Ocean wave ray tracing v.1: User manual.

Trygve Halsne[1,2], Kai Haakon Christensen[1,3], Gaute Hope[1], and Øyvind Breivik[1,2]

[1]Norwegian Meteorological Institute, Oslo, Norway
[2]University of Bergen, Bergen, Norway
[3]University of Oslo, Oslo, Norway

Correspondence: Trygve Halsne (trygve.halsne@met.no)

S1 Introduction

Here we provide a tutorial on how to use the Python module for solving the wave ray equations for ocean waves at arbitrary depths in the presence of ambient currents. Focus is put on how to run the model, which includes preparation of the data as well as showing examples of using the optional arguments. Please note that a number of use examples, which are referred to in the "main paper", are given in the GitHub repository https://github.com/hevgyrt/ocean_wave_tracing. A generic use example is given in Alg. 1, and in the subsequent sections we will go through the optional options to the generic example in detail.

S2 Wave ray model initialization

In Alg. S1, all the default values in the `__init__` method is used, which includes the `depth` and `temporal_evolution`. However, any input current field can be given by editing the velocity fields `U,V`.

S2.1 Adding bathymetry

The generic example in Alg. S1 can be extended by including a bathymetry field like

```
depth = np.ones((nx,ny)) * 100
wt = Wave_tracing(U=U,V=np.zeros((ny,nx)),
                  nx=nx, ny=ny, nt=150,T=T,
                  dx=x[1]-x[0],dy=y[1]-y[0],
                  nb_wave_rays=20,
                  domain_X0=x[0], domain_XN=x[-1],
                  domain_Y0=y[0], domain_YN=y[-1],
                  d=depth)
```

The bathymetry must be a 2D numpy array, and must match the domain size. A dedicated method to check that the input field follow the bathymetry conventions is initialized automatically.

Algorithm 1 Generic workflow code example

```
import numpy as np
import maplotlib.pyplot as plt
from ocean_wave_tracing import Wave_tracing

# Defining some properties of the medium
nx = 100; ny = 100 # number of grid points in x- and y-direction
x = np.linspace(0,2000,nx) # size x-domain [m]
y = np.linspace(0,3500,ny) # size y-domain [m]
T = 250 # simulation time [s]
U=np.zeros((nx,ny))
U[nx//2:,:]=1

# Define a wave tracing object
wt = Wave_tracing(U=U,V=np.zeros((ny,nx)),
                  nx=nx, ny=ny, nt=150,T=T,
                  dx=x[1]-x[0],dy=y[1]-y[0],
                  nb_wave_rays=20,
                  domain_X0=x[0], domain_XN=x[-1],
                  domain_Y0=y[0], domain_YN=y[-1],
                  )

# Set initial conditions
wt.set_initial_condition(wave_period=10,
                         theta0=np.pi/8)
# Solve
wt.solve()
```

S2.1 Temporally varying current fields using xarray

In the GitHub repository, a set of idealized current input fields are given in a netCDF file in the folder `notebooks`. The fields span multiple time steps, such that a temporal varying current field can be invoked in the ray tracing by setting `temporal_evolution=True` like

```
import xarray as xa
ncin = xa.open_dataset('idealized_input.nc')
U = ncin.U
V = ncin.V
X = ncin.x.data
Y = ncin.y.data
nx = len(X)
ny = len(Y)

# Define a wave tracing object
wt = Wave_tracing(U=U,V=V,
                  nx=nx, ny=ny, nt=150,T=T,
                  dx=(X[1]-X[0]).values,dy=(Y[1]-Y[0]).values,
                  nb_wave_rays=20,
                  domain_X0=X[0].data, domain_XN=X[-1].data,
                  domain_Y0=Y[0].data, domain_YN=Y[-1].data,
                  temporal_evolution=True,
                  )
```

The ray model will take into account the current field from the associated model time step depending on the propagation time of the rays.

S2.2 Make input data follow the conventions

In the `Wave_tracing` object, the input fields follow some conventions that are listed in the main paper. Particularly relevant for the input velocity data is that the the dimensions must be named `x` and `y` if the input fields are of type xarray DataArray. For example, for ocean circulation models, typical dimension names can be called X and Y. This is, however, easy to overcome by renaming the dimension names at initialization like

```
wt = Wave_tracing(U=ncin.U.rename({'X':'x','Y':'y'}),
                  V=ncin.V.rename({'X':'x','Y':'y'}),
                  ...
```

Additional examples where the dimensions are reanamed are given in the `notebooks` folder in the repository.

S3 Initial conditions for the solver

In addition to the mandatory input values in the `set_initial_conditions` method in Alg. 1, it is possible to further specify the `theta0` as an array with values for each wave ray. For example, a uniform distribution of initial wave propagation angles in the range $[0, \pi)$ is given as `theta0=np.linspace(0,np.pi,nb_wave_rays)`. Initial positions for each wave ray can be given by specifying the "keys and values" (`kwargs`) arguments `ipx` and `ipy`,

```
wt.set_initial_condition(wave_period=10,
                         theta0=np.linspace(0,np.pi,nb_wave_rays),
                         ipx=np.linspace(10,20,nb_wave_rays),
                         ipy=np.linspace(1,5,nb_wave_rays)
                         )
```

provided that `incoming_wave_side` is not given since it trumps `ipx` and `ipy`.

S4 Numerical integration

When the equations are to be solved (see `solve()` in Alg 1), it is possible to choose between the finite-difference schemes available in the `util_solvers.py`. In v.1, two schemes are available i.e. a Forward-Euler scheme and a 4th order Runge-Kutta, where the latter is default.