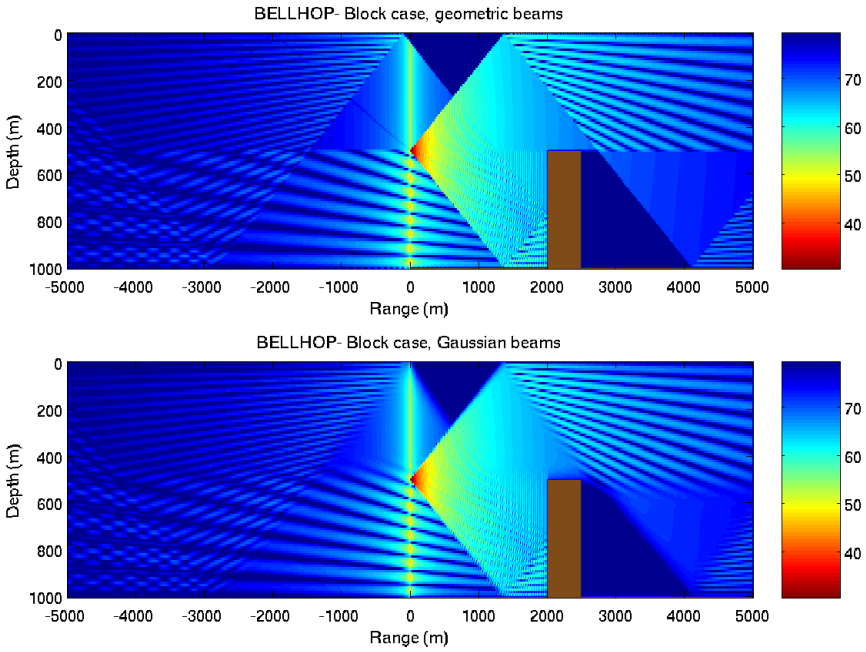
General description of

the BELLHOP ray tracing program

(June 2008 release)



Orlando Camargo Rodríguez
(http://w3.ualg.pt/orodrig, orodrig@ualg.pt)
Physics Department
Signal Processing Laboratory
Faculdade de Ciências e Tecnologia
Universidade do Algarve

Version 1.0 (13/06/2008)

Contents

1 Introduction

Bellhop is a highly efficient ray tracing program, written in Fortran by Michael Porter as part of the Acoustic Toolbox (available at the website of the Ocean Acoustic Library). Bellhop is designed in order to perform two-dimensional acoustic ray tracing for a given sound speed profile $c(z)$ or a given sound speed field $c(r, z)$, in ocean waveguides with flat or variable absorbing boundaries. Output options include ray coordinates, travel time, amplitude, eigenrays, acoustic pressure or transmission loss (either coherent, incoherent or semi-coherent). The calculation of acoustic pressure is based on the theory of Gaussian beams [1, 2], which can be applied using different approximations, namely:

- geometric beams (the default option) [3];
- beams with ray-centered coordinates;
- beams with Cartesian coordinates;
- Gaussian ray bundless approximation [4].

This description is intended to those working in Ocean Acoustics, interested in using Bellhop for particular applications. In the sections that will follow we are going to discuss briefly how to get and install the model, the equations that it solves, and to illustrate Bellhop capabilities through a set of examples.

2 Theoretical background

Ray tracing requires the solution of the ray equations to determine the ray coordinates. Amplitude and acoustic pressure requires the solution of the dynamic ray equations, which are described in detail in [1].

For a system with cylindrical symmetry the ray equations can be written as [2]

$$\begin{aligned} \frac{dr}{ds} &= c\xi(s) \ , \quad \frac{d\xi}{ds} = -\frac{1}{c^2}\frac{\partial c}{\partial r}, \\ \frac{dz}{ds} &= c\zeta(s) \ , \quad \frac{d\zeta}{ds} = -\frac{1}{c^2}\frac{\partial c}{\partial z}, \end{aligned} \tag{1}$$

where $r(s)$ and $z(s)$ represent the ray coordinates in cylindrical coordinates and s is the arclenght along the ray; the pair $c(s)\,[\xi(s), \zeta(s)]$ represents the tangent versor along the ray. Initial conditions for $r(s)$, $z(s)$, $\xi(s)$ and $\zeta(s)$ are

$$r(0) = r_s \quad , \quad z(0) = z_s \quad , \quad \xi(0) = \frac{\cos\theta_s}{c_s} \quad , \quad \zeta(0) = \frac{\sin\theta_s}{c_s} \quad ,$$

where θ_s represents the launching angle, (r_s, z_s) is the source position, and c_s is the sound speed at the source position. The coordinates are sufficient to obtain the ray travel time:

$$\tau = \int_\Gamma \frac{ds}{c(s)} \ , \tag{2}$$

which is calculated along the curve $[r(s), z(s)]$.

3 Installation

All the Fortran sources of the models included in the Acoustic Toolbox are provided with a makefile, plus a set of Matlab utilities to display ray coordinates, acoustic pressure or transmission loss. The toolbox has been sucessfully tested with the following compilers:

- Intel fortran compiler on linux;

- gfortran compiler;
- g95 compiler.

With minor modifications of the makefiles (which depend on the particular Fortran compiler available) typing `make clean` and `make install` on the command line should allow to compile all the models, including Bellhop. Once the binaries are created the models can be called using the scripts included in the Toolbox.

4 Numerical issues

Ray and dynamic equations are integrated in Bellhop using a two-step polygon method. For a given output option the model writes ray coordinates for every launching angle, or calculates progressively travel time data, acoustic pressure or transmission loss on the grid specified by the user.

5 Input and output files

Bellhop input files (with extension `*.env`) are required to be compatible with **Kraken** (a normal mode model) input files. Thus, some of the parameters are not used to perform the ray tracing. The general structure of the `*.env` input file used by **Bellhop** is as follows:

```
TITLE
Frequency (in Hz)
nmedia    (dummy integer < 20)
OPTIONS1
SURFACE-LINE
nmesh sigmas z(nssp)       |
z(1) cp(1) /               |
z(2) cp(2) /               |  Sound
 .    .                    |  Speed
 .    .                    |  Block
 .    .                    |
z(nssp) cp(nssp) /        _|_
OPTIONS2 sigmab            |  Bottom
BOTTOM-LINE               _|_ Block
nsources     (number of sources)                    |
source-depth(1) source-depth(nsources)     / (in m)  |
nrd          (number of receivers x depth)          |  Array
```

```
receiver-depth(1) receiver-depth(nrd)      / (in m)  |  Block
nrr          (number of receivers x range)           |
receiver-range(1) receiver-range(nrr)      / (in km)_|_
OPTIONS3                                             |
nbeams       (number of launching angles)            |  Output
theta(1) theta(nbeams) (launching angles in degrees) |  Block
ray-step  zmax  rmax                                _|_
OPTIONS4 epmult rloop isingl                         |  Beam
nimage ibwin component                              _|_ Block
```

`OPTIONS1` is a five-character string enclosed in single quotes. Here it follows the description of each character:

- `OPTIONS1(1)`: describes the method of interpolation used by Bellhop to calculate sound speed and its derivatives along the ray. This character can correspond to one of the following:
  - `'S'`: cubic spline interpolation;
  - `'C'`: C-linear interpolation;
  - `'N'`: N2-linear interpolation;
  - `'A'`: analytic interpolation (requires adaptation of the subroutine SSP and further model recompilation);
  - `'Q'`: quadratic approximation to the sound speed field (requires the creation of a `*.ssp` file containing the field).
- `OPTIONS1(2)`: describes the type of surface and can correspond to one of the following:
  - `'V'`: vacuum above surface (the `SURFACE-LINE` is not required);
  - `'R'`: perfectly rigid media above surface (the `SURFACE-LINE` is not required);
  - `'A'`: acoustic half-space; `SURFACE-LINE` should be written as
    `z-surface cp-surface cs-surface density-surface alpha-surface /`
  - `'F'`: read a list of reflection coefficients from a *irc file (requires running first the bounce program).
- `OPTIONS1(3)`: describes attenuation in the bottom (for more details see [5]) and can correspond to one of the following:

  - 'F': attenuation units correspond to (dB/m)kHz;
  - 'L': attenuation units correspond to the parameter loss;
  - 'M': attenuation units correspond to dB/m;
  - 'N': attenuation units correspond to Nepers/m;
  - 'Q': attenuation units correspond to Q-factor;
  - 'W': attenuation units correspond to dB/wavelength.

- OPTIONS1(4): optional parameter describing Thorpe volume attenuation in the watercolumn; if set it should correspond to 'T'.

- OPTIONS1(5): optional parameter describing the surface shape; if not specified the surface is considered flat, if specified it should correspond to '*'. In the former case the surface coordinates should be described in a *.ati file, with the following structure:

```
interpolation type
npoints
r(1)    z(1)
r(2)    z(2)
 .          .
 .          .
 .          .
r(npoints) z(npoints)
```

  The parameter interpolation type is a character, equal to 'L' (for a linear interpolation of the surface) or 'C' (for curvilinear interpolation); surface ranges should be specified in km, surface depths in m.

The parameters nmesh and sigmas are not used by Bellhop, while the parameter z(nssp) is used to detect the last point of the sound speed profile. The values of z() and cp() correspond to depth in m and P-wave speed in m/s.

OPTIONS2 is a two-character string enclosed in single quotes. Here it follows the description of each character:

- OPTIONS2(1): describes the type of media below the watercolumn and it can correspond to one of the following:

  - 'V': vacuum below watercolumn (the BOTTOM-LINE is not required);

  - 'R': rigid below watercolumn (the BOTTOM-LINE is not required);
  - 'A': acoustic half-space; the BOTTOM-LINE should be written as

    ```
    z-bottom cp-bottom cs-bottom density-bottom alpha-bottom
    /
    ```

    for obvious reasons z-bottom should be equal to z(nssp); cs(bottom) is ignored, density(bottom) should be specified in g/cm^3 and the units of alpha(bottom) depend on OPTIONS1(3).
  - 'F': read a list of reflection coefficients from a *.brc file (requires running first the bounce program).

- OPTIONS2(2): describes the shape of the bottom. It can be empty (which corresponds to a flat bottom) or it can correspond to '*'. In the former case the bottom coordinates should be described in a *.bty file, with the following structure:

  ```
  interpolation type
  npoints
  r(1)    z(1)
  r(2)    z(2)
   .          .
   .          .
   .          .
  r(npoints) z(npoints)
  ```

  again interpolation type can be equal to 'L' or 'C'; bottom ranges should be specified in km, bottom depths in m.

  The following six lines describe the number of sources and corresponding depths (in m), and the number of receivers along range and depth. Those lines are self-describing.

OPTIONS3 is a five-character string, which describes output options. Here it follows the description of each character:

- OPTIONS3(1): describes the type of information that should be written to the output file. It can correspond to
  - 'A': write amplitudes and travel times;
  - 'E': write eigenray coordinates;
  - 'R': write ray coordinates;
  - 'C': write coherent acoustic pressure;

  - `'I'`: write incoherent acoustic pressure;
  - `'S'`: write semi-coherent acoustic pressure.

- `OPTIONS3(2)`: describes the approximation used to calculate acoustic pressure; it can be empty or it can correspond to one of the following:
  - `'G'`: use geometric beams (default);
  - `'C'`: use Cartesian beams;
  - `'R'`: use ray-centered beams;
  - `'B'`: use Gaussian beam bundles.

- `OPTIONS3(3)`: selects the inclusion of the beam shift effect; it can be empty or it can correspond to one of the following:
  - `' '`: do not include beam shift effect (default);
  - `'S'`: include beam shift effect;
  - `'*'`: use a source beam pattern file (requires a `*.sbp` file, similar to the `*.ati` and `*.bty` files, with angles in degrees and amplitudes, instead of ranges and depths).

- `OPTIONS3(4)`: describes the type of source; it can be empty or it can correspond to one of the following:
  - `'R'`: point source in cylindrical coordinates (default);
  - `'X'`: line source in Cartesian coordinates.

- `OPTIONS3(5)`: describes the type of array; it can be empty or it can correspond to one of the following:
  - `'R'`: rectilinear receiver grid, receivers at rr(:) $\times$ rd(:) (default);
  - `'I'`: irregular grid, receivers at rr(:) , rd(:).

  The integer `nbeams` indicates the number of launching angles and the pair or reals `theta(1)` and `theta(nbeams)` determine the first and last launching angles, in degrees. Launching angles directed towards the bottom are considered positive, while launching angles directed towards the surface are considered negative. The parameters `ray-step` (in m), `zmax` (in m) and `rmax` (in km) define the ray step *ds* used in the integration of the ray and dynamic equations, and the "box" dimensions used to stop the tracing of rays leaving the box.

When the parameter `OPTIONS3` is composed of a single character there is no need to include more lines in the `*.env` input file. Otherwise the model expects two additional lines, which contain additional information regarding the beam characteristics. In those two lines one can find the `OPTIONS4` parameter, which is a string composed of two characters. Here it follows the description of each character:

- `OPTIONS4(1)`: describes the type of beam and it can correspond to one of the following:
  - `'C'`: Cerveny type;
  - `'F'`: space-filling;
  - `'M'`: minimum width;
  - `'W'`: WKB beams.
- `OPTIONS4(2)`: describes the type of beam curvature and it can correspond to one of the following:
  - `'D'`: use curvature doubling;
  - `'S'`: use standard curvature;
  - `'Z'`: use zeroing curvature.

The parameters `epmult` and `rloop` should be positive reals, while `isingl`, `nimage` and `ibwin` should be integers. The integer `nimage` can take the values 1, 2 or 3. `Component` is a single character, which is used only when the acoustic pressure is calculated using ray-centered coordinates (when `OPTIONS3(2) = 'R'`); it can be empty (acoustic pressure will be written to the output file), equal to `'H'` (write the horizontal component of pressure to the output file) or equal to `'V'` (write the vertical component of pressure to the output file).

6 Examples

In this section **Bellhop** capabilities are illustrated by providing the input file used for different types of calculations. In all cases we are going to consider a Munk deep water profile, between 0 and 5000 m depth, a 50 Hz source at 1000 m depth, a ray step of 100 m, and 70 rays between -13° and 13°. Sound speed at bottom is 1600 m/s, bottom density is 1.8 g/cm^3 and bottom attenuation is 0.8 dB/λ. The environment is illustrated in Fig.1.

