# Modal solution for a n2-profile

May 3, 2023

# 1 Analytical solution of the wave equation in a n$^2$-linear profile

**Jelle Assink** (KNMI) and **Roger Waxler** (National Center for Physical Acoustics (NCPA) at the University of Mississippi)

*jelle.assink@knmi.nl* – October 2020

This code computes the analytic solution for a n$^2$-linear profile, whose eigenfunctions are Airy functions.

## 1.1 Theoretical Background

### 1.1.1 Derivation

Consider the following Helmholtz equation in a two-dimensional coordinate system, as a function of range $r$ and altitude $z$:

$$\left[\nabla^2 + k^2\right] p(r, z, \omega) = 0 \tag{1}$$

Here, $k(z) = \frac{\omega}{c(z)}$ is the medium wavenumber, where $\omega$ and $c(z)$ correspond to the angular frequency ($\omega = 2\pi f$) and sound speed, respectively. Separation in variables ($p(r, z) = \psi(z)e^{ik_H r}$) leads to the following one-dimensional vertical wave equation:

$$\left[\frac{d^2}{dz^2} + k^2 - k_H^2\right] \psi(z) = 0 \tag{2}$$

where $k_H$ is the horizontal wavenumber. Note that the frequency dependence of $\psi(z)$ is kept implicit.

In the case of a $n^2$-linear profile, the medium wavenumber squared has the form $k^2(z) = b - az$, where $a$ and $b$ are constants and $z$ is the altitude. Substituting and rearranging leads to:

$$\left[\frac{d^2}{dz^2} - (az - b + k_H^2)\right] \psi(z) = 0 \tag{3}$$

It can be shown that this equation can be written in the form of the Airy equation:

$$\left[\frac{d^2}{d\zeta^2} - \zeta\right] \psi(\zeta) = 0 \tag{4}$$

For this, we need a variable transformation $\zeta = \zeta(z)$. Consider $\alpha\zeta(z) = az - b + k_H^2 = k_H^2 - k^2(z)$, where $\alpha$ is a constant.

Using the chain rule:

$$\frac{d^2\psi(\zeta(z))}{dz^2} = \left(\frac{d\zeta}{dz}\right)^2 \frac{d^2\psi}{d\zeta^2} + \frac{d^2\zeta}{dz^2}\frac{d\psi}{d\zeta} = a_0\frac{d^2\psi}{d\zeta^2} + a_1\frac{d\psi}{d\zeta} \tag{5}$$

where $a_0 = \left(\frac{d\zeta}{dz}\right)^2$ and $a_1 = \frac{d^2\zeta}{dz^2}$. From the definition of $\zeta(z)$, it follows that $a_0 = \left(\frac{a}{\alpha}\right)^2$ and $a_1 = 0$. Substituting Equation 5 in Equation 3 leads to:

$$\left[\left(\frac{a}{\alpha}\right)^2 \frac{d^2}{d\zeta^2} - \alpha\zeta\right]\psi(\zeta) = 0 \tag{6}$$

Solving $\left(\frac{a}{\alpha}\right)^2 = \alpha$ leads to $\alpha = a^{\frac{2}{3}}$ and thus:

$$a^{\frac{2}{3}}\left[\frac{d^2}{d\zeta^2} - \zeta\right]\psi(\zeta) = 0 \tag{7}$$

Omitting trivial solution $a = 0$ leaves the homogeneous Airy equation. Independent solutions to the homogeneous Airy equation are a linear combination of the Airy functions $Ai(\zeta)$ and $Bi(\zeta)$ where:

$$\zeta(z) = a^{-\frac{2}{3}}\left[k_H^2 - k^2(z)\right] \tag{8}$$

so:

$$\psi_a(k_H, z) = Ai\left(a^{-\frac{2}{3}}\left[k_H^2 - k^2(z)\right]\right) \tag{9}$$

$$\psi_b(k_H, z) = Bi\left(a^{-\frac{2}{3}}\left[k_H^2 - k^2(z)\right]\right) \tag{10}$$

therefore, the total solution becomes:

$$\psi(k_H, z) = C_1\psi_a(k_H, z) + C_2\psi_b(k_H, z) \tag{11}$$

For the problem of interest, we require solutions to satisfy the following boundary conditions:

$$\left.\frac{d\psi}{dz}\right|_{z=0} = 0 \qquad\qquad \lim_{z\to\infty}\psi(k_H, z) = 0 \tag{12}$$

Airy functions $Ai(\zeta)$ satisfy these conditions, and therefore $C_2 = 0$. $C_1$ is chosen to be one.

### 1.1.2 Phase and group speed

Phase velocity $c_{ph}$ and group velocity $c_g$ can be computed for each solution $(k_H, \psi)$ as follows:

$$c_{ph} = \frac{\omega}{k_H} \tag{13}$$

$$c_g = \frac{k_H}{\omega \int_0^\infty \psi(k_H, z)dz} = \frac{1}{c_{ph} \int_0^\infty \psi(k_H, z)dz} \tag{14}$$

For each mode, the phase speed provides information on the local horizontal propagation speed while the group speed quantifies the average horizontal speed of the mode through the waveguide. The phase speed corresponds to the observed trace velocity, whereas the group speed corresponds to the celerity.

## 1.2 $n^2$-linear Profile design

The two coefficients that determine a $n^2$-linear profile, $a$ and $b$, can be determined from two sound speed values at two altitudes, say at the bottom ($z_0 = 0$ km) and the top ($z_{max}$) of the domain. From these points, we obtain a set of two equations that can be solved for the two unknown variables $a$ and $b$.

