```
from scipy.special import pro_cv, pro_cv_seq, pro_ang1, pro_rad1, pro_rad2
import numpy as np
from numpy import *

MAX = 500
NTRY = 500
KMAX = 500
def SWFevalue(m,n,sign,c,smax,eps):
    csq = sign*c*c
    c2D2 = 0.5*csq
    c4 = csq*csq
    nMm = n-m
    fm2M1 = 4*m*m- 1
    r1Start = 0 if nMm%2 == 0 else 1
    r = r1Start
    fr = r
    tmPr = 2*m+r
    mPr = m+r
    tmPtrM1 = 2*mPr-1
    for s in range(r1Start,(nMm+1)//2+1):
        B1[s] =
(fr+2)*(fr+1)*(tmPr+2)*(tmPr+1)*c4/((tmPtrM1+4)*(tmPtrM1+4)*(tmPtrM1+2)*(tmPtrM1
+6))
        G1[s] = mPr*(mPr+1)+c2D2*(1-fm2M1/(tmPtrM1*(tmPtrM1+4)))
        fr += 2
        tmPr += 2
        tmPtrM1 += 4
        mPr += 2
    r2Start = nMm+2*smax
    fr = r2Start
    tmPr = 2*m+r2Start
    mPr = m+r2Start
    tmPtrM1=2*mPr-1
    for s in range(1,smax+1):
        B2[s]= fr*(fr-1)*tmPr*(tmPr-1)*c4/(tmPtrM1*tmPtrM1*(tmPtrM1-
2)*(tmPtrM1+2))
        G2[s]= mPr*(mPr+1)+c2D2*(1-fm2M1/(tmPtrM1*(tmPtrM1+4)))
        fr -= 2
        tmPr -= 2
        tmPtrM1 -= 4
        mPr -= 2
    if n == 0:
        if abs(c) < 3:
            Lest = LPS(m,n,csq)
        else:
            Lest = LAS(m,n,sign,c)
    else:
        if abs(c) < 4:
            Lest = LPS(m,n,csq)
        else:
            Lest = LAS(m,n,sign,c)
    Lu = Lest+0.1
    Ld = Lest-0.1
    Uu = U(B1, G1, B2, G2, m, n, r1Start, r2Start, smax, Lu)
    Ud = U(B1, G1, B2, G2, m, n, r1Start, r2Start, smax, Ld)
    k = 1
    while Uu*Ud > 0:
        Lstep= Lu- Ld
        if abs(Uu) < abs(Ud):
            Lu += Lstep
            Uu = U(B1, G1, B2, G2, m, n, r1Start, r2Start, smax, Lu)
        else:
            Ld -= Lstep
            Ud = U(B1, G1, B2, G2, m, n, r1Start, r2Start, smax, Ld)
```

```
        if k > NTRY:
            print(f'\n SWFevalue bracket > {NTRY} tries : 0 returned')
            return(0)
        k = k+1
    Uu = U(B1, G1, B2, G2, m, n, r1Start, r2Start, smax, Lu)
    Umid = U(B1, G1, B2, G2, m, n, r1Start, r2Start, smax, Ld)
    if Uu < 0:
        dL = Ld-Lu
        Lrt = Lu
    else:
        dL = Lu-Ld
        Lrt = Ld
    k = 0
    while (abs(dL) >= eps) and (Umid != 0):
        dL *= 0.5
        Lmid = Lrt+dL
        Umid = U(B1, G1, B2, G2, m, n, r1Start, r2Start, smax, Lmid)
        if (Umid <= 0):
            Lrt = Lmid
        k = k+1
        if k > KMAX:
            print(f'\n > {KMAX} tries to bisect root')
            Umid = 0
    return(Lrt)

def LAS(m, n, sign, c):
    if sign > 0:
        q = 2*(n - m) + 1
        q2 = q*q
        q3 = q2*q
        q4 = q3*q
        q5 = q4*q
        q6 = q5*q
        fm = m
        m2 = fm*fm
        m4 = m2*m2
        m6 = m4*m2
        L0 = c*q+m2-(q2+5)/8
        L1 = -q*(q2+11-32*m2)/64
