100 lines of the SBM Python codes for 2D and 3D Laplace equations

1) Consider Laplace equation $\Delta u = 0$ with Dirichlet boundary conditions on a unit circular domain. The analytical solution is $u = x_1^2 - x_2^2$. In the SBM implementation, the empirical formula $\square_{L1}^{j} = -\dfrac{\ln\left(L_j/2\pi\right)}{2\pi}$ is used to determine the OIFs.

```
# SBM for 2D Laplace equation
# Copyright (C) 2020 Zhuojia FU, Hohai University
import numpy as np
import math
from scipy.special import *
from scipy.spatial.distance import cdist
from scipy.sparse.linalg import lgmres
import timeit
nknot=np.array([100,400])
error1=np.zeros([1, len(nknot)])
cputime=np.zeros([1, len(nknot)])
for itt in range(0,len(nknot)):
    Ra=1.0
    ntknot=nknot[itt]
    the=np.linspace(0.0,2*np.pi-2*np.pi/ntknot,ntknot)
    the.shape=(ntknot,1)
    meansA=2*np.pi*Ra/ntknot
    diagAU=-1.0/2.0/np.pi*(np.log(meansA/2/np.pi))
    Axy=np.column_stack((Ra*np.cos(the), Ra*np.sin(the)))
    distA=cdist(Axy,Axy,metric='euclidean')
    Alog=-1.0/2.0/np.pi*(np.log(distA))
    Alog[np.diag_indices_from(Alog)] = diagAU
    theta0=0.0
    bxy=Axy[:,0]**2-Axy[:,1]**2
    start = timeit.default_timer()
    rbfcoeff, exitCode = lgmres(Alog, bxy,atol=1.0E-10)
    print(exitCode)
    rbfcoeff.shape=(ntknot,1)
    cputime[0,itt] = (timeit.default_timer() - start)
    rho=0.8*Ra
    nnknot=201
    theta=np.linspace(0.0,2*np.pi-2*np.pi/nnknot,nnknot)
    theta.shape=(nnknot,1)
    sAxy=np.column_stack((rho*np.cos(theta), rho*np.sin(theta)))
    ff=sAxy[:,0]**2-sAxy[:,1]**2
    DMT = cdist(sAxy,Axy,metric='euclidean')
```

```
DMFS=-1.0/2.0/np.pi*(np.log(DMT))
fZI=np.dot(DMFS,rbfcoeff)
error1[0,itt]=np.linalg.norm(fZI.T-ff)/np.linalg.norm(ff)
```

2) Consider Laplace equation $\Delta u = 0$ with Dirichlet boundary conditions on a cylindrical domain with $a = 2$ and $h = 4$ as shown in Fig. A1. The analytical solution is $u = x_1^2 + x_2^2 - 2x_3^2$. In the SBM implementation, the SABT-based approach associated with the equation $\Xi(\mathbf{x}_m) = \int_{\partial\Omega} \phi_L(\mathbf{x}_m, \mathbf{s}) d(\partial\Omega) \approx \sum_{j=1}^{J} L_j \phi_L(\mathbf{x}_m, \mathbf{s}_j)$ is used to determine the OIFs, and the FFT is used to accelerate the SBM computing.

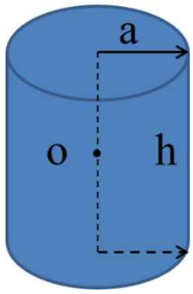

Fig. A1. Sketch of a finite circular cylinder.

