CINTAL - Centro de Investigação Tecnológica do Algarve
Universidade do Algarve

SiPLAB internal report

cTraceo - User Manual

Emanuel Ey
Orlando C. Rodriguez

Rep. 01/11 - SiPLAB
30/November/2011

University of Algarve
Campus da Penha
8005-139, Faro,
Portugal

tel: +351-289800131
fax: +351-289864258
cintal@ualg.pt
http://www.cintal.ualg.pt/

intentionally blank

Work requested by	CINTAL Universidade do Algarve, Campus da Penha, 8005-139, Faro, Portugal tel: +351-289800131, cintal@ualg.pt, www.cintal.ualg.pt
Laboratory performing the work	SiPLAB - Signal Processing Laboratory Universidade do Algarve, FCT, Campus de Gambelas, 8000 Faro, Portugal tel: +351-289800949, info@siplab.fct.ualg.pt, http://www.siplab.fct.ualg.pt
Projects	SiPLAB internal report SENSOCEAN Grant Fellowship (PTDC/EEA-ELC/104561/2008)
Title	The cTraceo User Manual.
Authors	Emanuel Ey, Orlando C. Rodriguez
Date	November 30, 2011
Reference	01/11 - SiPLAB
Number of pages	53 (fifty three)
Abstract	This internal report describes the C port of the gaussian beam model TRACEO, raytracing model, from Fortran to C.
Clearance level	UNCLASSIFIED
Distribution list	CINTAL(1), SiPLAB(1), FCT(1)
Total number of copies	3 (three)

intentionally blank

Contents

List of Figures

intentionally blank

Chapter 1

Introduction

This manual describes the C port of the TRACEO Fortran-77 Raytracing Model [1] hereafter called *cTraceo*. Being feature-identical to the Fortran implementation, cTraceo's development was motivated by the need to approach the code base from a software engineering view and focus on higher performance.

The cTraceo model is a two-dimensional raytracing model that can handle irregular surfaces and complex bathymetries with range-dependent properties (including both compressional and shear velocities and attenuations), and the optional inclusion of objects in the waveguide. A set of analytical sound speed profiles, as well as both range-independent profiles and range-dependent sound speed fields are supported. The model allows for free positioning of the source, as well as of the receiver array, which can be horizontal, vertical, rectangular or be arbitrarily shaped; such options allow to model cases with unequally spaced or irregular receiver arrays. A comprehensive list of output options is available, including ray paths and amplitudes, eigenray search, arrival patterns, acoustic pressure, coherent transmission loss, and particle velocity components. For ease of use, all results are written to Matlab ".mat" files, without requiring a Matlab license to be available.

This manual is structured as follows: first, a brief overview of the raytracing theory behind cTraceo is given in Chapter 2, followed by topics of numerical issues in Chapter 3. Chapter 4 describes how to compile and install the model, while Chapter 5 describes its usage. Finally, in Chapter 6 several examples are described along with some comparisons to other acoustic propagation models.

Chapter 2

Theoretical Background

This chapter will give a brief overview of the raytracing theory on which the TRACEO and cTraceo models are based.

2.1 The Acoustic Wave Equation

For a medium with constant density the Acoustic Wave Equation is given by [2]:

$$\nabla^2 p(\vec{r}, t) - \frac{1}{c^2} \frac{\partial p(\vec{r}, t)}{\partial t^2} = 0, \tag{2.1}$$

where $p(\vec{r}, t)$ is the pressure of the acoustic wave and c is the sound speed. Applying a Fourier Transform to both sides of Eq. (2.1), one can obtain the Helmholtz Equation:

$$\left[\nabla^2 + \frac{\omega^2}{c^2}\right] P(\vec{r}, \omega) = 0. \tag{2.2}$$