        L2 = -(5*(q4+26*q2+21)-384*m2*(q2+1))/1024
        L3 = -q*((33*q4+1594*q2+5621)/128-m2*(37*q2+167)+m4/8)/128
        L4 =
-((63*q6+4940*q4+43327*q2+22470)/65536-m2*(115*q4+1310*q2+735)/512+3*m4*(q2+1)/
8)
        L5 = -q*((527*q6+61529*q4+1043961*q2+2241599)/1048576-
m2*(5739*q4+127550*q2+298951)/32768+m4*(355*q2+1505)/512-m6/16)
    else:
        nMm = n-m
        if nMm % 2 == 0:
            nu = nMm/2
            q = n + 1
        else:
            nu = (nMm-1)/2
            q= n
        q2 = q*q
        q3 = q2*q
        q4 = q3*q
        q5 = q4*q
        q6 = q5*q
        fm = m
        m2 = fm*fm
        m4 = m2*m2
        m6 = m4*m2
        L0 = -c*c+2*c*(2*nu+fm+1)-2*nu*(nu+fm+1)-(fm + 1)
```

```
        L1 = -q*(q2+1- m2)/8
        L2 = -(5*q4+10*q2+1-2*m2*(3*q2+1)+m4)/64
        L3 = -q*(33*q4+114*q2+37-2*m2*(23*q2+25)+13*m4)/512
        L4 = -(63*q6+340*q4+239*q2+14-10*m2*(10*q4+23*q2+3)+m4*(39*q2-18)-
2*m6)/1024
        L5 = 0
    cR = 1/c
    LLAS = L0+(L1+(L2+(L3+(L4+L5*cR)*cR)*cR)*cR)*cR
    return(LLAS)

def LPS(m, n, csq):
# Suite de puissances pour valeurs propres de fonctions d onde spheroidales
    L0=n*(n+1)
    tm = 2*m
    tn = 2*n
    tnM1 = tn-1
    tnM3 = tn-3
    tnM5 = tn-5
    tnM7 = tn-7
    tnP1 = tn+1
    tnP3 = tn+3
    tnP5 = tn+5
    tnP7 = tn+7
    tnP9 = tn+9
    nMm = n-m
    nMmM1 = nMm-1
    nMmM2 = nMm-2
    nMmM3 = nMm-3
    nMmP1 = nMm+1
    nMmP2 = nMm+2
    nMmP3 = nMm+3
    nMmP4 = nMm+4
    nPm = n+m
    nPmM1 = nPm-1
    nPmM2 = nPm-2
    nPmM3 = nPm-3
    nPmP1 = nPm+1
    nPmP2 = nPm+2
    nPmP3 = nPm+3
    nPmP4 = nPm+4
    L2 = 0.5*(1-((tm-1)*(tm+1))/(tnM1*tnP3))
    L4 = 0.5*(-nMmP1*nMmP2*nPmP1*nPmP2/(pow(tnP3,3)*tnP5)+nMmM1*nMm*nPmM1*nPm/
(tnM3*pow(tnM1,3)))/tnP1
    L6 = (tm*tm-1)*(nMmP1*nMmP2*nPmP1*nPmP2/(pow(tnP3,4)*tnP5*tnP7)-
nMmM1*nMm*nPmM1*nPm/(tnM5*tnM3*pow(tnM1,4)))/(tnM1*tnP1*tnP3)
    A = (nMmM1*nMm*nPmM1*nPm/(pow(tnM5,2)*tnM3*pow(tnM1,5))-
nMmP1*nMmP2*nPmP1*nPmP2/(pow(tnP3,5)*tnP5*tnP7*tnP7))/(tnM1*tnM1*tnP1*tnP3*tnP3)
    B =
(nMmM3*nMmM2*nMmM1*nMm*nPmM3*nPmM2*nPmM1*nPm/(tnM7*tnM5*tnM5*pow(tnM3,3)*pow(tnM
1,4))-nMmP1*nMmP2*nMmP3*nMmP4*nPmP1*nPmP2*nPmP3*nPmP4/
(pow(tnP3,4)*pow(tnP5,3)*tnP7*tnP7*tnP9))/tnP1
    C = (pow(nMmP1*nMmP2*nPmP1*nPmP2,2)/(pow(tnP3,7)*tnP5*tnP5)-
pow(nMmM1*nMm*nPmM1*nPm,2)/(tnM3*tnM3*pow(tnM1,7)))/(tnP1*tnP1)
    D =
nMmM1*nMm*nMmP1*nMmP2*nPmM1*nPm*nPmP1*nPmP2/(tnM3*pow(tnM1*tnP3,4)*tnP1*tnP1*tnP
5)
    L8 = 2*pow(tm*tm-1,2)*A+B/16+C/8+D/2
    LLPS = L0+(L2+(L4+(L6+L8*csq)*csq)*csq)*csq
    return(LLPS)

def U (B1, Gl, B2, G2, m, n, r1Start, r2Start, smax, LL):
# Fraction continue avec suite U1 de valeurs propres de fonction d onde
spheroidale ; fini
