Bellhop_Introduction_2021

July 7, 2021

1 How to use an underwater acoustic propagation modeling with arlpy and Bellhop ?

```
[ ]: # Author : Jay Patel, Dalhousie University.
```

The underwater acoustic propagation modeling toolbox (`uwapm`) in `arlpy` is integrated with the popular Bellhop ray tracer distributed as part of the acoustics toolbox. In this notebook, we see how to use `arlpy.uwapm` to simplify the use of Bellhop for modeling.

-

1.1 Prerequisites

- Install arlpy (v1.6 or higher)

You can install arlpy (in overall system) by typing this in your command prompt: `bash python3 -m pip install arlpy`

- Install the acoustics toolbox (6 July 2020 version or later)

You can use this installation instructions to install acoustics toolbox in overall system.

1.2 Bellhop - Acoustic Toolbox

What is BELLHOP ?

- **BELLHOP** is a beam tracing model for predicting acoustic pressure fields in ocean environments.
- **BELLHOP** can produce a variety of useful outputs including transmission loss, eigenrays, arrivals, and received time-series. It also allows for range-dependence in the top and bottom boundaries (altimetry and bathymetry), as well as in the sound speed profile (SSP).
- **BELLHOP** is implemented in Fortran, Matlab, and Python and used on multiple platforms (Mac, Windows, and Linux).

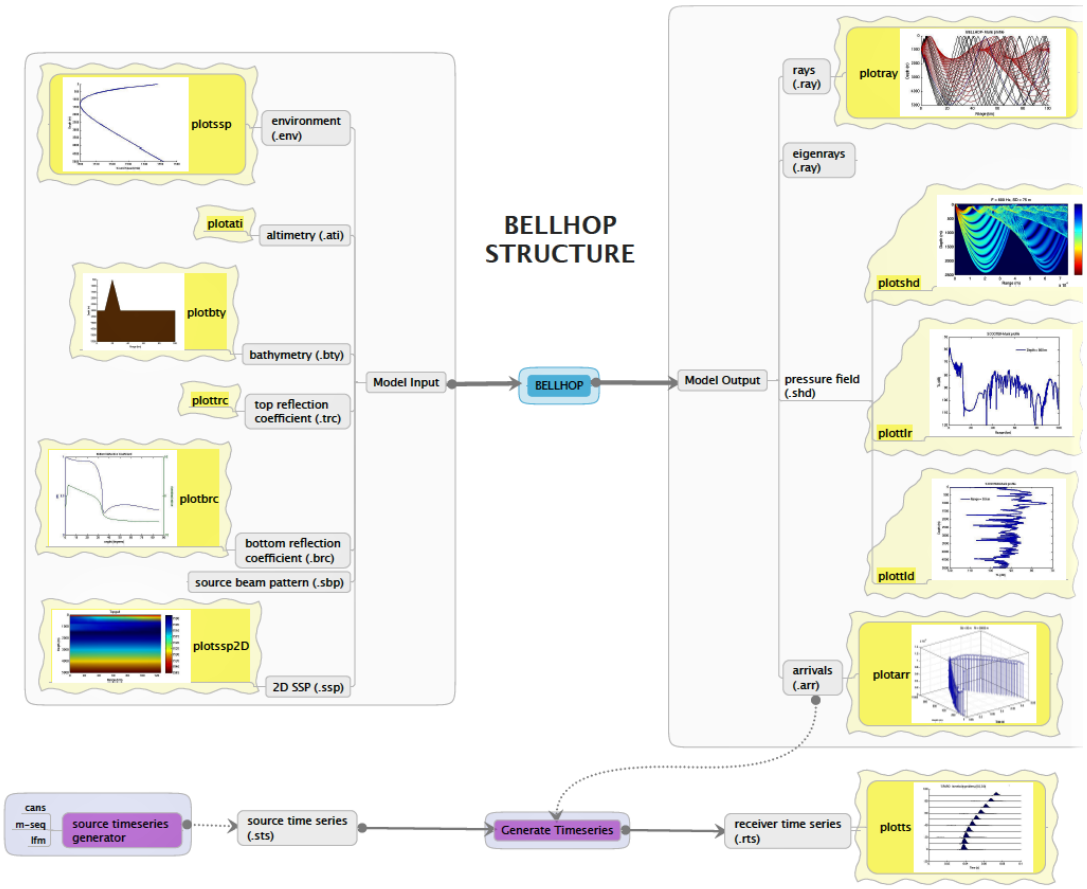


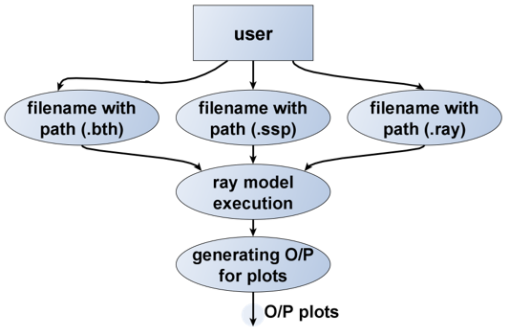
Figure 1: BELLHOP structure

2 WHY BELLHOP?

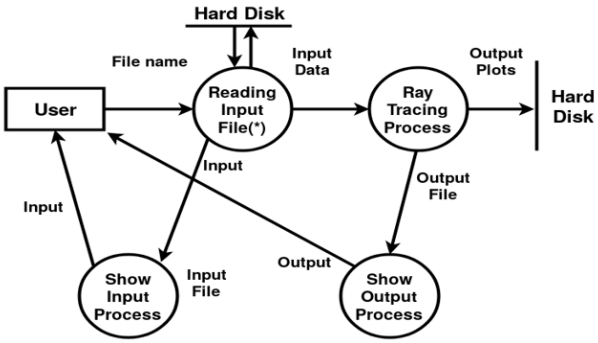
- Underwater communication channel is a relatively difficult transmission medium due to the variability of link quality depending on location and applications.
- Before deploying any kind of vehicles underwater, one should predict the underwater communication system performance which is based on the sound frequency transmitted underwater.

Why do you need to analyze the uw-comms performance?

- To analyze impact of channel characteristics on underwater communications,
- prior to deploying robots, predict communication system performance,
- provide guidance on best physical layout to deploy underwater vehicles,
- provide estimates on parameters for link budget calculation,
- **Because** it will provide you a rough idea about **how far you can communicate within network** which is also known as an **operation range for communication**.



Lower level direct-flow diagram



Top level direct-flow diagram

- **BELLHOP** reads these files depending on options selected within the main environmental file.
- i.e. Below is the example of the environmental file(*.env file).

[ ]:
```
'arlpy'                  / (Graph Title - just for Fortran/MATLAB)
25000.000000             / FREQUENCY
1                        / [UNUSED]
'SVWT*'                  / [(SSP INTERP METHOD)(TOP LAYER)(BOTTOM ATTENUATION␣
 ↪UNITS)(VOLUME ATTENUATION)]
1 0.0 100.000000         / [UNUSED] [UNUSED] [MAX DEPTH]
  0.000000 1540.400000   / [DEPTH] [SOUND SPEED]
  10.000000 1540.500000  / [DEPTH] [SOUND SPEED]
  20.000000 1540.700000  / [DEPTH] [SOUND SPEED]
  30.000000 1534.400000  / [DEPTH] [SOUND SPEED]
  50.000000 1523.300000  / [DEPTH] [SOUND SPEED]
  75.000000 1519.600000  / [DEPTH] [SOUND SPEED]
  100.000000 1518.500000 / [DEPTH] [SOUND SPEED]
'A' 0.000000             / [(BOTTOM LAYER)(EXTERNAL BATHYMETRY] [SIGMA BOTTOM␣
 ↪ROUGHNESS]
100.000000 1600.000000 0.0 1.200000 1.000000 / [MAX DEPTH] [SOUND SPEED BOTTOM]␣
 ↪[UNUSED] [DENSITY BOTTOM] [ATTENUATION BOTTOM]
1                        / [NUMBER SOURCES]
50.000000                / [SOURCE DEPTHS 1:N (m)]
1                        / [NUMBER RECEIVER DEPTHS]
8.000000                 / [RECEIVER DEPTHS 1:N (m)]
1                        / [NUMBER RECEIVER RANGES]
2.000000                 / [RECEIVER RANGES 1:N (km)]
'R *'                    / [(RUN TYPE)(BEAM TYPE)(EXTERNAL SOURCE BEAM PATTERN)]␣
 ↪; typically "R/C/I/S" - Refer next line
0                        / [NUMBER OF BEAMS] ; default is 0 that means bellhop␣
 ↪calculates it by itself using
-80.000000 80.000000     / [MIN ANGLE] [MAX ANGLE]
0.0 101.000000 2.020000  / [STEP SIZE] [DEPTH BOX] [RANGE BOX] ; default step␣
 ↪size is 0.
```

- There are various options for which you can run bellhop are: **(That can be found in the *.env file as RUN TYPE)**
  - ray tracing option (R),
  - eigenray option (E),
  - transmission loss option ,
    - Coherent TL calculations (C)
    - Incoherent TL calculations (I)
    - Semi-coherent TL calculations (S)
  - an arrivals calculation option in ascii (A) ; an arrivals calculation option in binary (a)

3 Sound Speed Profile of Bedford Basin (taken on 13-10-17)

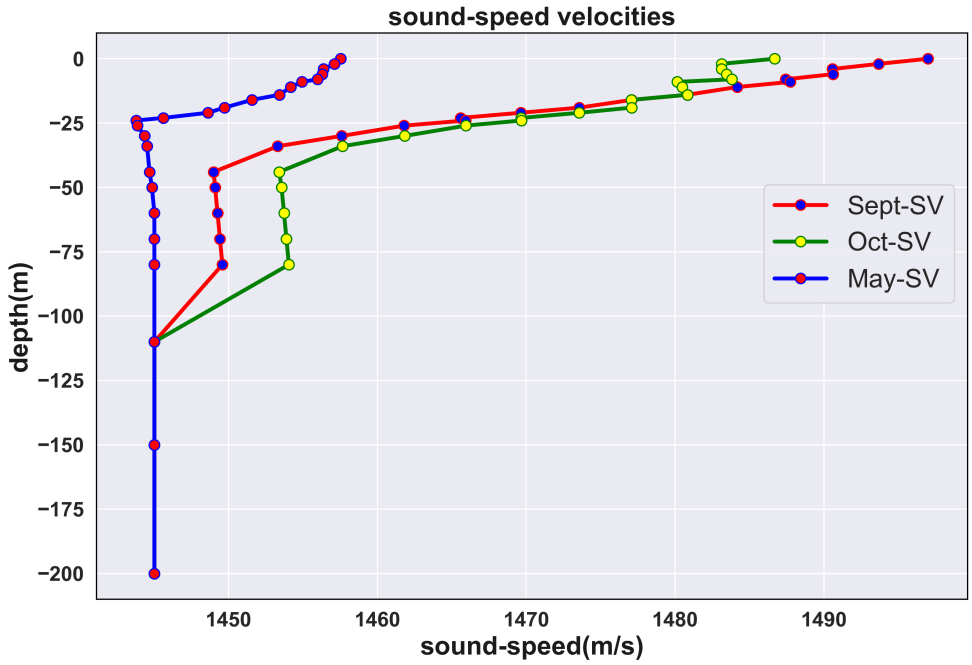
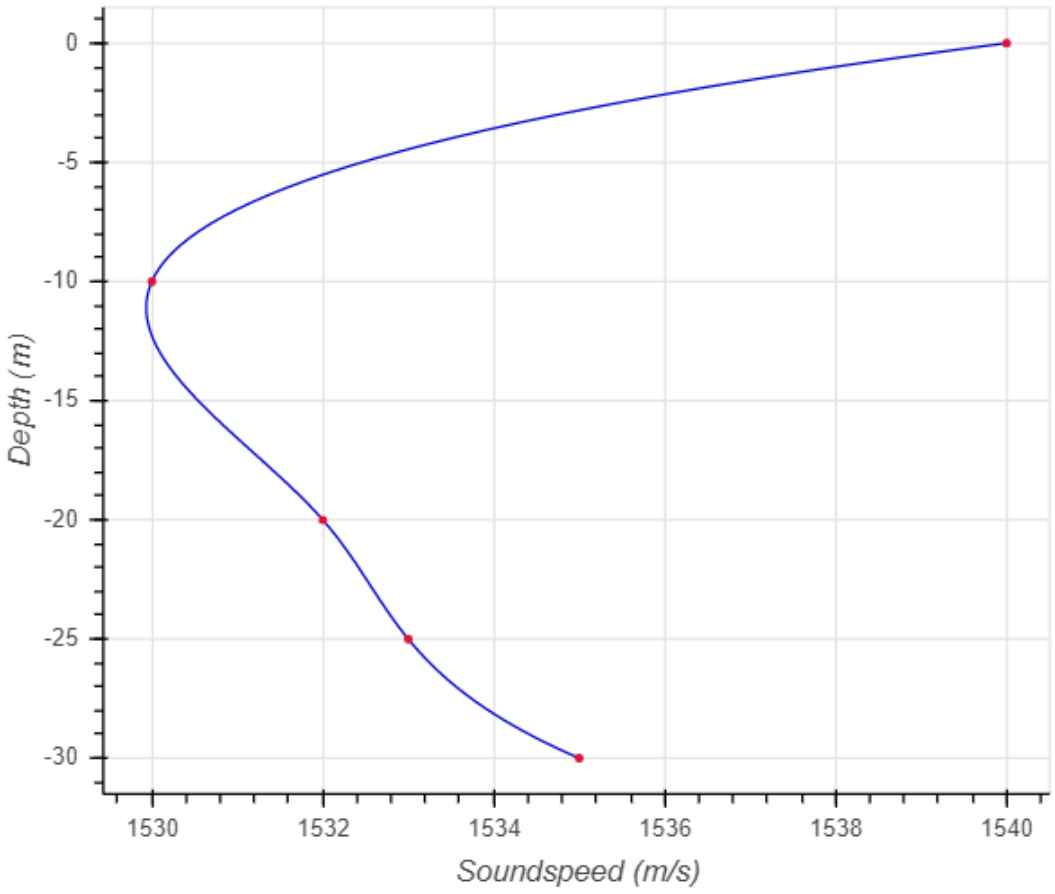