Assuming a plane wave approximation to the solution of Eq. (2.2), the expression for acoustic pressure can be written as:

$$P(\vec{r}, \omega) = A(\vec{r}) e^{-i\omega\tau(\vec{r})}, \tag{2.3}$$

where $\tau(\vec{r})$ is a rapidly varying phase function known as the *Eikonal*, and $A(\vec{r})$ is a much more slowly varying envelope function incorporating the effects of geometrical spreading and various loss mechanisms [3]. By substituting Eq. (2.3) in Eq. (2.2) and by considering the following high frequency approximation [1, 2],

$$\frac{\nabla^2 A(\vec{r})}{A(\vec{r})} \ll \frac{\omega^2}{c^2}, \tag{2.4}$$

from the resulting solution's real part the *Eikonal Equation* can be obtained:

$$(\nabla\tau)^2 = \frac{1}{c^2}, \tag{2.5}$$

while from its imaginary part the *Transport Equation* can be obtained:

$$2(\nabla A \cdot \nabla \tau) + A\nabla^2 \tau = 0. \tag{2.6}$$

The following sections will sum up the solution for Eq. (2.5) and Eq. (2.6).

2.2 Solution of the Eikonal Equation

From the solution of the Eq. (2.5) one can obtain the surfaces of constant phase (wavefronts). Ray paths are orthogonal to the wavefronts and indicate the direction of energy flow [3]. The solution of the Eikonal Equation requires solving the set of equations given by [1]:

$$\frac{d}{ds}\left(\frac{1}{c}\frac{dx}{ds}\right) = \frac{\partial}{\partial x}\left(\frac{1}{c}\right), \qquad \frac{d}{ds}\left(\frac{1}{c}\frac{dy}{ds}\right) = \frac{\partial}{\partial y}\left(\frac{1}{c}\right), \qquad \frac{d}{ds}\left(\frac{1}{c}\frac{dz}{ds}\right) = \frac{\partial}{\partial z}\left(\frac{1}{c}\right), \tag{2.7}$$

where s stands for the distance traveled by the acoustic wave. By replacing the sound speed c with sound slowness $\sigma = 1/c$, and after simplifying the set of equations given by Eq. (2.7) for a waveguide with cylindrical symmetry, the following set of equations can be obtained:

$$\frac{dr}{ds} = c\sigma_r(s), \qquad \frac{dz}{ds} = c\sigma_z(s), \qquad \frac{d\sigma_r}{ds} = -\frac{1}{c^2}\frac{\partial c}{\partial r}, \qquad \frac{d\sigma_z}{ds} = -\frac{1}{c^2}\frac{\partial c}{\partial z}. \tag{2.8}$$

The initial conditions for solving Eq. (2.8) are given by [1]:

$$r(0) = r_0, \qquad z(0) = z_0, \qquad \sigma_r(0) = \frac{\cos(\theta_0)}{c_0}, \qquad \sigma_z(0) = \frac{\sin(\theta_0)}{c_0}, \tag{2.9}$$

where θ_0 is the launching angle of the ray, $[r_0, z_0]$ is the source's position and c_0 is the sound speed at the source position. By solving Eq. (2.8) together with the initial conditions (2.9) the raypaths through the waveguide can thus be determined. The next step will be determining the amplitude of the rays; this is done by solving the transport equation.

2.3 Solution of the Transport Equation

The classical solution for the ray pressure can be written as [2]:

$$P(s,\omega) = \frac{1}{4\pi}\sqrt{\frac{c(s)}{c_0}\frac{\cos(\theta_0)}{J}}e^{-i\omega\tau(s)}, \tag{2.10}$$

where θ_0 is the ray's launching angle, c_0 is the sound speed at the source, and J stands for the Jacobian; $\tau(s)$ is the time delay along the ray. Unfortunately, the classical solution breaks down in the vicinity of caustics, due to the fact that the Jacobian becomes zero [1, 2, 3]. These singularities can be addressed using the Gaussian Beam Approximation, which is discussed in the following section.

2.4 The Gaussian Beam Approximation

In the Gaussian Beam Approximation the ray is considered to be the central axis of a beam, which features a Gaussian intensity distribution along the ray normal, as shown in Fig. 2.1.