    if n == m+r1Start:
```

```
        U1= G1[r1Start]-LL
    else:
        U1 = B1[r1Start]/(Gl[r1Start]-LL)
        sStop = (n-m-1)/2
        s = r1Start+1
        while s <= sStop:
            U1=B1[s]/(G1[s]-LL-U1)
            s = s+1
        U1 = G1[s]-LL-U1
    # suite U2 ; infinie, mais tronquee apres smax termes
    s = 1
    U2 = B2[1]/(G2[1]-LL)
    while s <= smax:
        U2 = B2[s]/(G2[s]-LL-U2)
        s = s+1
    return(U1-U2)

def SWFAngCoeff(m, n, sign, c, smax, TMNeps, dmn, tmn):
# coefficients de fonction angulaire spherique, utilisant tmn (cf Little et
Corbato [Stra56]) avec iteration backward
    csq = sign*c*c
    # valeur propre modifiee
    tmn = SWFevalTMN(m,n,csq, smax, TMNeps);
    # construire un tableau de coefficients
    nMm = n-m
    rSt = 0 if nMm % 2 == 0 else 1
    rLarge = nMm+14;
    dmn[rLarge] = 1;
    for r in range(rrlarge, rSt+2, -2):
        if csq == 0:
            dmn[r] = 1 if r == nMm else 0
        else:
            dmn[r-2] = -bbtmn(m,n,r-nMm,csq, tmn)*dmn[r]/cc(m,r,csq) if
r==rLarge else
-(bbtmn(m,n,r-nMm,csq,tmn)*dmn[r]+aa(m,r,csq)*dmn[r+2])/cc(m,r,csq)
    # Normalisation selon le schema de Flammer (cf [Fla57])
    tm = m+m
    tmM1 = tm - 1
    fm = m
    fnPm = n + m
    fnMm = nMm
    trSt = 2*rSt
    frSt = rSt
    term = exp(LogGamma(tm+1+trSt)-LogGamma(fm+frSt+1))/pow(2,frSt)
    sum = term*dmn[rSt]
    for r in range(rSt+2, rLarge, 2):
        fr = r
        term *= -(fr+tmM1+frSt)/(fr-frst)
        sum += term*dmn[r]
    phase = 1 if (nMm-rSt) % 4 == 0 else -1
    Norm = phase*exp(LogGamma(fnPm+frSt+1)-LogGamma(0.5*(fnMm-frSt)+1)-
LogGamma(0.5*(fnPm+frSt)+1))/(pow(2,fnMm)*sum)
    for r in range(rrst,rLarge+1,2):
         dmn[r] *= Norm
    return()

def aa(m, r, csq):
# terme alpha pour la recurrence des coeff des fonctions d onde spheroidale
    tmPrP1 = 2*m+r+1
    tmPtrP3 = 2*(m+r)+3
    return((tmPrP1+1)*tmPrP1*csq/(tmPtrP3*(tmPtrP3+2)))

def bbtmn(m, n, s, csq, tmn):
# terme beta pour la recurrence des coeff de fonction d onde spheroidale ;
```

```
modifie pour la fonction auxiliaire tmn
    fm = m
    tn = 2*n
    ts = 2*s
    fs = s
    tnPts = tn+ts

return(fs*(tn+fs+1)*(1+csq*(2*(4*fm*fm-1)/((tn-1)*(tn+3)*(tnPts-1)*(tnPts+3))))-
csq*tmn)

def cc(m, r, csq):
# terme gamma pour la recurrence des coeff de fonction d onde spheroidale
    fr = r
    tm = 2*m
    tmPtr = tm+2*fr
    return(fr*(fr-1)*csq/((tmPtr-3)*(tmPtr-1)))

def SWFevalTMN(m, n, csq, smax, TMNeps):
# valeur propre de fonction d onde spheroidale sous forme tmn pour le degre n, l
ordre m, l argument csq : utilise smax termes dans la fraction continue et
obtient la valeur propre modifiee avec precision TMNeps