Figure 2: Sound Speed Profile

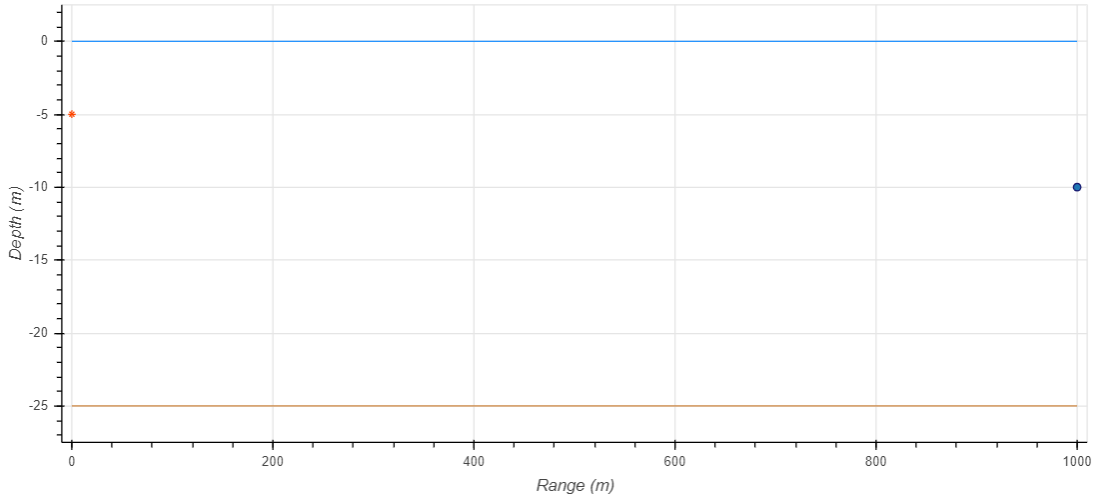
[1]:
```
import arlpy.uwapm as pm
import arlpy.plot as plt
import numpy as np

env = pm.create_env2d()
ssp = [
    [ 0, 1540],  # 1540 m/s at the surface
    [10, 1530],  # 1530 m/s at 10 m depth
    [20, 1532],  # 1532 m/s at 20 m depth
    [25, 1533],  # 1533 m/s at 25 m depth
```

```
    [30, 1535]   # 1535 m/s at the seabed
]
env = pm.create_env2d(soundspeed=ssp)
pm.plot_ssp(env, width=500)
```



4 Plotting an Environment

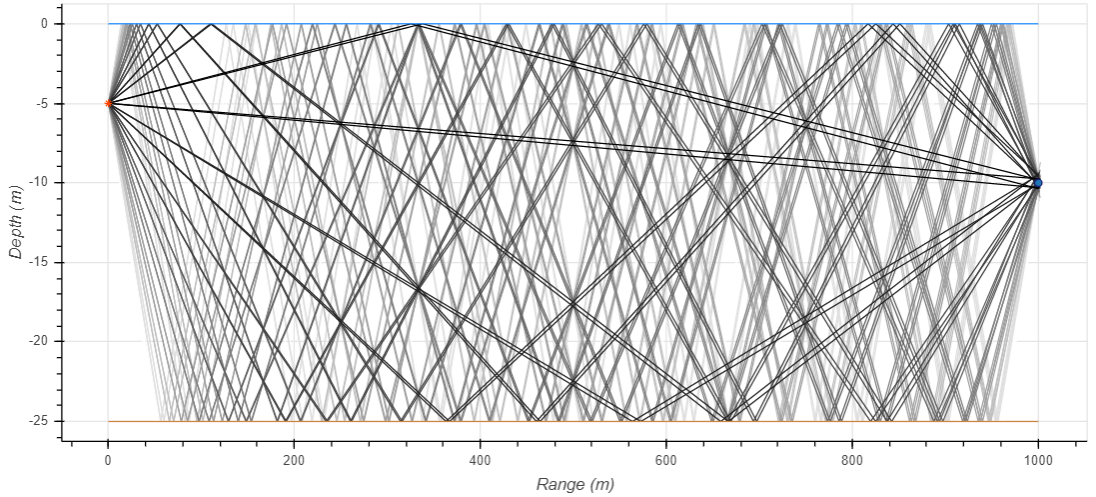
[3]:
```
# Plotting an Environment using ARLPY

pm.plot_env(env, width=900)
```



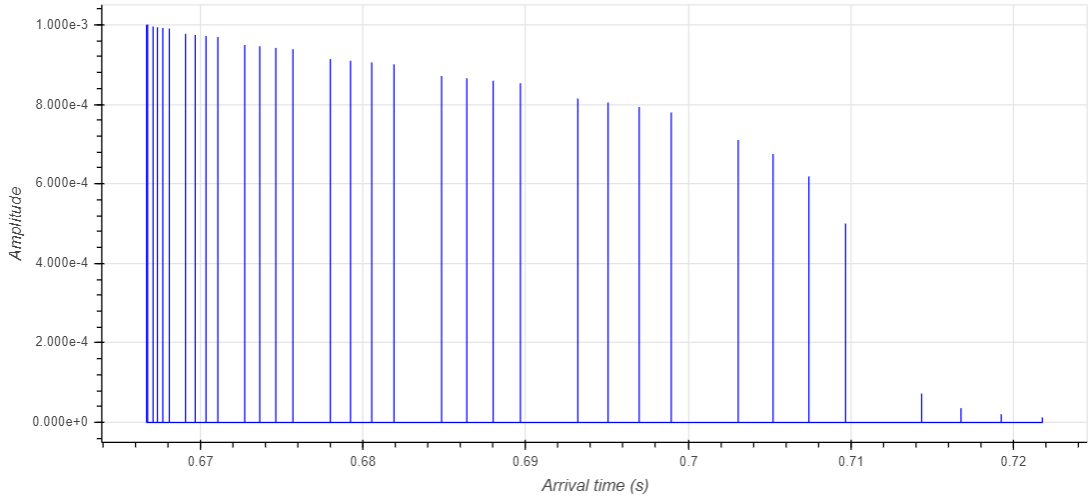
5 Eigenrays

- **Eigenray** plots show just the rays that connect the source to a receiver.

[47]:
```
# Eigenrays using ARLPY
import arlpy.uwapm as pm
import arlpy.plot as plt
import numpy as np

env = pm.create_env2d()
rays = pm.compute_eigenrays(env)
pm.plot_rays(rays, env=env, width=900)
```



[48]:
```
# compute the arrival structure at the receiver

arrivals = pm.compute_arrivals(env)
pm.plot_arrivals(arrivals, width=900)
```

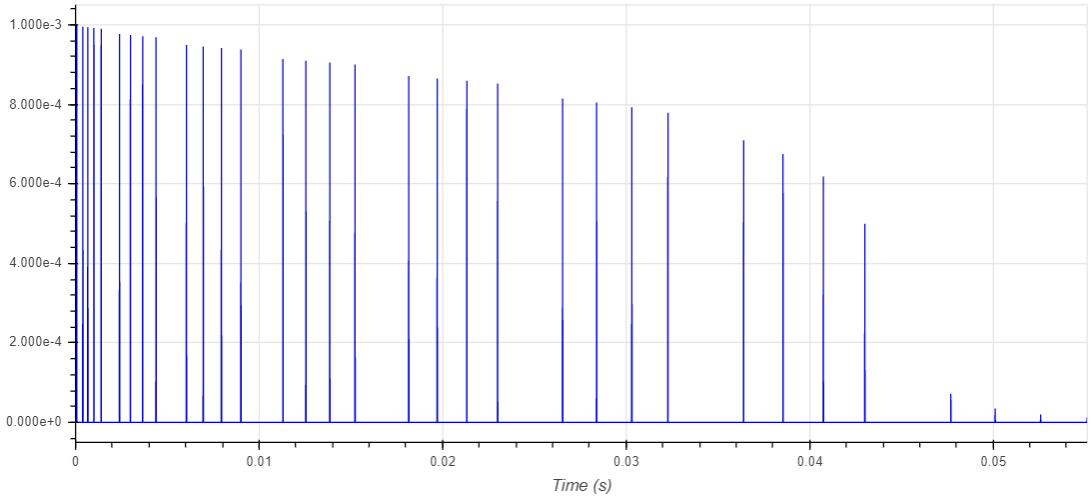


[49]:
```
arrivals[arrivals.arrival_number < 10][['time_of_arrival', 'angle_of_arrival',␣
↪'surface_bounces', 'bottom_bounces']]
```

[49]:

	time_of_arrival	angle_of_arrival	surface_bounces	bottom_bounces
1	0.721796	22.538254	9	8
2	0.716791	-21.553932	8	8
3	0.709687	20.052078	8	7
4	0.705226	-19.034414	7	7
5	0.698960	17.484421	7	6
6	0.695070	-16.436060	6	6
7	0.689678	14.842224	6	5
8	0.686383	-13.766296	5	5
9	0.681901	12.133879	5	4
10	0.679223	-11.034208	4	4

[50]:
```
# convert to a impulse response time series

ir = pm.arrivals_to_impulse_response(arrivals, fs=96000)
plt.plot(np.abs(ir), fs=96000, width=900)
```



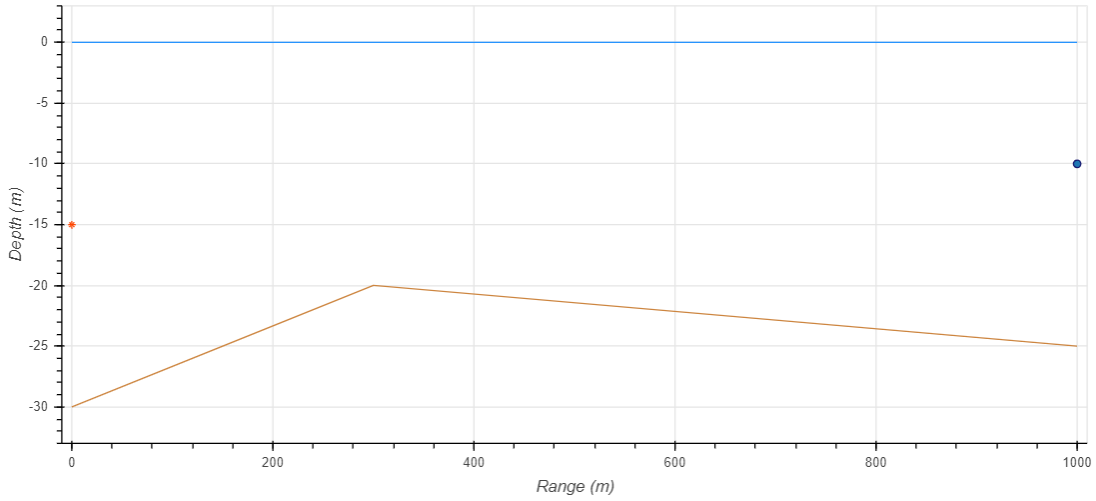
6 Bathymetry

Let's first start off by defining our bathymetry, a steep up-slope for the first 300 m, and then a gentle downslope:

[51]:
```
# add/change bathy to env
bathy = [
    [0, 30],     # 30 m water depth at the transmitter
    [300, 20],   # 20 m water depth 300 m away
    [1000, 25]   # 25 m water depth at 1 km
]

# add/change SSP to env
ssp = [
    [ 0, 1540],  # 1540 m/s at the surface
    [10, 1530],  # 1530 m/s at 10 m depth
    [20, 1532],  # 1532 m/s at 20 m depth
    [25, 1533],  # 1533 m/s at 25 m depth
    [30, 1535]   # 1535 m/s at the seabed
]

# Appending ssp and bathy to existing env file
env = pm.create_env2d(
    depth=bathy,
    soundspeed=ssp,
    bottom_soundspeed=1450,
    bottom_density=1200,
    bottom_absorption=1.0,
    tx_depth=15
)
pm.print_env(env)
```

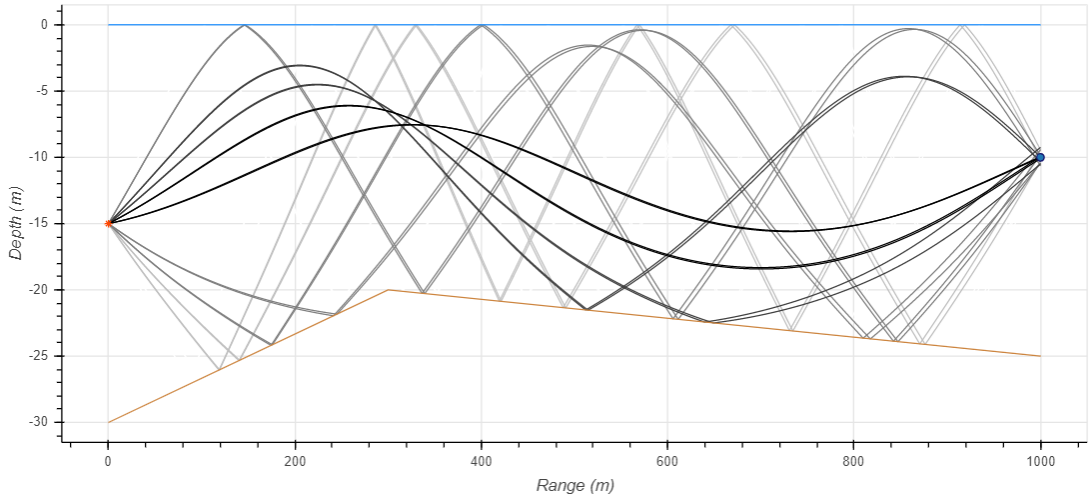
```
             name : arlpy
bottom_absorption : 1.0
   bottom_density : 1200
 bottom_roughness : 0
bottom_soundspeed : 1450
            depth : [[   0.   30.]
                     [ 300.   20.]
                     [1000.   25.]]
      depth_interp : linear
         frequency : 25000
         max_angle : 80
         min_angle : -80
            nbeams : 0
          rx_depth : 10
          rx_range : 1000
        soundspeed : [[   0. 1540.]
                      [  10. 1530.]
                      [  20. 1532.]
                      [  25. 1533.]
                      [  30. 1535.]]
soundspeed_interp : spline
           surface : None
    surface_interp : linear
          tx_depth : 15
tx_directionality : None
             type : 2D
```