Thus, we have for this $c(z)$ profile the following values:

$$c(z_0) = c_0 \tag{15}$$

$$c(z_{max}) = c_{max} \tag{16}$$

So:

$$k^2(z_0) = \frac{\omega^2}{c^2(z_0)} = b - az_0 \tag{17}$$

$$k^2(z_{max}) = \frac{\omega^2}{c^2(z_{max})} = b - az_{max} \tag{18}$$

Combining and re-arranging leads to the following expressions for $a$ and $b$:

$$a = \frac{k^2(z_0) - k^2(z_{max})}{z_{max} - z_0} \tag{19}$$

$$b = k^2(z_0) + az_0 \tag{20}$$

In most applications, $z_0 = 0$ km and $b$ can be immediately determined from the sound speed on the ground.

## 1.3 Algorithm

[1]:
```python
import numpy as np
from scipy.special import airy
```

[3]:
```python
%matplotlib inline
import matplotlib.pylab as plt
plt.rcParams['figure.dpi'] = 125
```

### 1.3.1 Classes and methods

[4]:
```python
from numpy.polynomial.polynomial import polyval

class Atmosphere(object):
    def __init__(self):
        return

    def get_n2_profile(self,**kwargs):
        z_ = self.z
        z0 = self.z_min
        z1 = self.z_max
        k0 = self.omega/self.c_min
        k1 = self.omega/self.c_max
        a = (k0**2 - k1**2)/(z1 - z0)
        b = k0**2 + a*z0
        c = self.omega/np.sqrt(b-a*z_)
        k2 = (self.omega/c)**2

        # Generate atmospheric state variable profiles
        gamma = 1.4
        R_universal = 8.31446261815324
        molar_mass_air = 0.0289644
        R_specific = R_universal/molar_mass_air
        rho = self.get_toy_density_profile()
        T = c**2 / (gamma*R_specific)
        P = rho*R_specific*T

        self.profile = { 'type': 'n**2 linear profile',
                         'a_coeff': float(a),
                         'b_coeff': float(b),
                         'c': c,
                         'k2': k2,
                         'rho': rho,
                         'P': P,
                         'T': T
                       }
        return

    def get_toy_density_profile(self):
```

```
        z_ = self.z
        rho_0 = 1.225

        A = np.array([ -3.9082017E-02, -1.1526465E-03,
                        3.2891937E-05, -2.0494958E-07,
                       -4.7087295E-02,  1.2506387E-03,
                       -1.5194498E-05,  6.5818877E-08 ])
        B = np.array([ -4.9244637E-03, -1.2984142E-06,
                       -1.5701595E-06,  1.5535974E-08,
                       -2.7221769E-02,  4.2474733E-04,
                       -3.9583181E-06,  1.7295795E-08 ])
        Pa_coeff = A[:4]
        Pb_coeff = B[:4]
        Pa_coeff = np.insert(Pa_coeff, 0, 0)
        Pb_coeff = np.insert(Pb_coeff, 0, 1)
        rho = rho_0*10**(polyval(z_/1e3,Pa_coeff)/polyval(z_/1e3,Pb_coeff))
        return rho

    def write_profile(self,fid):
        z_ = self.z/1e3
        u = np.zeros(self.z.shape)
        v = u
        w = u
        T = self.profile['T']
        rho = self.profile['rho']/1e3
        P = self.profile['P']/1e2
        line = np.c_[z_, u, v, w, T, rho, P ]
        line_fmt = '%9.3f %8.3f %8.3f %8.3f %10.3f %11.4e %11.4e'
        np.savetxt(fid, line, fmt=line_fmt)
        return
```

[5]:
```
class Propagation(Atmosphere):
    def __init__(self):
        return

    def set_parameters(self):
        """
        Based on program input, setup variables and arrays
        """
        self.omega = 2*np.pi*self.freq
        self.k0 = self.omega/self.c_0

        self.cph = np.arange(self.cph_max,
                             self.cph_min-self.dcph,
                             -self.dcph
                            )
        self.kH = self.omega / self.cph
```

```
        return

    def set_altitude_grid(self):
        """
        Build altitude grid based on the wavelength of the signal
        A good choice is to take 10 samples per wavelength
        """
        self.dz = 2*np.pi/self.k0/10
        self.z = np.arange(self.z_min,self.z_max+self.dz,self.dz)
        self.z_max = self.z[-1]
        self.nz = len(self.z)
        self.nzrcv = int(np.floor(self.zrcv/self.dz))
        self.nzsrc = int(np.floor(self.zsrc/self.dz))

    def set_range_grid(self):
        """
        Build range grid based on the wavelength of the signal
        A good choice is to take 10 samples per wavelength
        """
        r_min = self.dr
        r_max = self.r_max + self.dr
        self.r = np.arange(r_min, r_max, self.dr)
        self.nr = len(self.r)

    def compute_transmission_loss(self, p):
        return 20.0*np.log10(np.abs(4*np.pi*p))
```