    if csq == 0:
        return(0)
    # construire les tableaux de termes dans les fractions continues qui sont
independantes de la valeur propre
    c2D2 = 0.5*csq
    tcsq = 2*csq
    c4 = csq*csq
    nMm = n-m
    fm2M1 = 4*m*m-1
    # Elements pour la fonction U1
    r1Start = 0 if nMm % 2==0 else 1
    r = r1Start
    fr = r
    tmPr = 2*m+r
    mPr= m+r
    tmPtrM1= 2*mPr-1
    tn = n+n
    tnM13 = (tn-1)*(tn+3)
    s = fr - nMm
    for t in range(r1Start,(nMm+1)//2+1):
        B1[t] =
(fr+2)*(fr+1)*(tmPr+2)*(tmPr+1)*c4/((tmPtrM1+4)*(tmPtrM1+4)*(tmPtrM1+2)*(tmPtrM1
+6))
        tnPtsMl = tn+2*s-1
        GTT1[t] = s*(tn+s+1)*(1+tcsq*fm2M1/(tnM13*tnPtsM1*(tnPtsM1+4)))
        fr += 2
        tmPr += 2
        tmPtrM1 += 4
        mPr += 2
        s += 2
    # Elements pour la fonction U2 ; r commence en n-m+2smax pour smax termes
    r2Start=nMm+2*smax
    fr = r2Start;
    tmPr = 2*m+r2Start
    mPr = m+r2Start
    tmPtrM1 = 2*mPr-1
    s= 2*smax
    for t in range(1, smax+1):
        B2[t] = fr*(fr-1)*tmPr*(tmPr-1)*c4/(tmPtrM1*tmPtrM1*(tmPtrM1-
2)*(tmPtrM1+2))
        tnPtsMl = tn+2*s-1
        GTT2[t] = s*(tn+s+1)(1+tcsq*fm2M1/(tnM13*tnPtsM1*(tnPtsM1+4)))
        fr -= 2
```

```
        tmPr -= 2
        tmPtrM1 -= 4
        mPr -= 2
        s -= 2;
        # valeur propre modifiee estimee multipliee par le carre de c
        cTMNest = -0.01*csq
        # encadrement de la valeur propre modifiee
        cTMNu = cTMNest*1.1
        cTMNd = cTMNest*0.9
        Uu = U(B1, GTT1, B2, GTT2, m, n, r1start, r2Start, smax, cTMNu)
        Ud = U(B1, GTT1, B2, GTT2, m, n, r1Start, r2Start, smax, cTMNd)
        k= 1
        while Uu*Ud > 0:
            cTstep = cTMNu - cTMNd
            if abs (Uu) < abs (Ud):
                cTMNu += cTstep
                Uu = U(B1, GTT1, B2, GTT2, m, n, r1Start, r2Start, smax, cTMNu)
            else:
                cTMND -= cTstep
                Ud = U(B1, GTT1, B2, GTT2, m, n, r1Start, r2Start, smax, cTMNd)
            if k> NTRY:
                print(f'\n SWFevalTMN bracket > {NTRY} tries : 0 returned')
                return(0)
            k = k+1
        # Raffiner la racine a une precision fractionnaire de TMNeps par la
methode de bisection ; commencer en cTMNu et TMNd
        Uu = U(B1, GTT1, B2, GTT2, m, n, r1Start, r2Start, smax, cTMNu)
        Umid = U(B1, GTT1, B2, GTT2, m, n, r1Start, r2Start, smax, cTMNd);
        # trouver la direction telle que U>0 pour CTMNrt+dcTMN
        if Uu < 0:
            dcTMN = cTMNd - cTMNu
            cTMNrt = cTMNu
        else:
            dcTMN = cTMNu - cTMNd
            cTMNrt = cTMNd
        k = 0
        while (abs(dcTMN) >= TMNeps) and (Umid != 0):