[52]:
```
pm.plot_env(env, width=900)
```



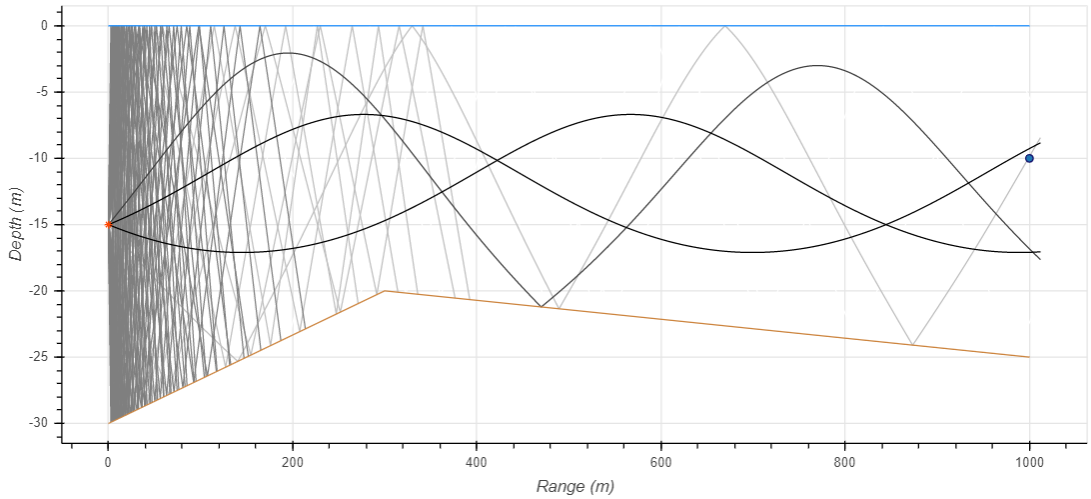
Looks more interesting! Let's see what the eigenrays look like, and also the arrival structure:

[53]:
```
rays = pm.compute_eigenrays(env)
pm.plot_rays(rays, env=env, width=900)
```



We could also ignore the receiver, and plot rays launched at various angles:

[54]:
```
rays = pm.compute_rays(env)
pm.plot_rays(rays, env=env, width=900)
```



7 Complex Bathymetry Example

[55]:
```
import numpy as np
from scipy.interpolate import griddata
import scipy.ndimage as ndimage
from scipy.ndimage import gaussian_filter
import scipy
# from scipy.misc import imsave
```

```
from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from stl import mesh, Mode
import matplotlib.tri as mtri
from mpl_toolkits.mplot3d.axes3d import get_test_data
from pandas import read_csv



data = read_csv('bathy.txt', sep='\s+', header=None, names=['x', 'y', 'depth'])

x = np.arange(data.x.min(), data.x.max()+1)
y = np.arange(data.y.min(), data.y.max()+1)

X, Y = np.meshgrid(x, y)

Z = griddata(data[['x','y']].values, -data['depth'].values, (X, Y),␣
 ↪method='linear')

# make the grid square
Z[np.isnan(Z)] = 0

fig = plt.figure(figsize=(14, 8))
ax = fig.add_subplot(111)
plt.imshow(Z)
plt.show()
```

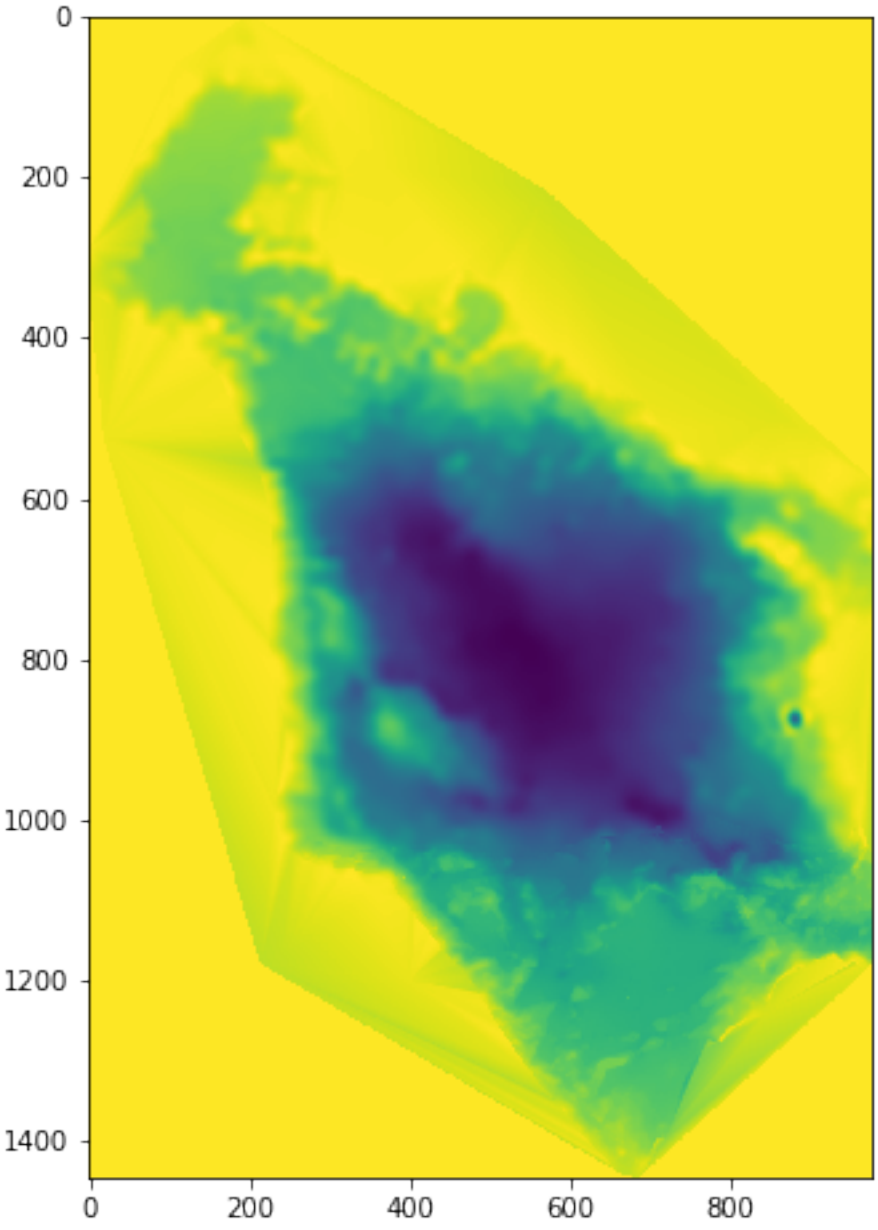


Figure 3: Bedford Basin Bathy 2D

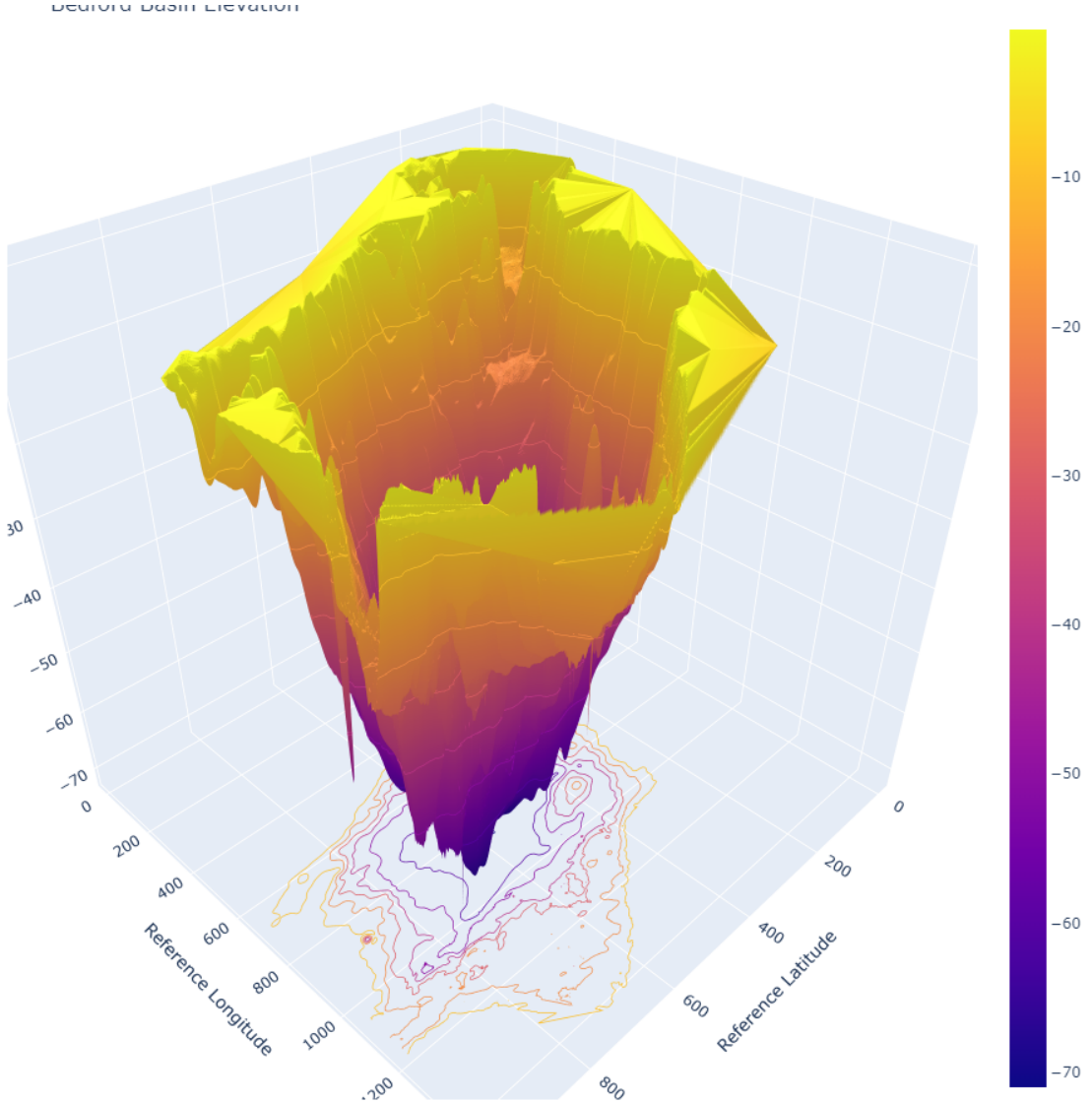


Figure 4: Bedford Basin Bathy 3D

or place lots of receivers in a grid to visualize the acoustic pressure field (or equivalently transmission loss). We can modify the environment (env) without having to recreate it, as it is simply a Python dictionary object:

[56]:
```
env['rx_range'] = np.linspace(0, 1000, 1001)
env['rx_depth'] = np.linspace(0, 30, 301)
```

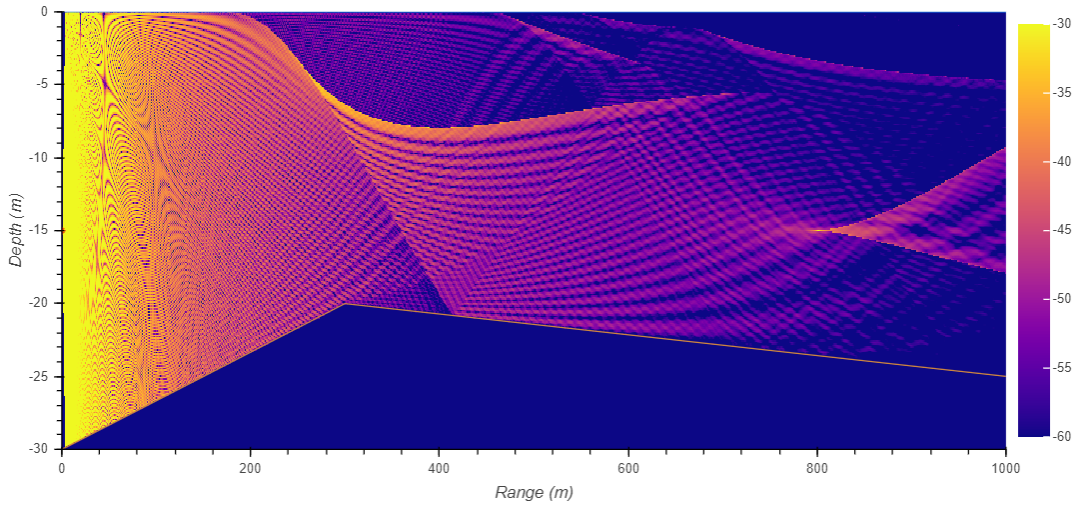
8 Transmission Loss

[ ]:
```
- RUN TYPE BELLHOP

OPTION(1:1):            'R' generates a ray file
                        'E' generates an eigenray file
```

```
                        'A' generates an amplitude-delay file (ascii)
                        'a' generate  an amplitude-delay file (binary)
                        'C' Coherent     TL calculation
                        'I  Incoherent   TL calculation
                        'S' Semicoherent TL calculation
                             (Lloyd mirror source pattern)

- The number of beams, NBeams, should normally be set to 0, allowing BELLHOP to␣
 ↪automatically select the appropriate value. The number
needed increases with frequency and the maximum range to a receiver.
```

