```
"""
module to compute the integrand in ccf

todo:
  - insert into sfTrident
  - connect to kinClass
  - use Integrator (asimps)
  - compare to myCalc with CCF (is it possible? -> boundarys)
"""

from phaseInt import B0ccf


def func(rStar,kinObj):
    photoNumC=rStar #insert translator
    photoNumBW = 1-photoNumC #insert BW
    alphasC = kinObj.getAlpha('c')
    alphasBW = kinObj.getAlpha('bw')
    return B0ccf(photoNumC,alphaC[0]/2.0,alphaC[2]/3.0)*B0ccf(photoNumBW,alphaBW
[0]/2.0,alphaBW[2]/3.0)

def integFunc(rStar,kinObj):
    return (func(rStar,kinObj) - func(-rStar,kinObj))/rStar
```

```
"""
module to compute the general phase integral in CCF with airy functions
"""
import numpy as np
from scipy.special import airy
import matplotlib.pylab as plt

def b(e,c1,c2):
    return 2.0*np.pi*np.exp(1j*e)/((3.0*c2)**(1.0/3.0))

def eta(r,c1,c2):
    return -r*c1/3.0/c2 + 2.0*c1**3/(27*c2**2)

def mu(r,c1,c2):
    return (r-c1**2/(3.0*c2))/((3.0*c2)**(1.0/3.0))


def B0ccf(r,c1=0.0,c2=0.0):
    et = eta(r,c1,c2)
    prefac = b(et,c1,c2)
    arg = mu(r,c1,c2)
    return prefac*airy(arg)[0]

def B1ccf(r,c1,c2):
    et = eta(r,c1,c2)
    prefac = b(et,c1,c2)
    arg = mu(r,c1,c2)
    fac1 = c1/3.0/c2*airy(arg)[0]
    fac2 =1j/((3.0*c2)**(1.0/3.0))*airy(arg)[1]
    return -prefac*(fac1 + fac2)

def B2ccf(r,c1,c2):
    et = eta(r,c1,c2)
    prefac = b(et,c1,c2)
    arg = mu(r,c1,c2)
    fac1 = ((c1/3.0/c2)**2 - arg/((3.0*c2)**(2.0/3.0)))*airy(arg)[0]
    fac2 =1j*2.0*c1/((3.0*c2)**(4.0/3.0))*airy(arg)[1]
    return prefac*(fac1 + fac2)

def B0reg(r,c1,c2):
    return -1.0/r*(2.0*c1*B1ccf(r,c1,c2) + 3.0*c2*B2ccf(r,c1,c2))

def f(rStar):
    return (rStar +15.2666666667
)/4.0970436

if __name__=='__main__':
    args = np.linspace(-30,30,2000)
    print args.shape
    #func0=lambda x: B0ccf(f(x),0.976313749749,0.325437916583)#*B0ccf(6.89928383
156-f(x),-2.92894124925,0.976313749749)
    func0=lambda x: B0ccf(x,1.0,0.3)
    #func0reg=lambda x: B0reg(f(x),0.976313749749,0.325437916583)#*B0reg(6.89928
383156-f(x),-2.92894124925,0.976313749749)
    func0reg=lambda x: B0reg(x,1.0,0.3)
    integFunc0 = lambda x: np.imag((func0(x) - func0(-x))/x)
    integFunc0reg = lambda x: np.imag((func0reg(x) - func0reg(-x))/x)
    #vals = np.array(map(func,args))
    #als0=integFunc0(args)
    vals0=func0(args)
    #vals0reg=integFunc0reg(args)
    vals0reg=func0reg(args)
    #print vals.shape
    print vals0reg/vals0
    plt.plot(args,vals0,label='analytic')
    plt.plot(args,vals0reg,label='reg.')
```



```
    plt.legend()
    plt.show()
```