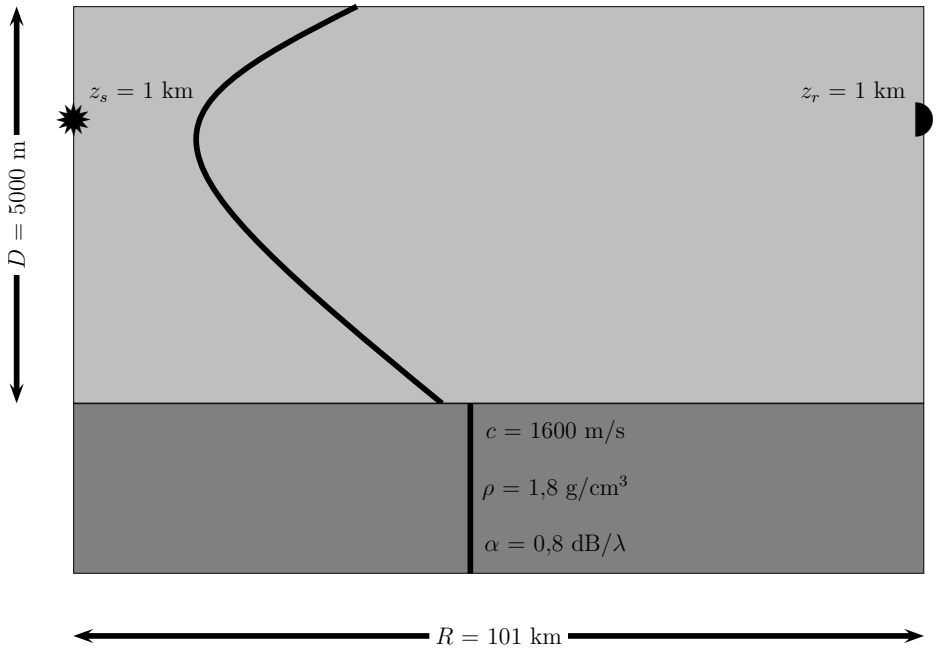


Figure 1: Schematic of the baseline deep water scenario considered in the examples.

6.1 A flat waveguide

Let us start with a simple ray trace between 0 and 101 km, using an input file called `flatwav.env`. The file looks like this:

```
'Munk profile/Flat waveguide'
50.0
1
'SVW'
51  0.0  5000.0
    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  /
'A'  0.0
 5000.0  1600.00 0.0 1.8 .8
1
1000.0 /
1
```

```
1000.0 /
1
101.0 /
'R'
70
 -13.0 13.0  /
100.0  5500.0  102.0
```

On the command line one invokes Bellhop by writing

```
whatevershell$ bellhop flatwav <ENTER>
```

Once **Bellhop** finishes with calculations one can verify that two files were created: the first is called `flatwav.prt`, with general information regarding the waveguide characteristics, number of launching angles, calculation time, etc, etc. The second one is called `flatwav.ray`, and it is an ASCII file containing ray coordinates. One can plot them using the M-file `plotray.m`:

```
>> plotray( 'flatwav' ) <ENTER>
```

which generates Fig.1. Changing `OPTIONS3(1) = 'R'` to `OPTIONS3(1) = 'E'` and proceeding as before one can obtain the eigenrays shown in Fig.2.

The calculation of coherent transmission loss requires some minor modifications to the input file. First, we set `OPTIONS3(1) = 'C'`; second, let us consider that transmission loss will be calculated on a rectangular grid with 501×501 points in range and depth. Finally, let us let set `nbeams` $= 0$ and let **Bellhop** decide how many rays are required. The input file then becomes:

```
'Munk profile/Flat waveguide'
50.0
1
'SVW'
51  0.0  5000.0
    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  /
```

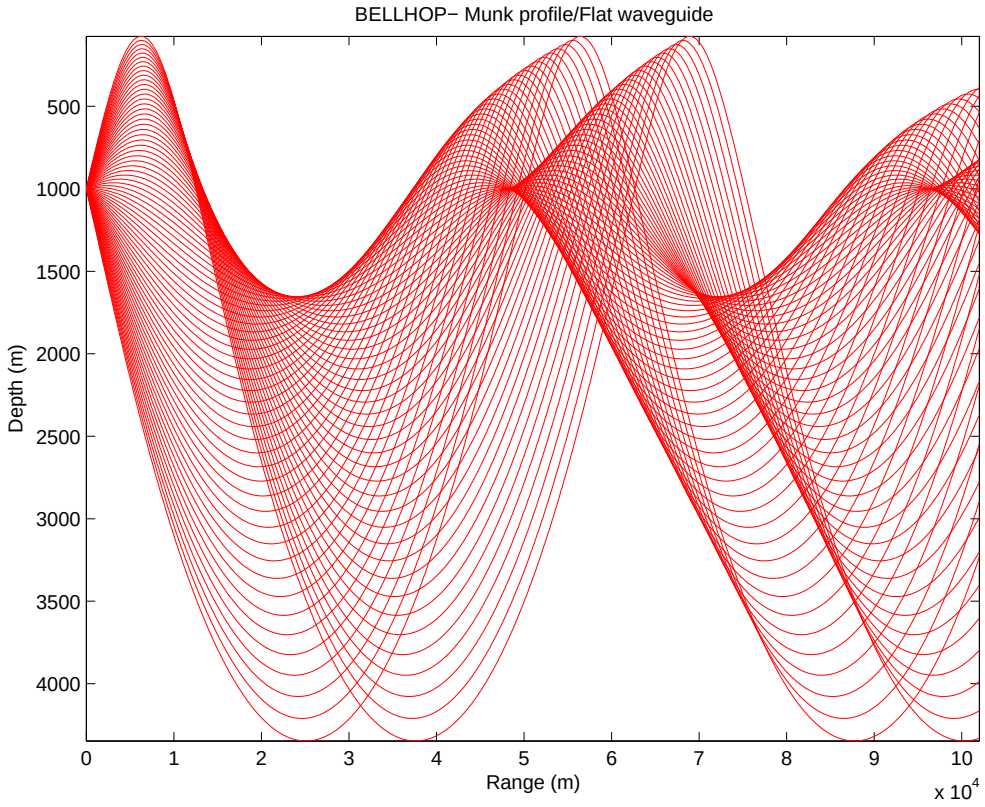


Figure 2: Rays calculated by **Bellhop** in a flat deep water waveguide.

```
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  /
```

