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

Getting started

Start off with checking that everything is working correctly:

In [1]:
```
pip freeze
```

```
arlpy==1.7.0
attrs==19.3.0
backcall==0.1.0
bleach==3.1.4
bokeh==2.0.1
certifi==2020.4.5.1
decorator==4.4.2
defusedxml==0.6.0
entrypoints==0.3
importlib-metadata==1.6.0
ipykernel==5.2.1
ipython==7.13.0
ipython-genutils==0.2.0
jedi==0.17.0
Jinja2==2.11.1
jsonschema==3.2.0
jupyter-client==6.1.3
jupyter-core==4.6.3
MarkupSafe==1.1.1
mistune==0.8.4
mkl-fft==1.0.15
mkl-random==1.1.0
mkl-service==2.3.0
nbconvert==5.6.1
nbformat==5.0.6
notebook==6.0.3
numpy==1.18.1
olefile==0.46
packaging==20.3
pandas==1.0.3
pandocfilters==1.4.2
parso==0.7.0
pexpect==4.8.0
pickleshare==0.7.5
Pillow==7.0.0
prometheus-client==0.7.1
prompt-toolkit==3.0.5
ptyprocess==0.6.0
Pygments==2.6.1
pyparsing==2.4.6
pyrsistent==0.16.0
python-dateutil==2.8.1
pytz==2019.3
PyYAML==5.3.1
pyzmq==19.0.0
scipy==1.4.1
Send2Trash==1.5.0
six==1.14.0
terminado==0.8.3
testpath==0.4.4
tornado==6.0.4
traitlets==4.3.3
typing-extensions==3.7.4.1
utm==0.5.0
wcwidth==0.1.9
webencodings==0.5.1
zipp==3.1.0
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import arlpy.uwapm as pm
import arlpy.plot as plt
import numpy as np
```

In [3]:
```
pm.models()
```

Out[3]:
```
['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 [4]:
```
env = pm.create_env2d()
pm.print_env(env)
```

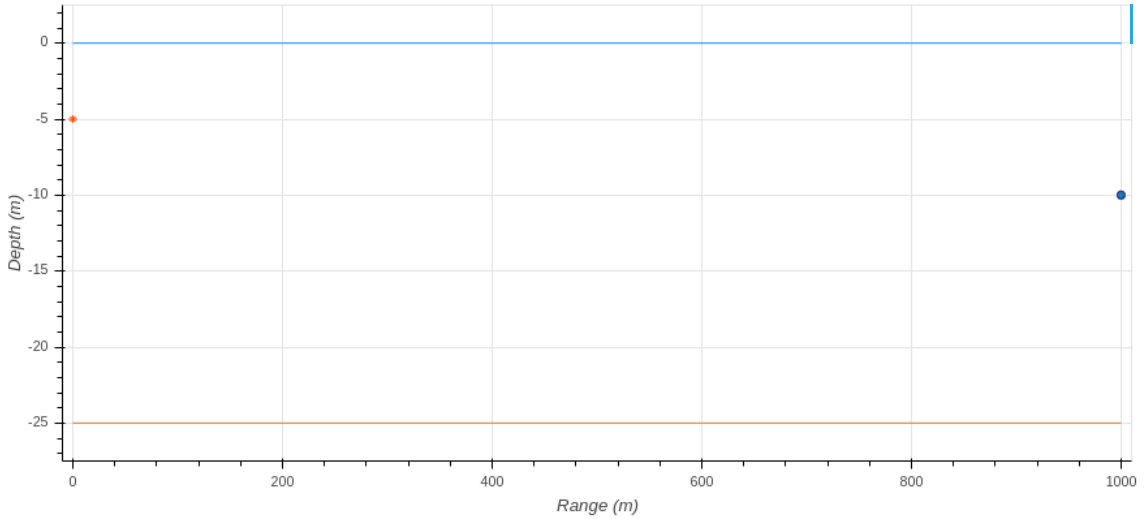
```
              name : arlpy
bottom_absorption : 0.1
   bottom_density : 1600
 bottom_roughness : 0
bottom_soundspeed : 1600
             depth : 25
      depth_interp : linear
         frequency : 25000
         max_angle : 80
         min_angle : -80
            nbeams : 0
          rx_depth : 10
          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 [5]:
```
pm.plot_env(env, width=900)
```



