```
import numpy as np
import scipy.linalg
import sys

from nbtypes import unit_x, unit_y, unit_z

def kvec_to_symmetric_gradient(kvec):
    '''Calculates 6x3 symmetric gradient matrix for plane wave of wavevector kvec.

  See Auld I, eq 1.53
  '''
    kx, ky, kz = kvec

    nabla_I_J = np.array([
        [kx,   0.0, 0.0],
        [0.0,  ky,  0.0],
        [0.0,  0.0, kz],
        [0,    kz,  ky],
        [kz,   0.0, kx],
        [ky,   kx, 0.0]
    ])
    return nabla_I_J


def power_flux_christoffel(kapv, v_p, evec, c_stiff):
    '''Evaluates the power flux P=-v^* \cdot T for a given unit wavevector kapv, eigenvector evec and implicit wa
venumber k and frequency omega.

  Factors of 2 seem to be correct here, but good to write out in full in docs.
  '''

    # S_I = \nabla_Ij (uj e^i(k kap . r)) = i k (\nabla_Ij  e^i(k kap . r)) . uj
    # Evaluate the S_I 6x1 vector, Auld 1.50, 1.49
    S_I = 1j * np.matmul(kvec_to_symmetric_gradient(kapv), evec)

    T_I = np.matmul(c_stiff.as_zerobase_matrix(), S_I)  # Auld 3.20   # Indices
off by 1 from zero count

    # Pcomp = - 1/2 v^* . T ,   Auld 2.30       , 5.77
    #       =  -1/2 (i \omega u^*) .  T
    #om = 1.0 # unit k and omega
    vsx, vsy, vsz = 1j*np.conj(evec)
    Pcomp = - 0.5 * np.array([
        vsx*T_I[0] + vsy*T_I[5] + vsz*T_I[4],
        vsx*T_I[5] + vsy*T_I[1] + vsz*T_I[3],
        vsx*T_I[4] + vsy*T_I[3] + vsz*T_I[2] ])

    # u_s = 1/2  k^2 S_I c_IJ SJ  # Auld, 5.35
    u_s = .5*np.matmul(np.matmul(S_I, c_stiff.as_zerobase_matrix()), S_I)  # rea
l vs complex fields?

    # vg = Pcomp/u_s ->  Pcomp/us   (omega k)/(k^2) = v_p Pcomp/us
    v_g = - np.real(v_p * Pcomp/u_s)
    #print('powers', kapv, v_p, u_s, Pcomp, v_g)
    return v_g



def Gamma_christoffel(vkap, c_stiff, rho):
    '''Returns Gamma_ij = 1/V0^2  mD.Cij.md^T/rho  in units of (km/s)^2
  vkap is unit wavevector.
  V0=1km/s
```



```
  See Auld V1. Sec 7.D
  '''
    (kapx, kapy, kapz) = vkap
    v0sq = 1e6

    mD = np.array([
        [kapx, 0,    0,    0,    kapz, kapy],
        [0,    kapy, 0,    kapz, 0,    kapx],
        [0,    0,    kapz, kapy, kapx, 0]])

    m_Gamma = np.matmul(np.matmul(mD, c_stiff.as_zerobase_matrix()), mD.T)/(v0sq
*rho)

    return m_Gamma

def chareq_christoffel(vkap, c_stiff, rho, v_p):
    '''Returns Om=|Gamma_ij -\omega^2 I|, the characteristic function of the Christoffel equation.

  v_p is in km/s
  See Auld V1. Sec 7.D
  '''
    return scipy.linalg.det(Gamma_christoffel(vkap, c_stiff, rho)-v_p**2*np.eye(
3))

def solve_christoffel(vkap, c_stiff, rho):
    '''Solve eigenproblem of Christoffel equation in the direction vkap (a 2D unit vector). Returns for each of 3 mo
des:
    phase velocity          v_phase[m]
    polarisation eigenvetors    evecs[:,m]
    group velocity vectors.     v_group[m:x/y/z]

  Modes are sorted by decreasing phase velocity.

  See Auld V1. Sec 7.D
  '''

    m_Gamma = Gamma_christoffel(vkap, c_stiff, rho)

    # Solve and normalise
    evals, evecs = scipy.linalg.eig(m_Gamma)
    for i in range(3):
        evecs[:, i] /= np.linalg.norm(evecs[:, i])
        # TODO: make a oneliner:"
        # evecs *= 1/np.sqrt(np.diag(np.real(evecs.T @ evecs)))

    vels = np.sqrt(np.real(evals))  # result is in km/s

    # orthos = np.array([
    #     np.dot(evecs[:,0], evecs[:,1]),
    #     np.dot(evecs[:,0], evecs[:,2]),
    #     np.dot(evecs[:,1], evecs[:,2]) ])
    # print(np.abs(orthos).max())

    # Sort according to velocity

    ivs = np.argsort(-vels)  # most likely get pwave first

    v_vphase = np.sqrt(np.real(evals[ivs])).copy()
    v_evecs = evecs[:, ivs].copy()

    # now look for vg here
```

```
    # vg = - nabla_k Om/ dOm/dom = nabla_kappa Om/ dOm/dvp =

    v_vgroup = np.zeros([3,3])  # first index is mode, second is component of \v
ec{v}_g


    dkap = 0.005  # about 0.5 %
    dvel = 0.005   # about 0.5 %

    # with open('tt.dat', 'w') as fout:

    #     for jj in range(1000):
    #         v_p = 7*jj/1000.
    #         dOmdkapx = (chareq_christoffel(vkap+dkap*unit_x, c_stiff, rho, v_p
)-
    #              chareq_christoffel(vkap-dkap*unit_x, c_stiff, rho, v_p))/(2*d
kap)
    #         dOmdvp = (chareq_christoffel(vkap, c_stiff, rho, v_p+dvel)-
    #           chareq_christoffel(vkap, c_stiff, rho, v_p-dvel))/(2*dvel)
    #         rat = -dOmdkapx/(dOmdvp+1e-14)

    #         fout.write(f'{jj}  {v_p}  {dOmdkapx} {dOmdvp} {rat}  \n')

    # sys.exit(0)
    for m in range(3): # for each mode at this vkap
        v_p = v_vphase[m]
        v_g = power_flux_christoffel(vkap, v_p, v_evecs[:,m], c_stiff)
        v_vgroup[m,:] = v_g

        # for ii in range(10):
        #     dkk = dkap /2**ii
        #     print('dk005', ii, (chareq_christoffel(vkap+dkk*unit_x, c_stiff, r
ho, v_p)-
        #          chareq_christoffel(vkap-dkk*unit_x, c_stiff, rho, v_p))/(2*dk
k))


        # dOmdkapx = (chareq_christoffel(vkap+dkap*unit_x, c_stiff, rho, v_p)-
        #          chareq_christoffel(vkap-dkap*unit_x, c_stiff, rho, v_p))/(2*d
kap)

        # dOmdkapy = (chareq_christoffel(vkap+dkap*unit_y, c_stiff, rho, v_p)-
        #           chareq_christoffel(vkap-dkap*unit_y, c_stiff, rho, v_p))/(2*
dkap)

        # dOmdkapz = (chareq_christoffel(vkap+dkap*unit_z, c_stiff, rho, v_p)-
        #          chareq_christoffel(vkap-dkap*unit_z, c_stiff, rho, v_p))/(2*d
kap)

        # dOmdvp = (chareq_christoffel(vkap, c_stiff, rho, v_p+dvel)-
        #       chareq_christoffel(vkap, c_stiff, rho, v_p-dvel))/(2*dvel)

        # vg = -np.array([dOmdkapx, dOmdkapy, dOmdkapz])/dOmdvp
        # v_vgroup[m,:] = vg


        # print('\n\n',vkap, vkap+dkap*unit_x, v_p, v_g, '\n        ',
        #       #dOmdkapx, dOmdkapy, dOmdkapz, dOmdvp, '\n        ',
        #       Gamma_christoffel(vkap, c_stiff, rho),
        #       chareq_christoffel(vkap, c_stiff, rho, v_p),
        #       np.matmul(Gamma_christoffel(vkap, c_stiff, rho), v_evecs[:,m])
        #                 -v_p**2*v_evecs[:,m],  v_evecs[:,m]
```



```
        #       )

     #print(v_p,
      #    chareq_christoffel(vkap, c_stiff, rho, v_p),
       #   chareq_christoffel(vkap, c_stiff, rho, v_p+dvel)
        #  )


    #print('got vg', vg, np.sqrt(np.linalg.norm(vg)) )
    return v_vphase, v_evecs, v_vgroup
```