```
# Import the required modules
import numpy as np
import scipy as sc
import sympy as sym
#import matplotlib; matplotlib.use('agg') ##comment out to show figures
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.patches as patches
import pickle
import itertools
import numpy.polynomial.polynomial as poly

###   Setup to suppress unwanted outputs and reduce size of output file
####  " The user can ignore this section - this section speeds "
####  " up simulation by surpressung text output and warnings "
import os
import sys
import contextlib
import warnings
warnings.filterwarnings('ignore', 'The iteration is not making good progress')
###

def fileno(file_or_fd):
    fd = getattr(file_or_fd, 'fileno', lambda: file_or_fd)()
    if not isinstance(fd, int):
        raise ValueError("Expected a file (`.fileno()`) or a file descriptor")
    return fd

@contextlib.contextmanager
def stdout_redirected(to=os.devnull, stdout=None):
    """
    https://stackoverflow.com/a/22434262/190597 (J.F. Sebastian)
    """
    if stdout is None:
       stdout = sys.stdout

    stdout_fd = fileno(stdout)
    # copy stdout_fd before it is overwritten
    #NOTE: `copied` is inheritable on Windows when duplicating a standard stream
    with os.fdopen(os.dup(stdout_fd), 'wb') as copied:
        stdout.flush()  # flush library buffers that dup2 knows nothing about
        try:
            os.dup2(fileno(to), stdout_fd)  # $ exec >&to
        except ValueError:  # filename
            with open(to, 'wb') as to_file:
                os.dup2(to_file.fileno(), stdout_fd)  # $ exec > to
        try:
            yield stdout # allow code to be run with the redirected stdout
```

```
        finally:
            # restore stdout to its previous value
            #NOTE: dup2 makes stdout_fd inheritable unconditionally
            stdout.flush()
            os.dup2(copied.fileno(), stdout_fd)  # $ exec >&copied

sys.setrecursionlimit(25000)




##########################################
c_i0 = 1.
vA_i0 = 1.2*c_i0   #1.2*c_i #-coronal        #1.9*c_i  -photospheric

vA_e = 3.*c_i0   #3.*c_i #-coronal        #0.8*c_i -photospheric
c_e = 0.4*c_i0    #0.4*c_i #- coronal          #1.3*c_i  -photospheric

cT_i0 = np.sqrt((c_i0**2 * vA_i0**2)/(c_i0**2 + vA_i0**2))

gamma=5./3.

rho_i0 = 1.
rho_e = rho_i0*(c_i0**2+gamma*0.5*vA_i0**2)/(c_e**2+gamma*0.5*vA_e**2)

print('rho_e    =', rho_e)


P_0 = c_i0**2*rho_i0/gamma
P_e = c_e**2*rho_e/gamma

T_0 = P_0/rho_i0
T_e = P_e/rho_e

B_0 = vA_i0*np.sqrt(rho_i0)
B_e = vA_e*np.sqrt(rho_e)


P_tot_0 = P_0 + B_0**2/2.
P_tot_e = P_e + B_e**2/2.

print('PT_i   =', P_tot_0)
print('PT_e   =', P_tot_e)

Kmax = 3.5

ix = np.linspace(-1, 1, 500)  # inside slab x values
ix2 = np.linspace(-1, 0, 500)  # inside slab x values
```

```
x0=0.  #mean
dx=1.25 #standard dev

xx=sym.symbols('x')   #In order to differentiate profile we need to use sympy notation using
symbols

rho_A = 1.   #Amplitude of internal density
A = 1.

def profile(x):       # Define the internal profile as a function of variable x   (Inverted Gaussian)
    return (B_e + ((B_0 - B_e)*sym.exp(-(x-x0)**2/dx**2)))


####################
#a = 1.   #inhomogeneity width for epstein profile
#def profile(x):       # Define the internal profile as a function of variable x   (Epstein Profile)
#    return ((rho_i0 - rho_e)/(sym.cosh(x/a)**4)**2 + rho_e)
####################



#def profile(x):       # Define the internal profile as a function of variable x   (Periodic threaded
waveguide)
#    return (3.*rho_e + (rho_i0 - 3.*rho_e)*sym.cos(2.*sym.pi*x + sym.pi))    #2.28 = W ~ 0.6
#3. corresponds to W ~ 1.5




##################################################
#def rho_i(x):                # Define the internal density profile
#    return rho_A*profile(x)

#def rho_i(x):                # nope
#    return sym.sqrt(B_i(x)*sym.sqrt(rho_i0)*vA_i0/(B_0*vA_i(x)))

def rho_i(x):               # nope
    return rho_i0

################################################

def P_i(x):
    return ((c_i(x))**2*rho_i(x)/gamma)

#def P_i(x):   #constant temp
#    return ((c_i0)**2*rho_i(x)/gamma)
```

```
##############################################

def T_i(x):     #use this for constant P
    return (P_i(x)/rho_i(x))

#def T_i(x):    # constant temp
#    return (P_i(x)*rho_i0/(P_0*rho_i(x)))


##############################################
#def vA_i(x):                # Define the internal alfven speed     #This works!!!!
#    return vA_i0*sym.sqrt(rho_i0)/sym.sqrt(profile(x))


#def vA_i(x):                # keep rho const
#    return vA_i0*sym.sqrt(rho_i0)*B_i(x)/sym.sqrt(rho_i(x))


def vA_i(x):                # consatnt temp
    return B_i(x)/(sym.sqrt(rho_i(x)))

##############################################
#def B_i(x):
#    return (vA_i(x)*sym.sqrt(rho_i(x)))

def B_i(x):  #constant rho
    return (A*profile(x))

####################################

def PT_i(x):
    return P_i(x) + B_i(x)**2/2.