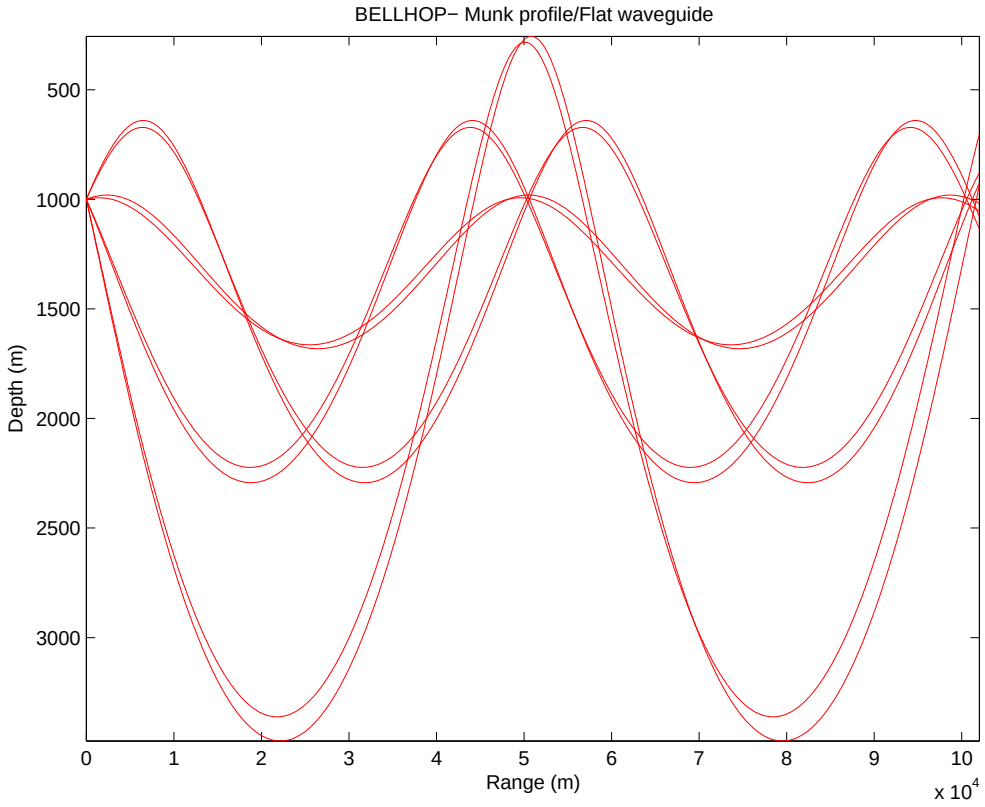


Figure 3: Eigenrays calculated by **Bellhop** in a flat deep water waveguide.

```
 5000.0  1551.91  /
'A'  0.0
 5000.0  1600.00 0.0 1.8 .0 /
1
  1000.0 /
 501
   0.0 5000.0 /
 501
    0.0 101.0 /
'C'
0
 -14.0 14.0  /
100.0  5500.0  102.0
```

Now, after running **Bellhop** one obtains a binary file called `flatwav.shd`, which in fact contains the acoustic pressure, calculated coherently. We can plot the transmission loss using the M-file `plotshd.m`:

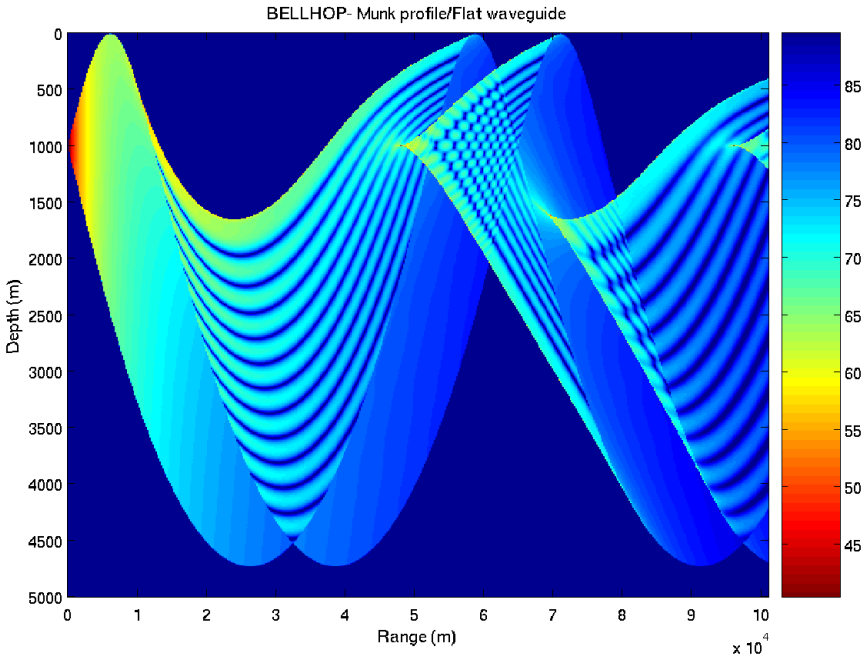


Figure 4: Coherent transmission loss calculated by **Bellhop**.

```
>> plotshd( 'flatwav.shd' ) <ENTER>
```

which produces Fig.4.

Finally, let us set `OPTIONS3(1)` = `'A'` and modify `flatwav.env` as shown bellow:

```
'Munk profile/Flat waveguide'
50.0
1
'SVW'
51  0.0  5000.0
    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  /
'A'  0.0
 5000.0  1600.00 0.0 1.8 .0 /
1
  1000.0 /
 1
   1000.0 /
 1
    101.0 /
'A'
101
 -14.0 14.0  /
100.0  5500.0  102.0
```

After running **Bellhop** one obtains an ascii file called `flatwav.arr`, with the amplitudes and travel times of the rays that arrive at the receiver position (we indicated only one, but the model is able to calculate travel times and amplitudes at all points of the array indicated in the array block). The data contained in the `*.arr` file can be read using the M-file `read_arrivals_asc.m`:

```
>>  [ a, tau, theta0, thetaR, srefl, brefl, narr, rz ] = ...
read_arrivals_asc( 'flatwav.arr' ) <ENTER>
```

6.2 A Gaussian seamountain (variable bottom)

Let us illustrate ray calculations with **Bellhop** with a non-flat bottom. We idealize a seamount using a Gaussian function and define it in a file called `seamount.bty`, which looks like this:

```
'L'
101
   0.0000000e+00   4.9971624e+03
   1.0100000e+00   4.9963474e+03
   2.0200000e+00   4.9953223e+03
   3.0300000e+00   4.9940400e+03
   4.0400000e+00   4.9924447e+03
   5.0500000e+00   4.9904711e+03
         .               .
         .               .
         .               .
   9.6960000e+01   4.9924447e+03
   9.7970000e+01   4.9940400e+03
   9.8980000e+01   4.9953223e+03
   9.9990000e+01   4.9963474e+03
   1.0100000e+02   4.9971624e+03
```

Next we make a copy of `flatwav.env` called `seamount.env`, and modify it like shown below:

```
'Munk profile/Seamount'
50.0
1
'SVW'
51  0.0  5000.0
    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  /
'A*'  0.0
 5000.0  1600.00 0.0 1.8 .0 /
1
  1000.0 /
 1
   5000.0 /
 1
    101.0 /
'R'
71
 -14.0 14.0  /
100.0  5500.0  102.0
```

After running **Bellhop** we can plot the rays (shown in Fig.5) using `plotrays.m`. As in the previous case eigenray calculations or coherent transmission loss can be obtained by replacing `OPTIONS3(1) = 'R'` with `OPTIONS3(1) = 'E'` and `OPTIONS3(1) = 'C'`, respectively. However, keep in mind that the array block should be written accordingly for every of those options.

