```
import  numpy as np
import  pylab as pl
from  scipy.linalg import expm
from  scipy.sparse.linalg import eigs

def  tebd(B_list,s_list,U,chi_list,chi_max):
    " Updates the B and s matrices using U_bond and the TEBD protocol "
    d = U.shape[0]

    for ibond in [0,1]:
        ia = np.mod(ibond,2); ib = np.mod(ibond+1,2)

        # Construct theta matrix (see Hastings)#
        C = np.tensordot(B_list[ia][:,:chi_list[ib],:chi_list[ia]],B_list[ib][:,:chi_list[ia],:chi_list[ib]],axes=(2,1))
        C = np.tensordot(C,U,axes=([0,2],[0,1]))
        theta = np.reshape(np.transpose(np.transpose(C)*s_list[ib][:chi_list[ib]],(1,3,0,2)),(d*chi_list[ib],d*chi_list[ib]))
        C = np.reshape(np.transpose(C,(2,0,3,1)),(d*chi_list[ib],d*chi_list[ib]))

        # Schmidt deomposition #
        theta[np.abs(theta)<10**(-10)] = 0
        X, Y, Z = np.linalg.svd(theta,full_matrices=0)
        Z = Z.T
        W = np.dot(C,Z.conj())
        chi_list[ia] = np.min([np.sum(Y>10.**(-8)), chi_max])

        # Obtain the new values for B and l #
        invsq = np.sqrt(sum(Y[:chi_list[ia]]**2))
        s_list[ia][:chi_list[ia]] = Y[0:chi_list[ia]]/invsq
        B_list[ia][:,:chi_list[ib],:chi_list[ia]] = np.reshape(W[:,:chi_list[ia]],(d,chi_list[ib],chi_list[ia]))/invsq
        B_list[ib][:,:chi_list[ia],:chi_list[ib]] = np.transpose(np.reshape(Z[:,:chi_list[ia]],(d,chi_list[ib],chi_list[ia])),(0,2,1))

def  representation(B_list,R,chi):
    # Construct the mixed transfermatrix
    T = np.tensordot(R,B_list[0],axes=(0,0))
    T = np.tensordot(T,np.conj(B_list[0]),axes=(0,0))
    T = np.tensordot(T,B_list[1],axes=(1,1))
    T = np.tensordot(R,T,axes=(0,3))
    T = np.tensordot(T,np.conj(B_list[1]),axes=([0,3],[0,1]))
    T = np.reshape(T,(chi**2,chi**2))

    # Obtain the dominant eigenvector
    eta,v = eigs(T,k=1)
    return eta,np.reshape(v,(chi,chi))

######## Define the simulation parameter ######################
chi_max=20; delta=0.1; N=2000;d=3

########### Define Ising Hamiltonian and get U ################
Sx = np.array([[0,1.,0],[1.,0,1.],[0,1.,0]])/np.sqrt(2)
Sy = np.array([[0,-1j,0],[1j,0,-1j],[0,1j,0]])/np.sqrt(2)
Sz = np.array([[1.,0,0],[0,0,0],[0,0,-1.]])

for   D in np.arange(0.0,2.0,0.4):
    ################ Get H and imaginary time evolution U #######
    H = np.kron(Sx,Sx) + np.kron(Sy,Sy) + np.kron(Sz,Sz) + D*np.kron(np.dot(Sz,Sz),np.eye(d))
    U = np.real(np.reshape(expm(-delta*H),(d,d,d,d)))

    ############### Initial state : |+-+-> + |0000> #############
    BA = np.zeros((d,chi_max,chi_max),dtype=float);BA[0,0,0] = 1.;BA[1,1,1] = 1.
    BB = np.zeros((d,chi_max,chi_max),dtype=float);BB[2,0,0] = 1.;BB[1,1,1] = 1.
    B_list = [BA,BB]
    s = np.zeros((chi_max));s[0:2] = 1.
    s_list = [s,s]
    chi_list = [np.array([2]),np.array([2])]

    ############### Find the Ground state #######################
    for step in range(1, N):
        tebd(B_list,s_list,U,chi_list,chi_max)

    ############### Get the topological invariant ###############
    eta_x,U_x = representation(B_list,expm(1j*np.pi*Sx),chi_max)
    eta_z,U_z = representation(B_list,expm(1j*np.pi*Sz),chi_max)
    I = np.trace(np.dot(np.dot(np.dot(U_x,U_z),np.conj(U_x.T)),np.conj(U_z.T)))
    print D
    print "--> Overlaps (Rx and Rz)   ", np.real(eta_x),np.real(eta_z)
    print "--> Entanglement spectrum: ",s_list[0][0:4]
    print "--> UxUzUx^+Uz^+:          ",np.real(I/np.abs(I))
    print
```