####################################

def c_i(x):                 # Define the internal sound speed
    return sym.sqrt((rho_e*(c_e**2 + 0.5*gamma*vA_e**2)/rho_i(x)) - 0.5*gamma*vA_i(x)**2)

#def c_i(x):                # keeps c_i constant
#    return c_i0*sym.sqrt(P_i(x))*sym.sqrt(rho_i0)/(sym.sqrt(P_0)*sym.sqrt(rho_i(x)))

#def c_i(x):                # keeps c_i constant
#    return sym.sqrt(P_i(x))/sym.sqrt(gamma*rho_i(x))

#####################################################
def cT_i(x):                # Define the internal tube speed
    return sym.sqrt(((c_i(x))**2 * (vA_i(x))**2) / ((c_i(x))**2 + (vA_i(x))**2))
```

```
def cT_e():
    return np.sqrt(c_e**2 * vA_e**2 / (c_e**2 + vA_e**2))

#########################################################
speeds = [c_i0, c_e, vA_i0, vA_e, cT_i0, cT_e()]
print('speeds  =', speeds)

speeds.sort()
print('sorted speeds  =', speeds)

print(len(speeds))
speed_diff = [0]

for i in range(len(speeds)-1):
    speed_diff.append(speeds[i+1] - speeds[i])


print('speed diff    =', speed_diff)
print(len(speed_diff))
#########################################################

rho_i_np=sym.lambdify(xx,rho_i(xx),"numpy")   #In order to evaluate we need to switch to
numpy
cT_i_np=sym.lambdify(xx,cT_i(xx),"numpy")   #In order to evaluate we need to switch to
numpy
c_i_np=sym.lambdify(xx,c_i(xx),"numpy")   #In order to evaluate we need to switch to numpy
vA_i_np=sym.lambdify(xx,vA_i(xx),"numpy")   #In order to evaluate we need to switch to
numpy
P_i_np=sym.lambdify(xx,P_i(xx),"numpy")   #In order to evaluate we need to switch to
numpy
T_i_np=sym.lambdify(xx,T_i(xx),"numpy")   #In order to evaluate we need to switch to
numpy
B_i_np=sym.lambdify(xx,B_i(xx),"numpy")   #In order to evaluate we need to switch to
numpy
PT_i_np=sym.lambdify(xx,PT_i(xx),"numpy")   #In order to evaluate we need to switch to
numpy

#####   SPEEDS AT BOUNDARY    ###

rho_bound = rho_i_np(-1)
c_bound = c_i_np(-1)
vA_bound = vA_i_np(-1)
cT_bound = np.sqrt(c_bound**2 * vA_bound**2 / (c_bound**2 + vA_bound**2))

P_bound = P_i_np(-1)
print(P_bound)
internal_pressure = P_i_np(ix)
print('internal gas pressure  =', internal_pressure)
```

```
print('internal density    =', rho_i_np(ix))

print('c_i0 ratio   =',   c_e/c_i0)
print('c_i ratio   =',   c_e/c_bound)
print('c_i boundary   =', c_bound)
print('vA_i boundary   =', vA_bound)
print('rho_i boundary   =', rho_bound)

print('temp = ', T_i_np(ix))

##################       NEED TO NOTE
########################################
############     It is seen that the density profile created is       #############
############     actually the profile for sqrt(rho_i) as c_i0 &       #############
############     vA_i0 are divided by this and not sqrt(profile)      #############
################################################################################
#########


plt.figure()
plt.title("width = 1.5")
#plt.xlabel("x")
#plt.ylabel("$\u03C1_{i}$",fontsize=25)
ax = plt.subplot(331)
ax.title.set_text("Density")
#ax.plot(ix,rho_i_np(ix));
ax.annotate( '$\u03C1_{e}$', xy=(1, rho_e),fontsize=25)
ax.annotate( '$\u03C1_{i}$', xy=(1, rho_i0),fontsize=25)
ax.axhline(y=rho_i0, color='k', label='$\u03C1_{i}$', linestyle='dashdot')
ax.axhline(y=rho_e, color='k', label='$\u03C1_{e}$', linestyle='dashdot')
ax.axhline(y=rho_i_np(ix), color='b', label='$\u03C1_{i}$', linestyle='solid')
ax.set_ylim(0., 1.1)

#plt.figure()
#plt.xlabel("x")
#plt.ylabel("$c_{i}$")
ax2 = plt.subplot(332)
ax2.title.set_text("$c_{i}$")
ax2.plot(ix,c_i_np(ix));
ax2.annotate( '$c_e$', xy=(1, c_e),fontsize=25)
ax2.annotate( '$c_{i0}$', xy=(1, c_i0),fontsize=25)
ax2.annotate( '$c_{B}$', xy=(1, c_bound), fontsize=20)
ax2.axhline(y=c_i0, color='k', label='$c_{i0}$', linestyle='dashdot')
ax2.axhline(y=c_e, color='k', label='$c_{e}$', linestyle='dashdot')
ax2.fill_between(ix, c_i0, c_bound, color='green', alpha=0.2)    # fill between uniform case
and boundary values
```

```
#plt.figure()
#plt.xlabel("x")
#plt.ylabel("$vA_{i}$")
ax3 = plt.subplot(333)
ax3.title.set_text("$v_{Ai}$")
ax3.plot(ix,vA_i_np(ix));
ax3.annotate( '$v_{Ae}$', xy=(1, vA_e),fontsize=25)
ax3.annotate( '$v_{Ai}$', xy=(1, vA_i0),fontsize=25)
ax3.annotate( '$v_{AB}$', xy=(1, vA_bound), fontsize=20)
ax3.axhline(y=vA_i0, color='k', label='$v_{Ai}$', linestyle='dashdot')
ax3.axhline(y=vA_e, color='k', label='$v_{Ae}$', linestyle='dashdot')
ax3.fill_between(ix, vA_i0, vA_bound, color='orange', alpha=0.2)    # fill between uniform
case and boundary values


#plt.figure()
#plt.xlabel("x")
#plt.ylabel("$cT_{i}$")
ax4 = plt.subplot(334)
ax4.title.set_text("$c_{Ti}$")
ax4.plot(ix,cT_i_np(ix));
ax4.annotate( '$c_{Te}$', xy=(1, cT_e()),fontsize=25)
ax4.annotate( '$c_{Ti}$', xy=(1, cT_i0),fontsize=25)