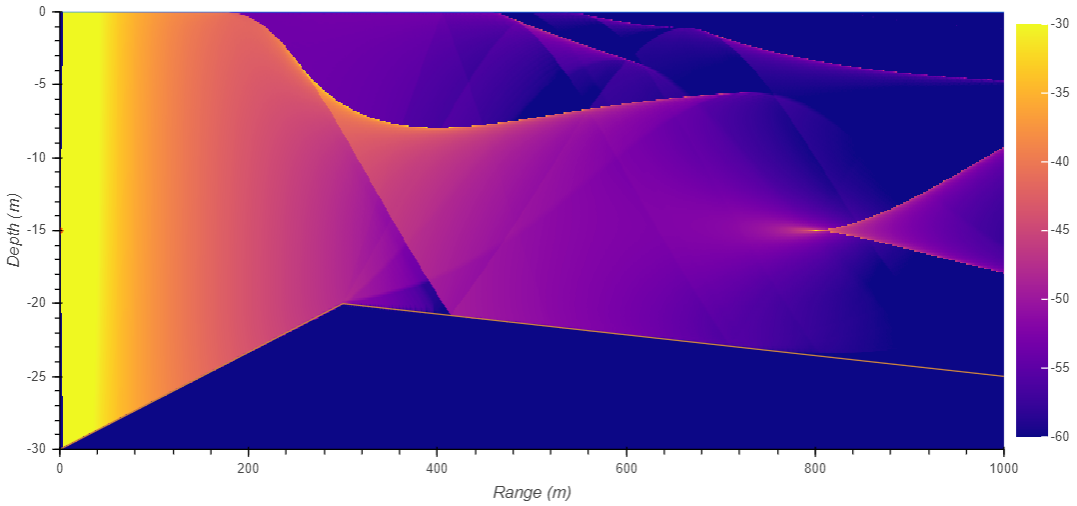
- The pressure field, p, is then calculated for the specified grid of receivers, with a scaling such that 20 $log10(|p|)$ is the transmission loss in dB.

[57]:
```
tloss = pm.compute_transmission_loss(env)
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```



We see a complicated interference pattern, but an interesting focusing at 800 m at a 15 m depth. The detailed interference pattern is of course sensitive to small changes in the environment. A less sensitive, but more averaged out, transmission loss estimate can be obtained using the incoherent mode:

[58]:
```
tloss = pm.compute_transmission_loss(env, mode='incoherent')
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```

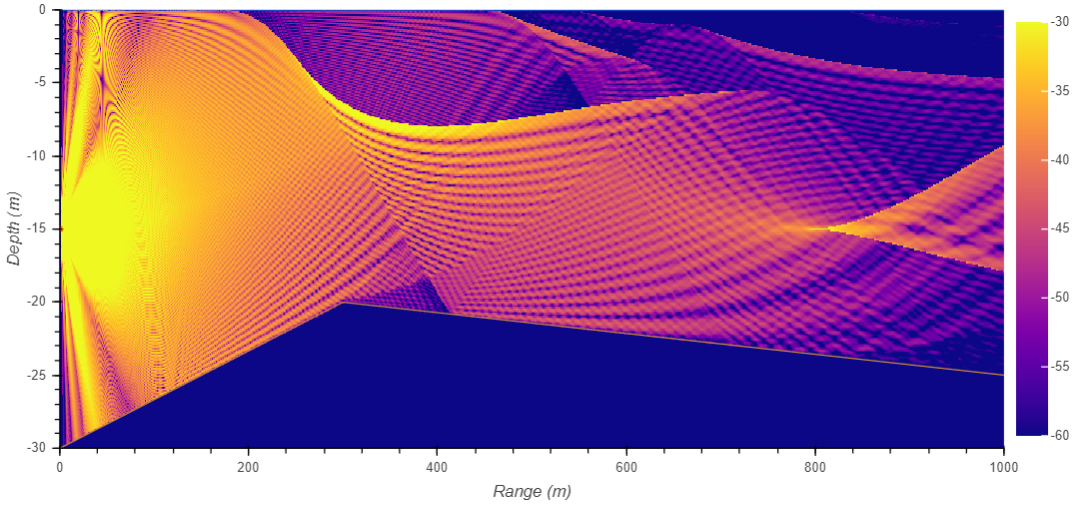


9 Source directionality

Now, let's use a directional transmitter instead of an omni-directional one:

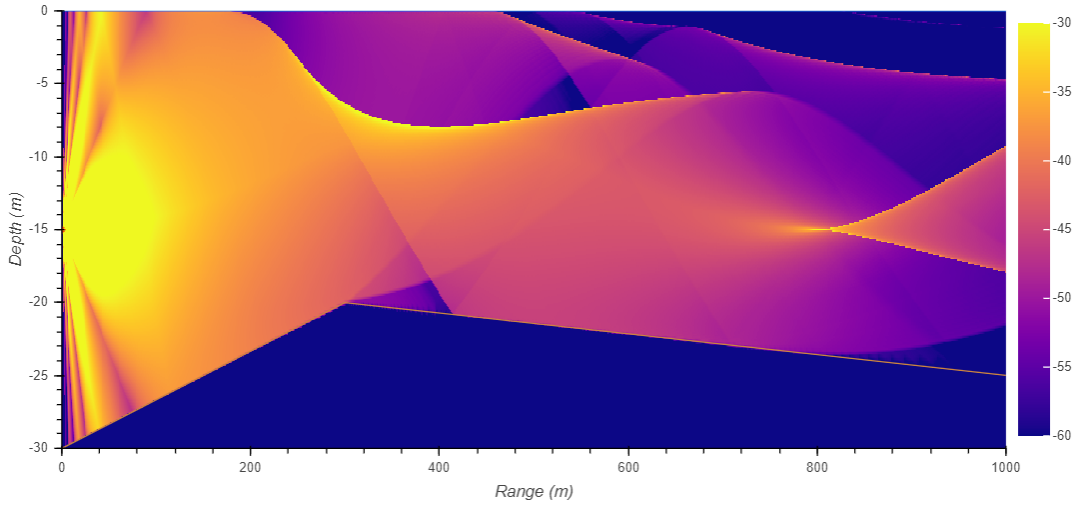
[59]:
```
beampattern = np.array([
    [-180,  10], [-170, -10], [-160,   0], [-150, -20], [-140, -10], [-130, -30],
    [-120, -20], [-110, -40], [-100, -30], [-90 , -50], [-80 , -30], [-70 , -40],
    [-60 , -20], [-50 , -30], [-40 , -10], [-30 , -20], [-20 ,   0], [-10 , -10],
    [  0 ,  10], [ 10 , -10], [ 20 ,   0], [ 30 , -20], [ 40 , -10], [ 50 , -30],
    [ 60 , -20], [ 70 , -40], [ 80 , -30], [ 90 , -50], [100 , -30], [110 , -40],
    [120 , -20], [130 , -30], [140 , -10], [150 , -20], [160 ,   0], [170 , -10],
    [180 ,  10]
])
env['tx_directionality'] = beampattern
```

[60]:
```
tloss = pm.compute_transmission_loss(env)
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```



Now you can see the directionality and the sidelobe structure of the transmitter.

[61]:
```
tloss = pm.compute_transmission_loss(env, mode='incoherent')
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```

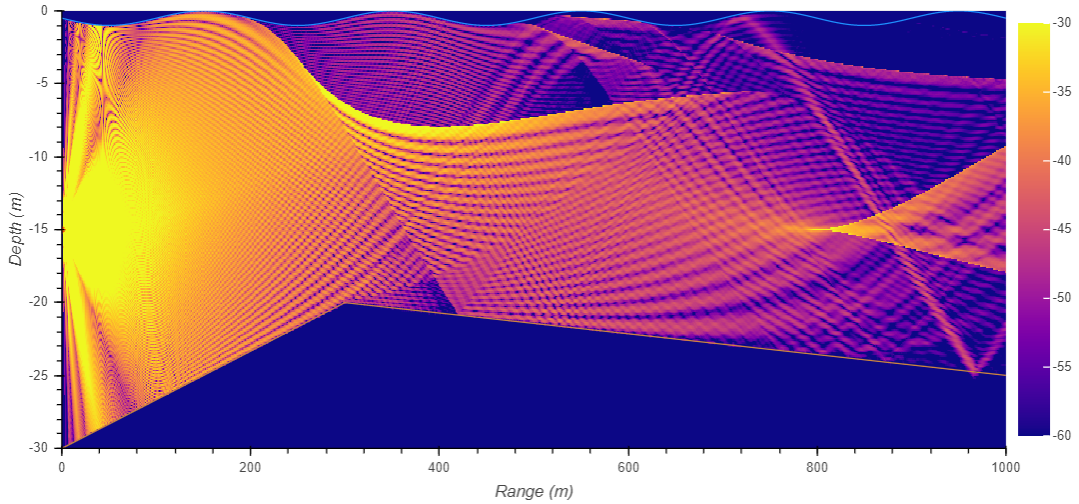


10 Undulating water surface

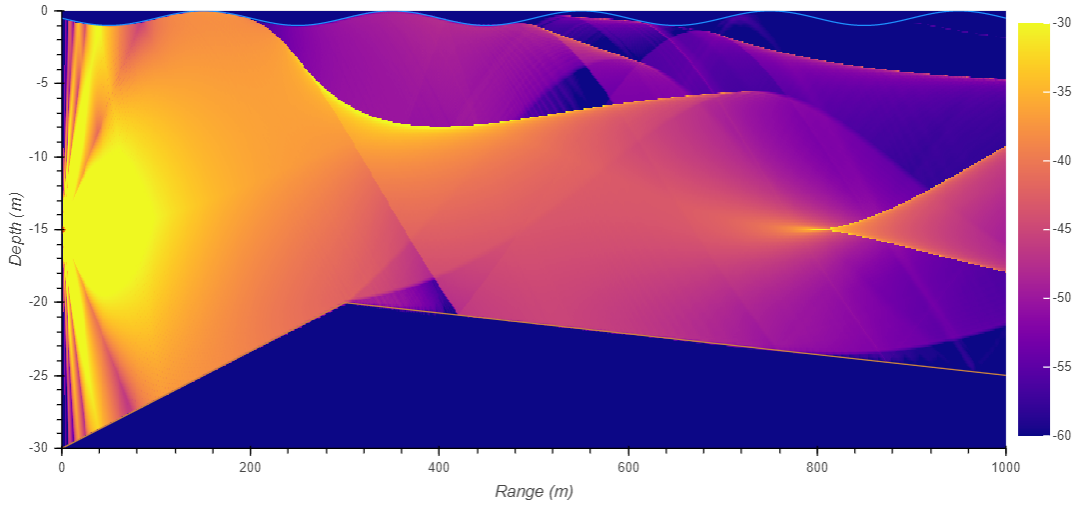
Finally, let's try adding a long wavelength swell on the water surface:

[62]:
```
surface = np.array([[r, 0.5+0.5*np.sin(2*np.pi*0.005*r)] for r in np.
 ↪linspace(0,1000,1001)])
env['surface'] = surface
```

[63]:
```
tloss = pm.compute_transmission_loss(env)
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```



[64]:
```
tloss = pm.compute_transmission_loss(env, mode='incoherent')
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```

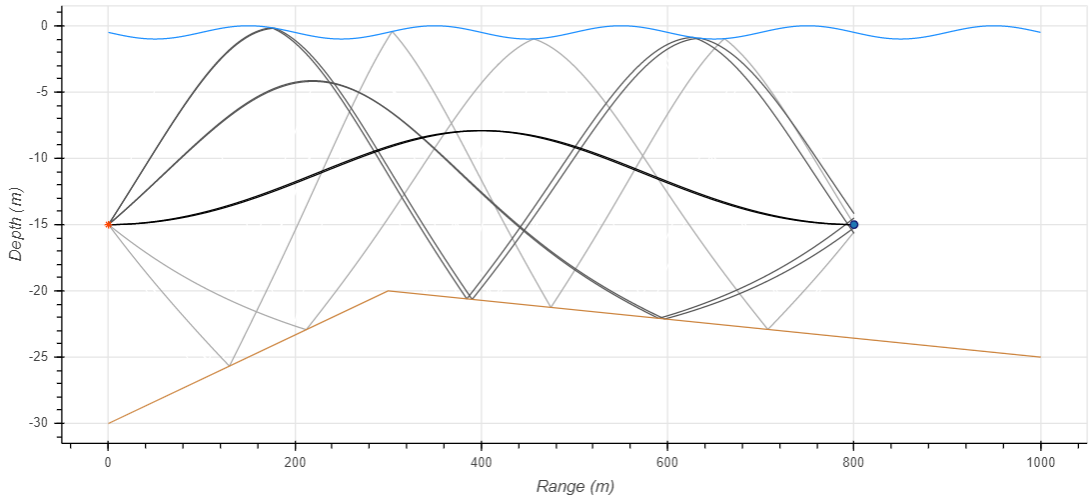


Now, if I placed a receiver at 800 m, and 15 m depth, roughly where we see some focusing, what would the eigenrays and arrival structure look like?

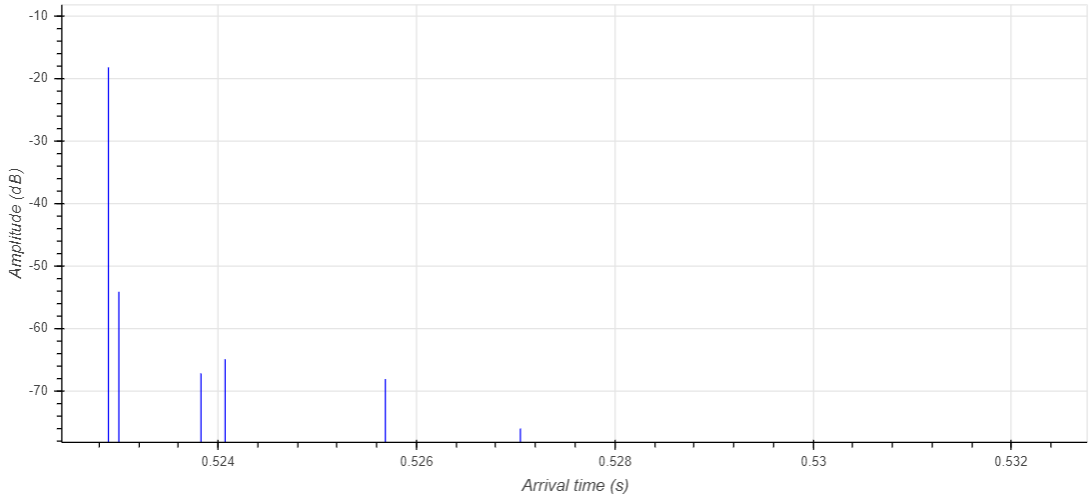
[ ]:
```
Now, if I placed a receiver at 800 m, and 15 m depth, roughly where we see some␣
↪focusing, what would the eigenrays and arrival structure look like
```

[65]:
```
env['rx_range'] = 800
env['rx_depth'] = 15
```

[66]:
```
rays = pm.compute_eigenrays(env)
pm.plot_rays(rays, env=env, width=900)
```



[67]:
```
arrivals = pm.compute_arrivals(env)
pm.plot_arrivals(arrivals, dB=True, width=900)
```



Note: We plotted the amplitudes in dB, as the later arrivals are much weaker than the first one, and better visualized in a logarithmic scale.

11 Bellhop3D

- **BELLHOP3D** is a beam tracing model for predicting acoustic pressure fields in ocean environments.
- It is an extension to 3D environments of the popular BELLHOP model and includes (optionally) horizontal refraction in the lat-long plane.
- 3D pressure fields can be calculated by a 2D model simply by running it on a series of radials (bearing lines) from the source.(This is the so-called Nx2D or 2.5D approach.)
- **BELLHOP3D** includes 4 different types of beams:
  - Cerveny Beams,
  - Geometric Hat-Beams,
  - Geometric Gaussian-Beams,
  - Geometric Hat-Beams in Cartesian Coordinates

11.0.1 P.S. : Not Supported right now on the python, still works great on MATLAB.

[ ]:
```
----------------------------------------------------------------------------------------------------
```

12 Another simulation case :

Start off with checking that everything is working correctly:

[1]:
```
# Author : Jay Patel, Dalhousie University
# ECED 6575
import arlpy.uwapm as pm
import arlpy.plot as plt
import numpy as np
import pandas as pd
```

[2]:
```
pm.models()
```

[2]:
```
['bellhop']
```

[3]:
```
# We next create an underwater 2D environment (with default settings) to model:

env = pm.create_env2d()
pm.print_env(env)
```

```
              name : arlpy
  bottom_absorption : 0.1
     bottom_density : 1600
   bottom_roughness : 0
  bottom_soundspeed : 1600
             depth : 25
       depth_interp : linear
          frequency : 25000
          max_angle : 80
          min_angle : -80
             nbeams : 0
           rx_depth : 10
           rx_range : 1000
         soundspeed : 1500
  soundspeed_interp : spline
            surface : None
     surface_interp : linear
           tx_depth : 5
  tx_directionality : None
               type : 2D
```

[9]:
```
# ============================================================================
# Sound speed profile
# ============================================================================
# ssp = 1500

# Depth dependent sound speed as an array
ssp = [
    [0, 1540.4],   # Speed at the surface
    [10, 1540.5],
    [20, 1540.7],
    [30, 1534.4],
    [50, 1523.3],
    [75, 1519.6],
    [100, 1518.5]  # Speed at the seabed
]
```

[10]:
```
depth = 100
```

[11]:
```
# Bottom properties
bottom_absorption = 1.0
bottom_density = 1200
```

[12]:
```
# ============================================================================
# Surface parameters
# ============================================================================
surface = None

# Surface profile
# surface = np.array([[r, 0.5+0.5*np.sin(2*np.pi*0.005*r)]
#                     for r in np.linspace(0, scenario.rx_range, 1+scenario.
 ↪rx_range)])
```

[13]:
```
# ============================================================================
# Ambient noise
# ============================================================================
sea_state = 3

# Ambient noise table
# TODO: Convert to lookup table based on sea_state and scenario.tx_frequency
an = pd.DataFrame({
        1: [34],                # profile at SS1
        2: [39],                # profile at SS2
        3: [47],                # profile at SS3
        4: [50],                # profile at SS4
        5: [52],                # profile at SS5
        6: [54]},               # profile at SS6
        index=[20000])          # frequency of profiles in Hz
```

[14]:
```
# ============================================================================
# TX properties
# ============================================================================
# tx_beampattern = np.array([
#     [-180,  10], [-170, -10], [-160,   0], [-150, -20], [-140, -10], [-130,␣
 ↪-30],
#     [-120, -20], [-110, -40], [-100, -30], [-90 , -50], [-80 , -30], [-70 ,␣
 ↪-40],
#     [-60 , -20], [-50 , -30], [-40 , -10], [-30 , -20], [-20 ,   0], [-10 ,␣
 ↪-10],
#     [  0 ,  10], [ 10 , -10], [ 20 ,   0], [ 30 , -20], [ 40 , -10], [ 50 ,␣
 ↪-30],
#     [ 60 , -20], [ 70 , -40], [ 80 , -30], [ 90 , -50], [100 , -30], [110 ,␣
 ↪-40],
#     [120 , -20], [130 , -30], [140 , -10], [150 , -20], [160 ,   0], [170 ,␣
 ↪-10],
#     [180 ,  10]
```

```
# ])
tx_beampattern = None
tx_depth = 50           # m
tx_frequency = 20000    # Hz
tx_source_level = 150   # dB
tx_speed_range = 20     # knots
```

[15]:
```
# ============================================================================
# RX properties
# ============================================================================
rx_bandwidth = 10           # Hz
rx_depth = 15               # m
rx_detection_threshold = 5  # dB
rx_directivity_index = 10   # dB
rx_range = 2000             # m
```

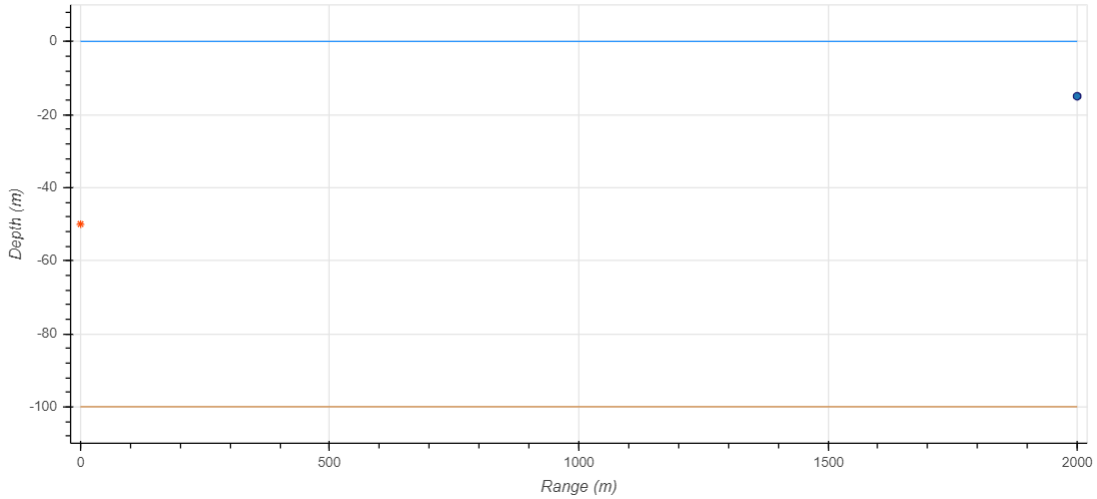
[16]:
```
env = pm.create_env2d(
        bottom_absorption=bottom_absorption,
        bottom_density=bottom_density,
        depth=depth,
        soundspeed=ssp,
        surface=surface,
        rx_depth=rx_depth,
        rx_range=rx_range,
        tx_depth=tx_depth,
        tx_directionality=tx_beampattern
    )
```

[17]:
```
pm.print_env(env)
```

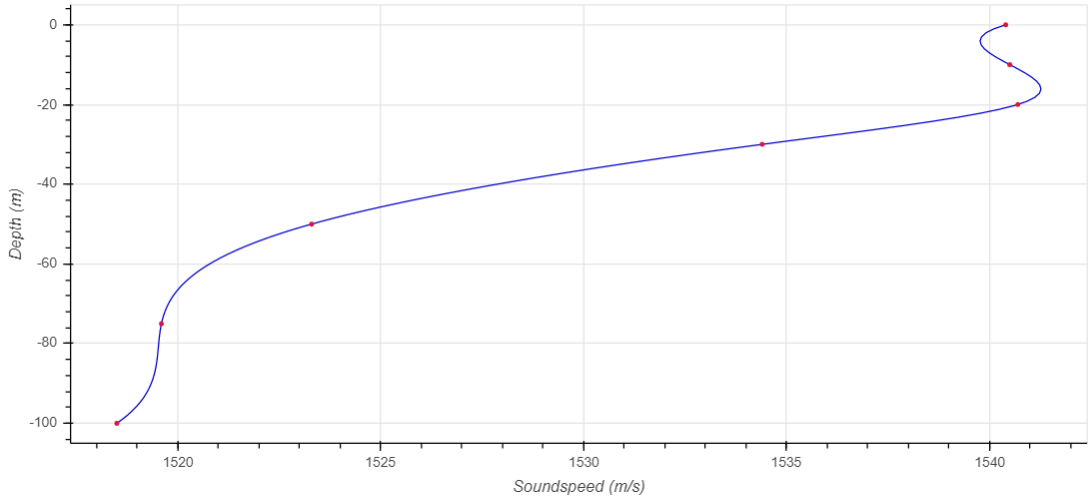
```
              name : arlpy
  bottom_absorption : 1.0
     bottom_density : 1200
   bottom_roughness : 0
  bottom_soundspeed : 1600
             depth : 100
       depth_interp : linear
          frequency : 25000
          max_angle : 80
          min_angle : -80
             nbeams : 0
           rx_depth : 15
           rx_range : 2000
         soundspeed : [[   0.  1540.4]
                       [  10.  1540.5]
                       [  20.  1540.7]
                       [  30.  1534.4]
                       [  50.  1523.3]
```

```
                    [  75.  1519.6]
                    [ 100.  1518.5]]
soundspeed_interp : spline
          surface : None
   surface_interp : linear
        tx_depth : 50
tx_directionality : None
             type : 2D
```

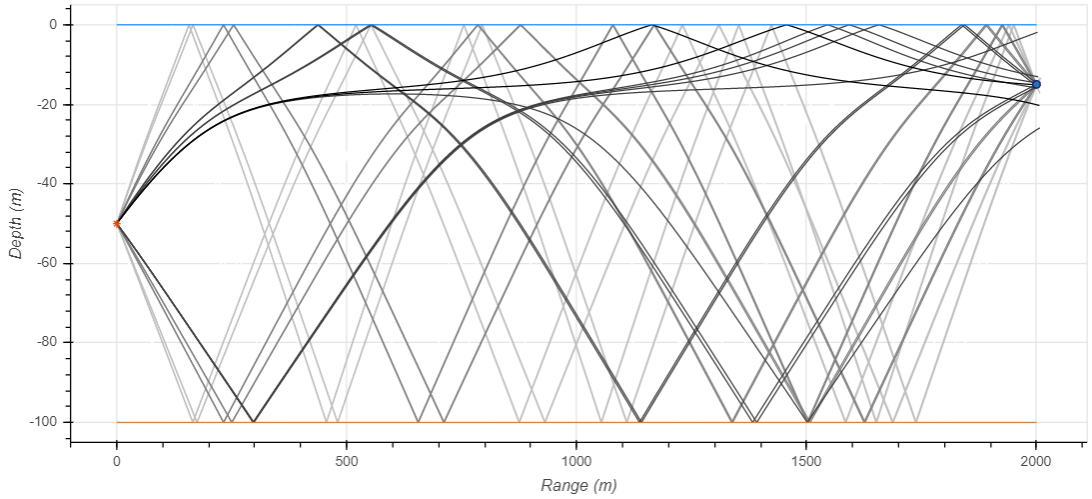
[18]:
```
pm.plot_env(env, width=900)
```



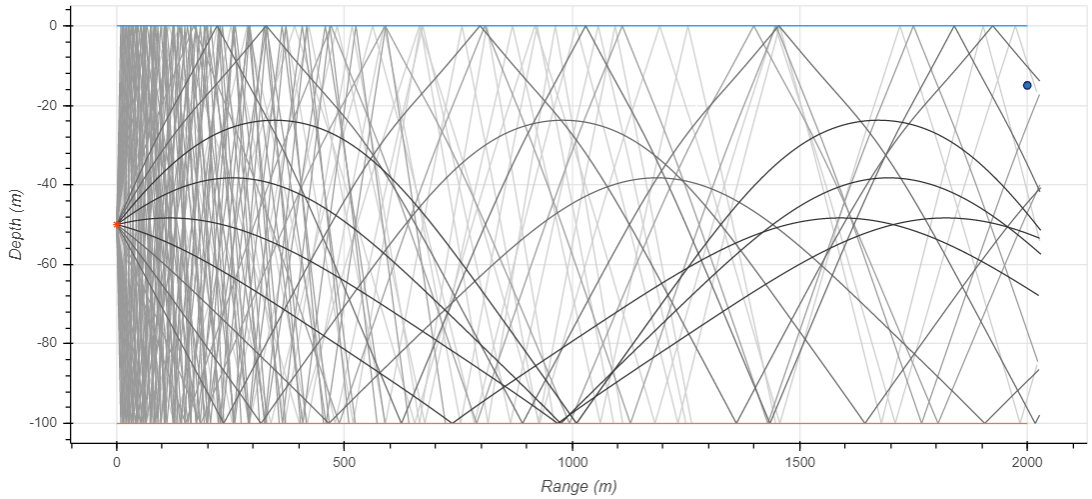
[19]:
```
pm.plot_ssp(env, width=900)
```



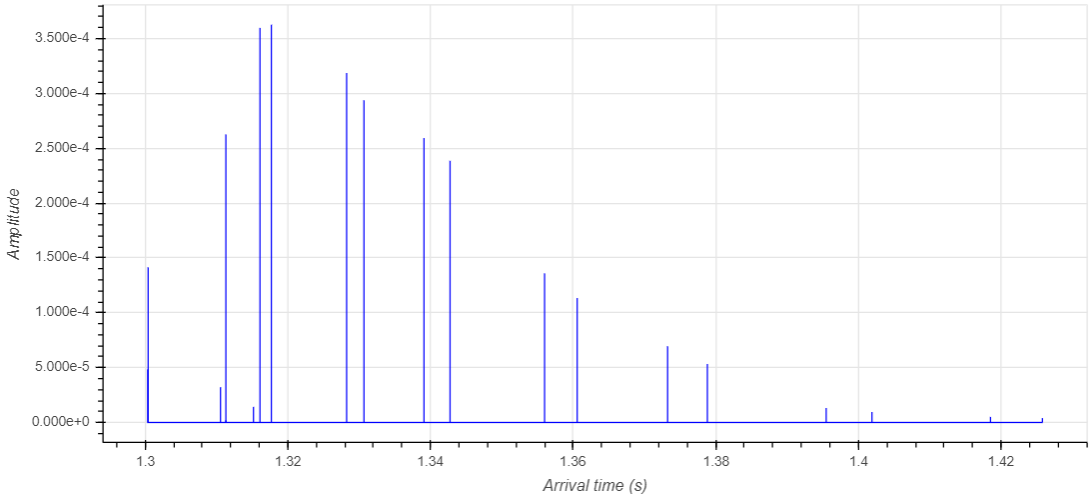
[20]:
```
rays = pm.compute_eigenrays(env)
pm.plot_rays(rays, env=env, width=900)
```



[21]:
```
rays = pm.compute_rays(env , debug=False)
pm.plot_rays(rays, env=env, width=900)
```



[22]:
```
arrivals = pm.compute_arrivals(env)
pm.plot_arrivals(arrivals, width=900)
```

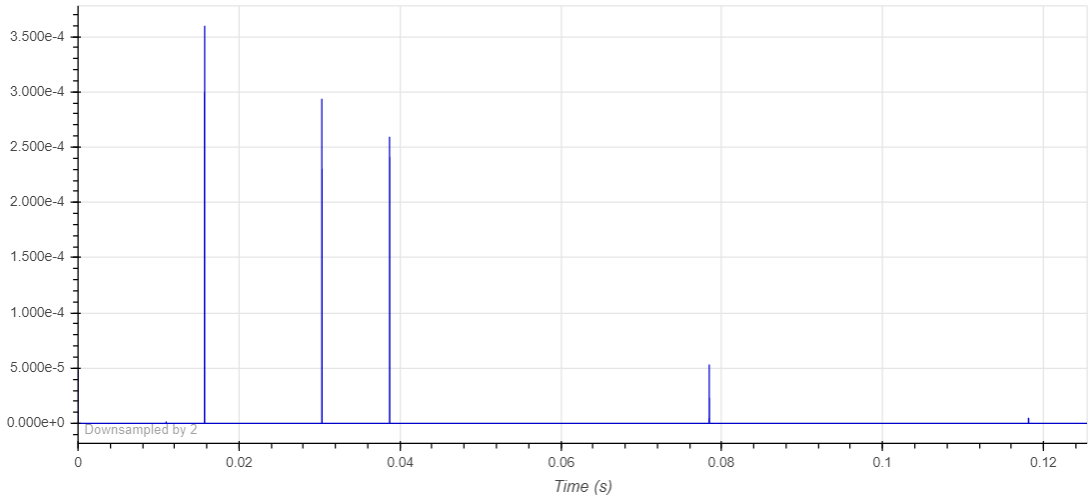


[23]:
```
arrivals[arrivals.arrival_number < 10][['time_of_arrival', 'angle_of_arrival',␣
↪'surface_bounces', 'bottom_bounces']]
```

[23]:

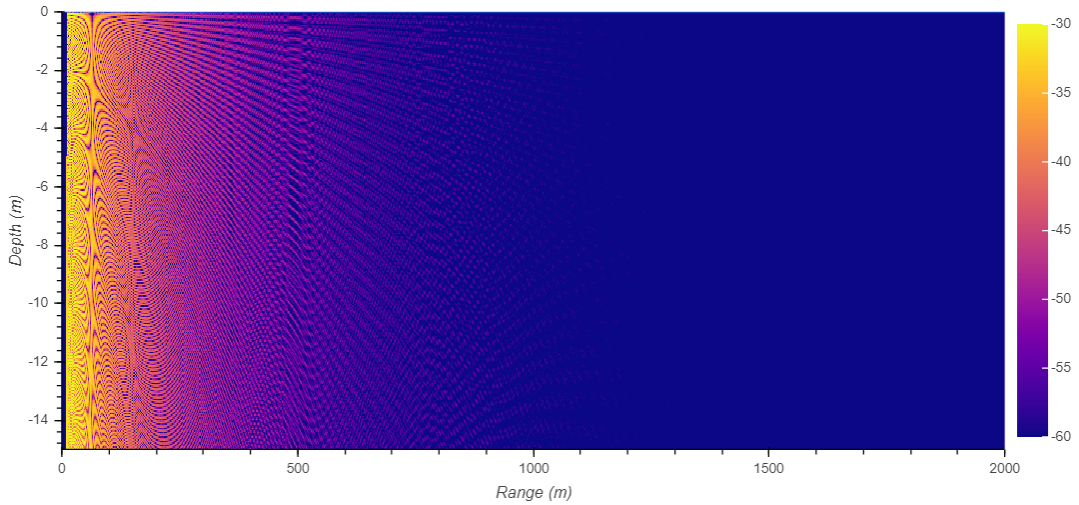
	time_of_arrival	angle_of_arrival	surface_bounces	bottom_bounces
1	1.425794	22.258884	5	4
2	1.418515	-21.452885	4	4
3	1.378822	16.974577	4	3
4	1.373249	-16.041763	3	3
5	1.342748	11.180706	3	2
6	1.339094	-10.067203	2	2
7	1.317695	4.996730	2	1
8	1.316098	-3.503432	1	1
9	1.300367	1.719691	1	0
10	1.300419	0.426278	1	0

[24]:
```
ir = pm.arrivals_to_impulse_response(arrivals, fs=96000)
plt.plot(np.abs(ir), fs=96000, width=900)
```



[25]:
```
env['rx_range'] = np.linspace(0, 2000, 1001)
env['rx_depth'] = np.linspace(0, 15, 301)
```

[26]:
```
tloss = pm.compute_transmission_loss(env)
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```

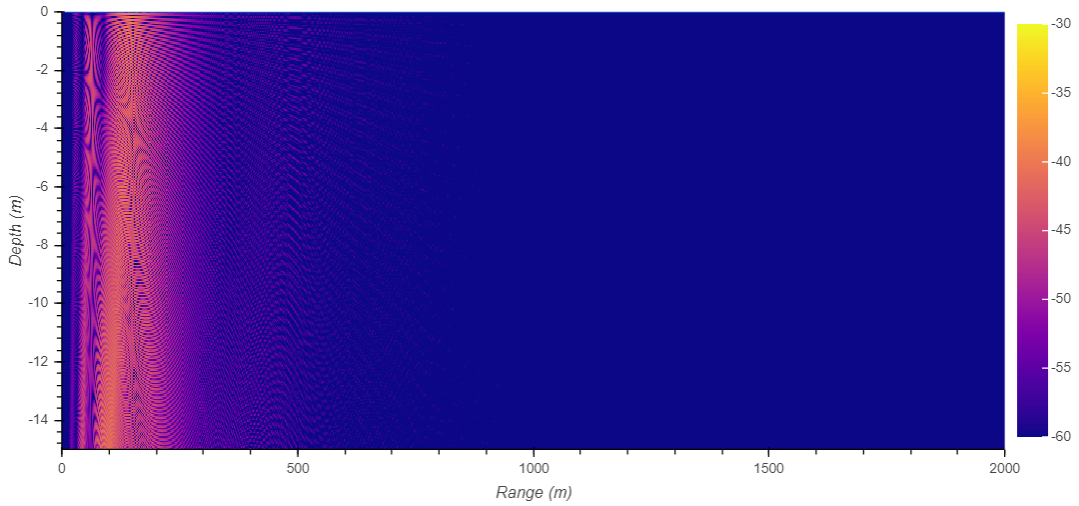


13 More complex environments

[34]:
```
env['rx_range'] = np.linspace(0, 2000, 1001)
env['rx_depth'] = np.linspace(0, 15, 301)
```

[35]:
```
beampattern = np.array([
    [-180,  10], [-170, -10], [-160,   0], [-150, -20], [-140, -10], [-130, -30],
    [-120, -20], [-110, -40], [-100, -30], [-90 , -50], [-80 , -30], [-70 , -40],
    [-60 , -20], [-50 , -30], [-40 , -10], [-30 , -20], [-20 ,   0], [-10 , -10],
    [  0 ,  10], [ 10 , -10], [ 20 ,   0], [ 30 , -20], [ 40 , -10], [ 50 , -30],
    [ 60 , -20], [ 70 , -40], [ 80 , -30], [ 90 , -50], [100 , -30], [110 , -40],
    [120 , -20], [130 , -30], [140 , -10], [150 , -20], [160 ,   0], [170 , -10],
    [180 ,  10]
])
env['tx_directionality'] = beampattern
```

[36]:
```
tloss = pm.compute_transmission_loss(env)
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```



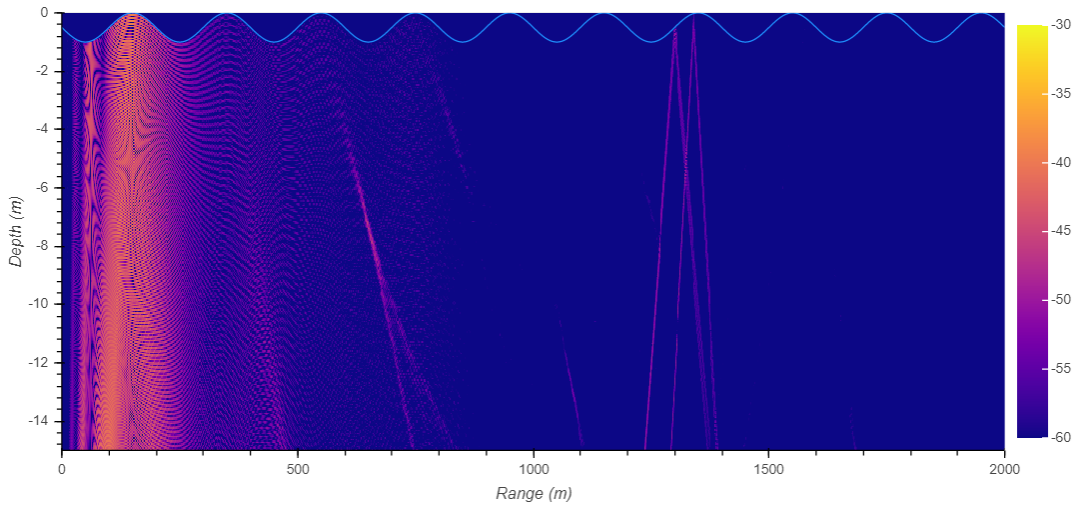
Now you can see the directionality and the sidelobe structure of the transmitter.