```
# SBM for 3D Laplace equation
# Copyright (C) 2020 Zhuojia FU, Hohai University
import numpy as np
import math
from scipy.special import *
from scipy.spatial.distance import cdist
from scipy import linalg
from scipy.integrate import quad
import timeit
nknot=np.array([1,2,3])
error1=np.zeros([1, len(nknot)])
cputime=np.zeros([1, len(nknot)])
def generateA(Ra,Rc,hkk,kk,rkk):
        rho=Ra*np.sqrt(1.0/2.0/rkk*(2*(np.linspace(1,rkk,rkk))-1));
        theta=np.linspace(np.pi/kk,2*np.pi-np.pi/kk,kk);
        XI, YI = np.meshgrid(rho,theta)
        XI = XI.reshape(kk*rkk,order='F')
        YI = YI.reshape(kk*rkk,order='F')
        Rx=(XI*np.cos(YI)).T
        Ry=(XI*np.sin(YI)).T
        Rz=Rc/2*np.ones((1, rkk*kk))
        Rx.shape=(1,rkk*kk)
```

```
        Ry.shape=(1,rkk*kk)
        Rmatrix1= np.concatenate((Rx,Ry,Rz),axis=0)
        Rmatrix2= np.concatenate((Rx,Ry,-Rz),axis=0)
        XI=np.kron(np.ones((1, hkk)), Ra*np.cos(theta))
        YI=np.kron(np.ones((1, hkk)), Ra*np.sin(theta))
        ZI=np.kron(np.linspace(Rc/2-Rc/2/hkk,-Rc/2+Rc/2/hkk,hkk), np.ones((1, kk)))
        Imatrix= np.concatenate((XI,YI,ZI),axis=0)
        Axy=np.concatenate((Rmatrix1,Imatrix,Rmatrix2[:,::-1]),axis=1)
        return Axy,theta
def intergalcir(Ellipt1,A,Ra,Rc):
x=1/2/np.pi*Ra*np.log(Rc/2-A[2,0]+np.sqrt(Ra**2+A[0,0]**2+A[1,0]**2-2*Ra*np.sqrt(A[0,0]**2+A[1,0]**2)*np.cos(Ellipt1)+(Rc/2-A[2,0])**2))-1/2/np.pi*Ra*np.log(-Rc/2-A[2,0]+np.sqrt(Ra**2+A[0,0]**2+A[1,0]**2-2*Ra*np.sqrt(A[0,0]**2+A[1,0]**2)*np.cos(Ellipt1)+(-Rc/2-A[2,0])**2))+1/2/np.pi*Ra*np.sin(Ellipt1)*np.log(Ra*np.sin(Ellipt1)+np.sqrt(Ra**2+A[0,0]**2+A[1,0]**2-2*Ra*np.sqrt(A[0,0]**2+A[1,0]**2)*np.cos(Ellipt1)+(Rc/2-A[2,0])**2))+1/2/np.pi*Ra*np.sin(Ellipt1)*np.log(Ra*np.sin(Ellipt1)+np.sqrt(Ra**2+A[0,0]**2+A[1,0]**2-2*Ra*np.sqrt(A[0,0]**2+A[1,0]**2)*np.cos(Ellipt1)+(-Rc/2-A[2,0])**2))
        return x
def intergalcylinder(A,Ra,Rc):
x=-1/4/np.pi*((2*(Rc/2-A[2,0])*np.arctan((Ra-np.sqrt(A[0,0]**2+A[1,0]**2))/(Rc/2-A[2,0]))+(Ra-np.sqrt(A[0,0]**2+A[1,0]**2))*np.log((Ra-np.sqrt(A[0,0]**2+A[1,0]**2))**2+(Rc/2-A[2,0])**2)-2*Ra)-(2*(Rc/2-A[2,0])*np.arctan((-Ra-np.sqrt(A[0,0]**2+A[1,0]**2))/(Rc/2-A[2,0]))+(-Ra-np.sqrt(A[0,0]**2+A[1,0]**2))*np.log((-Ra-np.sqrt(A[0,0]**2+A[1,0]**2))**2+(Rc/2-A[2,0])**2)+2*Ra)+(2*(-Rc/2-A[2,0])*np.arctan((Ra-np.sqrt(A[0,0]**2+A[1,0]**2))/(-Rc/2-A[2,0]))+(Ra-np.sqrt(A[0,0]**2+A[1,0]**2))*np.log((Ra-np.sqrt(A[0,0]**2+A[1,0]**2))**2+(-Rc/2-A[2,0])**2)-2*Ra)-(2*(-Rc/2-A[2,0])*np.arctan((-Ra-np.sqrt(A[0,0]**2+A[1,0]**2))/(-Rc/2-A[2,0]))+(-Ra-np.sqrt(A[0,0]**2+A[1,0]**2))*np.log((-Ra-np.sqrt(A[0,0]**2+A[1,0]**2))**2+(-Rc/2-A[2,0])**2)+2*Ra))