ax4.annotate( '$c_{TB}$', xy=(1, cT_bound), fontsize=20)
ax4.axhline(y=cT_i0, color='k', label='$c_{Ti}$', linestyle='dashdot')
ax4.axhline(y=cT_e(), color='k', label='$c_{Te}$', linestyle='dashdot')
ax4.fill_between(ix, cT_i0, cT_bound, alpha=0.2)    # fill between uniform case and boundary
values



#plt.figure()
#plt.xlabel("x")
#plt.ylabel("$P$")
ax5 = plt.subplot(335)
ax5.title.set_text("Gas Pressure")
ax5.plot(ix,P_i_np(ix));
ax5.annotate( '$P_{0}$', xy=(1, P_0),fontsize=25)
ax5.annotate( '$P_{e}$', xy=(1, P_e),fontsize=25)
ax5.axhline(y=P_0, color='k', label='$P_{i}$', linestyle='dashdot')
ax5.axhline(y=P_e, color='k', label='$P_{e}$', linestyle='dashdot')
#ax5.axhline(y=P_i_np(ix), color='b', label='$P_{e}$', linestyle='solid')

#plt.figure()
#plt.xlabel("x")
#plt.ylabel("$T$")
ax6 = plt.subplot(336)
```

```
ax6.title.set_text("Temperature")
ax6.plot(ix,T_i_np(ix));
ax6.annotate( '$T_{0}$', xy=(1, T_0),fontsize=25)
ax6.annotate( '$T_{e}$', xy=(1, T_e),fontsize=25)
ax6.axhline(y=T_0, color='k', label='$T_{i}$', linestyle='dashdot')
ax6.axhline(y=T_e, color='k', label='$T_{e}$', linestyle='dashdot')
#ax6.axhline(y=T_i_np(ix), color='b', linestyle='solid')
#ax6.set_ylim(0., 1.1)

#plt.figure()
#plt.xlabel("x")
#plt.ylabel("$B$")
ax7 = plt.subplot(337)
ax7.title.set_text("Mag Field")
ax7.plot(ix,B_i_np(ix));
ax7.annotate( '$B_{0}$', xy=(1, B_0),fontsize=25)
ax7.annotate( '$B_{e}$', xy=(1, B_e),fontsize=25)
ax7.axhline(y=B_0, color='k', label='$B_{i}$', linestyle='dashdot')
ax7.axhline(y=B_e, color='k', label='$B_{e}$', linestyle='dashdot')
#ax7.axhline(y=B_i_np(ix), color='b', label='$B_{e}$', linestyle='solid')


#plt.xlabel("x")
#plt.ylabel("$P_T$")
ax8 = plt.subplot(338)
ax8.title.set_text("Tot Pressure (uniform)")
ax8.annotate( '$P_{T0}$', xy=(1, P_tot_0),fontsize=25,color='b')
ax8.annotate( '$P_{Te}$', xy=(1, P_tot_e),fontsize=25,color='r')
ax8.axhline(y=P_tot_0, color='b', label='$P_{Ti0}$', linestyle='dashdot')
ax8.axhline(y=P_tot_e, color='r', label='$P_{Te}$', linestyle='dashdot')

#ax8.axhline(y=PT_i_np(ix), color='k', label='$P_{Ti}$', linestyle='solid')
#ax8.axhline(y=B_i_np(ix), color='b', label='$P_{Te}$', linestyle='solid')

ax9 = plt.subplot(339)
ax9.title.set_text("Tot Pressure  (Modified)")
ax9.annotate( '$P_{T0},  P_{Te}$', xy=(1, P_tot_0),fontsize=25,color='b')
#ax9.plot(ix,PT_i_np(ix));

ax9.axhline(y=PT_i_np(ix), color='k', label='$P_{Ti}$', linestyle='solid')

plt.suptitle("W = 1.75", fontsize=14)
plt.tight_layout()
#plt.savefig("photospheric_profiles_09.png")
#
#plt.show()
#exit()
```

```
wavenumber = np.linspace(0.01,3.5,20)      #(1.5, 1.8), 5


########   READ IN VARIABLES    #########

with open('B_width125_coronal.pickle', 'rb') as f:
    sol_omegas1, sol_ks1, sol_omegas_kink1, sol_ks_kink1 = pickle.load(f)



### SORT ARRAYS IN ORDER OF WAVENUMBER ###
sol_omegas1 = [x for _,x in sorted(zip(sol_ks1,sol_omegas1))]
sol_ks1 = np.sort(sol_ks1)

sol_omegas1 = np.array(sol_omegas1)
sol_ks1 = np.array(sol_ks1)

sol_omegas_kink1 = [x for _,x in sorted(zip(sol_ks_kink1,sol_omegas_kink1))]
sol_ks_kink1 = np.sort(sol_ks_kink1)

sol_omegas_kink1 = np.array(sol_omegas_kink1)
sol_ks_kink1 = np.array(sol_ks_kink1)



####   CORONAL    ######

### FAST BODY    ( vA_i  <  w/k  <  vA_e )
fast_body_sausage_omega = []
fast_body_sausage_k = []
fast_body_kink_omega = []
fast_body_kink_k = []

### SLOW BODY    ( cT_i  <  w/k  <  c_i )
slow_body_sausage_omega = []
slow_body_sausage_k = []
slow_body_kink_omega = []
slow_body_kink_k = []


for i in range(len(sol_ks1)):   #sausage mode
  v_phase = sol_omegas1[i]/sol_ks1[i]

  if v_phase > vA_i0 and v_phase < vA_e:
      fast_body_sausage_omega.append(sol_omegas1[i])
      fast_body_sausage_k.append(sol_ks1[i])
```

```
  if v_phase > cT_i0 and v_phase < c_i0:  #vA_i0:   change after uniform
      slow_body_sausage_omega.append(sol_omegas1[i])
      slow_body_sausage_k.append(sol_ks1[i])


for i in range(len(sol_ks_kink1)):   #kink mode
  v_phase = sol_omegas_kink1[i]/sol_ks_kink1[i]

  if v_phase > vA_i0 and v_phase < vA_e:
      fast_body_kink_omega.append(sol_omegas_kink1[i])
      fast_body_kink_k.append(sol_ks_kink1[i])


  if v_phase > cT_i0 and v_phase < vA_i0:
      slow_body_kink_omega.append(sol_omegas_kink1[i])
      slow_body_kink_k.append(sol_ks_kink1[i])


fast_body_sausage_omega = np.array(fast_body_sausage_omega)
fast_body_sausage_k = np.array(fast_body_sausage_k)
fast_body_kink_omega = np.array(fast_body_kink_omega)
fast_body_kink_k = np.array(fast_body_kink_k)

slow_body_sausage_omega = np.array(slow_body_sausage_omega)
slow_body_sausage_k = np.array(slow_body_sausage_k)
slow_body_kink_omega = np.array(slow_body_kink_omega)
slow_body_kink_k = np.array(slow_body_kink_k)


cutoff = 7.    #9 for messing    #7 is good!