            dcTMN *= 0.5;
            cTMNmid = cTMNrt+dcTMN
            Umid = U(B1, GTT1, B2, GTT2, m, n, r1Start, r2Start, smax, cTMNmid)
            if Umid <= 0:
                cTMNrt = cTMNmid
            k = k+1
            if k > KMAX:
                print(f'\n > {KMAX} tries to bisect root')
                Umid = 0
        return(xTMNrt/csq)

def U(B1, GTT1, B2, GTT2, m, n, r1Start, r2Start, smax, cTMN):
# fraction continue avec valeur propre de la fonction d onde spherique modifiee
TMN fois csq
    # suite U1 ; finie
    if n == m:
       U1 = GTT1[0]-cTMN
    else:
       if n == m+1:
          U1 = GTT1[1]-cTMN
       else:
          U1 = B1[r1Start]/(GTT1[r1Start]-cTMN)
          sStop = (n-m-1)/2
          s = r1Start+1
          while s <= sStop:
             U1 = B1[s]/(GTT1[s]-cTMN-U1)
             s = s+1
```

```
        U1 = GTT1[s]-cTMN-U1
    # suite U2, infinie, mais tronquee apres smax termes
    s = 1;
    U2 = B2[1]/(GTT2[1]-cTMN)
    while s <= smax:
        U2 = B2[s]/(GTT2[s]-cTMN-U2)
        s = s+1
    return(U1-U2)

def S1MN(m, n, eta, dmn, Seps):
  # fonction spheroidale angulaire du premier type ; degre n, ordre m, argument
eta. Convergence vers une precision fractionnaire Seps
  nMm = n-m
  rStart = 0 if nMm % 2 == 0 else 1
  rLarge = nMm+14 # domaine des coefficients environ 10**15
  sum = 0
  ratio = 10
  r = rStart
  while ratio > Seps:
     if r > rLarge:
        print(f'\n\n S1NM unconverged to {Seps:.6E} in {rLarge} terms')
        return(sum)
     term = dmn[r]*PNM(m+r,m,eta)
     sum += term
     ratio = abs(term/sum)
     r += 2
     return(sum)

def S2MN(m, n, eta, dmn, Seps):
# fonction spheroidale angulaire du second type ; degre n, ordre m, argument
eta. Convergence vers une precision fractionnaire Seps
  nMm = n-m
  rStart = 0 if nMm % 2 == 0 else 1
  rLarge = nMm+14 # coefficient range about 10**15
  sum = 0
  ratio = 10
  r = rStart
  while ratio > Seps:
     if r > rLarge:
        print(f'\n\n S2NM unconverged to {Seps:.6} in {rLarge} terms')
        return(sum)
     term = dmn[r]*QNM(m+r,m,eta)
     sum += term
     ratio = abs(term/sum)
     r += 2
  return(sum)

B1 = np.zeros(1000)
B2 = np.zeros(1000)
G1 = np.zeros(1000)
G2 = np.zeros(1000)
dmn = np.zeros(1000)
GTT1 = np.zeros(1000)
GTT2 = np.zeros(1000)
signe = +1 # prolate
# signe = -1 # oblate
c = (2**0.5)/2
nbtermesfraccontinue = 4
epsilon = 0.01
m = 0
print('signe = ',signe, ' c = ',c,' nb termes frac cont. =
',nbtermesfraccontinue,' epsilon = ',epsilon,' m = ',m)
for n in range(-11,11):
    res = SWFevalue(m,n,signe,c,nbtermesfraccontinue,epsilon)
```

```
print('n = ',n,' --> ',res,' procv = ',pro_cv(0,n,c))
```