[6]:
```
from scipy.optimize import root_scalar
from tqdm.notebook import trange, tqdm

class Modes(Propagation):
    def __init__(self):
        return

    def compute_airy_zeta(self, kH, k2):
        a_coeff = self.profile['a_coeff']
        zeta = (kH**2 - k2) / a_coeff**(2./3)
        return zeta

    def compute_airy_function_argument(self, kH):
        """
        Computes the argument to the Airy function (Ai),
        i.e. zeta, as defined in the theoretical background.
        """
        zeta = self.compute_airy_zeta(kH, self.profile['k2'])
        return zeta
```

```python
    def compute_airy_bndcnd(self, kH):
        """
        Helper function to find roots, called from find_modes()
        """
        zeta = self.compute_airy_zeta(kH, self.profile['k2'][0])
        ai, aip, bi, bip = airy(zeta)
        return aip

    def compute_airy_ground(self, kH):
        """
        Returns the Airy function on the ground for a given
        wavenumber kH
        """
        zeta = self.compute_airy_zeta(kH, self.profile['k2'][0])
        ai, aip, bi, bip = airy(zeta)
        return ai

    def compute_airy_eigenfunction(self, kH):
        """
        Compute the normalized eigenfunction for a given mode
        with eigenvalue kH
        """
        zeta = self.compute_airy_zeta(kH, self.profile['k2'])
        ai, aip, bi, bip = airy(zeta)
        ain = self.normalize_function(ai)
        return ain

    def normalize_function(self, psi):
        """
        Helper function to normalize mode shape functions using
        orthonormality relation
        """
        psi_norm = np.sqrt(np.trapz(psi*psi, dx=self.dz))
        return psi / psi_norm

    def find_modes(self):
        """
        Find modes of waveguide by looking at roots of derivative
        of Airy function Brent's method is used to look at roots
        within wavenumber interval [a, b]
        """
        self.kH_modes = []
        self.psi_modes = []

        for ikh in tqdm(range(0,len(self.kH)-1)):
            a = self.kH[ikh]
            b = self.kH[ikh+1]
```

```
        try:
            sol = root_scalar(self.compute_airy_bndcnd,
                              bracket=[a, b],
                              method='brentq')
            kH_mode = sol.root
            #print('Mode isolated at {:6.2f}'.format(self.omega/sol.root))
            self.kH_modes.append(kH_mode)
            self.psi_modes.append(self.compute_airy_eigenfunction(kH_mode))
        except ValueError as e:
            #print('No mode found [ {} ]'.format(e))
            pass

    self.psi_modes = np.array(self.psi_modes)
    self.kH_modes = np.array(self.kH_modes)
    self.n_modes = len(self.kH_modes)
    print('{:d} mode(s) found!'.format(self.n_modes))
    return

def compute_phase_group_speeds(self):
    """
    Compute phase and group speed for a given eigenpair

    cph_j = omega/kH_j
    cg_j  = 1./0 / cph_j*int_0^z_max psi_j^2/c_eff^2 dz
    """
    self.cph_modes = self.omega/self.kH_modes
    self.cg_modes = np.zeros(self.cph_modes.shape)

    for ikh in tqdm(range(0,self.n_modes)):
        psi = self.psi_modes[ikh,:]
        c = self.profile['c']
        integral = np.trapz(psi**2/c**2, dx=self.dz)
        self.cg_modes[ikh] = 1.0/(self.cph_modes[ikh]*integral)
    return

def compute_modal_sum(self):
    """
    Computes modal sum for a selection of eigenpairs
    """
    (rr, _) = np.meshgrid(self.r, self.z)
    prefix = np.exp(-1j*np.pi*0.25)*np.sqrt(1./8./np.pi/rr)

    modal_sum = 0
    for m in range(0,self.n_modes):
        psi = ( np.repeat(self.psi_modes[m,:], self.nr
                     ).reshape(self.nz, self.nr) )
        kH = self.kH_modes[m]
```

```
            modal_sum += psi[self.nzsrc,0]*psi*np.exp(1j*kH*rr)/np.sqrt(kH)

        return prefix*modal_sum
```

### 1.3.2 Main program: A modal expansion for the $n^2$-linear profile as numerical solution

[7]:
```
# Set up propagation object and set attributes
pm = Modes()
pm.freq = 0.4
pm.c_0 = 340.0

pm.zrcv = 0.0*1e3
pm.zsrc = 2.0*1e3
pm.z_min = 0.0
pm.z_max = 25.0*1e3
pm.c_min = 330.0
pm.c_max = 450.0

pm.dr = 0.5e3
pm.r_min = pm.dr
pm.r_max = 1000.0*1e3

pm.cph_min = 330.0
pm.cph_max = 440.0
pm.dcph = 0.5
```

[8]:
```
pm.set_parameters()
pm.set_altitude_grid()
pm.set_range_grid()
```

### 1.3.3 Plotting sound speed profile

Using the two sound speed datapoints, it is now possible to generate a $n^2$-linear profile. The $a$ and $b$ coefficients are also stored for analysis and later use, as the $a$ parameter appears in the argument of the Airy function $Ai(\zeta)$.

From the form of the sound speed profile $c(z) = \frac{\omega}{\sqrt{b - a*z}}$, it follows that the sound speed at $z = 0$ is $c(z = 0) = \frac{\omega}{\sqrt{b}}$ while the function has a vertical asymptote at $z = \frac{b}{a}$, i.e. $\lim_{z \to \frac{b}{a}} c(z) = \infty$

At the altitude of the vertical asymptote, the medium wavenumber squared is 0.