13.0.1 Undulating water surface

Finally, let's try adding a long wavelength swell on the water surface:

[37]:
```
surface = np.array([[r, 0.5+0.5*np.sin(2*np.pi*0.005*r)] for r in np.
 ↪linspace(0,2000,2001)])
env['surface'] = surface
```

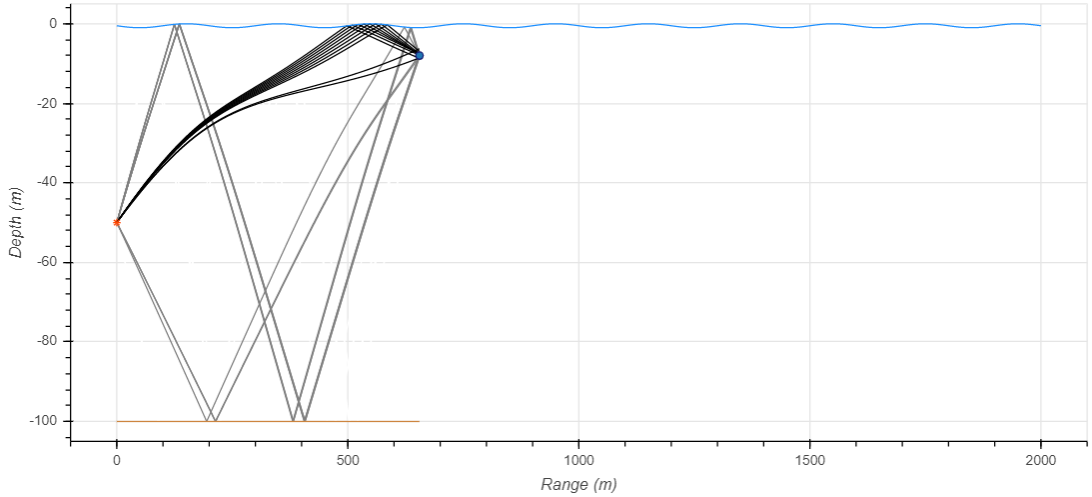
[38]:
```
tloss = pm.compute_transmission_loss(env)
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```



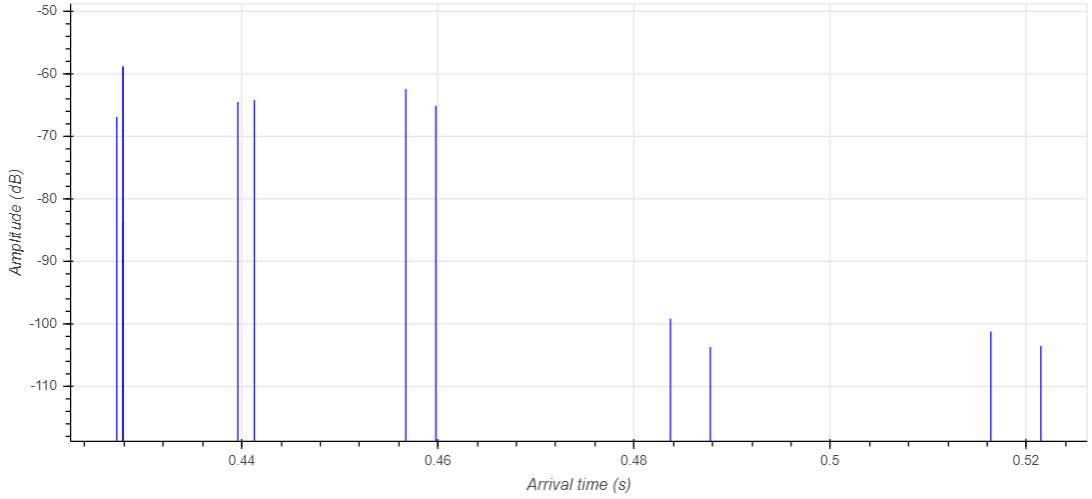
Now, if I placed a receiver at 655 m, and 8 m depth, roughly where we see some focusing, what would the eigenrays and arrival structure look like?

[39]:
```
env['rx_range'] = 655
env['rx_depth'] = 8
```

[40]:
```
rays = pm.compute_eigenrays(env)
pm.plot_rays(rays, env=env, width=900)
```



[41]:
```
arrivals = pm.compute_arrivals(env)
pm.plot_arrivals(arrivals, dB=True, width=900)
```



We plotted the amplitudes in dB, as the later arrivals are much weaker than the first one, and better visualized in a logarithmic scale.

14 Generate Chirp Signal and check response of channel:

[46]:
```
from scipy import signal
import matplotlib.pyplot as mplt
```

[55]:
```
ir = pm.arrivals_to_impulse_response(arrivals, fs=96000, abs_time=True)
```

We'll create a chirp signal 10-20 kHz in 1/4 second and then send that signal through the channel:

[56]:
```
t = np.linspace(0,1//4,96000//4+1)
chirp = signal.chirp(t,10000,1/4,20000)
ch_resp = signal.fftconvolve(chirp, ir)
```

Spectrogram for 500m range channel:

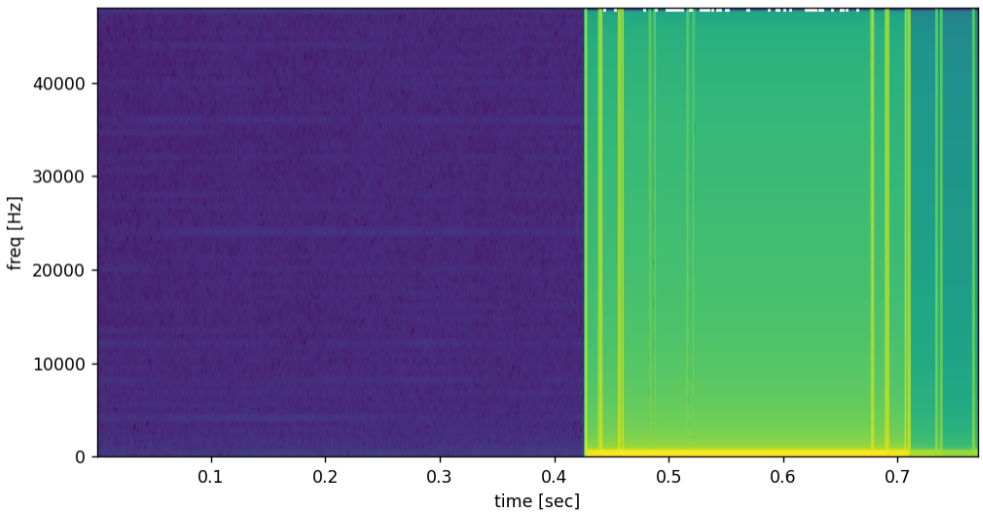
[57]:
```
%matplotlib notebook
```

[58]:
```
env['rx_range'] = 655
arrivals = pm.compute_arrivals(env)
ir = pm.arrivals_to_impulse_response(arrivals, fs=96000, abs_time=True)
ch_resp = signal.fftconvolve(chirp, ir)
mplt.figure()
Pxx, freqs, bins, im = mplt.specgram(ch_resp, Fs=96000,sides='onesided')
mplt.ylabel('freq [Hz]')
mplt.xlabel('time [sec]')
```

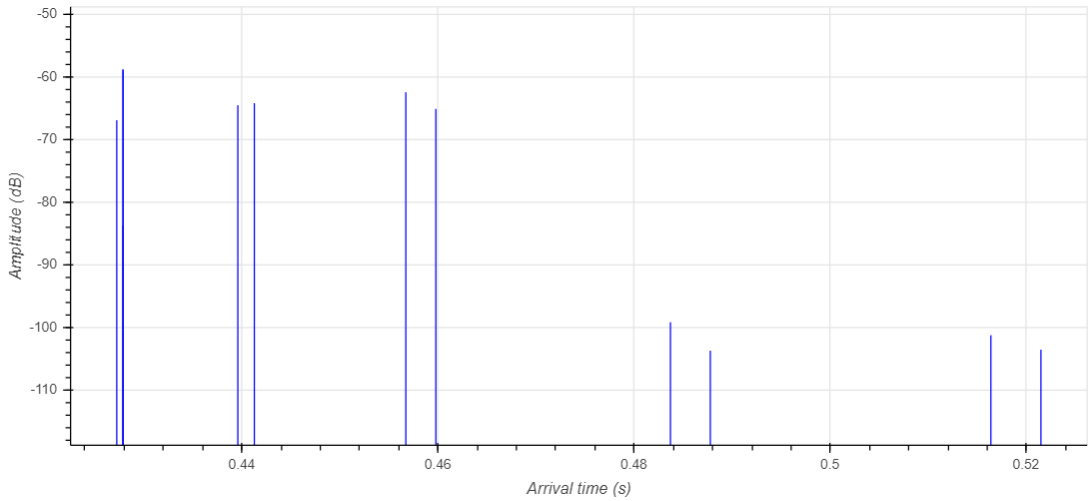
```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[58]:
```
Text(0.5, 0, 'time [sec]')
```



[60]:
```
# first 10 arrivals : compare the figures
pm.plot_arrivals(arrivals, dB=True, width=900)
```



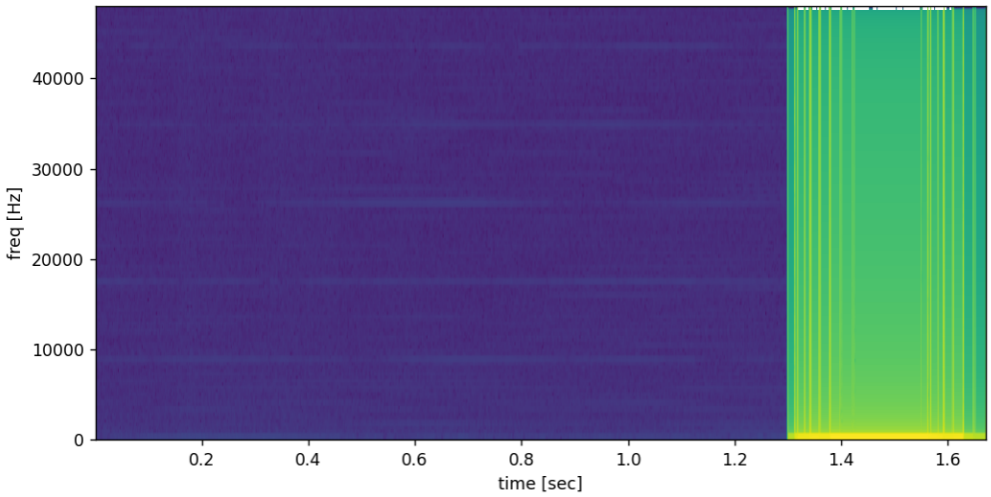
Spectrogram for 2000m channel:

[61]:
```
env['rx_range'] = 2000
arrivals = pm.compute_arrivals(env)
ir = pm.arrivals_to_impulse_response(arrivals, fs=96000, abs_time=True)
mplt.figure()
ch_resp = signal.fftconvolve(chirp, ir)
Pxx, freqs, bins, im = mplt.specgram(ch_resp, Fs=96000, sides='onesided')
mplt.ylabel('freq [Hz]')
mplt.xlabel('time [sec]')
```

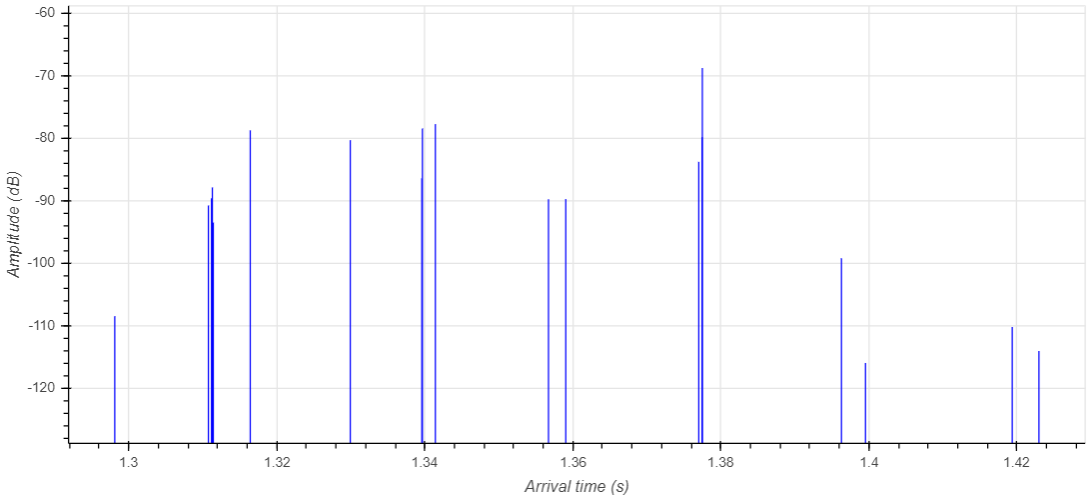
```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[61]:
```
Text(0.5, 0, 'time [sec]')
```



[62]:
```
# first 10 arrivals : compare the figures
pm.plot_arrivals(arrivals, dB=True, width=900)
```



[63]:
```
rays = pm.compute_rays(env , debug=True)
```

```
[DEBUG] Model: bellhop
[DEBUG] Bellhop working files: C:\Users\jay_p\AppData\Local\Temp\tmpj5oio0k_.*
```

15 Appendix

[ ]:
```
This is bellhop *.env file example:
```

[ ]:
```
'arlpy'                 / (Graph Title - just for Fortran/MATLAB)
25000.000000            / FREQUENCY
1                       / [UNUSED]
```

```
'SVWT*'                  / [(SSP INTERP METHOD)(TOP LAYER)(BOTTOM ATTENUATION␣
 ↪UNITS)(VOLUME ATTENUATION)]
1 0.0 100.000000         / [UNUSED] [UNUSED] [MAX DEPTH]
  0.000000 1540.400000   / [DEPTH] [SOUND SPEED]
  10.000000 1540.500000  / [DEPTH] [SOUND SPEED]
  20.000000 1540.700000  / [DEPTH] [SOUND SPEED]
  30.000000 1534.400000  / [DEPTH] [SOUND SPEED]
  50.000000 1523.300000  / [DEPTH] [SOUND SPEED]
  75.000000 1519.600000  / [DEPTH] [SOUND SPEED]
  100.000000 1518.500000 / [DEPTH] [SOUND SPEED]
'A' 0.000000             / [(BOTTOM LAYER)(EXTERNAL BATHYMETRY] [SIGMA BOTTOM␣
 ↪ROUGHNESS]
100.000000 1600.000000 0.0 1.200000 1.000000 / [MAX DEPTH] [SOUND SPEED BOTTOM]␣
 ↪[UNUSED] [DENSITY BOTTOM] [ATTENUATION BOTTOM]
1                        / [NUMBER SOURCES]
50.000000                / [SOURCE DEPTHS 1:N (m)]
1                        / [NUMBER RECEIVER DEPTHS]
8.000000                 / [RECEIVER DEPTHS 1:N (m)]
1                        / [NUMBER RECEIVER RANGES]
2.000000                 / [RECEIVER RANGES 1:N (km)]
'R *'                    / [(RUN TYPE)(BEAM TYPE)(EXTERNAL SOURCE BEAM PATTERN)]␣
 ↪; typically "R/C/I/S" - Refer next line
0                        / [NUMBER OF BEAMS] ; default is 0 that means bellhop␣
 ↪calculates it by itself using
-80.000000 80.000000     / [MIN ANGLE] [MAX ANGLE]
0.0 101.000000 2.020000  / [STEP SIZE] [DEPTH BOX] [RANGE BOX] ; default step␣
 ↪size is 0.
```

[ ]:
```
# Another Sample Input (Environmental) File:

'Munk profile'          ! TITLE
50.0                    ! FREQ (Hz)
1                       ! NMEDIA
'SVN'                   ! SSPOPT (Analytic or C-linear interpolation)
51  0.0  5000.0        ! DEPTH of bottom (m)
    0.0  1548.52  /
  200.0  1530.29  /
  250.0  1526.69  /
  400.0  1517.78  /
  600.0  1509.49  /
  800.0  1504.30  /
 1000.0  1501.38  /
 1200.0  1500.14  /
 1400.0  1500.12  /
 1600.0  1501.02  /
 1800.0  1502.57  /
 2000.0  1504.62  /
```

```
 2200.0  1507.02  /
 2400.0  1509.69  /
 2600.0  1512.55  /
 2800.0  1515.56  /
 3000.0  1518.67  /
 3200.0  1521.85  /
 3400.0  1525.10  /
 3600.0  1528.38  /
 3800.0  1531.70  /
 4000.0  1535.04  /
 4200.0  1538.39  /
 4400.0  1541.76  /
 4600.0  1545.14  /
 4800.0  1548.52  /
 5000.0  1551.91  /
'V'  0.0
1                       ! NSD
1000.0 /                ! SD(1:NSD) (m)
2                       ! NRD
0.0 5000.0 /            ! RD(1:NRD) (m)
501                     ! NRR
0.0  100.0 /            ! RR(1:NR ) (km)
'R'                     ! Run-type: 'R/C/I/S'
51                      ! NBEAMS
-11.0 11.0 /            ! ALPHA(1:NBEAMS) (degrees)
200.0  5500.0  101.0    ! STEP (m)  ZBOX (m)  RBOX (km)
```

15.0.1 Description of Inputs

The first 6 blocks in the ENVFIL are common to all the programs in the Acoustics Toolbox. The following blocks should be appended for BELLHOP:
(7) - SOURCE/RECEIVER DEPTHS AND RANGES Syntax:

```
      NSD
      SD(1:NSD)
      NRD
      RD(1:NRD)
      NR
      R(1:NR )

   Description:

      NSD:  The number of source depths
      SD(): The source depths (m)
      NRD:  The number of receiver depths
      RD(): The receiver depths (m)
      NR:   The number of receiver ranges
      R():  The receiver ranges (km)
```

This data is read in using list-directed I/O you can type it just about any way you want, e.g. on one line or split onto several lines. Also if the depths or ranges are equally spaced then you can type just the first and last depths followed by a '/' and the intermediate depths will be generated automatically. You can specify a receiver at zero range; however, the BELLHOP field is singular there— the pressure is returned as zero. Some of the subroutines that calculate the beam influence allow an arbitrary vector of receiver ranges; others require it to be equally spaced in range. In particular, only the following allow an arbitrary range vector:

```
'G' GeoHatCart
'B' GeoGaussianCart
CerveyRayCen
```

(8) • RUN TYPE Syntax:

OPTION

Description:

OPTION(1:1): 'R' generates a ray file 'E' generates an eigenray file 'A' generates an amplitude-delay file (ascii) 'a' generate an amplitude-delay file (binary) 'C' Coherent TL calculation 'I Incoherent TL calculation 'S' Semicoherent TL calculation (Lloyd mirror source pattern)

OPTION(2:2): 'G' Geometric hat beams in Cartesian coordinates (default) 'g' Geometric hat beams in ray-centered coordinates 'B' Geometric Gaussian beams

OPTION(3:3): '*' read in a source beam pattern file 'O' don't (default)

OPTION(4:4): 'R' point source (cylindrical coordinates) (default) 'X' line source (cartesian coordinates)

OPTION(5:5): 'R' rectilinear grid (default) 'I' irregular grid

The ray file and eigenray files have the same simple ascii format and can be plotted using the Matlab script plotray.m. The eigenray option seems to generate a lot of questions. The way this works is that BELLHOP simply writes the trajectories for all the beams that contribute at a given receiver location. To get a useful picture you normally want to use a very fine fan, only one receiver location, and the geometric beam option. See the examples in at/tests. The amplitude-delay file can be used with the Matlab script stackarr.m to 'stack the arrivals', i.e. to convolve them with the source spectrum and plot the channel response. stackarr.m can also be used to simple plot the impulse response.

For TL calculations, the output is in the shdfil format used by all the codes in the Acoustics Toolbox and can be plotted using the Matlab script, plotshd.m. (Use toasc.f to convert the binary shade files to ascii format for use by plotshd.m or whatever plot package you're using.) The pressure field is normally calculated on a rectilinear grid formed by the receiver ranges and depths. If an irregular grid is selected, then the receiver ranges and depths are interpreted as a coordinate pair for the receivers. This option is useful for reverberation calculations where the receivers need to follow the bottom terrain. This option has not been used much. The plot routines (plotarr) have not been modified to accomodate it. There may be some other limitations. There are actually several different types of Gaussian beam options (OPTION(2:2)) implemented in the code. Only the two described above are fully maintained.

The source beam pattern file has the format

```
NSBPPts
angle1  power1
angle2  power2
 ...
```

with angle following the BELLHOP convention, i.e. declination angle in degrees (so that 90 degrees points to the bottom). The power is in dB. To match a standard point source calculation one would used anisotropic source with 0 dB for all angles. (See at/tests/BeamPattern for an example.) (9) - BEAM FAN Syntax:

```
NBEAMS ISINGLE
ALPHA(1:NBEAMS)

Description:

          NBEAMS: Number of beams
      (use 0 to have the program calculate a value automatically, but conservatively).

ISINGLE: If the option to compute a single beam in the fan is selected (top option)
    then this selects the index of the beam that is traced.

ALPHA(): Beam angles (negative angles toward surface)
```

For a ray trace you can type in a sequence of angles or you can type the first and last angles followed by a '/'.
For a TL calculation, the rays must be equally spaced otherwise the results will be incorrect.

(10) • NUMERICAL INTEGRATOR INFO Syntax:

STEP ZBOX RBOX

Description:

```
STEP:  The step size used for tracing the rays (m). (Use 0 to let BELLHOP choose the step s
ZBOX:  The maximum depth to trace a ray        (m).
RBOX:  The maximum range to trace a ray       (km).
```

The required step size depends on many factors.
This includes frequency, size of features in the SSP (such as surface ducts), range of receivers, and whether a coherent or incoherent TL calculation is performed. If you use STEP=0.0 BELLHOP will use a default step-size and tell you what it picked. You should then halve the step size until the results are convergent to your required accuracy. To obtain a smooth ray trace you should use the spline SSP interpolation and a step-size less than the smallest distance between SSP data points. Rays are traced until they exit the box (ZBOX, RBOX). By setting ZBOX less than the water depth you can eliminate bottom reflections. Make ZBOX, RBOX a bit (say 1%) roomy too make sure rays are not killed the moment they hit the bottom or are just reaching your furthest receiver.

[ ]:
```
This is bellhop file run log example:
```

[ ]:
```
 BELLHOP/BELLHOP3D
```

```
BELLHOP- arlpy
frequency =  0.2500E+05 Hz

Dummy parameter NMedia =            1

    Spline approximation to SSP
    Attenuation units: dB/wavelength
    THORP attenuation added
    Altimetry file selected
    VACUUM

Depth =     100.00 m

Sound speed profile:
  z (m)     alphaR (m/s)   betaR  rho (g/cm^3)  alphaI     betaI

     0.00      1540.40      0.00     1.00       0.0000    0.0000
    10.00      1540.50      0.00     1.00       0.0000    0.0000
    20.00      1540.70      0.00     1.00       0.0000    0.0000
    30.00      1534.40      0.00     1.00       0.0000    0.0000
    50.00      1523.30      0.00     1.00       0.0000    0.0000
    75.00      1519.60      0.00     1.00       0.0000    0.0000
   100.00      1518.50      0.00     1.00       0.0000    0.0000

                                ( RMS roughness =   0.00     )
    ACOUSTO-ELASTIC half-space
   100.00      1600.00      0.00     1.20       1.0000    0.0000

--------------------------------------------------------------------------------


Number of source   depths =            1
Source depths (m)
  50.0000

Number of receiver depths =            1
Receiver depths (m)
  8.00000

Number of receiver ranges =            1
Receiver ranges (km)
  2.00000

Ray trace run
Geometric hat beams in Cartesian coordinates
Point source (cylindrical coordinates)
Rectilinear receiver grid: Receivers at rr( : ) x rd( : )
```

```
Number of beams in elevation   =           50
Beam take-off angles (degrees)
 -80.0000      -76.7347      -73.4694      -70.2041      -66.9388
 -63.6735      -60.4082      -57.1429      -53.8776      -50.6122
 -47.3469      -44.0816      -40.8163      -37.5510      -34.2857
 -31.0204      -27.7551      -24.4898      -21.2245      -17.9592
 -14.6939
 ...    80.000000000000000

------------------------------------------------------------------------


Step length,      deltas =    0.0000000000000000      m

Maximum ray depth, Box%z =    101.00000000000000      m
Maximum ray range, Box%r =    2020.0000000000000      m

*********************************
Using top-altimetry file
Piecewise linear interpolation
Number of altimetry points        2001

 Range (km)  Depth (m)
  0.00      0.500
 0.100E-02  0.516
 0.200E-02  0.531
 0.300E-02  0.547
 0.400E-02  0.563
 0.500E-02  0.578
 0.600E-02  0.594
 0.700E-02  0.609
 0.800E-02  0.624
 0.900E-02  0.639
 0.100E-01  0.655
 0.110E-01  0.669
 0.120E-01  0.684
 0.130E-01  0.699
 0.140E-01  0.713
 0.150E-01  0.727
 0.160E-01  0.741
 0.170E-01  0.755
 0.180E-01  0.768

------------------------------
Using source beam pattern file
Number of source beam pattern points          37
```

```
 Angle (degrees)  Power (dB)
 -180.       10.0
 -170.      -10.0
 -160.       0.00
 -150.      -20.0
 -140.      -10.0
 -130.      -30.0
 -120.      -20.0
 -110.      -40.0
 -100.      -30.0
 -90.0      -50.0
 -80.0      -30.0
 -70.0      -40.0
 -60.0      -20.0
 -50.0      -30.0
 -40.0      -10.0
 -30.0      -20.0
 -20.0       0.00
 -10.0      -10.0
  0.00       10.0
  10.0      -10.0
  20.0       0.00
  30.0      -20.0
  40.0      -10.0
  50.0      -30.0
  60.0      -20.0
  70.0      -40.0
  80.0      -30.0
  90.0      -50.0
  100.      -30.0
  110.      -40.0
  120.      -20.0
  130.      -30.0
  140.      -10.0
  150.      -20.0
  160.       0.00
  170.      -10.0
  180.       10.0

 Step length,      deltas =    10.000000000000000      m (automatically selected)

Tracing beam       1    -80.00
Tracing beam       2    -76.73
Tracing beam       3    -73.47
Tracing beam       4    -70.20
Tracing beam       5    -66.94
```

```
Tracing beam       6    -63.67
Tracing beam       7    -60.41
Tracing beam       8    -57.14
Tracing beam       9    -53.88
Tracing beam      10    -50.61
Tracing beam      11    -47.35
Tracing beam      12    -44.08
Tracing beam      13    -40.82
Tracing beam      14    -37.55
Tracing beam      15    -34.29
Tracing beam      16    -31.02
Tracing beam      17    -27.76
Tracing beam      18    -24.49
Tracing beam      19    -21.22
Tracing beam      20    -17.96
Tracing beam      21    -14.69
Tracing beam      22    -11.43
Tracing beam      23     -8.16
Tracing beam      24     -4.90
Tracing beam      25     -1.63
Tracing beam      26      1.63
Tracing beam      27      4.90
Tracing beam      28      8.16
Tracing beam      29     11.43
Tracing beam      30     14.69
Tracing beam      31     17.96
Tracing beam      32     21.22
Tracing beam      33     24.49
Tracing beam      34     27.76
Tracing beam      35     31.02
Tracing beam      36     34.29
Tracing beam      37     37.55
Tracing beam      38     40.82
Tracing beam      39     44.08
Tracing beam      40     47.35
Tracing beam      41     50.61
Tracing beam      42     53.88
Tracing beam      43     57.14
Tracing beam      44     60.41
Tracing beam      45     63.67
Tracing beam      46     66.94
Tracing beam      47     70.20
Tracing beam      48     73.47
Tracing beam      49     76.73
Tracing beam      50     80.00

 CPU Time =       0.250    s
```

[ ]:
```
This is *.ati file example.
```

[ ]:
```
'L'
2001
0.000000 0.500000
0.001000 0.515705
0.002000 0.531395
0.003000 0.547054
0.004000 0.562667
0.005000 0.578217
0.006000 0.593691
0.007000 0.609072
0.008000 0.624345
0.009000 0.639496
0.010000 0.654508
0.011000 0.669369
0.012000 0.684062
0.013000 0.698574
0.014000 0.712890
0.015000 0.726995
0.016000 0.740877
0.017000 0.754521
1.998000 0.468605
1.999000 0.484295
2.000000 0.500000
```

16 Reference

1. **M. Chitre 2021, "ARLPY python toolbox"** , https://github.com/org-arl/arlpy
2. **Ocean Acoustics Library.**, https://oalib-acoustics.org/, Retrieved July 07, 2021.
3. **Jay Patel and Mae Seto,** Underwater channel characterization for shallow water multi-domain communications, Proceedings of Meetings on Acoustics 40, 070014 (2020), https://doi.org/10.1121/2.0001316, Retrieved July 07, 2021.