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

In [6]:
```
rays = pm.compute_eigenrays(env)
pm.plot_rays(rays, env=env, width=900)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-1af3b30f110e> in <module>
----> 1 rays = pm.compute_eigenrays(env)
      2 pm.plot_rays(rays, env=env, width=900)

~/anaconda3/envs/test_env/lib/python3.6/site-packages/arlpy/uwapm.py in compute_eigenrays(env, tx_depth_ndx, rx_depth
_ndx, rx_range_ndx, model, debug)
    352     if debug:
    353         print('[DEBUG] Model: '+model_name)
--> 354     return model.run(env, eigenrays, debug)
    355
    356 def compute_rays(env, tx_depth_ndx=0, model=None, debug=False):

~/anaconda3/envs/test_env/lib/python3.6/site-packages/arlpy/uwapm.py in run(self, env, task, debug)
    605             else:
    606                 try:
--> 607                     results = taskmap[task][1](fname_base)
    608                 except FileNotFoundError:
    609                     print('[WARN] Bellhop did not generate expected output file')

~/anaconda3/envs/test_env/lib/python3.6/site-packages/arlpy/uwapm.py in _load_rays(self, fname_base)
    819                         'ray': [ray]
    820                     }))
--> 821         return _pd.concat(rays)
    822
    823     def _load_shd(self, fname_base):

~/anaconda3/envs/test_env/lib/python3.6/site-packages/pandas/core/reshape/concat.py in concat(objs, axis, join, ignor
e_index, keys, levels, names, verify_integrity, sort, copy)
    279         verify_integrity=verify_integrity,
    280         copy=copy,
--> 281         sort=sort,
    282     )
    283

~/anaconda3/envs/test_env/lib/python3.6/site-packages/pandas/core/reshape/concat.py in __init__(self, objs, axis, joi
n, keys, levels, names, ignore_index, verify_integrity, copy, sort)
    327
    328         if len(objs) == 0:
--> 329             raise ValueError("No objects to concatenate")
    330
    331         if keys is None:

ValueError: No objects to concatenate
```

We can also compute the arrival structure at the receiver:

In [18]:
```
arrivals = pm.compute_arrivals(env)
pm.plot_arrivals(arrivals, width=900)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-3843dd94e0a3> in <module>
----> 1 arrivals = pm.compute_arrivals(env)
      2 pm.plot_arrivals(arrivals, width=900)

~/.local/lib/python3.8/site-packages/arlpy/uwapm.py in compute_arrivals(env, model, debug)
    323     if debug:
    324         print('[DEBUG] Model: '+model_name)
--> 325     return model.run(env, arrivals, debug)
    326
    327 def compute_eigenrays(env, tx_depth_ndx=0, rx_depth_ndx=0, rx_range_ndx=0, model=None, debug=False):

~/.local/lib/python3.8/site-packages/arlpy/uwapm.py in run(self, env, task, debug)
    605             else:
    606                 try:
--> 607                     results = taskmap[task][1](fname_base)
    608                 except FileNotFoundError:
    609                     print('[WARN] Bellhop did not generate expected output file')

~/.local/lib/python3.8/site-packages/arlpy/uwapm.py in _load_arrivals(self, fname_base)
    792                                 'bottom_bounces': [data[7]]
    793                             }, index=[len(arrivals)+1]))
--> 794         return _pd.concat(arrivals)
    795
    796     def _load_rays(self, fname_base):

~/.local/lib/python3.8/site-packages/pandas/core/reshape/concat.py in concat(objs, axis, join, ignore_index, keys, le
vels, names, verify_integrity, sort, copy)
    269     ValueError: Indexes have overlapping values: ['a']
    270     """
--> 271     op = _Concatenator(
    272         objs,
    273         axis=axis,

~/.local/lib/python3.8/site-packages/pandas/core/reshape/concat.py in __init__(self, objs, axis, join, keys, levels,
 names, ignore_index, verify_integrity, copy, sort)
    327
    328         if len(objs) == 0:
--> 329             raise ValueError("No objects to concatenate")
    330
    331         if keys is None:

ValueError: No objects to concatenate
```

The arrivals are returned in pandas (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 [ ]:
```
arrivals[arrivals.arrival_number < 10][['time_of_arrival', 'angle_of_arrival', 'surface_bounces', 'bottom_bounces']]
```

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

In [ ]:
```
ir = pm.arrivals_to_impulse_response(arrivals, fs=96000)
plt.plot(np.abs(ir), fs=96000, 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 [19]:
```
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
]
```

and then our soundspeed profile: