Underwater acoustic propagation modeling with arlpy and Bellhop

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

Prerequisites ¶

- Install [arlpy (https://pypi.org/project/arlpy/)](https://pypi.org/project/arlpy/) (v1.5 or higher)
- Install the [acoustics toolbox (https://oalib-acoustics.org/)](https://oalib-acoustics.org/) (6 July 2018 version or later)

You can install arlpy (in overall system) by typing this in your command prompt:

In [ ]:
```
python3 -m pip install arlpy
```

Getting started

Start off with checking that everything is working correctly:

In [14]:
```
# 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
```

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

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

The `bellhop` model should be listed in the list of models above, if everything is good. If it isn't listed, it means that `bellhop.exe` is not available on the PATH, or it cannot be correctly executed. Ensure that `bellhop.exe` from the acoustics toolbox installation is on your PATH (updated

`.profile` or equivalent, if necessary, to add it in).

From here on we assume that the `bellhop` model is available, and proceed...

We next create an underwater 2D environment (with default settings) to model:

In [3]:
```
env = pm.create_env2d()
pm.print_env(env)
```

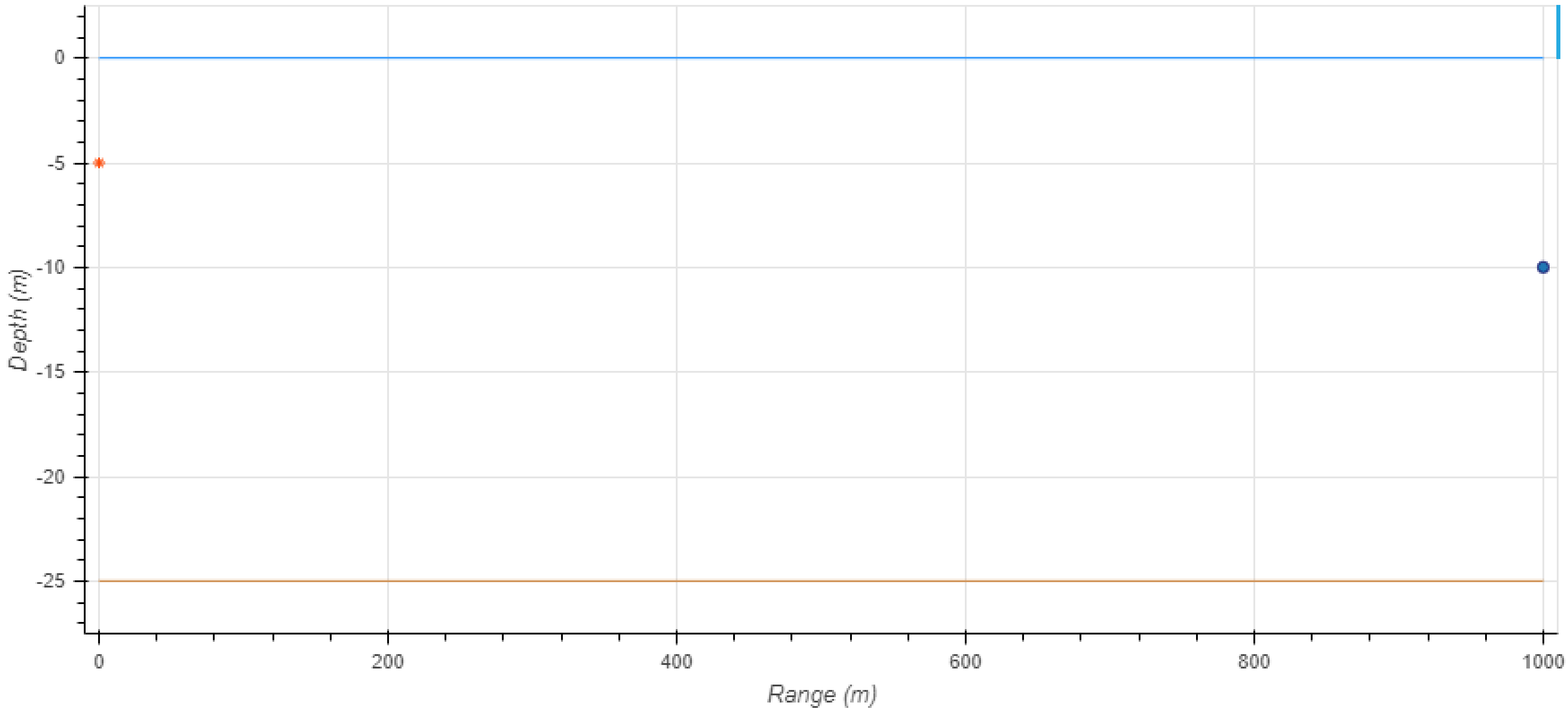
```
              name : arlpy
 bottom_absorption : 0.1
    bottom_density : 1600
  bottom_roughness : 0
 bottom_soundspeed : 1600
             depth : 25
      depth_interp : linear
         frequency : 25000
         max_angle : 80
         min_angle : -80
            nbeams : 0
          rx_depth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

We can see the default values above. Most numbers are in SI units. The only ones that aren't are:

- `bottom_absoption` is in dB/wavelength
- `min_angle` and `max_angle` are in degrees

The default environment is an isovelocity Pekeris waveguide with a water depth of 25 m, a omnidirectional transmitter at 5 m depth, and a receiver at 10 m depth 1 km away. We can visualize the environment (not to scale):

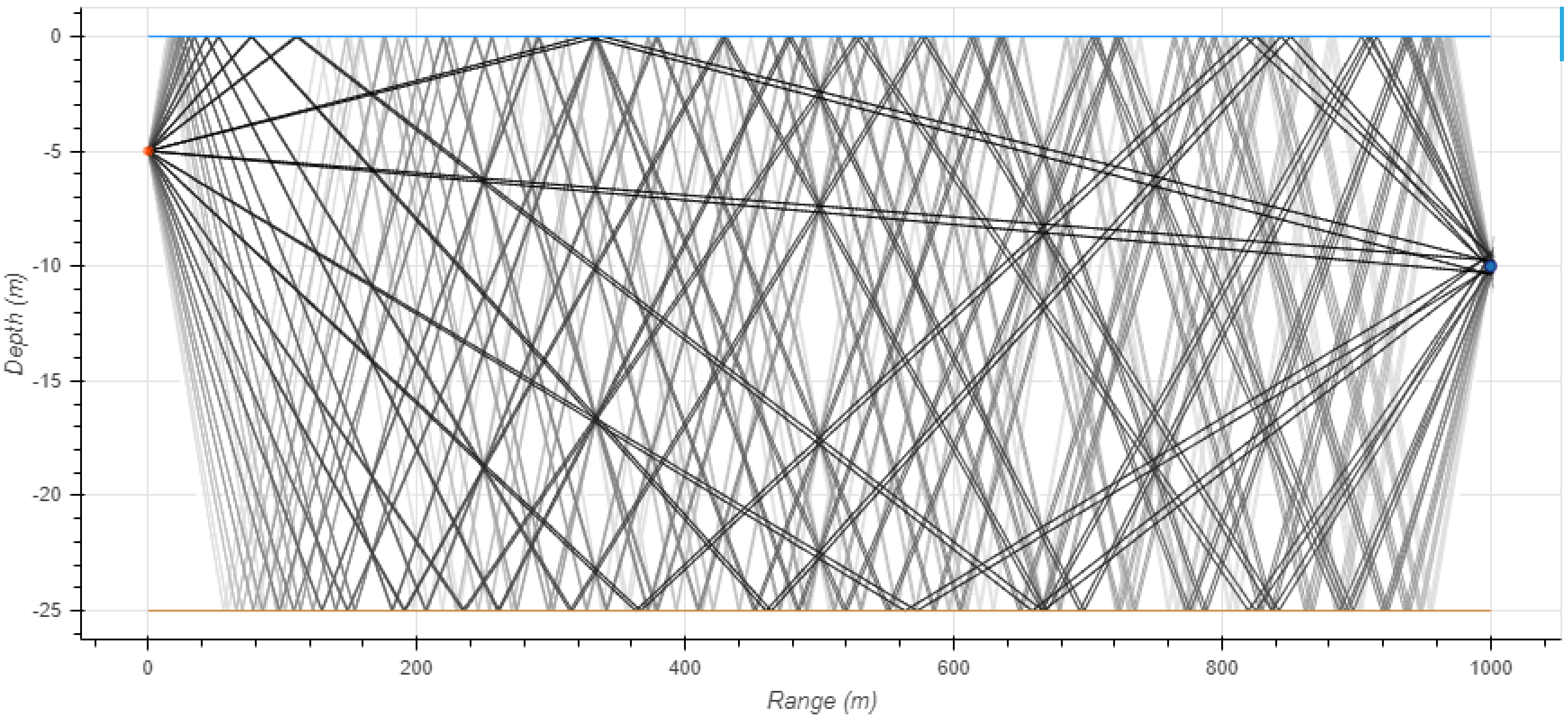
In [4]:
```
pm.plot_env(env, width=900)
```



Let's simulate it and see what the eigenrays between the transmitter and receiver look like:

In [5]:
```
rays = pm.compute_eigenrays(env , debug=True)
pm.plot_rays(rays, env=env, width=900)
```

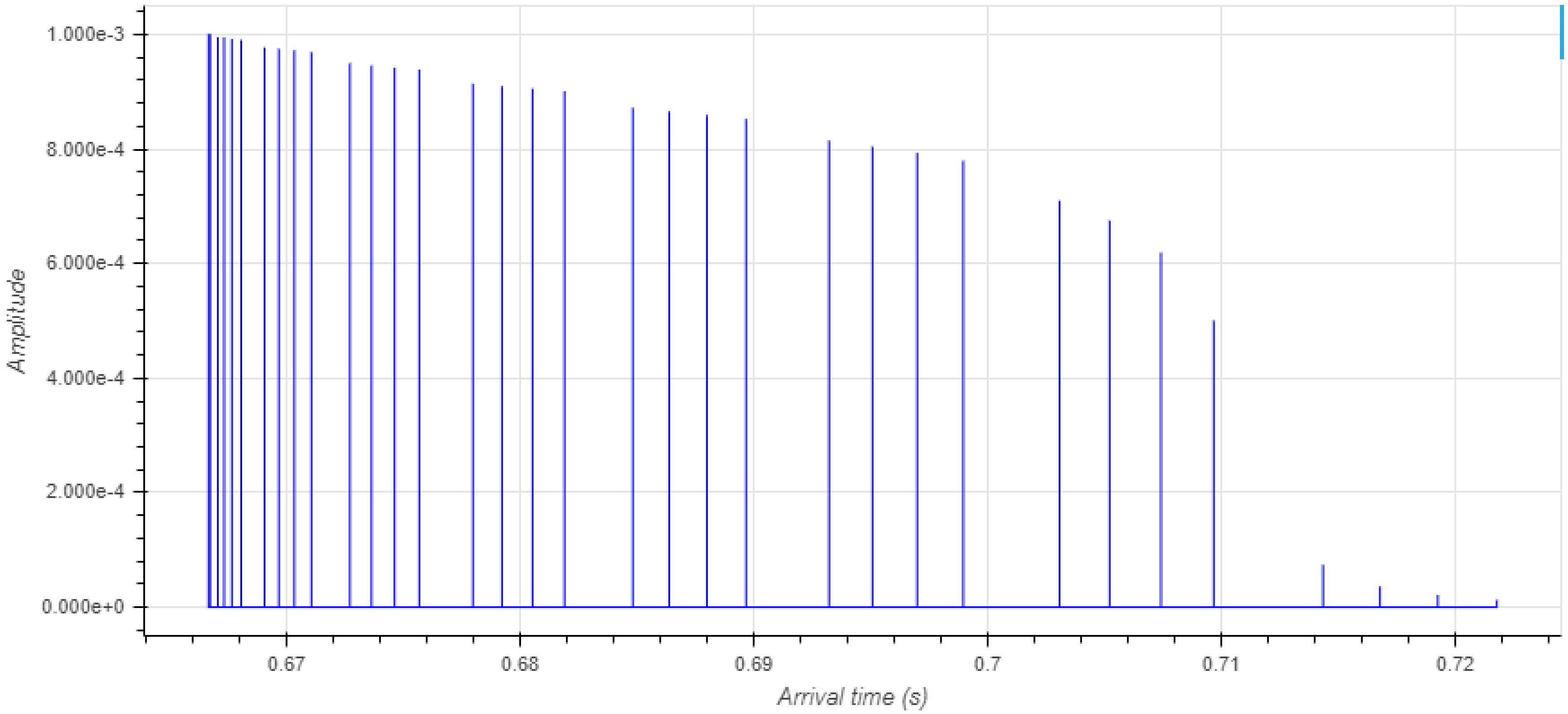
```
[DEBUG] Model: bellhop
[DEBUG] Bellhop working files: C:\Users\jay_p\AppData\Local\Temp\tmpl1fz28q0.*
```



We can also compute the arrival structure at the receiver:

In [6]:

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



The arrivals are returned in [pandas (https://pandas.pydata.org)](https://pandas.pydata.org) data frame that we can query, if we like. For example, we can look up the time of arrival, angle of arrival, and the number of surface/bottom bounces for the first 10 arrivals:

In [7]:

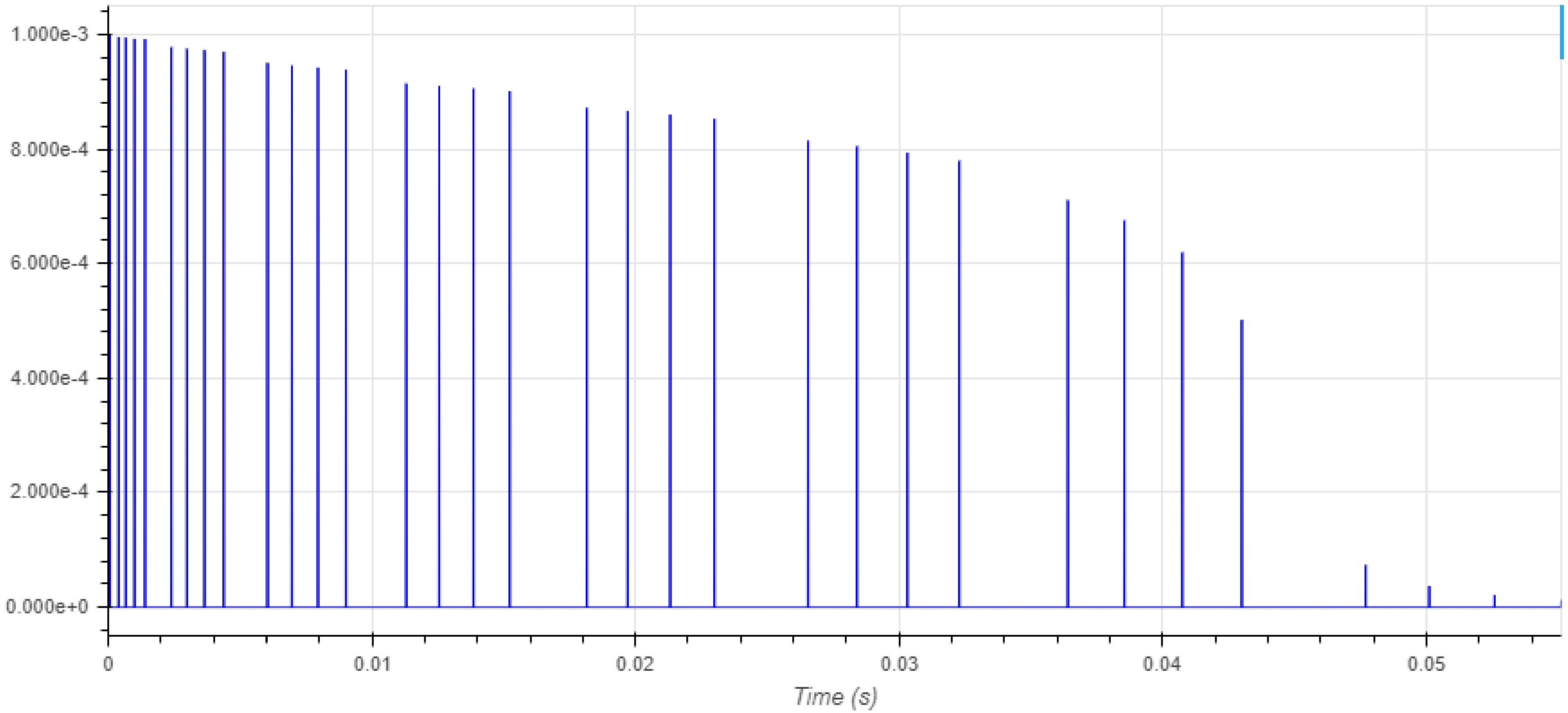
```
arrivals[arrivals.arrival_number < 10][['time_of_arrival', 'angle_of_arrival', 'surface_bounces', 'bottom_bounces']]
```

Out[7]:

	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

We can also convert to a impulse response time series, if we want to use it for further signal processing:

In [8]:
```
ir = pm.arrivals_to_impulse_response(arrivals, fs=96000)
plt.plot(np.abs(ir), fs=96000, width=900)
```



Another Test Case

In [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
]
```

In [10]:

```
depth = 100
```

In [11]:

```
# Bottom properties
bottom_absorption = 1.0
bottom_density = 1200
```

In [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)])
```

In [15]:

```python
# =============================================================================
# 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
```

In [17]:

```python
# =============================================================================
# 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
```

In [18]:

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

In [20]:

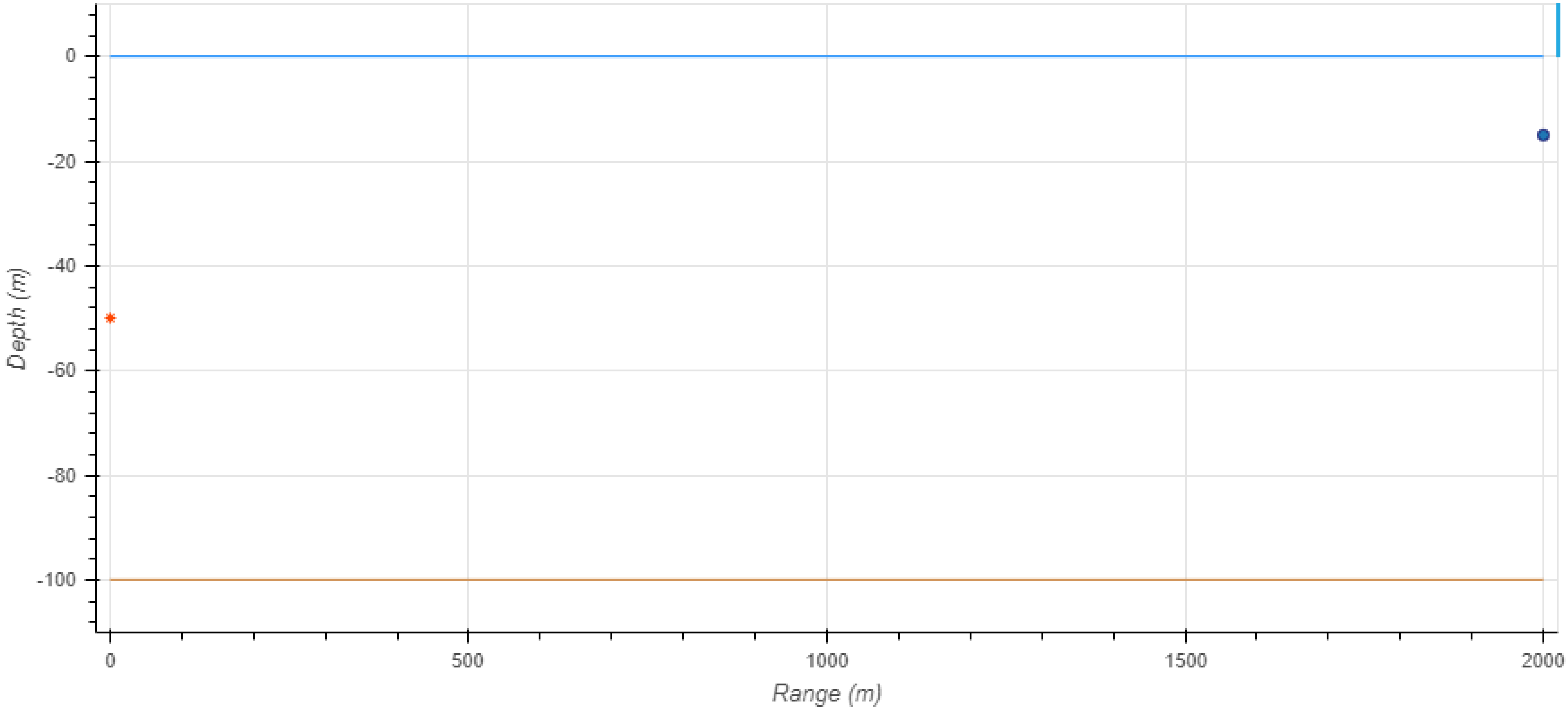
```
env = pm.create_env2d(
        bottom_absorption=bottom_absorption,
        bottom_density=bottom_density,
        depth=depth,
        soundspeed=ssp,
        surface=surface,
        rx_depth=rx_depth,
        rx_range=rx_range,
        tx_depth=tx_depth,
        tx_directionality=tx_beampattern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

In [21]:
```
pm.print_env(env)
```

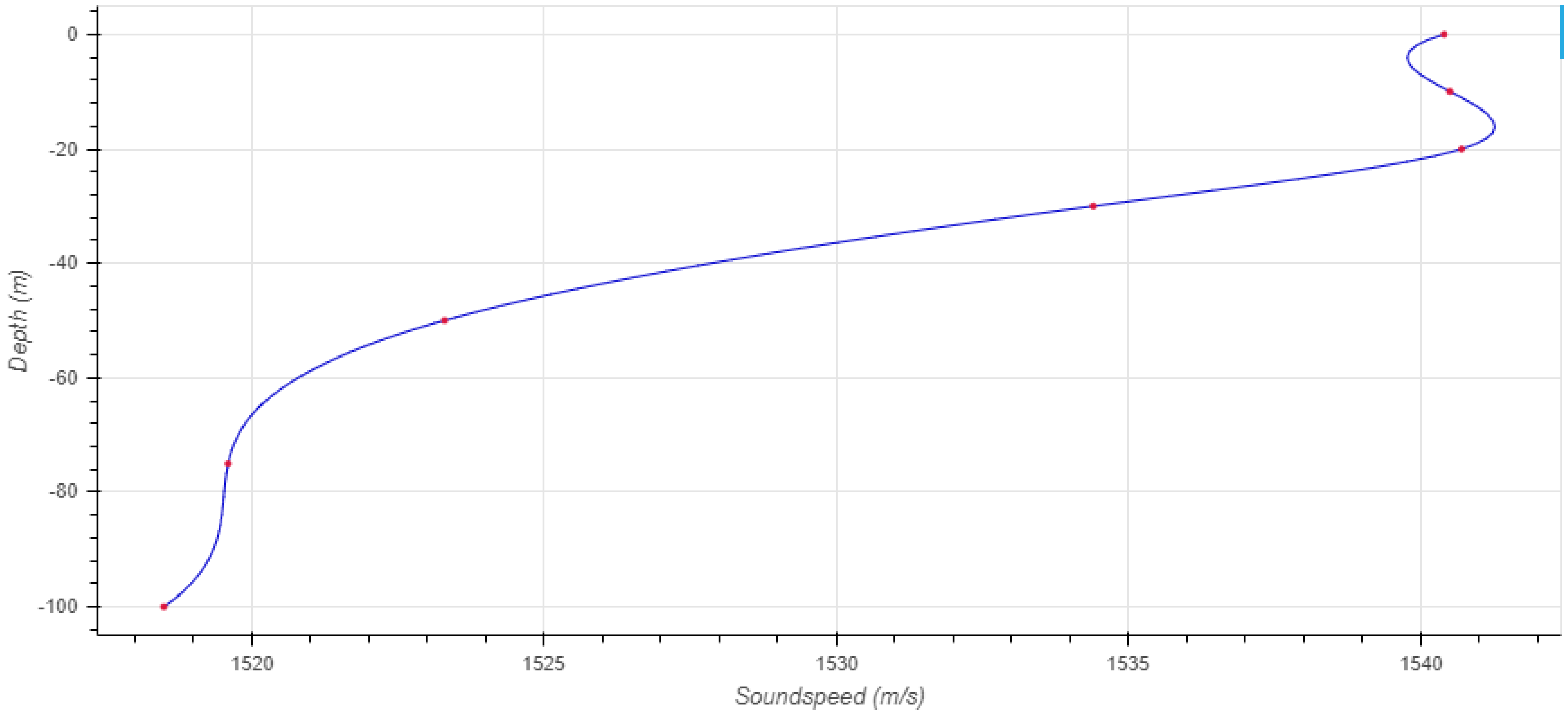
```
               name : arlpy
  bottom_absorption : 1.0
     bottom_density : 1200
   bottom_roughness : 0
  bottom_soundspeed : 1600
              depth : 100
       depth_interp : linear
          frequency : 25000
          max_angle : 80
          min_angle : -80
             nbeams : 0
           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
```

In [23]:

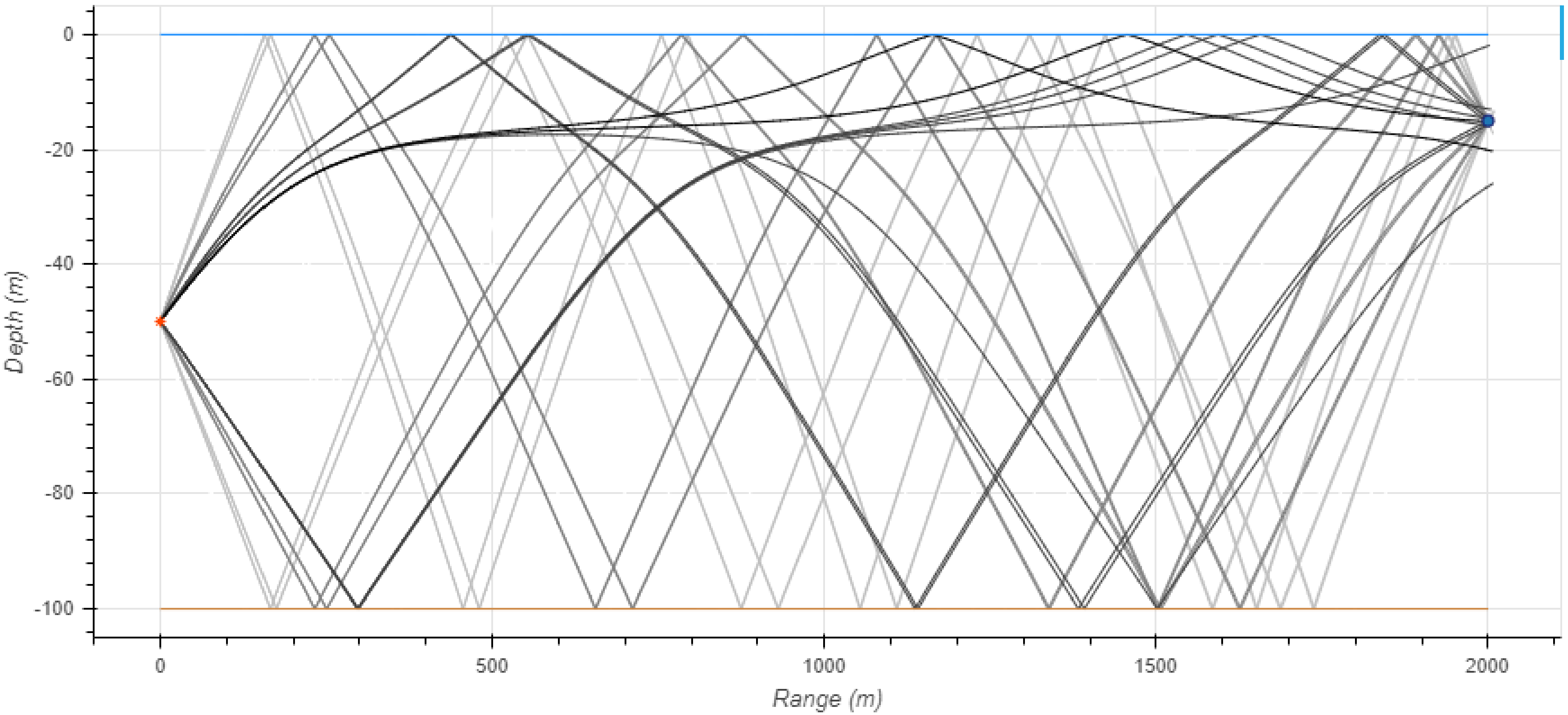
```
pm.plot_env(env, width=900)
```



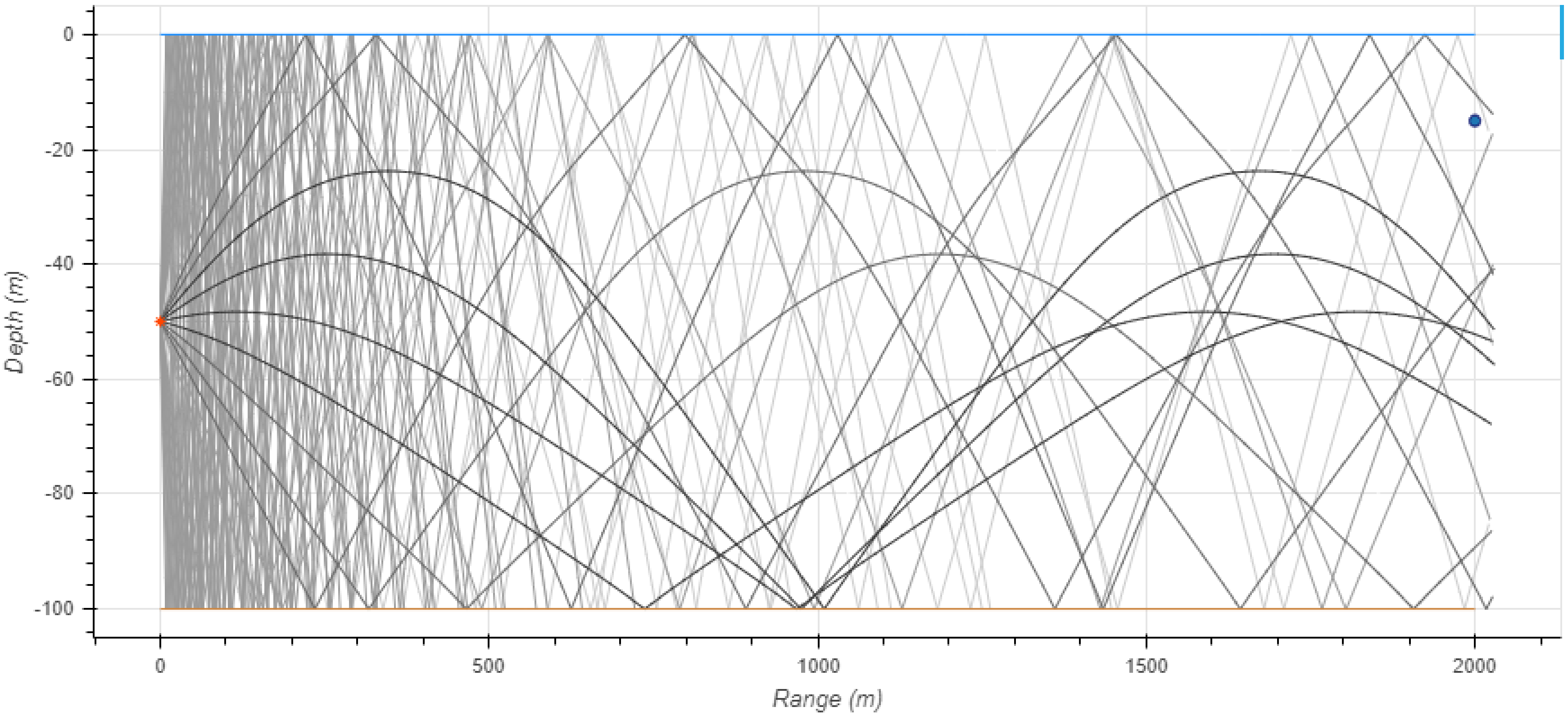
In [25]:
```
pm.plot_ssp(env, width=900)
```



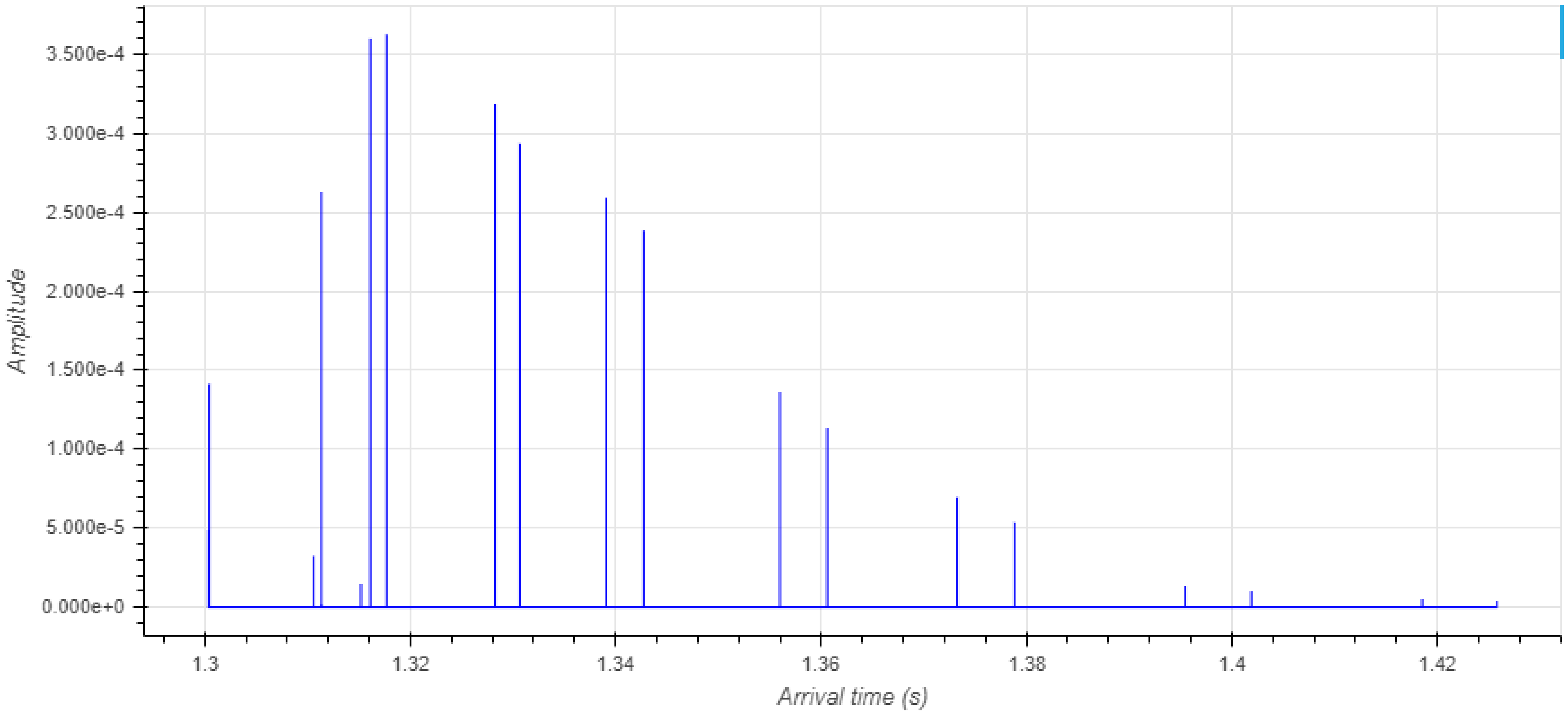
In [26]:
```
rays = pm.compute_eigenrays(env)
pm.plot_rays(rays, env=env, width=900)
```