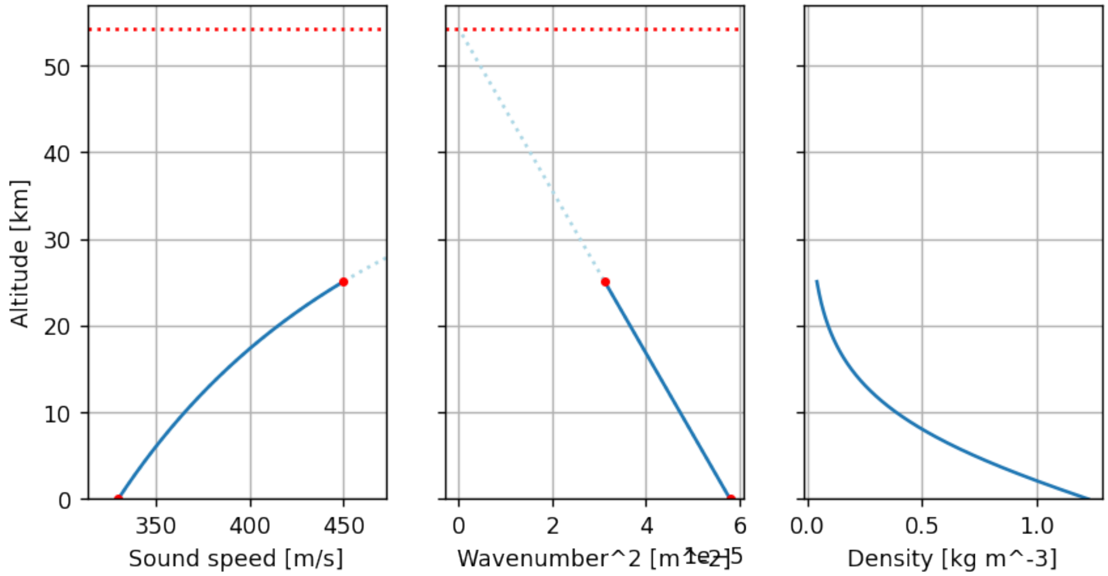
[9]:
```
pm.get_n2_profile()

a=pm.profile['a_coeff']
b=pm.profile['b_coeff']
z_asymptote = b/a

print('n2-linear a-coefficent is {:15.8e} (s^2/m^3)'.format(a))
```

```
print('n2-linear b-coefficent is {:15.8e} (s^2/m^2)'.format(b))
print('')
print('Sound speed at z = 0 km is {:6.2f} m/s'.format(pm.omega/np.sqrt(b)))
print('The vertical asymptote of the sound speed profile is found at altitude {:
↪6.2f} km'.format(z_asymptote/1e3))
```

```
n2-linear a-coefficent is  1.06920682e-09 (s^2/m^3)
n2-linear b-coefficent is  5.80031847e-05 (s^2/m^2)

Sound speed at z = 0 km is 330.00 m/s
The vertical asymptote of the sound speed profile is found at altitude  54.25 km
```

[10]:
```
fig, ax = plt.subplots(1, 3, figsize=(8, 4), sharey=True)

z_tmp = np.arange(pm.z_min,z_asymptote,pm.dz)
c_tmp = pm.omega / np.sqrt(b-a*z_tmp)
k2_tmp = b-a*z_tmp

# plot sound speed
ax[0].plot(c_tmp,z_tmp/1e3, color='lightblue', linestyle=':')
ax[0].plot(pm.profile['c'],pm.z/1e3)
ax[0].plot(pm.c_min,pm.z_min/1e3,'.r')
ax[0].plot(pm.c_max,pm.z_max/1e3,'.r')
ax[0].axhline(y=z_asymptote/1e3, color='r', linestyle=':')
ax[0].set_xlim(pm.c_min*0.95,pm.c_max*1.05)
ax[0].set_ylim(pm.z_min/1e3,1.05*z_asymptote/1e3)
ax[0].set_xlabel('Sound speed [m/s]')
ax[0].set_ylabel('Altitude [km]')
ax[0].grid()

# medium wavenumber
k2_max = (pm.omega/pm.c_min)**2
k2_min = (pm.omega/pm.c_max)**2

ax[1].plot(k2_tmp,z_tmp/1e3, color='lightblue', linestyle=':')
ax[1].plot(pm.profile['k2'],pm.z/1e3)
ax[1].plot(k2_max,pm.z_min/1e3,'.r')
ax[1].plot(k2_min,pm.z_max/1e3,'.r')
ax[1].axhline(y=z_asymptote/1e3, color='r', linestyle=':')
ax[1].set_xlabel('Wavenumber^2 [m^-2]')
ax[1].grid()

# medium wavenumber
k2_max = (pm.omega/pm.c_min)**2
k2_min = (pm.omega/pm.c_max)**2
z_tmp = np.arange(pm.z_min,z_asymptote+pm.dz,pm.dz)
k2_tmp = b-a*z_tmp
```

```
ax[2].plot(pm.profile['rho'],pm.z/1e3)
ax[2].set_xlabel('Density [kg m^-3]')
ax[2].grid()

plt.show()
```




### 1.3.4 Write out profile to disk

This routine writes the profile to a standard 1-D atmosphere format that can be used with NCPA-prop.

[11]:
```
pm.write_profile('profile_n2.dat')
```

### 1.3.5 Analysis of the vertical solution

Recall that the variable of the Airy function reads:

$$\zeta = a^{-\frac{2}{3}}[k_H^2 - k^2(z)] = \frac{k_H^2 + az - b}{a^{\frac{2}{3}}} \tag{21}$$

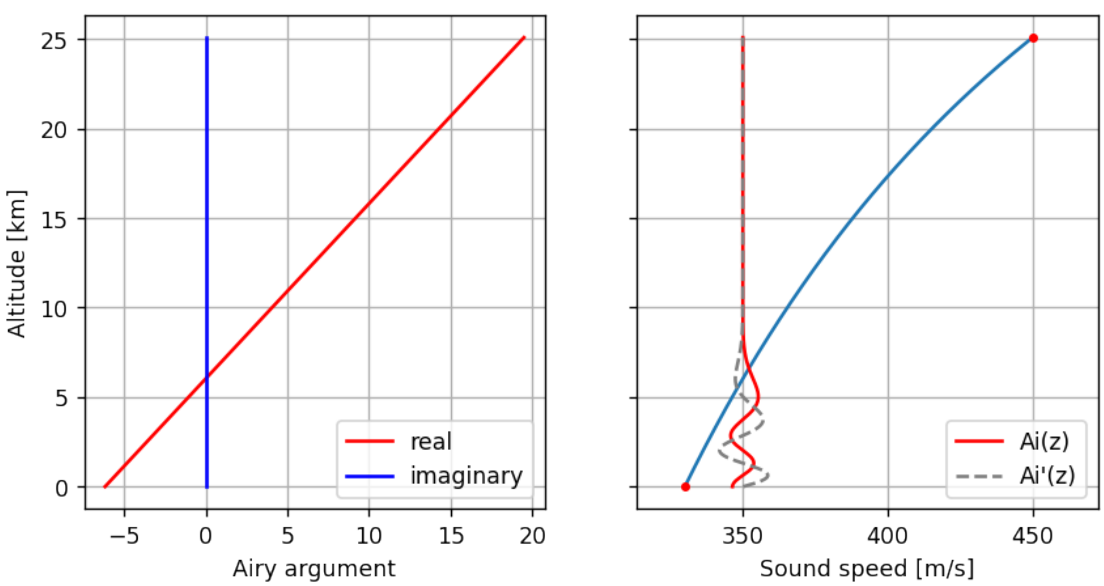
We can now visualize the behavior of the vertical solution as a function of horizontal wavenumber $k_H$.

[12]:
```
c_ph = 350.0

idx = (np.abs(pm.cph - c_ph)).argmin()
kH = pm.kH[idx]
```

```
c_ph = pm.cph[idx]

print('Computing vertical solution with phase speed: {:6.2f} m/s'.format(c_ph))

airy_arg = pm.compute_airy_function_argument(kH)
ai, aip, bi, bip = airy((airy_arg))
```

```
Computing vertical solution with phase speed: 350.00 m/s
```

[13]:
```
fig, ax = plt.subplots(1, 2, figsize=(8, 4), sharey=True)


ax[0].plot(np.real(airy_arg),pm.z/1e3,'r', label='real')
ax[0].plot(np.imag(airy_arg),pm.z/1e3,'b', label='imaginary')
ax[0].set_xlabel('Airy argument')
ax[0].set_ylabel('Altitude [km]')
ax[0].grid()
ax[0].legend(loc='lower right')

ax[1].plot(pm.profile['c'],pm.z/1e3)
ax[1].plot(pm.c_min,pm.z_min/1e3,'.r', label='')
ax[1].plot(pm.c_max,pm.z_max/1e3,'.r', label='')
ax[1].plot(c_ph+ai*10,pm.z/1e3, color='red', label='Ai(z)')
ax[1].plot(c_ph+aip*10,pm.z/1e3, linestyle='--', color='gray', label='Ai\'(z)')
ax[1].set_xlim(0.95*pm.c_min,1.05*pm.c_max)
ax[1].set_xlabel('Sound speed [m/s]')
ax[1].grid()
ax[1].legend(loc='lower right')

plt.show()
```



### 1.3.6 Finding the modes: finding the zero-crossings for the Airy derivative

Recall that vertical solutions that satisfy the following boundary conditions are eigenfunctions.

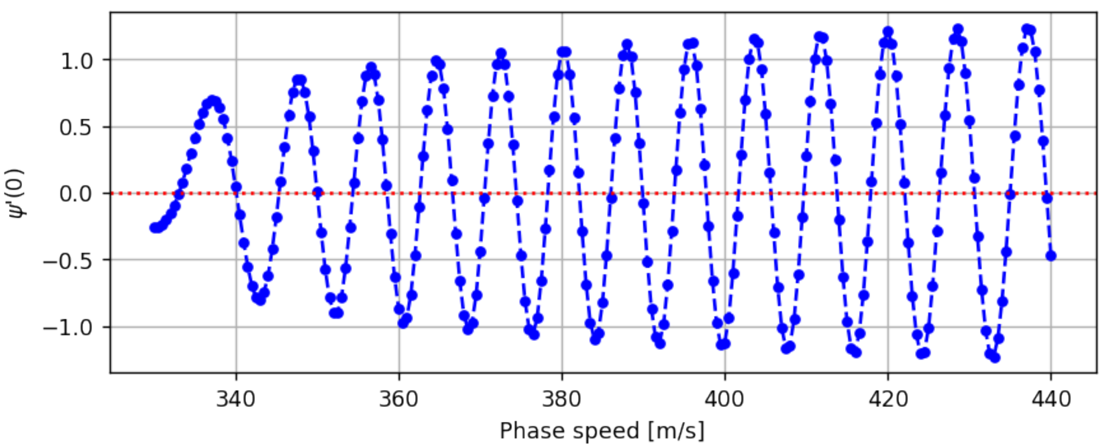
$$\left.\frac{d\psi}{dz}\right|_{z=0} = 0 \qquad\qquad \lim_{z\to\infty} \psi(k_H, z) = 0 \tag{22}$$

By evaluating $\left.\frac{d\psi}{dz}\right|_{z=0}$ for the range of wavenumbers in our domain, it is possible to identify the modes as the wavenumbers for which the value of $\frac{d\psi}{dz}$ changes sign.

As all Airy functions under consideration decay for $k_H > k(z)$, the boundary condition $\lim_{z\to\infty} \psi(k_H, z) = 0$ is satisfied for all wavenumbers $k_H$ considered here.

[14]:
```
boundary_condition = pm.compute_airy_bndcnd(pm.kH)
solution_ground = pm.compute_airy_ground(pm.kH)
```

[15]:
```
fig=plt.figure(figsize=(8, 3))
plt.plot(pm.cph, boundary_condition, linestyle='--',
         marker='o', color='b', markersize=4)
ax=fig.gca()
ax.axhline(y=0., color='r', linestyle=':')
ax.set_xlabel('Phase speed [m/s]')
ax.set_ylabel('$\psi$\'(0)')
ax.grid()
```




A root-finding routine (that makes use of Brent's method) is used to find the positions where the derivative of the Airy function changes sign. The routine returns the eigenpairs as part of the object, where `pm.kH_modes` and `pm.psi_modes` contain the eigenvalues and the eigenfunctions, respectively.

[16]:
```
pm.find_modes()
```

```
  0%|          | 0/220 [00:00<?, ?it/s]

26 mode(s) found!
```

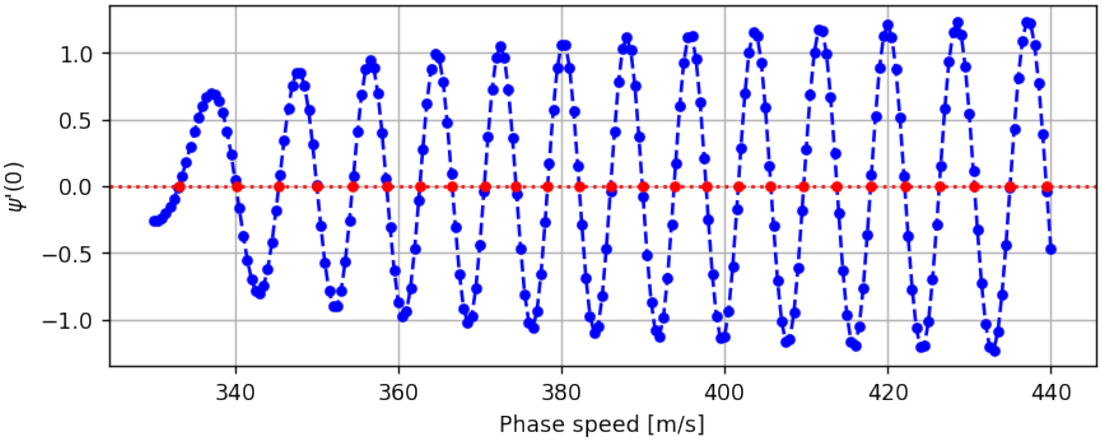
Using the eigenpairs, the phase speed $c_{ph}$ and group speed $c_g$ are computed. These are stored in the `pm.cph_modes` and `pm.cg_modes` arrays, respectively

[17]:
```
pm.compute_phase_group_speeds()
```

```
  0%|          | 0/26 [00:00<?, ?it/s]
```

The wavenumbers can now be plotted on top of the derivative versus phase speed curve, to verify that these are indeed at locations where the derivative of the airy function changes sign.

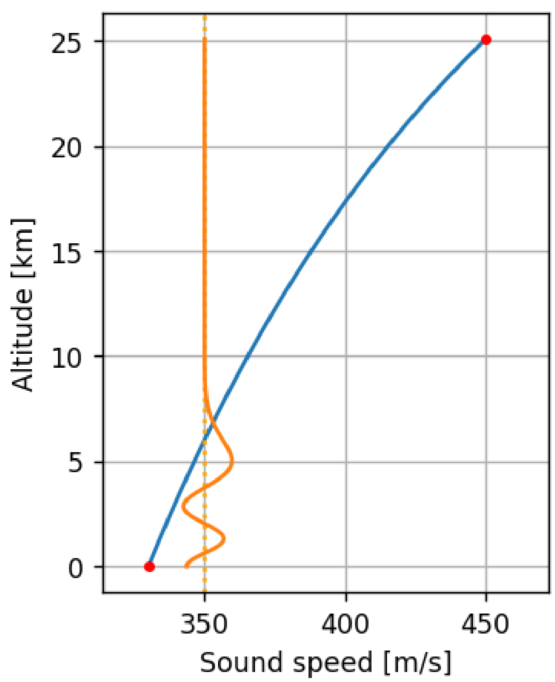
[18]:
```
fig=plt.figure(figsize=(8, 3))
plt.plot(pm.cph, boundary_condition, linestyle='--',
         marker='o', color='b', markersize=4)
plt.plot(pm.cph_modes, np.zeros(pm.cph_modes.shape),
         linestyle='', marker='o', color='red', markersize=4)
ax=fig.gca()
ax.axhline(y=0., color='r', linestyle=':')
ax.set_xlabel('Phase speed [m/s]')
ax.set_ylabel('$\psi$\'(0)')
#ax.set_ylim(-0.02,0.02)
#ax.set_xlim(350,400)
ax.grid()
```




[19]:
```
c_ph = 350.0

idx = (np.abs(pm.cph_modes - c_ph)).argmin()
mode_ = pm.psi_modes[idx,:]
```

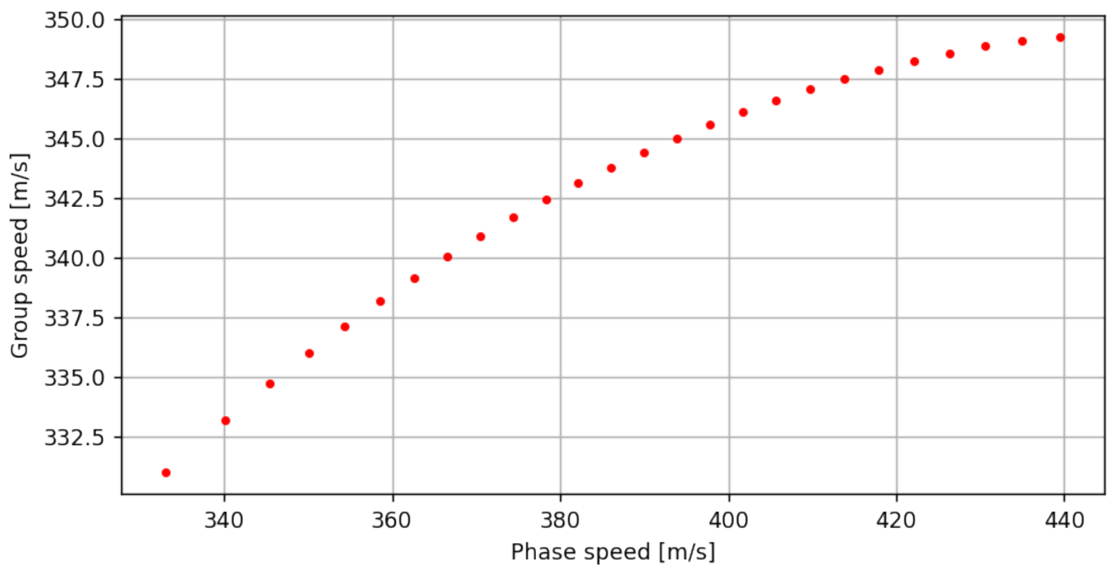
```
c_ph_ = pm.cph_modes[idx]
```

[20]:
```
fig=plt.figure(figsize=(3,4))
plt.plot(pm.profile['c'],pm.z/1e3)
plt.plot(pm.c_min,pm.z_min/1e3,'.r')
plt.plot(pm.c_max,pm.z_max/1e3,'.r')
ax = fig.gca()
ax.axvline(x=c_ph_, color='orange', linestyle=':')
ax.plot(c_ph_+mode_*5e2,pm.z/1e3)
ax.set_xlim(0.95*pm.c_min,1.05*pm.c_max)
ax.set_xlabel('Sound speed [m/s]')
ax.set_ylabel('Altitude [km]')
plt.grid()
plt.show()
```




### 1.3.7 Phase speed versus group speed

[21]:
```
fig=plt.figure(figsize=(8,4))
plt.plot(pm.cph_modes,pm.cg_modes,'.r')
ax = fig.gca()
ax.set_xlabel('Phase speed [m/s]')
ax.set_ylabel('Group speed [m/s]')
plt.grid()
plt.show()
```




### 1.3.8 Discrete spectrum and acoustic field

In general, the total field consists of a contribution from the discrete spectrum and the continuum integral. In the case of ground-to-ground propagation over rigid ground, the former corresponds to the downward refracting field, the latter represents the field in the near-field, i.e. the region above the source.

$$\hat{p}(r,z,\omega) = \overbrace{\frac{e^{-i\frac{\pi}{4}}}{\sqrt{8\pi r}} \sum_{j=1}^{N} \psi_j(z)\psi_j(z_{src}) \frac{e^{ik_{H,j}r}}{\sqrt{k_{H,j}}}}^{\text{discrete sum}} + \overbrace{\oint_{\mathcal{BC}} \psi(z,k_H) e^{ik_H r} \mathrm{dk_H}}^{\text{continuum integral}} \tag{23}$$

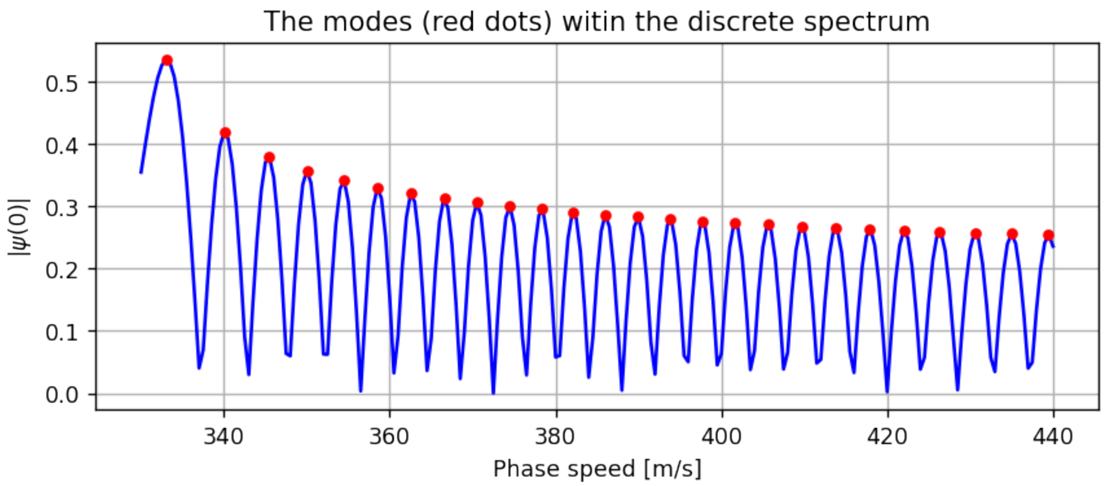
At longer ranges, the continuum integral can be neglected and the field is completely described by the discrete spectrum, i.e. by the modal sum. Using the set of eigenpairs $(k_{H,j}, \psi_j(z))$ we can now compute the Green's function and the transmission loss.

As the vertical part of the solutions are Airy functions, it is straightforward now to compute the full spectral solution for the bracketed wavenumbers and plot the modes as red dots.

It can be seen that the modes correspond to the peaks of the spectral solution. Note that in the case of a wavenumber integration or FFP code, an integration is performed along the blue curve.

[22]:
```
solution_ground_modes = pm.compute_airy_ground(pm.kH_modes)

fig=plt.figure(figsize=(8, 3))
plt.plot(pm.cph, np.abs(solution_ground), color='b')
plt.plot(pm.cph_modes, np.abs(solution_ground_modes),
         linestyle='', marker='o', color='red', markersize=4)
ax=fig.gca()
```

```
ax.set_title('The modes (red dots) witin the discrete spectrum')
ax.set_xlabel('Phase speed [m/s]')
ax.set_ylabel('|$\psi$(0)|')
ax.grid()
```



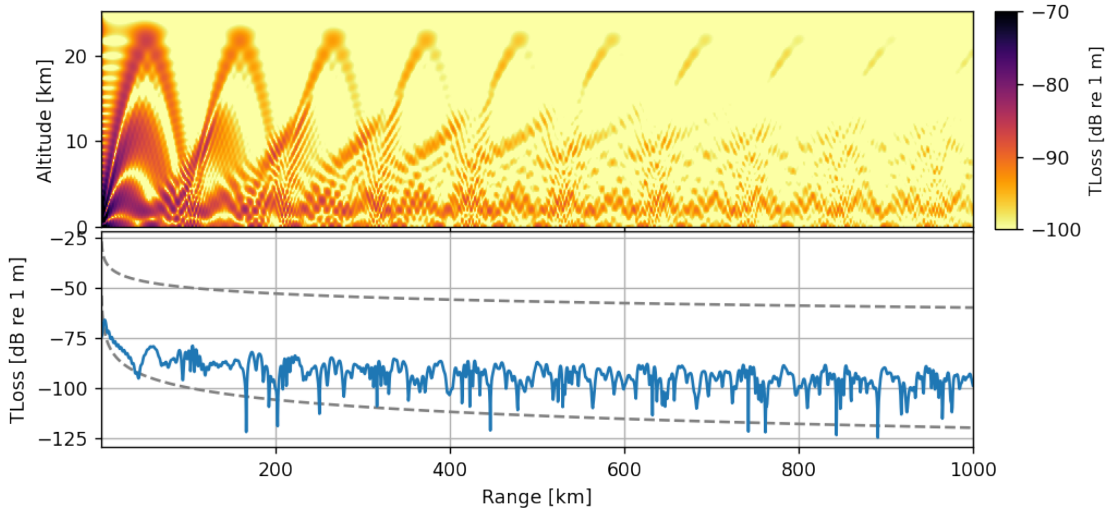

[23]:
```
p = pm.compute_modal_sum()
tl = pm.compute_transmission_loss(p)
tl_rcv = tl[pm.nzrcv,:]
```

It can be useful to compute the transmission loss for Green's functions for spherical spreading (no duct) and cylindrical spreading (ideal duct), to be included as bounds when plotting the transmission loss as a function of range and altitude

[24]:
```
# Compute Transmission loss
tl_spherical = 10*np.log10(1./pm.r)
tl_cylindrical = 20*np.log10(1./pm.r)
```

[25]:
```
plot_db_min = -100
plot_db_max = -70

fig, ax = plt.subplots(2, 1, sharex=True, figsize=(8,4))
fig.subplots_adjust(bottom=0.1, top=0.9, left=0.1,
                    right=0.9, wspace=0.02, hspace=0.02)

im = ax[0].pcolormesh(pm.r/1e3, pm.z/1e3, tl, cmap='inferno_r',
                      vmin=plot_db_min, vmax=plot_db_max,
                      shading='auto', snap=True
                      )
ax[0].set_ylabel('Altitude [km]')
cb_ax = fig.add_axes([0.92, 0.50, 0.02, 0.4])
```

```
cbar = fig.colorbar(im, cax=cb_ax)
cbar.set_label(label='TLoss [dB re 1 m]', size=9)

ax[1]
ax[1].plot(pm.r/1e3, tl_spherical, linestyle='--', color='gray')
ax[1].plot(pm.r/1e3, tl_cylindrical, linestyle='--', color='gray')
ax[1].plot(pm.r/1e3, tl_rcv)
ax[1].grid()
ax[1].set_xlabel('Range [km]')
ax[1].set_ylabel('TLoss [dB re 1 m]')
plt.show()
```




### 1.3.9 Data output

Write out ground transmission loss curve to be compared with other models

[26]:
```
np.savetxt('tl_analytic.dat', np.c_[pm.r/1e3, tl_rcv], fmt='%6.1f %9.3f')
```

[ ]:
```
```