        return x
for itt in range(0,len(nknot)):
    Ra,Rc,hkk,kk,rkk=2.0,4.0,10*nknot[itt],15*nknot[itt],10*nknot[itt]
    kka=kk
    kkb=2*rkk+hkk
    Axy,theta=generateA(Ra,Rc,hkk,kk,rkk)
    nxy=Axy.shape[1]
    meanoA=2.0*np.pi*Ra/kk*Rc/hkk*np.ones((1,hkk*kk));
    meantA=np.pi*Ra**2/rkk/kk*np.ones((1,rkk*kk));
    meansA=np.concatenate((meantA,meanoA,meantA[::-1]),axis=1)
    ttest1=np.zeros([1,int(nxy/kk)])
    ttest2=np.zeros([1,int(nxy/kk)])
    for ii in range(0,int(nxy/kk)):
        itemp=ii*kk+1
        Atemp=Axy[:,itemp]
        Atemp.shape=(3,1)
        ttest1[0,ii]=quad(lambda x: intergalcir(x,Atemp,Ra,Rc),0,np.pi)[0]
        ttest2[0,ii]=intergalcylinder(Atemp,Ra,Rc)
    sumdiagA1=ttest1+ttest2
    distA=cdist(Axy[:,np.arange(0, nxy, int(nxy/kkb))].T,Axy.T,metric='euclidean')
```

```
        Alog=1/4.0/np.pi/distA
        Alog[np.isinf(Alog)]=0.0
        diagAU=sumdiagA1/meansA[0,np.arange(0, nxy, int(nxy/kkb))]-(np.dot \
        (Alog,meansA.T).T)/meansA[0,np.arange(0, nxy, int(nxy/kkb))]
        iidiag = np.arange(0, 2*rkk+hkk, 1)
        Alog[iidiag, iidiag*int(nxy/kkb)]=diagAU[0,iidiag]
        TRa,TRc,Thkk,Tkk,Trkk=Ra*0.8,Rc*0.8,10,10,5
        sAxy,Ttheta=generateA(TRa,TRc,Thkk,Tkk,Trkk)
        Mxy=sAxy.shape[1]
        bxy=Axy[0,:]**2+Axy[1,:]**2-2*Axy[2,:]**2
        bxy.shape=(1,nxy)
        ff=sAxy[0,:]**2+sAxy[1,:]**2-2*sAxy[2,:]**2
        ff.shape=(Mxy,1)
        FmA=np.zeros([kkb,nxy])
        Fb=np.zeros([1,nxy])
        Frbfcoeff=np.zeros([1,nxy])
        rbfcoeff=np.zeros([1,nxy])
        start = timeit.default_timer()
        for iifft in range(0,kkb):
            for jjfft in range(0,kkb):
                FmA[iifft,jjfft*int(nxy/kkb)+np.arange(0, int(nxy/kkb), 1)]=np.fft.fft \
(Alog[iifft,jjfft*int(nxy/kkb)+np.arange(0, int(nxy/kkb), 1)])
            Fb[0,iifft*int(nxy/kkb)+np.arange(0, int(nxy/kkb), 1)]=np.fft.ifft(bxy[0,iifft* \
int(nxy/kkb)+np.arange(0, int(nxy/kkb), 1)])
        for iifft in range(0,kka):
            Frbfcoeff[0,np.arange(0, nxy, int(nxy/kkb))+iifft]=linalg.solve(FmA[:,(np.arange(0, 
nxy, int(nxy/kkb))+iifft)], Fb[0,np.arange(0, nxy, int(nxy/kkb))+iifft])
        for iifft in range(0,kkb):
            rbfcoeff[0,iifft*int(nxy/kkb)+np.arange(0, int(nxy/kkb), 1)]=np.fft.fft(Frbfcoeff \
[0,iifft*int(nxy/kkb)+np.arange(0, int(nxy/kkb), 1)])
        cputime[0,itt] = (timeit.default_timer() - start)
        DMT = cdist(sAxy.T,Axy.T,metric='euclidean')
        DMFS=1/4.0/np.pi/DMT
        fZI=np.dot(DMFS,rbfcoeff.T)
        error1[0,itt]=np.linalg.norm(fZI-ff)/np.linalg.norm(ff)
```