cutoff2 = 8.5   #8.7 for W < 2     # 8.5 otherwise
cutoff3 = 3.  #3.    #4.2 for W < 0.85
fast_sausage_branch1_omega = []
fast_sausage_branch1_k = []
fast_sausage_branch2_omega = []
fast_sausage_branch2_k = []
fast_sausage_branch3_omega = []
fast_sausage_branch3_k = []




##################################################
for i in range(len(fast_body_sausage_omega)):   #sausage mode

  if fast_body_sausage_omega[i] < cutoff and fast_body_sausage_omega[i] > cutoff3:
      fast_sausage_branch1_omega.append(fast_body_sausage_omega[i])
      fast_sausage_branch1_k.append(fast_body_sausage_k[i])
```

```
  elif fast_body_sausage_omega[i] > cutoff2:
     fast_sausage_branch2_omega.append(fast_body_sausage_omega[i])
     fast_sausage_branch2_k.append(fast_body_sausage_k[i])

  else:
     fast_sausage_branch3_omega.append(fast_body_sausage_omega[i])
     fast_sausage_branch3_k.append(fast_body_sausage_k[i])


index_to_remove = []
if len(fast_sausage_branch1_omega) > 1:     # some solutions may be in worng array so
separate if too far off curve

  #for i in range(len(fast_sausage_branch1_omega)-1):
  for i in range(len(fast_sausage_branch1_omega)):

    #ph_diff = abs((fast_sausage_branch1_omega[i+1]/fast_sausage_branch1_k[i+1]) -
(fast_sausage_branch1_omega[i]/fast_sausage_branch1_k[i]))
    ph_speed = fast_sausage_branch1_omega[i]/fast_sausage_branch1_k[i]

    #if ph_diff > 0.3:
    if ph_speed < 1.5:
       index_to_remove.append(i)
       fast_sausage_branch3_k.append(fast_sausage_branch1_k[i])
       fast_sausage_branch3_omega.append(fast_sausage_branch1_omega[i])

    #### uncomment for W < 0.8
fast_sausage_branch1_omega = np.delete(fast_sausage_branch1_omega,
index_to_remove)
fast_sausage_branch1_k = np.delete(fast_sausage_branch1_k, index_to_remove)


index_to_remove = []
if len(fast_sausage_branch3_omega) > 1:     # some solutions may be in worng array so
separate if too far off curve

  for i in range(len(fast_sausage_branch3_omega)-1):

    ph_diff = abs((fast_sausage_branch3_omega[i+1]/fast_sausage_branch3_k[i+1]) -
(fast_sausage_branch3_omega[i]/fast_sausage_branch3_k[i]))

    if ph_diff > 0.02:
       index_to_remove.append(i)

fast_sausage_branch3_omega = np.delete(fast_sausage_branch3_omega,
index_to_remove)
fast_sausage_branch3_k = np.delete(fast_sausage_branch3_k, index_to_remove)
```

```
##########################################################################
####

fast_sausage_branch1_omega = np.array(fast_sausage_branch1_omega)
fast_sausage_branch1_k = np.array(fast_sausage_branch1_k)
fast_sausage_branch2_omega = np.array(fast_sausage_branch2_omega)
fast_sausage_branch2_k = np.array(fast_sausage_branch2_k)
fast_sausage_branch3_omega = np.array(fast_sausage_branch3_omega)
fast_sausage_branch3_k = np.array(fast_sausage_branch3_k)


cutoff_kink = 5.     # 6.5 for messing
cutoff2_kink = 5.9   # 6 for W < 1.5     #5.8 otherwise     7 for W < 0.85


#####################################################   kink mode
fast_kink_branch1_omega = []
fast_kink_branch1_k = []
fast_kink_branch2_omega = []
fast_kink_branch2_k = []
fast_kink_branch3_omega = []
fast_kink_branch3_k = []
#####################################################
for i in range(len(fast_body_kink_omega)):   #sausage mode

  if fast_body_kink_omega[i] < cutoff_kink:
     fast_kink_branch1_omega.append(fast_body_kink_omega[i])
     fast_kink_branch1_k.append(fast_body_kink_k[i])

  elif fast_body_kink_omega[i] > cutoff2_kink:
     fast_kink_branch2_omega.append(fast_body_kink_omega[i])
     fast_kink_branch2_k.append(fast_body_kink_k[i])

  else:
     fast_kink_branch3_omega.append(fast_body_kink_omega[i])
     fast_kink_branch3_k.append(fast_body_kink_k[i])


index_to_remove = []
if len(fast_kink_branch1_omega) > 1:     # some solutions may be in worng array so separate
if too far off curve

  for i in range(len(fast_kink_branch1_omega)):

    ph_speed = (fast_kink_branch1_omega[i]/fast_kink_branch1_k[i])

    if ph_speed < c_bound:
```

```
      index_to_remove.append(i)

fast_kink_branch1_omega = np.delete(fast_kink_branch1_omega, index_to_remove)
fast_kink_branch1_k = np.delete(fast_kink_branch1_k, index_to_remove)



fast_kink_branch1_omega = np.array(fast_kink_branch1_omega)
fast_kink_branch1_k = np.array(fast_kink_branch1_k)
fast_kink_branch2_omega = np.array(fast_kink_branch2_omega)
fast_kink_branch2_k = np.array(fast_kink_branch2_k)
fast_kink_branch3_omega = np.array(fast_kink_branch3_omega)
fast_kink_branch3_k = np.array(fast_kink_branch3_k)

##################################################



############################ sausage body
body_sausage_branch1_omega = []
body_sausage_branch1_k = []
body_sausage_branch2_omega = []
body_sausage_branch2_k = []

#body_sausage_omega = slow_body_sausage_omega[::-1]
#body_sausage_k = slow_body_sausage_k[::-1]

if len(slow_body_sausage_omega) > 1:

  body_sausage_branch1_omega.append(slow_body_sausage_omega[0])
  body_sausage_branch1_k.append(slow_body_sausage_k[0])


  for i in range(len(slow_body_sausage_omega)-1):

    ph_diff = abs((slow_body_sausage_omega[i+1]/slow_body_sausage_k[i+1]) -
(body_sausage_branch1_omega[-1]/body_sausage_branch1_k[-1]))
    k_diff = abs((slow_body_sausage_k[i+1] - body_sausage_branch1_k[-1]))

    #if slow_body_sausage_omega[i+1] < body_sausage_branch1_omega[-1]:
    if ph_diff < 0.03 and k_diff < 0.5:
      body_sausage_branch1_omega.append(slow_body_sausage_omega[i+1])
      body_sausage_branch1_k.append(slow_body_sausage_k[i+1])

    else:
      body_sausage_branch2_omega.append(slow_body_sausage_omega[i+1])
      body_sausage_branch2_k.append(slow_body_sausage_k[i+1])
```

```
body_sausage_branch1_omega = np.array(body_sausage_branch1_omega)