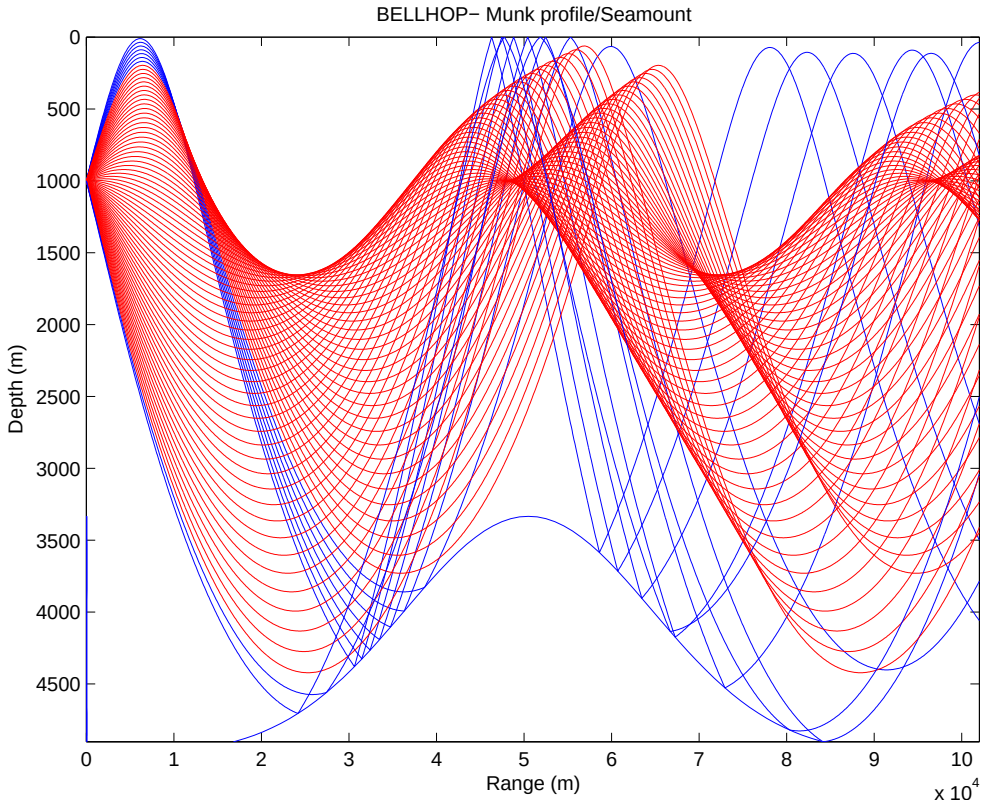


Figure 5: Rays calculated by **Bellhop** in a deep water waveguide with a Gaussian seamount.

6.3 Variable boundaries

Bellhop is not only able to handle a variable bottom, it can also deal simultaneously with a variable surface and bottom. Let us illustrate this by copying the `seamount.bty` file as `varbounds.bty`; further, we create a `varbounds.ati` file containing the coordinates of a wavy surface, which looks like this:

```
'L'
101
   0.0000000e+00   1.0000000e+02
   1.0100000e+00   1.3090170e+02
   2.0200000e+00   1.5877853e+02
   3.0300000e+00   1.8090170e+02
   4.0400000e+00   1.9510565e+02
   5.0500000e+00   2.0000000e+02
   6.0600000e+00   1.9510565e+02
   7.0700000e+00   1.8090170e+02
   8.0800000e+00   1.5877853e+02
         .               .
         .               .
         .               .
   9.6960000e+01   4.8943484e+00
   9.7970000e+01   1.9098301e+01
   9.8980000e+01   4.1221475e+01
   9.9990000e+01   6.9098301e+01
   1.0100000e+02   1.0000000e+02
```

Then, `seamount.env` is copied as `varbounds.env`, and modified to look like follows:

```
'Munk profile/Variable boundaries'
50.0
1
'SVW *'
51  0.0  5000.0
    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  /
'A*'  0.0
 5000.0  1600.00 0.0 1.8 .0 /
1
  1000.0 /
 1
   5000.0 /
 1
    101.0 /
'R'
71
 -14.0 14.0  /
100.0  5500.0  102.0
```

Running **Bellhop** with this input file we can get Fig.6.

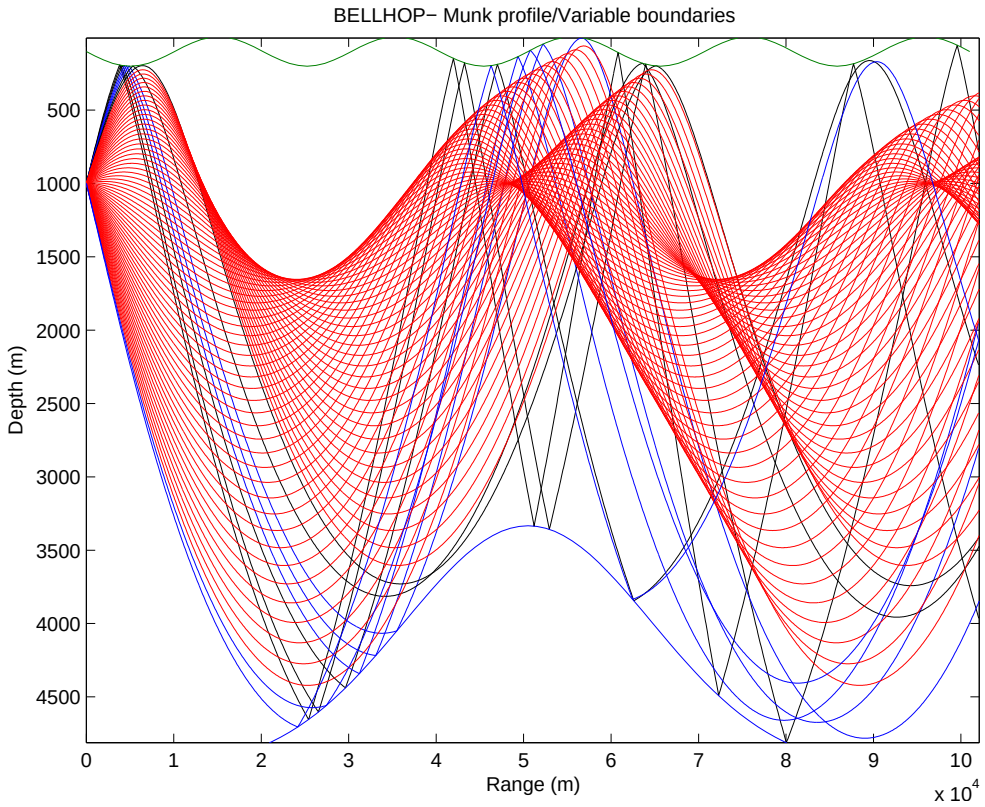


Figure 6: Rays calculated by **Bellhop** in a deep water waveguide with a wavy surface and a Gaussian seamount.