In [28]:
```
rays = pm.compute_rays(env , debug=False)
pm.plot_rays(rays, env=env, width=900)
```



In [29]:
```
arrivals = pm.compute_arrivals(env)
pm.plot_arrivals(arrivals, width=900)
```

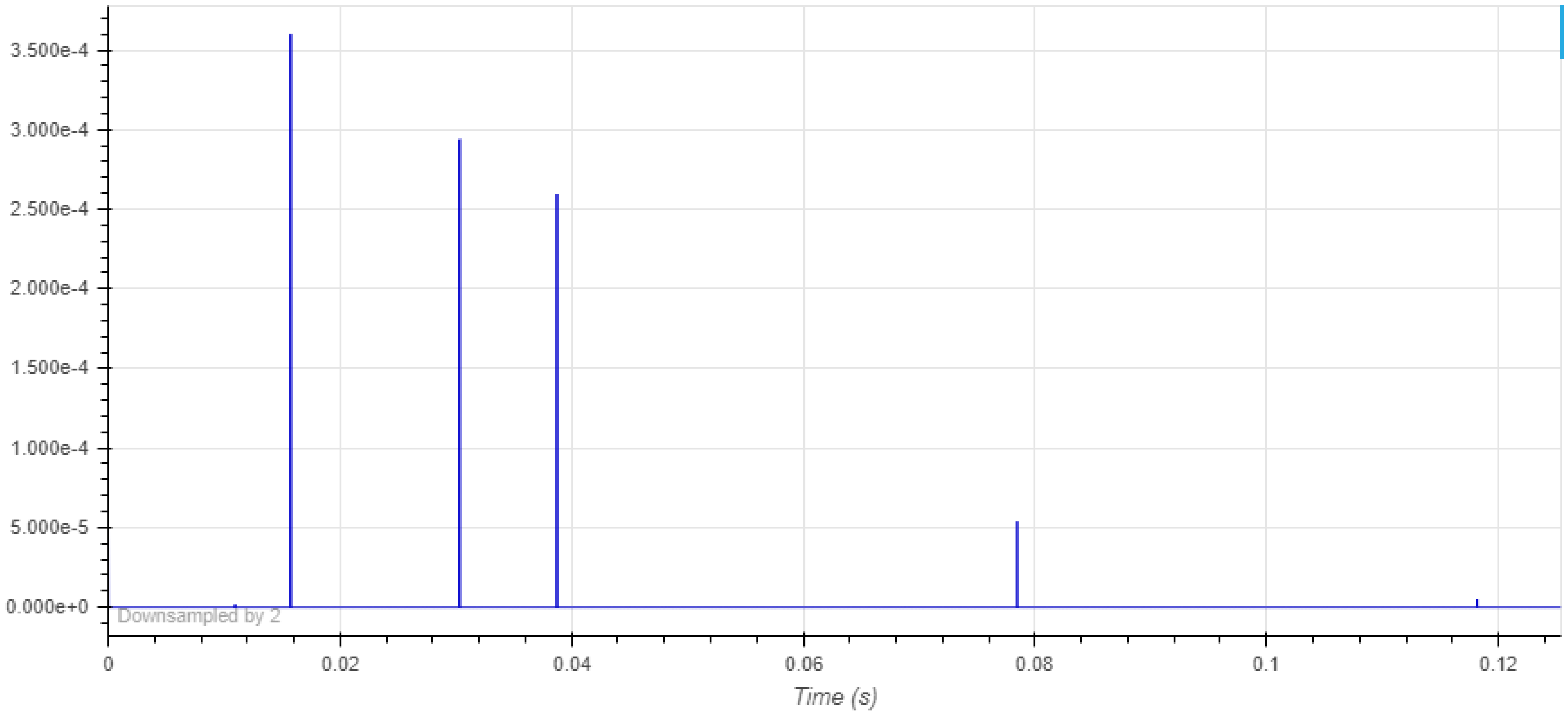


In [30]:
```
arrivals[arrivals.arrival_number < 10][['time_of_arrival', 'angle_of_arrival', 'surface_bounces', 'bottom_bounces']]
```

Out[30]:

	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

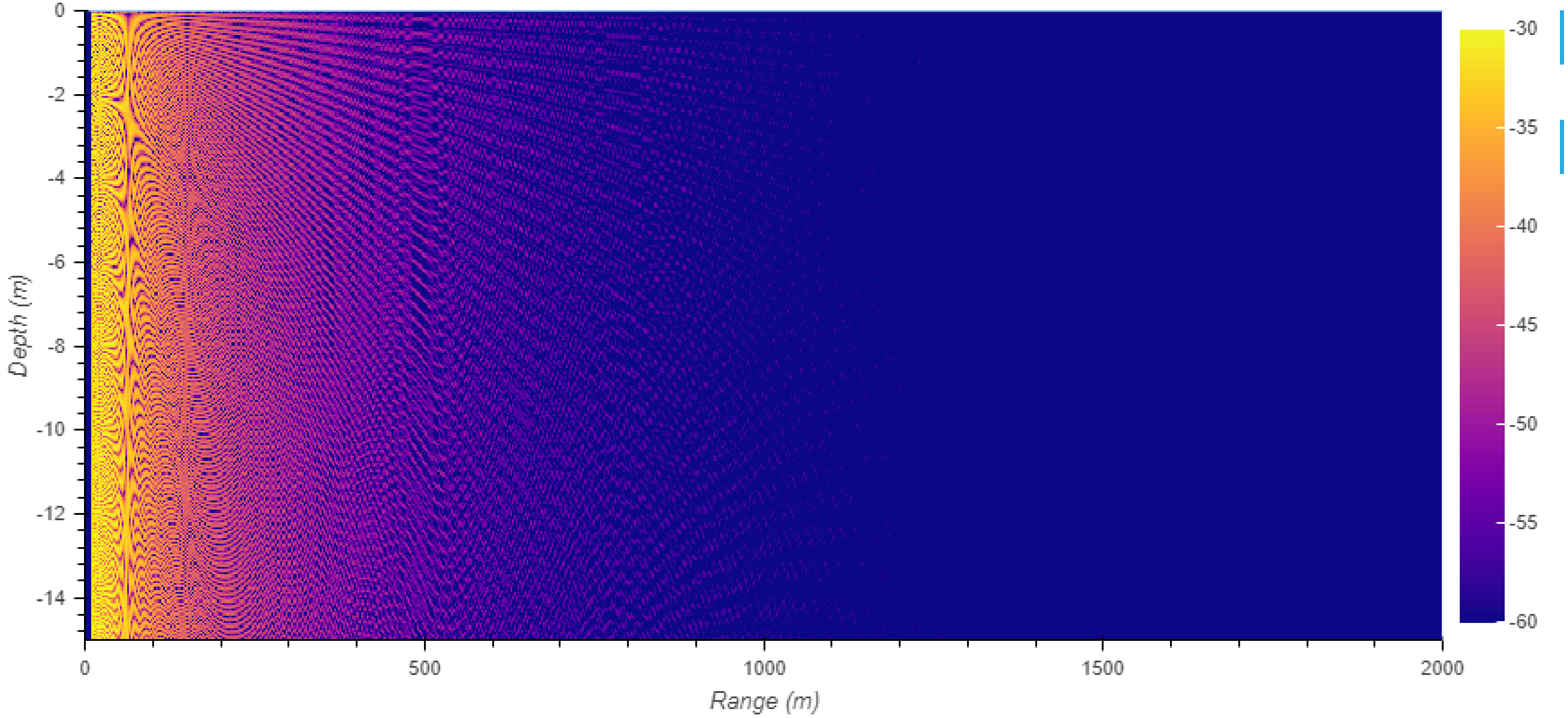
In [31]:
```
ir = pm.arrivals_to_impulse_response(arrivals, fs=96000)
plt.plot(np.abs(ir), fs=96000, width=900)
```