body_sausage_branch1_k = np.array(body_sausage_branch1_k)
body_sausage_branch2_omega = np.array(body_sausage_branch2_omega)
body_sausage_branch2_k = np.array(body_sausage_branch2_k)

body_sausage_branch1_omega = body_sausage_branch1_omega[::-1]
body_sausage_branch1_k = body_sausage_branch1_k[::-1]
body_sausage_branch2_omega = body_sausage_branch2_omega[::-1]
body_sausage_branch2_k = body_sausage_branch2_k[::-1]

index_to_remove = []

for i in range(len(body_sausage_branch2_omega)-1):

  ph_speed = body_sausage_branch2_omega[i]/body_sausage_branch2_k[i]
  ph_diff = abs((body_sausage_branch2_omega[i+1]/body_sausage_branch2_k[i+1]) -
(body_sausage_branch2_omega[i]/body_sausage_branch2_k[i]))

  if ph_diff > 0.03:   ###0.01
     index_to_remove.append(i)

  elif ph_speed > c_bound:
     index_to_remove.append(i)

body_sausage_branch2_omega = np.delete(body_sausage_branch2_omega,
index_to_remove)
body_sausage_branch2_k = np.delete(body_sausage_branch2_k, index_to_remove)


for i in range(len(body_sausage_branch1_omega)):   # dont like this but have to do it   ##
removes solutions close to c_i_bound (spurious solutions)
    ph_speed = body_sausage_branch1_omega[i]/body_sausage_branch1_k[i]
    if ph_speed > c_bound:
      index_to_remove.append(i)


body_sausage_branch1_omega = np.delete(body_sausage_branch1_omega,
index_to_remove)
body_sausage_branch1_k = np.delete(body_sausage_branch1_k, index_to_remove)


for i in range(len(body_sausage_branch2_omega)):   # dont like this but have to do it   ##
removes solutions close to c_i_bound (spurious solutions)
    ph_speed = body_sausage_branch2_omega[i]/body_sausage_branch2_k[i]
    if ph_speed > c_bound:
      index_to_remove.append(i)
```

```
body_sausage_branch2_omega = np.delete(body_sausage_branch2_omega,
index_to_remove)
body_sausage_branch2_k = np.delete(body_sausage_branch2_k, index_to_remove)

if len(body_sausage_branch1_omega) > 1:
  #   sausage body polyfit
  SB_phase = body_sausage_branch1_omega/body_sausage_branch1_k
  SB_k = body_sausage_branch1_k
  SB_k_new = np.linspace(SB_k[-1], SB_k[0], num=len(SB_k)*10)      # set [ SB_k[-1], Kmax
] for small samples

  SB_coefs = poly.polyfit(SB_k, SB_phase, 6)   # was 6    # use 1 for W <= 1
  SB_ffit = poly.polyval(SB_k_new, SB_coefs)

if len(body_sausage_branch2_omega) > 1:
  SB2_phase = body_sausage_branch2_omega/body_sausage_branch2_k
  SB2_k = body_sausage_branch2_k
#  SB2_k_new = np.linspace(SB2_k[0], SB2_k[-1], num=len(SB2_k)*10)
  SB2_k_new = np.linspace(SB2_k[-1], Kmax, num=len(SB2_k)*10)

  SB2_coefs = poly.polyfit(SB2_k, SB2_phase, 1)    # 6
  SB2_ffit = poly.polyval(SB2_k_new, SB2_coefs)


#################   kink body  #################
body_kink_branch1_omega = []
body_kink_branch1_k = []
body_kink_branch2_omega = []
body_kink_branch2_k = []

body_kink_omega = slow_body_kink_omega[::-1]
body_kink_k = slow_body_kink_k[::-1]

if len(slow_body_kink_omega) > 1:

  body_kink_branch1_omega.append(slow_body_kink_omega[0])
  body_kink_branch1_k.append(slow_body_kink_k[0])

  for i in range(len(slow_body_kink_omega)-1):

     #omega_diff = slow_body_sausage_omega[i+1] - slow_body_sausage_omega[i]
     ph_diff = abs((slow_body_kink_omega[i+1]/slow_body_kink_k[i+1]) -
(body_kink_branch1_omega[-1]/body_kink_branch1_k[-1]))

     if ph_diff < 0.01 and k_diff < 0.5:
       body_kink_branch1_omega.append(slow_body_kink_omega[i+1])
       body_kink_branch1_k.append(slow_body_kink_k[i+1])
```

```
      else:
        body_kink_branch2_omega.append(slow_body_kink_omega[i+1])
        body_kink_branch2_k.append(slow_body_kink_k[i+1])


body_kink_branch1_omega = np.array(body_kink_branch1_omega)
body_kink_branch1_k = np.array(body_kink_branch1_k)
body_kink_branch2_omega = np.array(body_kink_branch2_omega)
body_kink_branch2_k = np.array(body_kink_branch2_k)

body_kink_branch1_omega = body_kink_branch1_omega[::-1]
body_kink_branch1_k = body_kink_branch1_k[::-1]
body_kink_branch2_omega = body_kink_branch2_omega[::-1]
body_kink_branch2_k = body_kink_branch2_k[::-1]

index_to_remove = []

for i in range(len(body_kink_branch2_omega)-1):

  ph_speed = body_kink_branch2_omega[i]/body_kink_branch2_k[i]
  ph_diff = abs((body_kink_branch2_omega[i+1]/body_kink_branch2_k[i+1]) -
(body_kink_branch2_omega[i]/body_kink_branch2_k[i]))

  if ph_diff > 0.01:
     index_to_remove.append(i)

  #elif ph_speed > body_kink_branch2_omega[0]:
  #    index_to_remove.append(i)

  elif ph_speed > c_bound-0.01:
     index_to_remove.append(i)

body_kink_branch2_omega = np.delete(body_kink_branch2_omega, index_to_remove)
body_kink_branch2_k = np.delete(body_kink_branch2_k, index_to_remove)


#   kink slow body polyfit
if len(body_kink_branch1_omega) > 1:

  KB_phase = body_kink_branch1_omega/body_kink_branch1_k
  KB_k = body_kink_branch1_k
  KB_k_new = np.linspace(KB_k[-1], KB_k[0], num=len(KB_k)*10)

  KB_coefs = poly.polyfit(KB_k, KB_phase, 1)   #6
  KB_ffit = poly.polyval(KB_k_new, KB_coefs)
```

```
if len(body_kink_branch2_omega) > 1:

  KB2_phase = body_kink_branch2_omega/body_kink_branch2_k
  KB2_k = body_kink_branch2_k
  KB2_k_new = np.linspace(KB2_k[-1], Kmax, num=len(KB2_k)*10)

  KB2_coefs = poly.polyfit(KB2_k, KB2_phase, 1)   #6
  KB2_ffit = poly.polyval(KB2_k_new, KB2_coefs)


##########################################################################
#################



test_k_plot = np.linspace(0.01,Kmax,20)

fig=plt.figure()
ax = plt.subplot(111)
plt.xlabel("$k$", fontsize=18)
plt.ylabel('$\omega$', fontsize=22, rotation=0, labelpad=15)
vA_e_plot = test_k_plot*vA_e
vA_i_plot = test_k_plot*vA_i0
c_e_plot = test_k_plot*c_e
c_i_plot = test_k_plot*c_i0
cT_e_plot = test_k_plot*cT_e()
cT_i_plot = test_k_plot*cT_i0
#ax.plot(sol_ks1, sol_omegas1, 'b.', markersize=4.)
