0
    dyz=Rring*0.1
    plotaxial=3*Rring
    zvect=arange(-plotaxial,plotaxial,dyz)

    x=0
    y=0
    seglist=[[2,(-5.0,0.0,0.0),(5.0,0.0,0.0)]]
    result=[_Biot_Savart_LineSegments_at_Point(seglist,Point((x,y,z),type="CARTESIAN"))[1]
for z in zvect]
    import pylab
    pylab.plot(result)
    pylab.show()

def testline():
    from CoilViewer import seglistviewer
    from RingArray import fieldplot
    Rring=1.0
    seglist=[[2,(-5.0,0.0,0.0),(5.0,0.0,0.0)]]
    seglistviewer(seglist)
    filename="/tmp/tmp.txt"



    fieldfind(Rring,seglist,filename)
    fieldplot(filename,XYZS=[False,True,True,True])


def testing():
    #fieldonline()
    #testline()
    testring()


    #from fileaccess import *
    #zvect=[0.01*i for i in range(-30,31)]
    #fieldpoints=[[0,0,z] for z in zvect]
    #rings=[[1,0.2,.25]]
    #Rresult=Biot_Savart_Rings(rings,fieldpoints)
    #Rresult=[(z,mz) for z,(mx,my,mz) in zip(zvect,Rresult)]
    #Zresult=Biot_Savart_ZRings(rings,zvect)
    #Zresult=[(z,mz) for z,mz in zip(zvect,Zresult)]
    #writefloatarray("outR.txt",Rresult)
    #writefloatarray("outZ.txt",Zresult)
```

```
def commandlineapp():
    import sys
    from RingArray import fieldplot

    coilfile=sys.argv[1]
    fieldfile=sys.argv[2]

    print coilfile
    if coilfile.split(".")[-1] == "xml":
        c=smoothcoil().XML_load(coilfile,globals())
        print "opened using xml"
    elif coilfile.split(".")[-1] == "cPickle":
        c=smoothcoil().load(coilfile)
        print "opened using cPickle"

    Rring=c.Rformer
    seglist=c.seglist
    seglistviewer(seglist)
    fieldfind(Rring,seglist,fieldfile)
    fieldplot(fieldfile,XYZS=[True,False,True,True])

if __name__ == "__main__":
    #testing()
    commandlineapp()
```