6.4 Ray trace with a sound speed field

The latest version of **Bellhop** allows calculations with a sound speed field, which is going to be illustrated here with a ray calculation only. Let us create a `gulf.env` input file, which look like follows:

```
'Gulf sound speed field'
50.0
1
'QVW'
0 0.0 5000.0
    0.0  1536.00 /
  200.0  1506.00 /
  700.0  1503.00 /
  800.0  1508.00 /
 1200.0  1508.00 /
 1500.0  1497.00 /
 2000.0  1500.00 /
 3000.0  1512.00 /
 4000.0  1528.00 /
 5000.0  1545.00 /
'A' 0.0
 5000.00   1800.0  0.0  2.0  0.1 0.0
1 ! NSD
300.0 /          ! SD(1:NSD) (m)
101 ! NRD
0.0 5000.0 / ! RD(1:NRD) (m)
1001 ! NR
0.0  200.0 / ! R(1:NR ) (km)
'R'   ! 'R/C/I/S'
51 ! NBeams
-10.0 10.0 /          ! ALPHA1,2 (degrees)
0.0  5500.0 201.0 ! STEP (m), ZBOX (m), RBOX (km)
```

Setting `OPTIONS1(1) = 'Q'` allows to take into account a sound speed field, named `gulf.ssp` which looks like follows (the syntax is self describing):

```
8
0.0     12.5    25.0    37.5    50.0    75.0    100.0   125.0
1536 1536 1536 1536 1536 1536 1536 1536
1506 1508.75 1511.5 1514.25 1517 1520 1524 1528
1503 1503 1503 1502.75 1502.5 1502 1502 1502
```

```
1508 1507 1506 1505 1504 1503 1501.5 1500
1508 1506.6 1505 1503.75 1502.5 1500.5 1499 1497
1497 1497 1497 1497 1497 1497 1497 1497
1500 1500 1500 1500 1500 1500 1500 1500
1512 1512 1512 1512 1512 1512 1512 1512
1528 1528 1528 1528 1528 1528 1528 1528
1545 1545 1545 1545 1545 1545 1545 1545
```

Running **Bellhop** with this input file allows to get Fig.7.

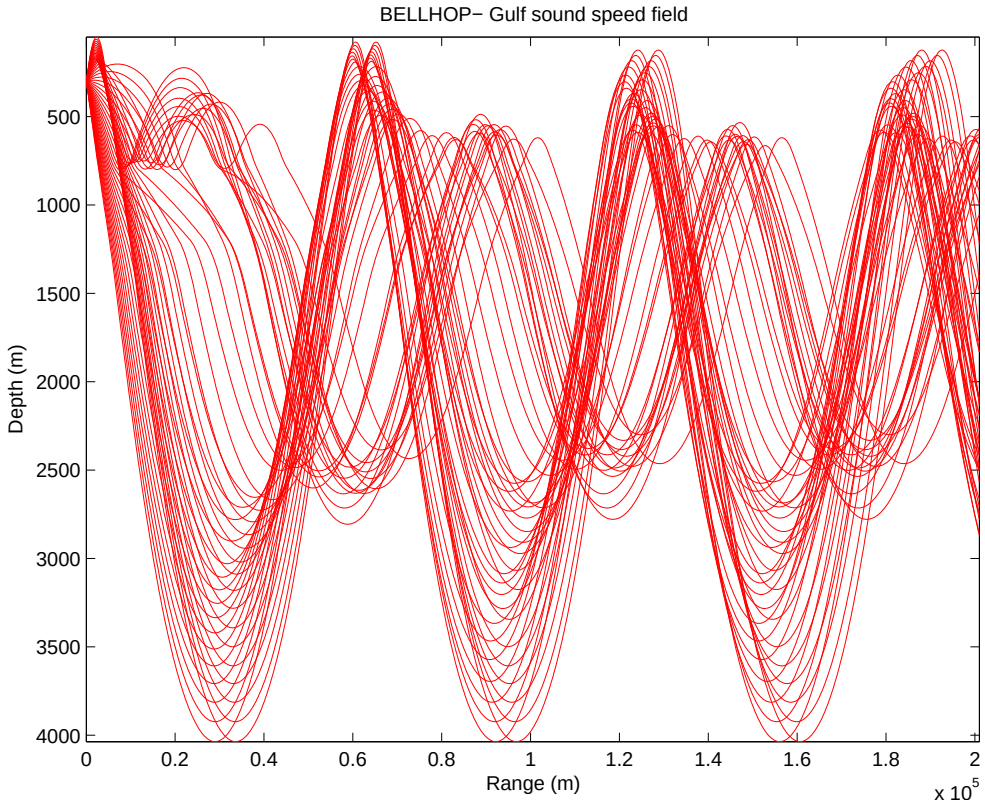


Figure 7: Rays calculated by **Bellhop** with a sound speed field.

6.5 Near source fields

The examples of the previous sections used **Bellhop**'s default option to calculate acoustic pressure, namely, geometric beams, which sometimes can be not sufficiently accurate for particular applications. To this effect **Bellhop** provides additional approximations to improve accuracy. Let us illustrate this through the calculation of coherent transmission loss near the source with different approximations, so the interference pattern reveals the accuracy of the approximation used. First, we copy `flatwav.env` to `nearsource.env`, which now looks like follows:

```
'Munk profile/Near source field'
50.0
1
'SVF'
51  0.0  5000.0
    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  /
'A'  0.0
 5000.0  1600.00 0.0 1.8 .8 /
1
  1000.0 /
501
    0.0 2000.0 /
501
  0.2  10.0 /
'C'
201
 -25.0 25.0  /
0.0  5500.0  102.0,
'MS' 1.0  100.0 0,
3  5
```

We can notice the inclusion of two additional lines (in fact, the `beam block`). When using geometric beams, by setting `OPTIONS3(1) = 'C'`, those lines are ignored, but we are including them in advance in order to make automatic the transition to the other approximations. Running **Bellhop** with `nearsource.env` allows us to obtain Fig.8. As shown by the figure the acoustic field is strictly confined between the propagating rays and the interference pattern of acoustic pressure becomes visible only at a large distance from the source. Similar results will be obtained using Gaussian beam bundles, when we change `OPTIONS3(1) = 'C'` to `OPTIONS3(1:2) = 'CB'` in `nearsource.env`.

A completely different pattern is revealed when we switch on the calculation of beams influence using Cartesian coordinates, by setting `OPTIONS3(1:2) = 'CC'` and `OPTIONS4(1:2) = 'MS'`. As shown in Fig.9 the acoustic field reveals an accurate pattern of interference, which now spans over the entire watercolumn. A similar result, although with minor differences in amplitude, can be obtained using ray centered coordinates (`OPTIONS3(1:2) = 'CR'`, see Fig.10). However, the structure of the pattern depends on the type of beam curvature. For instance, setting `OPTIONS4(1:2) = 'CZ'` will lead us to Fig.11.

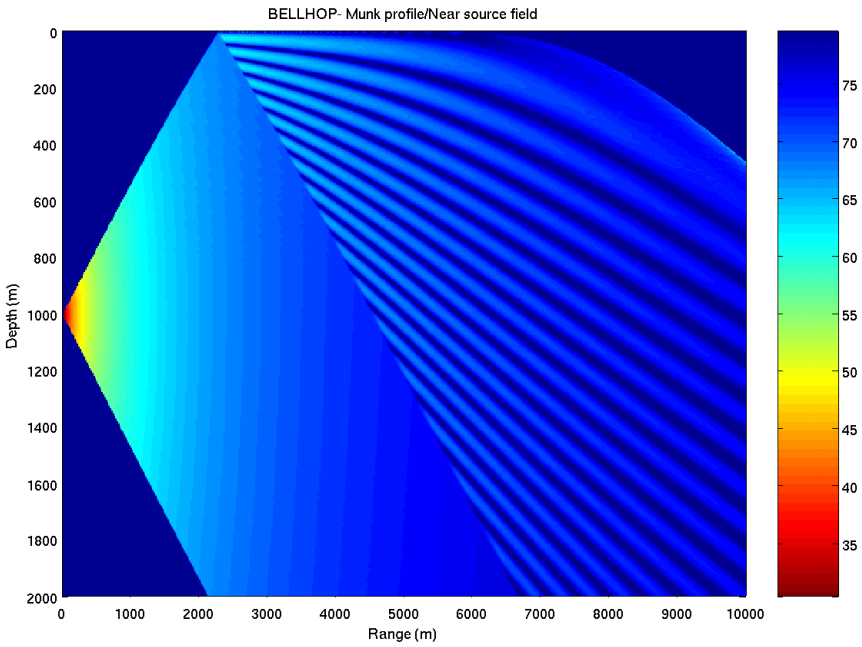


Figure 8: Coherent transmission loss calculated by **Bellhop** near the source using geometric beams.