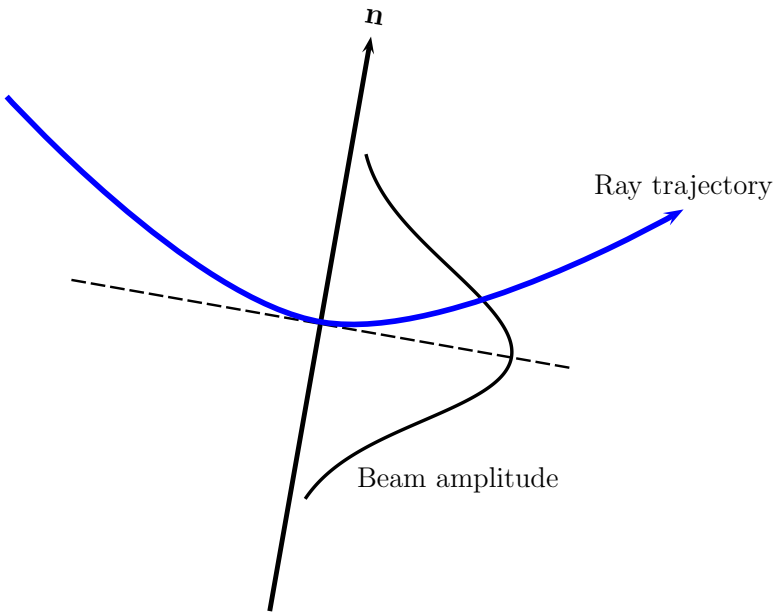


Figure 2.1: Gaussian beams: amplitude decay along the normal.

Thus a particular Gaussian beam solution for acoustic pressure, compatible with cylindrical spreading, can be written as [4]:

$$P(s,n) = \frac{1}{4\pi}\sqrt{\frac{\cos\theta_0}{c_0}\frac{c(s)}{rq(s)}}\; e^{-i\omega\left[\tau(s)+\frac{1}{2}\frac{p(s)}{q(s)}n^2\right]}, \tag{2.11}$$

where r is the range coordinate, n is the normal distance to the beam's central axis, and $p(s)$ and $q(s)$ are auxiliary functions, derived by solving a system of differential equations in the neighborhood of the ray axis [4]; $q(s)$ is also proportional to the Jacobian [2]. The approximation given by Eq. (2.11) solves the issues of the singularities at caustics, but it introduces other artifacts. Specifically, rays being returned to the source will correspond to focusing (instead of reflected) waves. And since the cylindrical approximation breaks down in the vicinity of the origin, the field of such backpropagating rays will increase as the ray aproaches the source, instead of decaying as raylength increases. A solution for the drawbacks of Eq. (2.11) is presented in the following section.

2.5 The Gaussian Beam Aproximation in Cartesian Coordinates

The typical Gaussian Beam Approximation for a waveguide with cylindrical symmetry is given by [4]:

$$P(s,n) = \frac{1}{4\pi}\sqrt{\frac{c(s)}{c_0}\frac{\cos(\theta_0)}{rq(s)}}\; e^{-i\omega\left[\tau(s)+\frac{1}{2}\frac{p(s)}{q(s)}n^2\right]}. \tag{2.12}$$

By relying on cylindrical simmetry range values are not expected to decrease, which imposes some limitations to the application of Eq. (2.12) for backscattering problems. After a carefull review of the beam expressions on the (x,z) plane, the following expression for a ray centered Gaussian Beam Approximation can be obtained [1]:

$$P(s,n) = \frac{1}{4\pi}\sqrt{\frac{c(s)}{c_0}\frac{\cos(\theta_0)}{q_\perp(s)q(s)}}\; e^{-i\omega\left[\tau(s)+\frac{1}{2}\frac{p(s)}{q(s)}n^2\right]}, \tag{2.13}$$

where n is the normal distance to the central axis of the Gaussian beam, and $p(s)$, $q(s)$ and $q_\perp(s)$ are auxilliary parameters, which determine the beamwidth along the arclength s.
In contrast to the solution for a waveguide with cylindrical simmetry as given by Eq. (2.12), $q_\perp(s)$ stands in the place of the radial coordinate r. Because the three-dimensional Gaussian beam expression is solved on (x,z) plane instead of on the (r,z) plane, backscattering can now be accomodated.
The approximation given by Eq. (2.13) is the basis for the calculation of ray amplitudes in both TRACEO and cTraceo.

2.6 Attenuation

The solution for the acoustic pressure can be written in the form:

$$P(\vec{r},\omega) = A(\vec{r})e^{-i\omega\tau(\vec{r})}, \tag{2.14}$$

where A is the ray's amplitude, which can be written as:

$$A = \frac{1}{4\pi}\sqrt{\frac{c(s)}{c_0}\frac{\cos(\theta_0)}{q_\perp(s)q(s)}}. \tag{2.15}$$

The solution given by Eq. (2.15) does not incorporate the losses introduced by volume attenuation and reflections at media interfaces. To include these loss mechanisms, A is replaced by a corrected amplitude a, given by:

$$a = A\phi_r\phi_V, \tag{2.16}$$

where ϕ_r represents the total decay due to interface reflections, and ϕ_V represents the volumetric attenuation. Both factors are described in the following sections.

2.6.1 Volumetric Attenuation

The volumetric attenuation factor ϕ_V is given by:

$$\phi_V = e^{-\alpha_T s}, \tag{2.17}$$

where s is the ray arclength and α_T is the frequency dependent Thorpe attenuation in dB/m, as given by [5]:

$$\alpha_T = \frac{40f^2}{4100 + f^2} + \frac{0.1f^2}{1 + f^2}, \tag{2.18}$$

where the frequency f is given in kHz.

2.6.2 Boundary Reflection

The total amplitude decay due to reflections at media interfaces is given by:

$$\phi_r = \begin{cases} 1 & n_r = 0 \\ \prod_{i=1}^{n_r} R_i & n_r > 0 \end{cases}, \tag{2.19}$$

where n_r is the total number of reflections at interfaces, and R_i is the complex reflection coefficient at the ith reflection. cTraceo considers four types of interfaces:

- Absorbent: the wave energy is transmitted completely to the medium beyond the interface, so $R = 0$, thus terminating the ray propagation.
- Rigid: the wave energy is reflected completely, with no phase change, so $R = 1$.
- Vacuum: the wave energy is reflected completely, with a phase change of π radians, so $R = -1$.
- Elastic: the wave energy is partially reflected, with R being a complex value and $|R| < 1$.

The reflection coefficient for an elastic medium is calculated through the following set of equations [6]:

$$R(\theta) = \frac{D(\theta)\cos\theta - 1}{D(\theta)\cos\theta + 1}, \qquad \text{and} \qquad D(\theta) = A_1\left(A_2\frac{1 - A_7}{\sqrt{1 - A_6^2}} + A_3\frac{A_7}{\sqrt{1 - A_5/2}}\right), \tag{2.20}$$

with

$$A_1 = \frac{\rho_2}{\rho_1}, \qquad A_2 = \frac{\widetilde{c}_{p_2}}{c_{p_1}}, \qquad A_3 = \frac{\widetilde{c}_{s_2}}{c_{p_1}}, \tag{2.21}$$
$$A_4 = A_3\sin\theta, \qquad A_5 = 2A_4^2, \qquad A_6 = A_2\sin\theta, \quad A_7 = 2A_5 - A5^2,$$

where

$$\widetilde{c}_{p_2} = \frac{c_{p_2}}{1 + \widetilde{\alpha}_{c_p}^2}\left(1 - i\widetilde{\alpha}_{c_p}\right), \qquad \widetilde{c}_{s_2} = \frac{c_{s_2}}{1 + \widetilde{\alpha}_{c_s}^2}\left(1 - i\widetilde{\alpha}_{c_s}\right), \tag{2.22}$$

$$\tilde{\alpha}_{c_p} = \frac{\alpha_{c_p}}{40\pi \log e}, \qquad \tilde{\alpha}_{c_s} = \frac{\alpha_{c_s}}{40\pi \log e},$$

where the attenuation values are given in dB/λ. The reflection coefficient is real when there is no attenuation and the angle of incidence is less than the critical angle

$$\theta_c = \arcsin\left(\frac{c_{p_1}}{c_{p_2}}\right).$$

2.7 Particle Velocity

The particle velocity $\vec{v}$ can be calculated in the freqency domain from the relationship [7]:

$$\vec{v} = -\frac{i}{\omega\rho}\nabla P, \tag{2.23}$$

where ω is the frequency of the propagating wave, ρ is the medium's density, P is the acoustic pressure and ∇ is the nabla operator.

Chapter 3

Numerical Issues

This chapter addresses some of the numerical issues implemented in TRACEO and cTraceo. Section 3.1 covers the methods available for eigenray search, Section 3.2 addresses Amplitudes and Arrivals and Section 3.3 focuses on the method used for calculating particle velocity components.

3.1 Eigenray Search

Although the search for eigenrays is easily defined as the task of finding rays which connect source and receiver, it is one of the most difficult raytracing problems to tackle. It is important to keep in mind that rays may be backscattered towards the source, and that even if a ray misses the receiver when propagating forward, it still has a chance of hitting the receiver on the way back to the source. Additional difficulties arise when rays can be absorbed by boundaries, or when small variations of the launching angle lead to non-linear variations of ray trajectories (a typical situation when objects are placed inside the waveguide). Two methods for eigenray search are implemented:

1. The Regula Falsi Method: If *all* rays are propagating forwards, the model can interpolate the ray depth at each array range r_i for every launching angle θ_j and generate a matrix of the form:

$$\begin{bmatrix} & r_1 & r_2 & r_3 & \dots & r_m \\ \theta_1 & z_1(\theta_1) & z_2(\theta_1) & z_3(\theta_1) & \dots & z_m(\theta_1) \\ \theta_2 & z_1(\theta_2) & z_2(\theta_2) & z_3(\theta_2) & \dots & z_m(\theta_2) \\ \theta_3 & z_1(\theta_3) & z_2(\theta_3) & z_3(\theta_3) & \dots & z_m(\theta_3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_n & z_1(\theta_n) & z_2(\theta_n) & z_3(\theta_n) & \dots & z_m(\theta_n) \end{bmatrix}; \tag{3.1}$$

   each row of the matrix allows to calculate the function

$$f(\theta) = z_h - z_i(\theta); \tag{3.2}$$

   if an eigenray exists in the interval i and $i+1$, the function $f(\theta)$ will switch signs between θ_i and θ_{i+1}; in such case the *Regula Falsi* method is used to find the zero of the function. Once a zero is found, the ray is calculated and written to the

output file as an eigenray. In order to avoid an infinite loop, the eigenray search is interrupted if the number of iterations is greater than a given threshold. Particular care is taken in order to deal with rays which for some reason do not reach the given array range. The search is interrupted if a ray which is being reflected back towards the source is detected. The *Regula Falsi* method is computationally accurate and efficient, as long as the function $z(\theta)$ can be properly computed, which won't be the case for rays which are reflected back towards the source.

2. The Proximity Method: A less accurate method of eigenray search by *Proximity* can be used as an alternative to the Regula Falsi method. Ray depth z is calculated at every hydrophone range, and for each depth (if there is more than one) the following difference is calculated:

$$|z_h - z|\,, \tag{3.3}$$

where z_h represents hydrophone depth; if the difference is less than a given threshold the ray is written to the output file as an eigenray. The accuracy of the method depends on the choice of the threshold (the lower, the better), and on the number of launching angles (the more, the better). Until a better approach is idealized for dealing with returning rays the proximity method seems to offer a reasonable compromise between accuracy and stability.

In contrast with TRACEO, cTraceo groups eigenrays together according to the hydrophones at which they arrive (more on this can be found in Section 5.3.3).

3.2 Amplitudes and Arrivals

Ray amplitudes and travel times are calculated using the same methods used for eigenray search, with the difference that instead of the entire ray, only the eigenray's last values are written to the output file. As with eigenrays, arrivals are grouped together according to the hydrophones at which they arrive (more on this can be found in Section 5.3.4).

3.3 Calculating Particle Velocity

When calculating particle velocity, both ω and ρ factors of Eq. (2.23) are ignored. This way, calculating particle velocity at a given point is simply a matter of determining the horizontal and vertical derivatives of the acoustic pressure and multiplying them by i. To approximate the vertical derivative at a specific point, the acoustic pressure is first calculated above and below of the chosen point; in a second step a parabolic interpolator is used to approximate the derivative at the chosen point. A similar strategy with three horizontal points is used to obtain the horizontal derivative at the center. The spacing between the points is defined, arbitrarily, as being equal to $\lambda/10$ unless the hydrophone spacing is less than this value, in which case the distance will be equal to half the shortest distance between hydrophones.

Chapter 4

Installation

The code of the cTraceo model is licensed under the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License*[1] and is not distributed in binary form, which requires the user to compile the program from source. The following sections will describe compiler requirements, library dependencies as well as code compilation.

4.1 Compiler Requirements

Code development of the cTraceo model was done with the Clang compiler but the code can also be compiled with the GNU C Compiler. Compilation of cTraceo requires a C compiler which supports the C99 standard because of its support for complex math. The code has been compiled and tested successfully with versions 2.8 and 2.9 of Clang. The GNU C Compiler supports C99 partially since version 4.3 and fully since version 4.5. The code has been compiled and tested successfully with versions 4.4.4 and newer of GCC on Linux, and with version 4.5.3 on Windows (using Cygwin[2]).

4.2 Library Dependencies

For ease of use, cTraceo writes all results as Matlab ".mat" files. In previous versions, the model required a Matlab license to be available for compilation and usage. Currently, a set of internal functions has come to replace the reliance on Matlab's API for matfile output. Nevertheless it is still optionally possible to revert to the old behaviour of linking the code with Matlab, should the user so desire, although this is only supported in Linux and requires extra configuration.

[1] A summary as well as the full text of the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License* can be found at `http://creativecommons.org/licenses/by-nc-sa/3.0/`

[2] Cygwin provides Windows versions of essential Linux development tools like "make" and "gcc". For more information visit `http://www.cygwin.com`.

4.3 Makefile Configuration

Before compiling the model, the user must first adapt the included *makefile* to the target system. All system dependent makefile variables are listed at the top of the file, along with examples and allowable values, as shown below:

Makefile Variable	Description
CC	Sets the command to invoke for compilation.
OS	Defines for which operating system to compile.
ARCH	Defines whether to compile for 32 bit or 64 bit systems.
USE_MATLAB	When compiling cTraceo on Linux it is possible to choose between using Matlab or the internal functions to write the result files. In case doubt, leave at "0".
MATLAB_DIR	As the name suggest, this variable should contain the path to the base directory of the system's Matlab installation. This is needed when linking with Matlab for writing ".mat" files, and is only possible on Linux. Only Relevant when `"USE_MATLAB" = 1`.
MATLAB_VERSION	This option is needed for selecting the appropriate variable types as required by different versions of Matlab. Only Relevant when `"USE_MATLAB" = 1`.

After setting the makefile variables, the method of compilation varies according to the operating system, as described in the following subsections.

4.3.1 Linux

In Linux operating systems, actual compilation is started by opening a command line, changing to the directory which contains the source files and typing:

```
$> make
```

This will compile the model and place the resulting binary called "ctraceo" in the "bin/" subdirectory. Finally, for convenience, the user should add the resulting binary's location to the system's *$PATH* environment variable.

4.3.2 Windows

To compile cTraceo on windows, Cygwin is required, so the first step will be downloading the most recent version from `www.cygwin.com` and installing it.
During the installation of Cygwin, the setup program allows choosing aditional packages. At his point the user must add *devel>gcc4* and *devel>make* to the list of selected packages before continuing with the installation. Several other packages will be installed for dependencies.
After the installation of Cygwin and it's additional packages, the folder `c:/cygwin/bin` (or whichever folder Cygwin was installed to) will need to be added to the system's PATH environment variable.
From within the Cygwin command line, the user will need to navigate to the location which contains cTraceo's source code; this will look similar to:

```
$> cd /cygdrive/c/Users/username/Documents/models/cTraceo
```

and run:

```
$> cd make
```

which will produce an executable called `ctraceo.exe` in the "bin/" subdirectory which can now be used like any other Windows executable. Finally, for convenience, the user should add the resulting binary's location to the system's *$PATH* environment variable.

Chapter 5

Using The Model

This chapter focuses on the model's usage. cTraceo requires an input file, hereafter called *infile*. After model execution two output files are produced, the first is a Matlab ".mat" file with all the results and the second is an auxilliary log file (the file with the *.log extension). The infile is described in Section 5.1, running the model is described in Section 5.2, and the matlab files described in Section 5.3.

