```
"""
Simulation of a Quantum Gate for a Superconductor-Based Quantum Processor

This work was performed under the supervision of Dr. Johannes Heinsoo (at Quantum Device Lab, ETH Zurich).
The thesis is titled "Simulations of CPHASE Gate Adiabaticity", which is
also in my google scholar page https://scholar.google.com/citations?user=MJVeIfQAAAAJ&hl=en

We consider two qutrits, a and b.
ee = state of two qutrits where both are in the excited state
fg = state of two qutrits where one of the qutrit is excited out of the computational subsystem and another is in the ground state

Qutrit frequencies are given in units of GHz, and time in units of ns. The CPHASE gate is executed by tuning the frequencies of two transmon-type qutrits to
resonance using magnetic flux pulse.

N.B. In this class, alpha parameter is introduced to add anharmonicity to the qutrit-spectrum, as required for the reasons of quantum information processing.
To keep the code short and concise for the purpose of delivering to Xanadu, only rabi oscillation for the ee and fg state is demonstrated. The ideal CPHASE
gate implementation requires 100% of the ee state population to come back to the computational subsystem. In this simulation, at gate execution time of 61 ns,
we see that over 99% of the population comes back to the computational subsystem, which is an encouraging result.

The script takes less than 20 seconds to produce the results!
"""

from __future__ import division

__author__ = 'Manish J. Thapa'

import math
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA
from scipy.special import erf
from scipy.linalg import expm

class CPHASE():
    """Defines parking frequencies for qutrits as well as flux pulse parameters"""
    def __init__(self,offset,sigma,delta,n=3,wa1=5.3724,wb1=4.8167,alpha=0.3,J=0.005893,h=1):
        self.offset=offset
        self.sigma=sigma
        self.delta=delta
        self.alpha=alpha
        self.wa1=wa1
        self.wb1=wb1
        self.J=J
        self.n=n
        self.h=h

    def fluxpulse(self,t,len):
        """Defines flux pulse which brings the two qutrits to resonance"""

        errf1=erf((len+2*self.offset-2*t)/(2*math.sqrt(2)*self.sigma))
        errf2=erf((len-2*self.offset+2*t)/(2*math.sqrt(2)*self.sigma))
        return 1/2*self.delta*(errf1+errf2)

    def Ha(self,t,len):
        """Bare Hamiltonian for qutrit a"""
        assert self.alpha != 0
```

```
        ae=self.wa1+self.fluxpulse(t,len)
        af=2*ae-self.alpha
        ha1=np.diag([0,ae,af])
        ha2=np.diag(np.ones(self.n))
        return self.h*np.kron(ha1,ha2)

    def Hb(self):
        """Bare Hamiltonian for qutrit b"""
        assert self.alpha != 0
        hb1=np.diag(np.ones(self.n))
        hb2=np.diag([0,self.wb1,2*self.wb1-self.alpha])
        return self.h*np.kron(hb1,hb2)

    def Hab(self):
        """Defining interaction Hamiltonian for qutrits a and b via J-coupling
        in terms of the projection operators
        """
        proj1a=np.zeros((self.n,self.n))
        proj1a[0,1] = 1
        proj2a=np.zeros_like(proj1a)
        proj2a[1,2] = math.sqrt(2)
        proja=proj1a+proj2a

        proj1b=np.zeros((self.n,self.n))
        proj1b[1,0]=1
        proj2b=np.zeros_like(proj1b)
        proj2b[2,1]=math.sqrt(2)
        projb=proj1b+proj2b
        return self.h*np.kron(proja,projb)

    def H(self,t,len):
        """Total Hamiltonian of the two-qutrit system"""
        return self.Ha(t,len)+self.Hb()+self.J*self.Hab()+self.J*self.Hab().T

    def spectrum(self,t,len):
        """Energy levels of the total Hamiltonian"""
        eigvals=LA.eigvalsh(self.H(t,len))
        ee=eigvals[4]
        fg=eigvals[5]
        return ee,fg #these states exhibit avoided crossing near their "sweet spot"

    def unitary(self,len):
        """Using Baker-Campbell-Hausdorff formula to simulate CPHASE gate"""
        ham = lambda t : self.H(t,len)
        approxU=np.diag(np.ones(self.n**2))
        N=600
        tinit=0
        tfinal=120 #simulation window
        dt=(tfinal-tinit)/N
        evolutiongrid=np.linspace(tinit,tfinal,N)
        for t in evolutiongrid:
            approxU=np.dot(expm(-1j*2*math.pi*ham(t)*dt),approxU)
        return approxU

if __name__ == "__main__":

    cphase=CPHASE(delta=-0.255,offset=57.3571,sigma=1.7857) #defining more gate parameters

    pulselength=50 #example pulse length (or gate time)
    f=lambda t : cphase.fluxpulse(t,pulselength)
    tgrid= np.linspace(0,50,50) #propagation window
    fgrid=f(tgrid) #flux pulse amplitudes
```

```
    ee=np.zeros_like(tgrid)
    fg=np.zeros_like(tgrid)

    for i,proptime in enumerate(tgrid):
        spec = lambda t : cphase.spectrum(t,pulselength)
        ee[i],fg[i]=spec(proptime)
        print 'energies ee = %g, energies fg = %g' % (ee[i], fg[i])

    f, ax = plt.subplots(2, sharex=False,figsize=(2.8,4.5))

    ax[0].plot(tgrid, ee,'r',label='ee')
    ax[0].plot(tgrid, fg,'g',label='fg')
    ax[0].set_xlabel('$t (\\mathrm{ns})$',fontsize=15)
    ax[0].set_ylabel('$\\nu (\\mathrm{GHz})$',fontsize=15)
    ax[0].legend(loc='upper right', borderpad=0.25, labelspacing=0.3,fontsize ='small')

    ee=np.array([0,0,0,0,1,0,0,0,0]) #qutrits in ee state
    fg=np.array([0,0,0,0,0,0,1,0,0]) #qutrits in fg state

    tlen=np.linspace(0,61,30) #array of gate execution times

    popee=np.zeros_like(tlen)
    popfg=np.zeros_like(tlen)

    for i,gatetime in enumerate(tlen):
        popee[i]=abs(np.dot(np.dot(ee.transpose(),cphase.unitary(gatetime)),ee))**2
        popfg[i]=abs(np.dot(np.dot(fg.transpose(),cphase.unitary(gatetime)),ee))**2
        if popee[i] < 0:
            print 'WARNING: negative populations!'
            exit()
        print 'population of state ee for gate time '+str(gatetime)+' ns =' , popee[i]

    ax[1].plot(tlen, popee,'r--',label='ee')
    ax[1].plot(tlen, popfg,'g--',label='fg')
    ax[1].set_xlabel('$t_{\mathrm{len}} (\\mathrm{ns})$',fontsize=15)
    ax[1].set_ylabel('$\\mathrm{Population}$',fontsize=15)
    ax[1].legend(loc='upper right', borderpad=0.25, labelspacing=0.3,fontsize ='small')
    plt.tight_layout(pad=0.1)
    plt.savefig('cphase.pdf', dpi=600)
    plt.show()
```