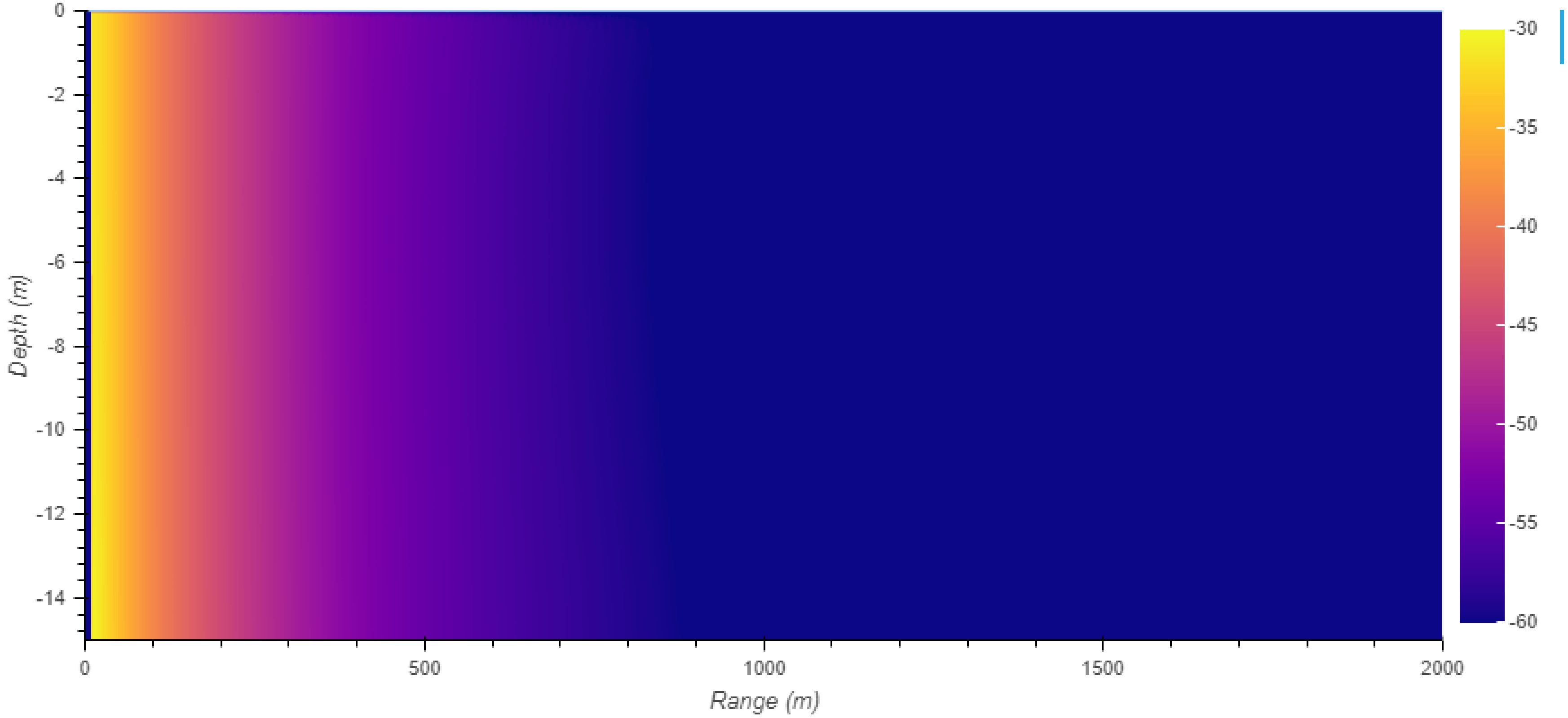
In [35]:
```
env['rx_range'] = np.linspace(0, 2000, 1001)
env['rx_depth'] = np.linspace(0, 15, 301)
```

In [36]:

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



In [40]:
```
tloss = pm.compute_transmission_loss(env, mode='incoherent')
pm.plot_transmission_loss(tloss, env=env, clim=[-60,-30], width=900)
```



More complex environments

So far, we have a simple isovelicty Pekeris waveguide. Let us next have something more interesting - an environment with some bathymetric structure and a more complicated soundspeed profile.

Bathymetry and soundspeed profile

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

In [42]:
```python
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
    [2000, 25]  # 25 m water depth at 2 km
]
```

and then our soundspeed profile:

In [44]:
```python
ssp
```

Out[44]:
```
[[0, 1540.4],
 [10, 1540.5],
 [20, 1540.7],
 [30, 1534.4],
 [50, 1523.3],
 [75, 1519.6],
 [100, 1518.5]]
```

Now we can create our environment with a muddy bottom, and a transmitter that is at 20 m depth:

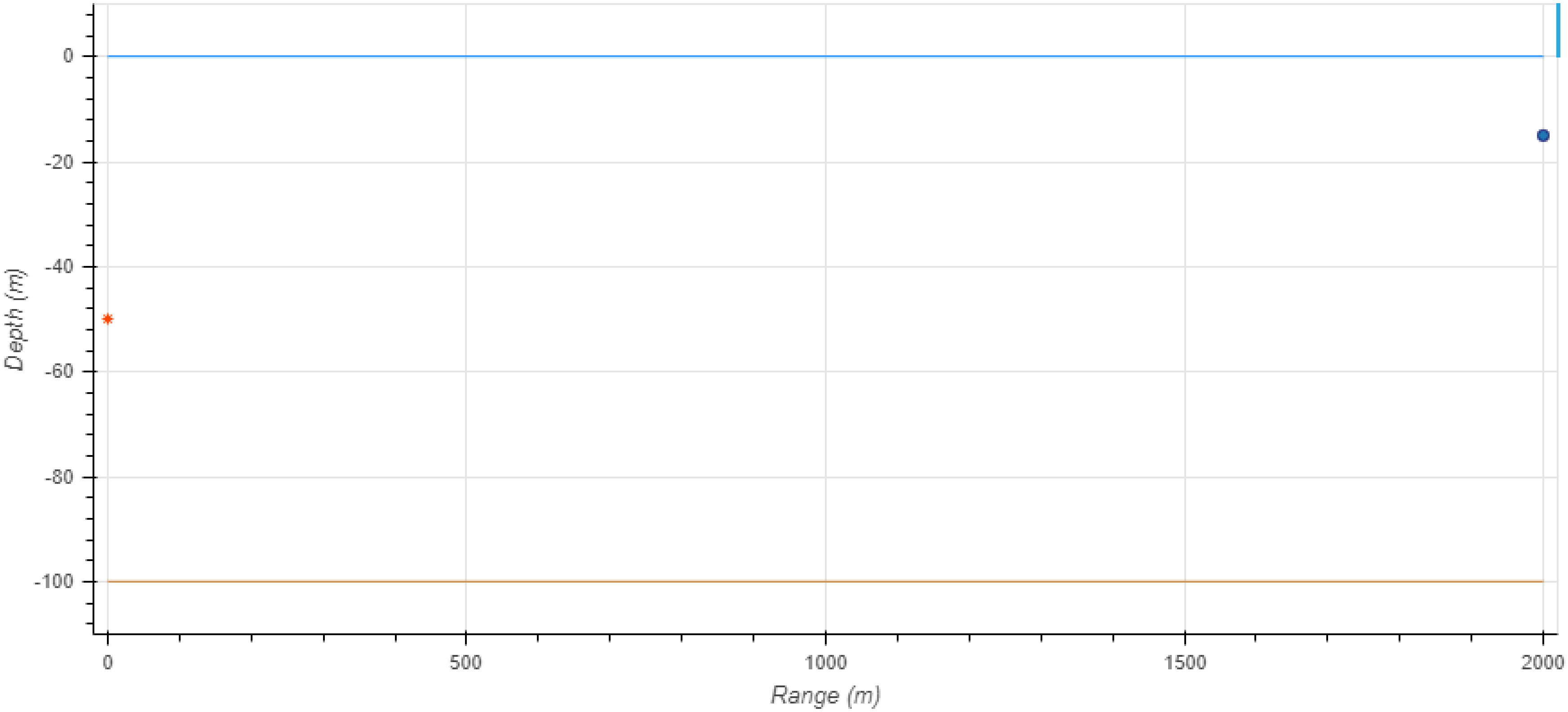
In [48]:
```python
env = pm.create_env2d(
        bottom_absorption=bottom_absorption,
        bottom_density=bottom_density,
        depth=depth,
        soundspeed=ssp,
        surface=surface,
        rx_depth=rx_depth,
        rx_range=rx_range,
        tx_depth=tx_depth,
        tx_directionality=tx_beampattern
    )
```

In [49]:
```
pm.print_env(env)
```

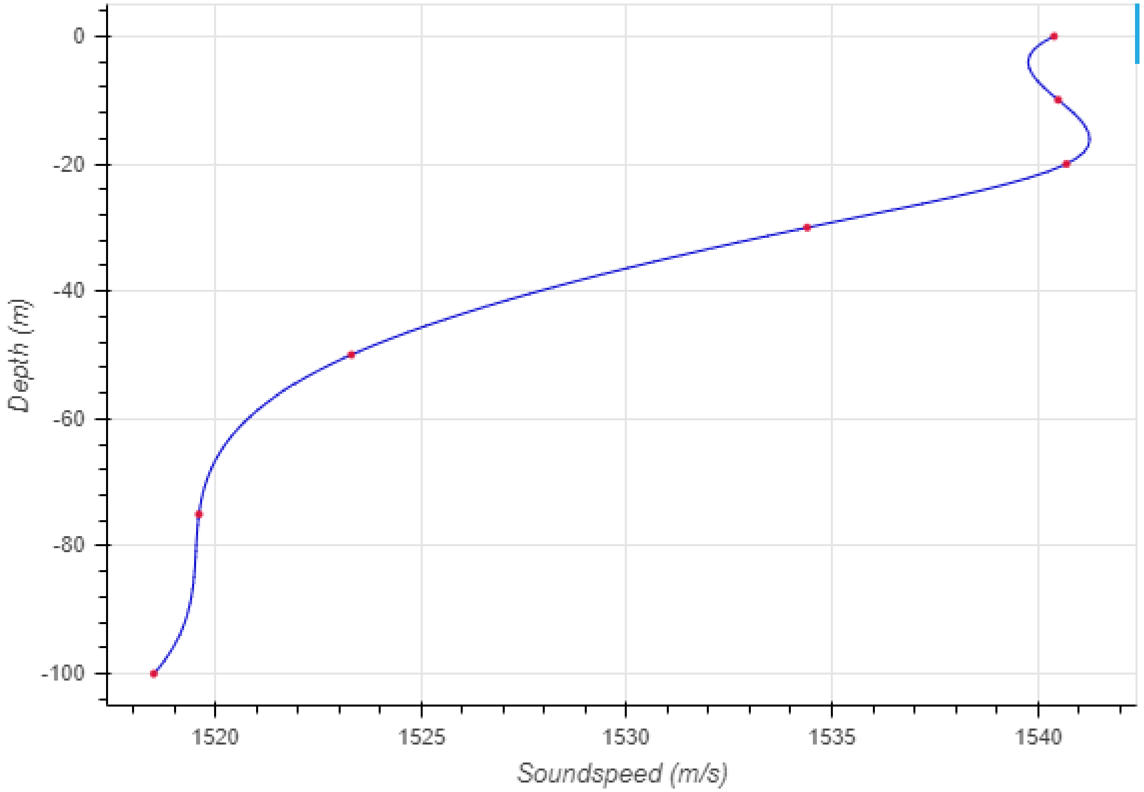
```
                name : arlpy
   bottom_absorption : 1.0
      bottom_density : 1200
    bottom_roughness : 0
   bottom_soundspeed : 1600
               depth : 100
        depth_interp : linear
           frequency : 25000
           max_angle : 80
           min_angle : -80
              nbeams : 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
```

In [50]:

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



In [51]:
```
pm.plot_ssp(env)
```



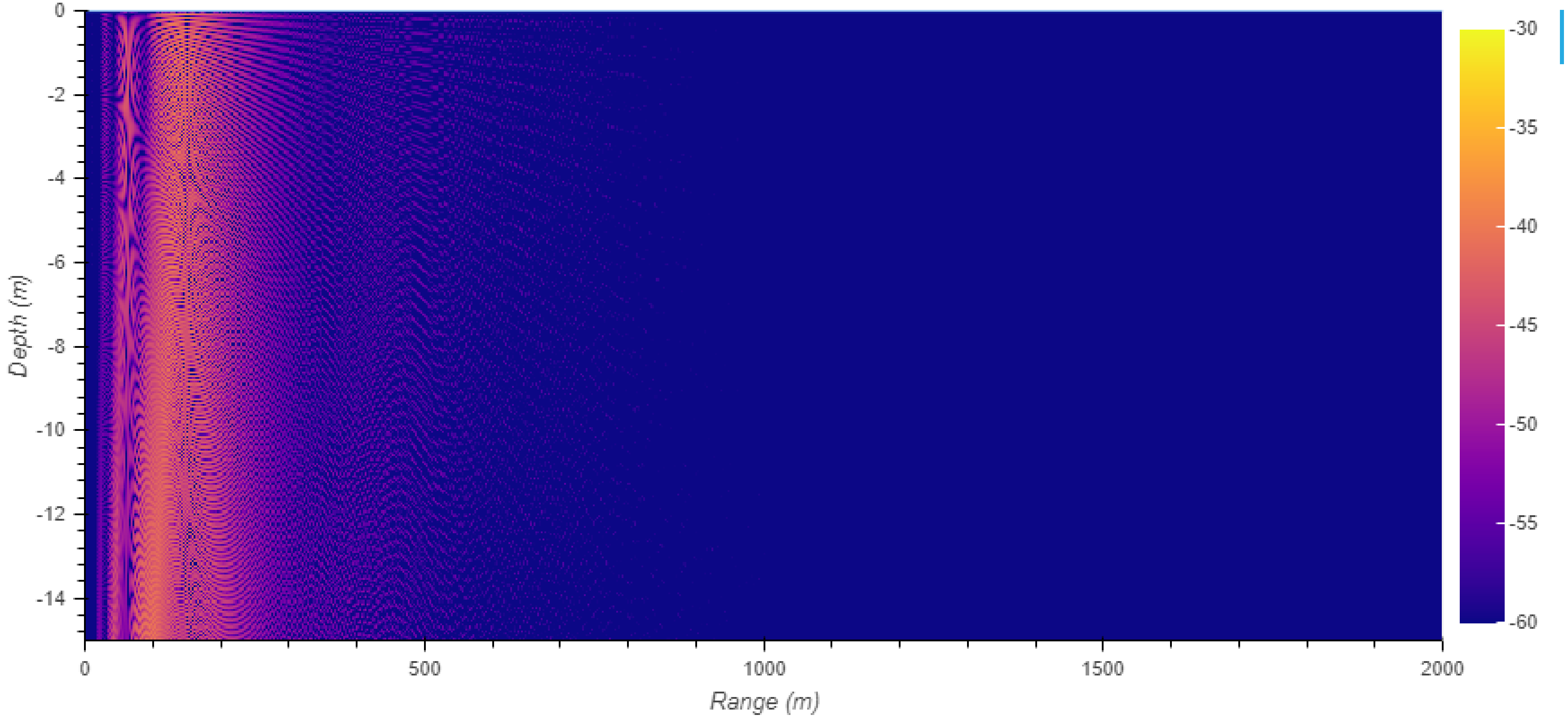
Source directionality

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

In [55]:
```
env['rx_range'] = np.linspace(0, 2000, 1001)
env['rx_depth'] = np.linspace(0, 15, 301)
```

In [56]:
```
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
```

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



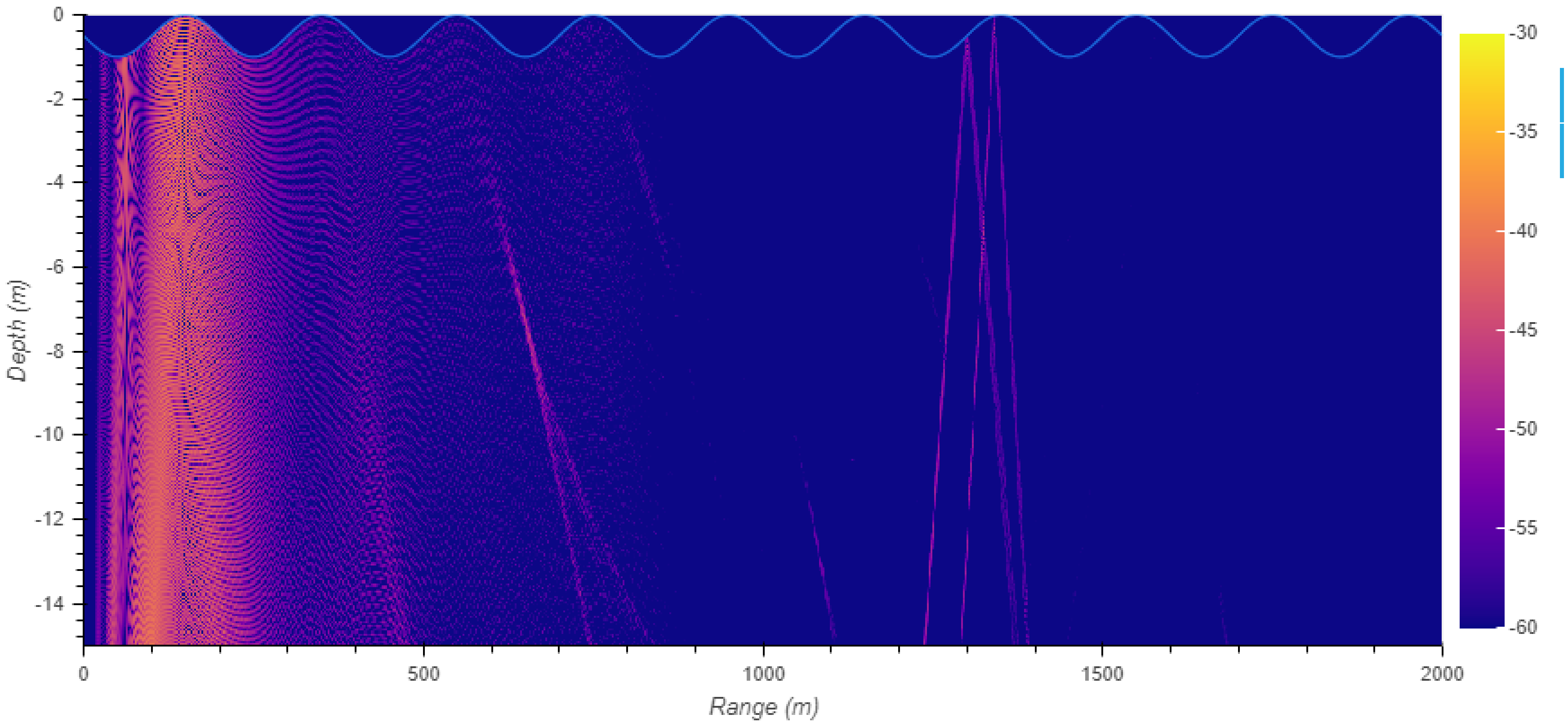
Now you can see the directionality and the sidelobe structure of the transmitter.