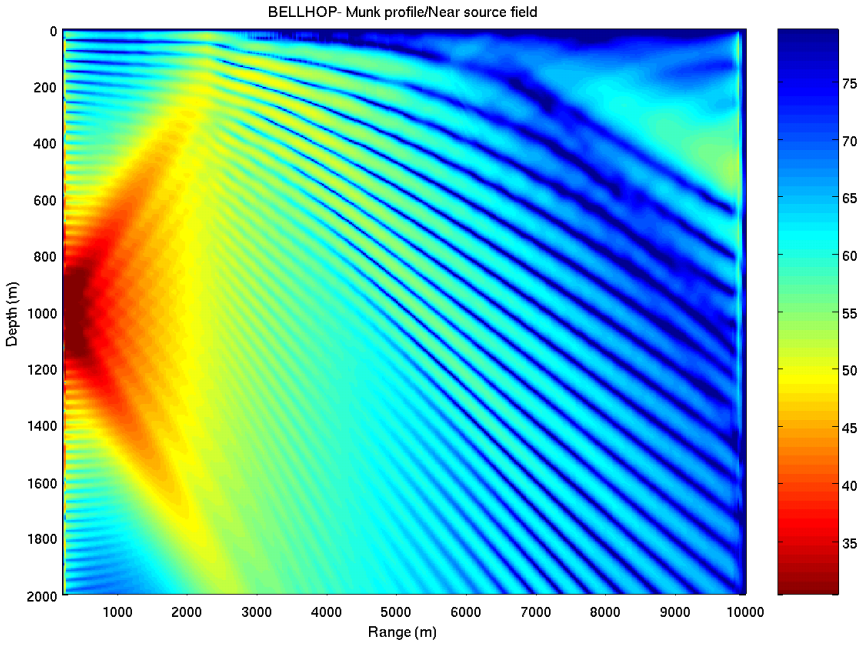


Figure 9: Coherent transmission loss calculated by **Bellhop** near the source using Cartesian coordinates to calculate the beams influence. Beam type is `'MS'`.

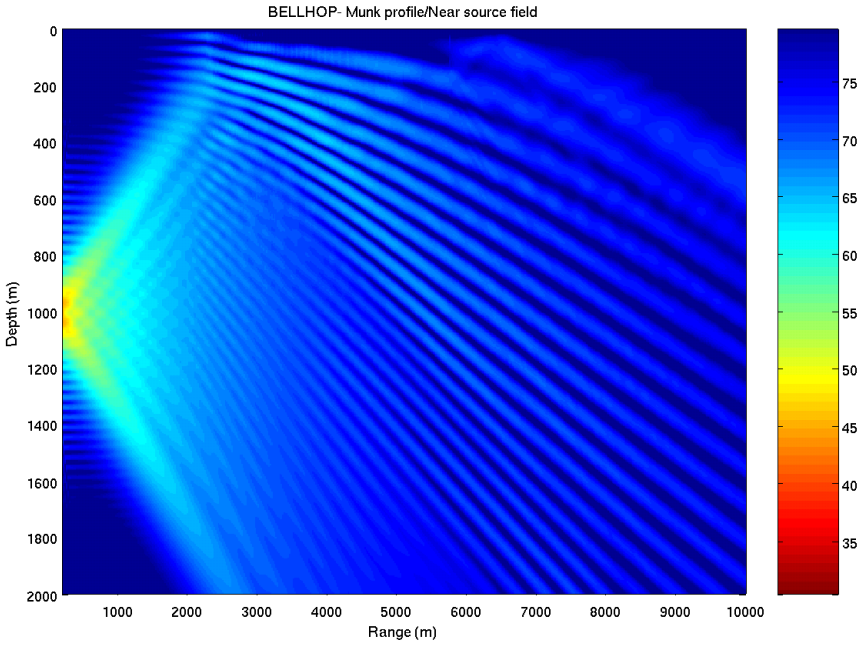


Figure 10: Coherent transmission loss calculated by **Bellhop** near the source using ray centered coordinates to calculate the beams influence. Beam type is `'MS'`.

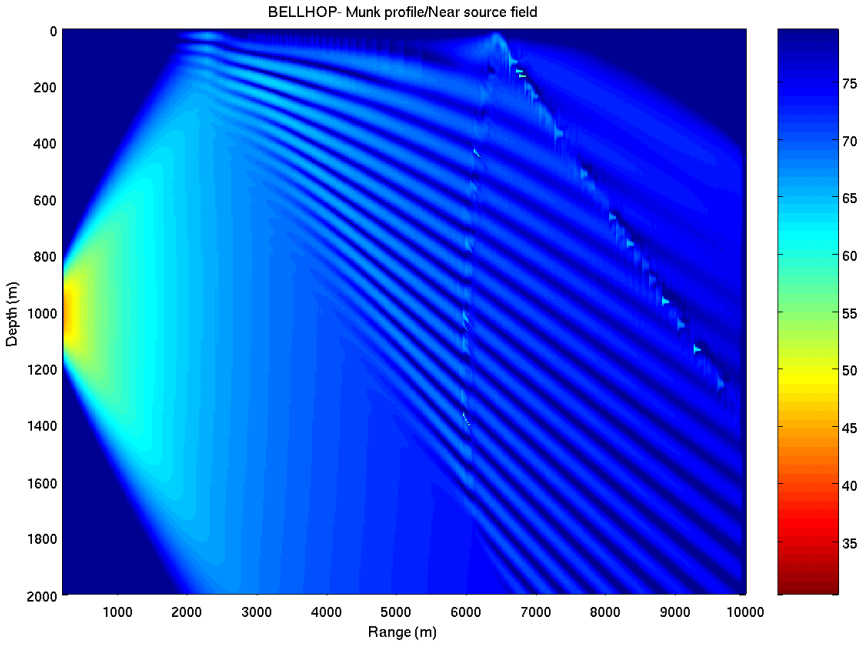


Figure 11: Coherent transmission loss calculated by **Bellhop** near the source using Cartesian coordinates to calculate the beams influence. Beam type is `'CZ'`.

6.6 Pressure and pressure components

The recent development of Vertical Sensor Arrays (VSAs) is leading to the integration of particle velocity calculations in the ray tracing models. **Bellhop** incorporates this feature through the calculation of pressure components, which are proportional to the components of particle velocity. Again, to illustrate this, let us modify `nearsource.env` as follows:

```
'Munk profile/Near source field'
50.0
1
'SVF'
51  0.0  5000.0
    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  /
'A'  0.0
 5000.0  1600.00 0.0 1.8 .8 /
```

```
1
  1000.0 /
501
    0.0 2000.0 /
501
  0.0  30.0 /
'CR'
201
 -25.0 25.0  /
100.0  5500.0  102.0,
'MS' 1.0  100.0 0,
3  5
```

Again, running **Bellhop** we can obtain Fig.12. Modifying the last line to

```
3  5 'H'
```

and

```
3  5 'V'
```

allows to obtain Fig.13 and Fig.14, respectively.

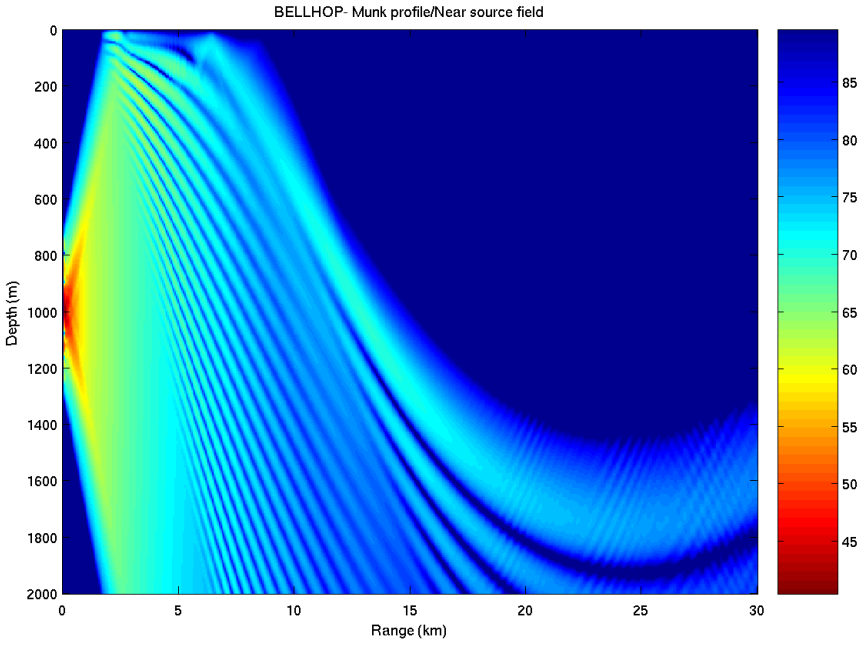


Figure 12: Coherent transmission loss calculated from acoustic pressure using ray centered coordinates.

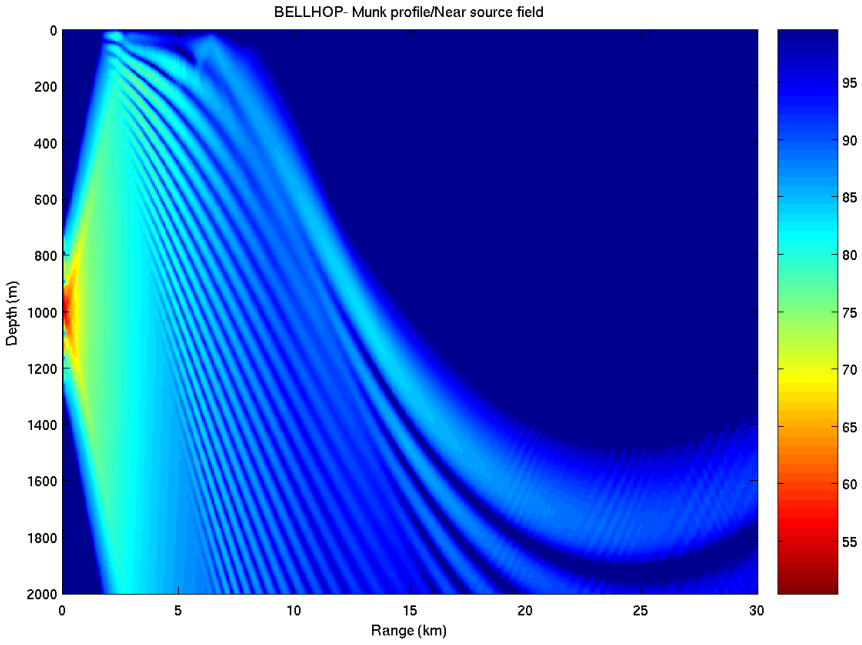


Figure 13: Coherent transmission loss calculated from the horizontal component of acoustic pressure.

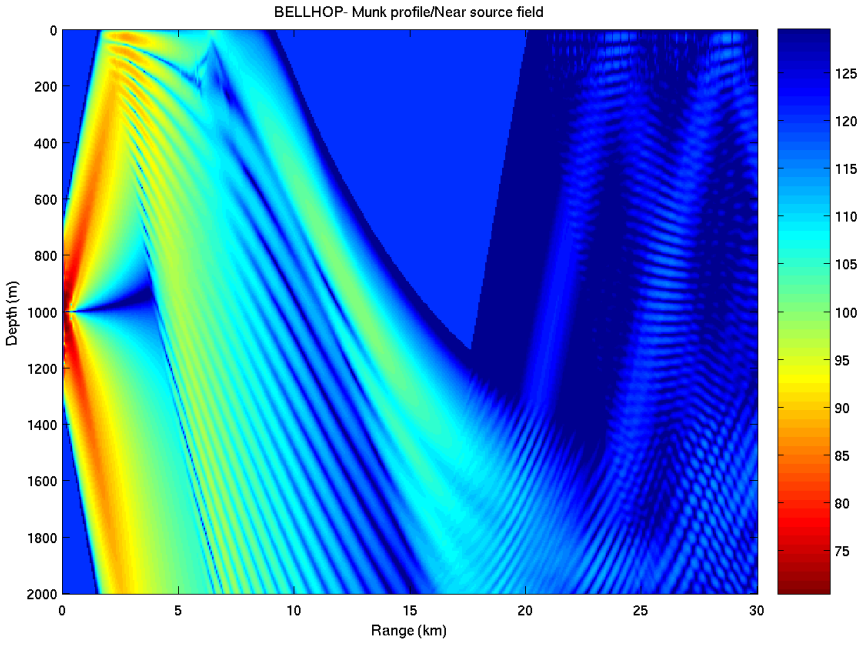


Figure 14: Coherent transmission loss calculated from the vertical component of acoustic pressure.

7 Concluding remarks

This introduction was written in the hope that it can make easier for a novice to run some preliminary tests with **Bellhop**, and also to introduce an updated manual of the model, since the documentation available is in fact rather sparse. Certainly, as the model evolves, future updates will be required. The theoretical aspects of the different approximations are not covered in detail in this document simply because the material available in the standard literature is out of date or non-existing. Any contributions to this matter (and any other contributions of interest regarding **Bellhop** applications) will be greatly appreciated.

References

[1] Porter M.B. and Bucker H.P. Gaussian beam tracing for computing ocean acoustic fields. *J. Acoust. Soc. America*, 82(4):1349–1359, 1987.

[2] Jensen F., Kuperman W., Porter M., and Schmidt H. *Computational Ocean Acoustics*. AIP Series in Modern Acoustics and Signal Processing, New York, 1994.

[3] Porter M. B. and Liu Y-C. Finite-Element Ray Tracing. In *Theoretical and Computational Acoustics, Vol. 2*, World Scientific Publishing Co., 1994.

[4] Weinberg H. and Keenan R.E. Gaussian ray bundless for modeling high-frequency propagation loss under shallow-water conditions. *J. Acoust. Soc. America*, 100(3):1421–1431, September 1996.

[5] Porter M. *The KRAKEN normal mode program*. SACLANT UNDERSEA RESEARCH (memorandum), San Bartolomeo, Italy, 1991.