ax.plot(fast_body_sausage_k, fast_body_sausage_omega, 'r.', markersize=4.)   #fast body
sausage
ax.plot(fast_body_kink_k, fast_body_kink_omega, 'b.', markersize=4.)   #fast body kink
#ax.plot(fast_sausage_branch1_k, fast_sausage_branch1_omega, 'r.', markersize=4.)   #fast
body branch 1
#ax.plot(fast_sausage_branch2_k, fast_sausage_branch2_omega, 'r.', markersize=4.)   #fast
body branch 2
#ax.plot(fast_sausage_branch3_k, fast_sausage_branch3_omega, 'r.', markersize=4.)   #fast
body branch 3
ax.plot(test_k_plot, vA_e_plot, linestyle='dashdot', color='k')
ax.plot(test_k_plot, vA_i_plot, linestyle='dashdot', color='k')
ax.plot(test_k_plot, c_e_plot, linestyle='dashdot', color='k')
ax.plot(test_k_plot, c_i_plot, linestyle='dashdot', color='k')
ax.plot(test_k_plot, cT_e_plot, linestyle='dashdot', color='k')
ax.plot(test_k_plot, cT_i_plot, linestyle='dashdot', color='k')
#ax.plot(slow_body_sausage_k, slow_body_sausage_omega, 'b.', markersize=4.)   #slow
body sausage
#ax.plot(slow_body_kink_k, slow_body_kink_omega, 'b.', markersize=4.)   #slow body kink

ax.annotate( '$c_{Ti}$', xy=(Kmax, cT_i_plot[-1]), fontsize=20)
```

```
#ax.annotate( '$c_{Te}$', xy=(Kmax, cT_e_plot[-1]), fontsize=20)
ax.annotate( '$c_{e},    c_{Te}$', xy=(Kmax, c_e_plot[-1]), fontsize=20)
ax.annotate( '$c_{i}$', xy=(Kmax, c_i_plot[-1]), fontsize=20)
ax.annotate( '$v_{Ae}$', xy=(Kmax, vA_e_plot[-1]), fontsize=20)
ax.annotate( '$v_{Ai}$', xy=(Kmax, vA_i_plot[-1]), fontsize=20)
ax.axhline(y=cutoff, color='r', linestyle='dashed', label='_nolegend_')
ax.axhline(y=cutoff2, color='r', linestyle='dashed', label='_nolegend_')
ax.annotate( '$\omega = {}$'.format(cutoff2), xy=(Kmax, cutoff2), fontsize=18, color='r')
ax.annotate( '$\omega = {}$'.format(cutoff), xy=(Kmax, cutoff), fontsize=18, color='r')
ax.axhline(y=cutoff3, color='r', linestyle='dashed', label='_nolegend_')
ax.annotate( '$\omega = {}$'.format(cutoff3), xy=(Kmax, cutoff3), fontsize=18, color='r')
ax.axhline(y=cutoff_kink, color='b', linestyle='dashed', label='_nolegend_')
ax.axhline(y=cutoff2_kink, color='b', linestyle='dashed', label='_nolegend_')
ax.annotate( '$\omega = {}$'.format(cutoff2_kink), xy=(Kmax, cutoff2_kink), fontsize=18,
color='b')
ax.annotate( '$\omega = {}$'.format(cutoff_kink), xy=(Kmax, cutoff_kink), fontsize=18,
color='b')


#########################################################################
sausage polyfit
FSB1_phase = fast_sausage_branch1_omega/fast_sausage_branch1_k
FSB1_k = fast_sausage_branch1_k
k_new = np.linspace(FSB1_k[0], FSB1_k[-1], num=len(FSB1_k)*10)

coefs = poly.polyfit(FSB1_k, FSB1_phase, 6)   #  # 6 is good
ffit = poly.polyval(k_new, coefs)

if len(fast_sausage_branch2_omega) > 1:
  FSB2_phase = fast_sausage_branch2_omega/fast_sausage_branch2_k
  FSB2_k = fast_sausage_branch2_k
  k_new_2 = np.linspace(FSB2_k[0], FSB2_k[-1], num=len(FSB2_k)*10)

  coefs_2 = poly.polyfit(FSB2_k, FSB2_phase, 6)    # 6 order     # 1st order for W < 1.5
  ffit_2 = poly.polyval(k_new_2, coefs_2)

if len(fast_sausage_branch3_omega) > 1:
  FSB3_phase = fast_sausage_branch3_omega/fast_sausage_branch3_k
  FSB3_k = fast_sausage_branch3_k
  #k_new_3 = np.linspace(FSB3_k[0], Kmax, num=len(FSB3_k)*25)
  k_new_3 = np.linspace(FSB3_k[0], FSB3_k[-1], num=len(FSB3_k)*10)   #0, Kmax

  coefs_3 = poly.polyfit(FSB3_k, FSB3_phase, 1)  #4
  ffit_3 = poly.polyval(k_new_3, coefs_3)

#########################################################################
kink polyfit
FSB1_kink_phase = fast_kink_branch1_omega/fast_kink_branch1_k
```

```
FSB1_kink_k = fast_kink_branch1_k
k_kink_new = np.linspace(FSB1_kink_k[0], FSB1_kink_k[-1], num=len(FSB1_kink_k)*10)
#FSB1_kink_k[-1]

coefs_kink = poly.polyfit(FSB1_kink_k, FSB1_kink_phase, 6) # 3 order for messing
ffit_kink = poly.polyval(k_kink_new, coefs_kink)

if len(fast_kink_branch2_omega) > 1:
  FSB2_kink_phase = fast_kink_branch2_omega/fast_kink_branch2_k
  FSB2_kink_k = fast_kink_branch2_k
  k_kink_new_2 = np.linspace(FSB2_kink_k[0], Kmax, num=len(FSB2_kink_k)*10)
#FSB2_kink_k[-1]

  coefs_2_kink = poly.polyfit(FSB2_kink_k, FSB2_kink_phase, 6)
  ffit_2_kink = poly.polyval(k_kink_new_2, coefs_2_kink)

#########################################################################
#

plt.figure()
plt.title("$ W = 1.25 $")
ax = plt.subplot(111)
plt.xlabel("$kx_{0}$", fontsize=18)
plt.ylabel(r'$\frac{\omega}{k}$', fontsize=22, rotation=0, labelpad=15)

ax.plot(sol_ks1, (sol_omegas1/sol_ks1), 'r.', markersize=4.)
ax.plot(sol_ks_kink1, (sol_omegas_kink1/sol_ks_kink1), 'b.', markersize=4.)

ax.plot(fast_sausage_branch1_k, fast_sausage_branch1_omega/fast_sausage_branch1_k,
'r.', markersize=4.)
ax.plot(fast_sausage_branch2_k, fast_sausage_branch2_omega/fast_sausage_branch2_k,
'r.', markersize=4.)
#ax.plot(fast_sausage_branch3_k, fast_sausage_branch3_omega/fast_sausage_branch3_k,
'r.', markersize=4.)

ax.plot(k_new, ffit, color='r')
ax.plot(k_new_2, ffit_2, color='r')
#ax.plot(k_new_3, ffit_3, color='r')

ax.plot(fast_kink_branch1_k, fast_kink_branch1_omega/fast_kink_branch1_k, 'b.',
markersize=4.)
ax.plot(fast_kink_branch2_k, fast_kink_branch2_omega/fast_kink_branch2_k, 'b.',
markersize=4.)

ax.plot(k_kink_new, ffit_kink, color='b')
ax.plot(k_kink_new_2, ffit_2_kink, color='b')
```

```
#ax.plot(slow_body_sausage_k, (slow_body_sausage_omega/slow_body_sausage_k), 'r.', markersize=4.)
#ax.plot(slow_body_kink_k, (slow_body_kink_omega/slow_body_kink_k), 'b.', markersize=4.)

ax.plot(body_sausage_branch1_k, body_sausage_branch1_omega/body_sausage_branch1_k, 'r.', markersize=4.)   # body sausage
ax.plot(body_kink_branch1_k, body_kink_branch1_omega/body_kink_branch1_k, 'b.', markersize=4.)   # body kink
ax.plot(body_sausage_branch2_k, body_sausage_branch2_omega/body_sausage_branch2_k, 'r.', markersize=4.)   # body sausage
ax.plot(body_kink_branch2_k, body_kink_branch2_omega/body_kink_branch2_k, 'b.', markersize=4.)   # body kink

#ax.plot(SB_k_new, SB_ffit, color='r')
#ax.plot(SB2_k_new, SB2_ffit, color='r')
#ax.plot(KB_k_new, KB_ffit, color='b')
#ax.plot(KB2_k_new, KB2_ffit, color='b')

ax.axhline(y=vA_e, color='k', linestyle='dashdot', label='_nolegend_')
#ax.axhline(y=vA_i0, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=c_e, color='k', linestyle='dashdot', label='_nolegend_')
#ax.axhline(y=c_i0, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=cT_e(), color='k', linestyle='dashdot', label='_nolegend_')
#ax.axhline(y=cT_i0, color='k', linestyle='dashdot', label='_nolegend_')

#ax.annotate( '$c_{Ti}$', xy=(Kmax, cT_i0), fontsize=20)
ax.annotate( '$c_{e},   c_{Te}$', xy=(Kmax, c_e), fontsize=20)
#ax.annotate( '$c_{i}$', xy=(Kmax, c_i0), fontsize=20)
ax.annotate( '$v_{Ae}$', xy=(Kmax, vA_e), fontsize=20)
#ax.annotate( '$v_{Ai}$', xy=(Kmax, vA_i0), fontsize=20)

ax.annotate( '$c_{B}$', xy=(Kmax, c_bound), fontsize=20, color='k')
ax.annotate( '$v_{AB}$', xy=(Kmax, vA_bound), fontsize=20, color='k')
ax.annotate( '$c_{TB}$', xy=(Kmax, cT_bound), fontsize=20, color='k')
ax.axhline(y=c_bound, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=vA_bound, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=cT_bound, color='k', linestyle='dashdot', label='_nolegend_')


ax.fill_between(wavenumber, cT_i0, cT_bound, alpha=0.2)    # fill between uniform case and boundary values
ax.fill_between(wavenumber, c_i0, c_bound, color='green', alpha=0.2)    # fill between uniform case and boundary values
ax.fill_between(wavenumber, vA_i0, vA_bound, color='orange', alpha=0.2)    # fill between uniform case and boundary values
```

```
ax.set_ylim(0., vA_e+0.1)
#ax.set_yticks([])

box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width*0.95, box.height])

#plt.savefig("Bwidth125_coronal_curves_dots.png")
plt.show()
exit()


#############################################################################
end polyfit
#
##d_omega_sausage_branch1 = np.gradient(fast_sausage_branch1_omega)
##d_k_sausage_branch1 = np.gradient(fast_sausage_branch1_k)
#
#d_omega_sausage_branch1 = np.diff(fast_sausage_branch1_omega)
#d_k_sausage_branch1 = np.diff(fast_sausage_branch1_k)
#
##d_omega_kink = np.gradient(sol_omegas_kink1)
##d_k_kink = np.gradient(sol_ks_kink1)
#
#plt.figure()
#ax = plt.subplot(111)
#plt.xlabel("$kx_{0}$", fontsize=18)
#plt.ylabel(r'$\frac{d\omega}{dk}$', fontsize=22, rotation=0, labelpad=15)
#ax.plot((d_omega_sausage_branch1/d_k_sausage_branch1),fast_sausage_branch1_k[:-1],
'b.', markersize=4.)




##################    BEGIN FULL PLOTS   ###################################
!!!!!!!!!!!!!!!!          CORONAL            !!!!!!!!!!!!!!!!!!!!!
plt.figure()
ax = plt.subplot(111)
plt.title("Sausage Mode  $A = 1e5$")
plt.xlabel("$kx_{0}$", fontsize=18)
plt.ylabel(r'$\frac{\omega}{k}$', fontsize=22, rotation=0, labelpad=15)
#ax.plot(sol_ks1, (sol_omegas1/sol_ks1), 'b.', markersize=4.)
ax.plot(fast_body_sausage_k, (fast_body_sausage_omega/fast_body_sausage_k), 'b.',
markersize=4.)   #fast body
#ax.plot(slow_body_sausage_k, (slow_body_sausage_omega/slow_body_sausage_k), 'b.',
markersize=4.)   #slow body
ax.axhline(y=vA_e, color='k', linestyle='dashdot', label='_nolegend_')
```

```
ax.axhline(y=vA_i0, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=c_e, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=c_i0, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=cT_e(), color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=cT_i0, color='k', linestyle='dashdot', label='_nolegend_')

ax.annotate( '$c_{Ti}$', xy=(Kmax, cT_i0), fontsize=20)
ax.annotate( '$c_{e},   c_{Te}$', xy=(Kmax, c_e), fontsize=20)
ax.annotate( '$c_{i}$', xy=(Kmax, c_i0), fontsize=20)
ax.annotate( '$v_{Ae}$', xy=(Kmax, vA_e), fontsize=20)
ax.annotate( '$v_{Ai}$', xy=(Kmax, vA_i0), fontsize=20)

ax.set_ylim(0., vA_e+0.1)
ax.set_yticks([])

box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width*0.95, box.height])

#plt.savefig("test_dispersion_diagram_sausage_coronal.png")

plt.figure()
ax = plt.subplot(111)
plt.title("Kink Mode  $A = 1e5$")
plt.xlabel("$kx_{0}$", fontsize=18)
plt.ylabel(r'$\frac{\omega}{k}$', fontsize=22, rotation=0, labelpad=15)
#ax.plot(sol_ks_kink, (sol_omegas_kink/sol_ks_kink), 'b.', markersize=5.)
ax.plot(sol_ks_kink1, (sol_omegas_kink1/sol_ks_kink1), 'b.', markersize=4.)
#ax.plot(fast_body_kink_k, (fast_body_kink_omega/fast_body_kink_k), 'b.', markersize=4.)
#fast body
#ax.plot(slow_body_kink_k, (slow_body_kink_omega/slow_body_kink_k), 'b.',
markersize=4.)   #slow body
ax.axhline(y=vA_e, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=vA_i0, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=c_e, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=c_i0, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=cT_e(), color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=cT_i0, color='k', linestyle='dashdot', label='_nolegend_')

ax.annotate( '$c_{Ti}$', xy=(Kmax, cT_i0), fontsize=20)
ax.annotate( '$c_{e},   c_{Te}$', xy=(Kmax, c_e), fontsize=20)
ax.annotate( '$c_{i}$', xy=(Kmax, c_i0), fontsize=20)
ax.annotate( '$v_{Ae}$', xy=(Kmax, vA_e), fontsize=20)
ax.annotate( '$v_{Ai}$', xy=(Kmax, vA_i0), fontsize=20)

ax.set_ylim(0., vA_e+0.1)
ax.set_yticks([])

box = ax.get_position()
```

```
ax.set_position([box.x0, box.y0, box.width*0.95, box.height])

#plt.savefig("test_dispersion_diagram_kink_coronal.png")


#########   FULL TEST DISPERSION DIAGRAM      #################
plt.figure()
ax = plt.subplot(111)
plt.title("$W = 0.6$")
plt.xlabel("$kx_{0}$", fontsize=18)
plt.ylabel(r'$\frac{\omega}{k}$', fontsize=22, rotation=0, labelpad=15)
ax.plot(sol_ks1, (sol_omegas1/sol_ks1), 'r.', markersize=4.)
ax.plot(sol_ks_kink1, (sol_omegas_kink1/sol_ks_kink1), 'b.', markersize=4.)
ax.axhline(y=vA_e, color='k', linestyle='dashdot', label='_nolegend_')
#ax.axhline(y=vA_i0, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=c_e, color='k', linestyle='dashdot', label='_nolegend_')
#ax.axhline(y=c_i0, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=cT_e(), color='k', linestyle='dashdot', label='_nolegend_')
#ax.axhline(y=cT_i0, color='k', linestyle='dashdot', label='_nolegend_')

#ax.annotate( '$c_{Ti}$', xy=(Kmax, cT_i0), fontsize=20)
ax.annotate( '$c_{e},   c_{Te}$', xy=(Kmax, c_e), fontsize=20)
#ax.annotate( '$c_{i}$', xy=(Kmax, c_i0), fontsize=20)
ax.annotate( '$v_{Ae}$', xy=(Kmax, vA_e), fontsize=20)
#ax.annotate( '$v_{Ai}$', xy=(Kmax, vA_i0), fontsize=20)

ax.annotate( '$c_{B}$', xy=(Kmax, c_bound), fontsize=20, color='k')
ax.annotate( '$v_{AB}$', xy=(Kmax, vA_bound), fontsize=20, color='k')
ax.annotate( '$c_{TB}$', xy=(Kmax, cT_bound), fontsize=20, color='k')
ax.axhline(y=c_bound, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=vA_bound, color='k', linestyle='dashdot', label='_nolegend_')
ax.axhline(y=cT_bound, color='k', linestyle='dashdot', label='_nolegend_')


ax.fill_between(wavenumber, cT_i0, cT_bound, alpha=0.2)    # fill between uniform case and
boundary values
ax.fill_between(wavenumber, c_i0, c_bound, color='green', alpha=0.2)    # fill between
uniform case and boundary values
ax.fill_between(wavenumber, vA_i0, vA_bound, color='orange', alpha=0.2)    # fill between
uniform case and boundary values


ax.set_ylim(0., vA_e+0.1)
ax.set_yticks([])

box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width*0.95, box.height])
```

```
plt.savefig("smooth_width_coronal_06_dispersion_diagram_shaded.png")

plt.show()
#exit()
```