Undulating water surface

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

In [60]:
```
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
```

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



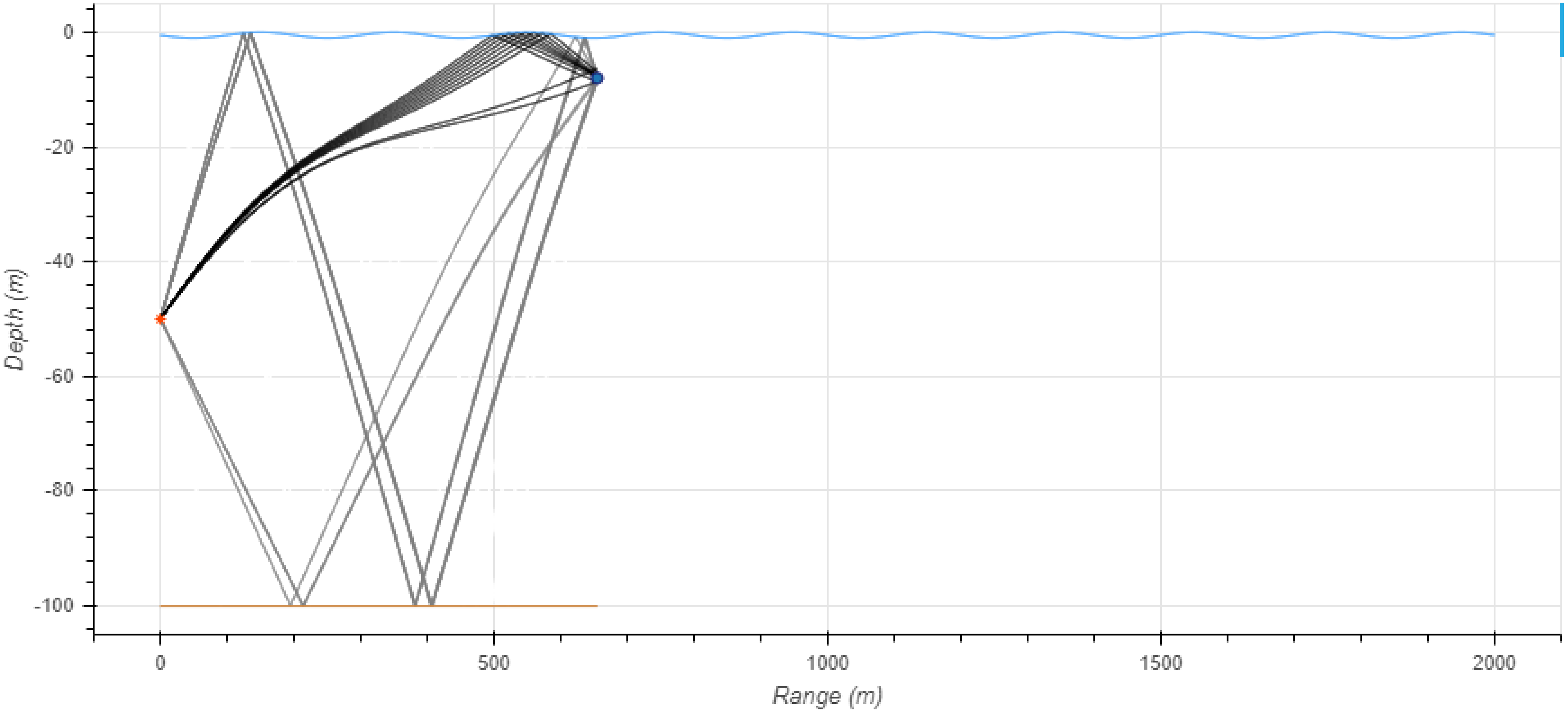
Now, if I placed a receiver at 655 m, and 8 m depth, roughly where we see some focusing, what would the eigenrays and arrival structure look like?

In [62]:

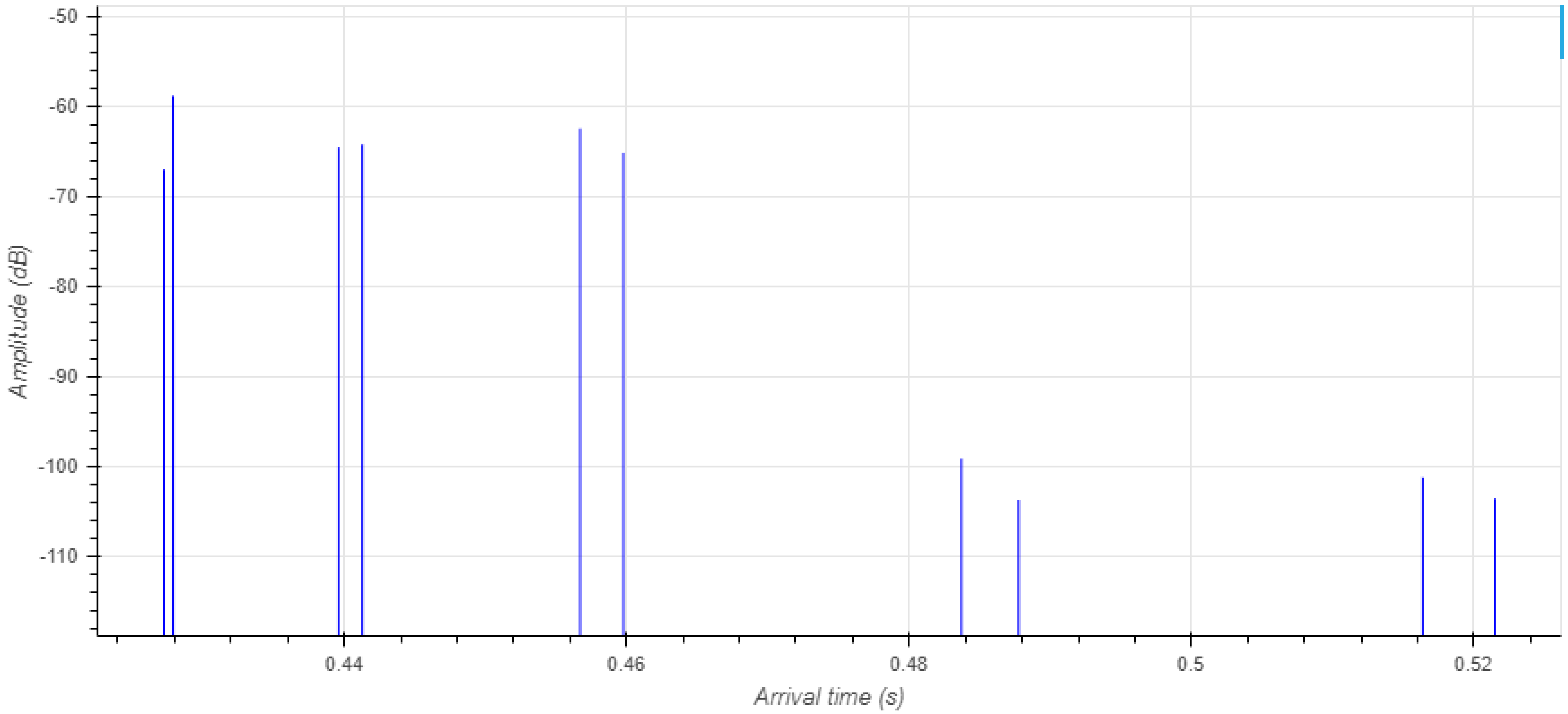
```
env['rx_range'] = 655
env['rx_depth'] = 8
```

In [63]:

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



In [64]:
```
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.