5.1 The Input file

The input file (hereafter called `infile`) is a plain text file composed of text blocks, with each block describing a particular element of the waveguide, from top to bottom. To make the file friendlier for human reading, blocks are separated by lines of dashes ("-"). The model's distribution package includes an M-File named `wtraceoinfil.m`, located in the `M-Files/` subdirectory, which simplifies the creation of infiles from within Matlab and should be sufficient for most cases. The documentation of `wtraceoinfil.m` is included in Appendix A. Even though the examples presented in Section 6.1 make use of `wtraceoinfil.m`, the block structure of the `infile` is described in detail below:

```
Title
Source        Block
Altimetry     Block
Sound Speed   Block
Objects       Block
Bathymetry    Block
Array         Block
Output        Block
```

The following subsections describe the structure of each block.

5.1.1 Title

The `Title` consists of a single line character string, which will be written to the output file and also to the log file.

5.1.2 Source Block

The infile's source block describes the parameters of the acoustic source as follows:

```
ds                      ray step [m]
xx zx                   source coordinates [m]
xbox(1) xbox(2)         range box [m]
freqx                   source frequency [Hz]
nthtas                  number of launching angles
theta(1) theta(nthtas)  first and last launching angles [degree]
```

Optionally, the user can set *ds* to zero, which results in cTraceo using the ranges of the `rbox` to automatically choose a step size.

5.1.3 Altimetry Block

The altimetry block contains the definition of the upper interface. The first five lines of the altimetry block define several parameters for the interface while the remaining lines contain the interface's coordinates and elastic properties. The first five lines are as follows:

```
 atype  interface type
aptype  interface properties
aitype  interpolation type
  atiu  attenuation units
  nati  number of interface coordinates
```

where:

- `atype` determines the interface type and can be one of the following characters:

`'A'`	absorbent interface
`'E'`	elastic interface
`'R'`	rigid interface
`'V'`	vacuum beyond interface

- `aptype` specifies if the interface's properties are homogeneous or not, and can be one of the following characters:

'H'	homogeneous interface
'N'	non-homogeneous interface

- `aitype` determines the type of interpolation to be used and can be one of the following strings:

'FL'	flat interface
'SL'	flat interface with a slope
'2P'	piecewise linear interpolation
'4P'	piecewise cubic interpolation

- `atiu` specifies the shear and compressional attenuation units, and can be one of the following characters:

'F'	dB/kHz
'M'	dB/meter
'N'	dB/neper
'Q'	Q factor
'W'	dB/λ

  These are the same units as used by Bellhop and KRAKEN, for specific details see [8].

- `nati` indicates the number of interface coordinates to read from the subsequent lines.

The remaning lines of the altimetry block depend on the value of `aptype`.
For homogeneous interface properties (`aptype = 'H'`) the first line contains the elastic properties of the medium beyond the interface (compressional wave speed, shear wave speed, density, compressional attenuation and shear attenuation), followed by lines containing the range and depth coordinates of the interface:

```
cpati(1) csati(1) rhoati(1) apati(1) asati(1)
rati(1) zati(1)
rati(2) zati(2)
rati(3) zati(3)
...
rati(nati) zati(nati)
```

Alternatively, for non-homogeneuous interface properties (`aptype = 'N'`), each subsequent line will contain the range and depth coordinates, as well as the elastic properties along the surface:

```
rati(1) zati(1) cpati(1) csati(1) rhoati(1) apati(1) asati(1)
rati(2) zati(2) cpati(2) csati(2) rhoati(2) apati(2) asati(2)
rati(3) zati(3) cpati(3) csati(3) rhoati(3) apati(3) asati(3)
...
rati(nati) zati(nati) cpati(nati) ...
```

5.1.4 Sound Speed Block

The sound speed block contains the definition of the sound speed properties, as shown below:

```
cdist    type of sound speed distribution
cclass   class of sound speed
nr0 nz0  number or points in range, number of points in depth
```

where:

- `cdist` determines the type of sound speed distribution, and can be one of the following strings:

`'c(z,z)'`	sound speed profile	$c = c(z)$
`'c(r,z)'`	sound speed field	$c = c(r, z)$

- `cclass` specifies the class of sound speed, and can be one of the following strings:

`'ISOV'`	isovelocity	profile
`'LINP'`	linear	profile
`'PARP'`	parabolic	profile
`'EXPP'`	exponential	profile
`'N2LP'`	n^2-linear	profile
`'ISQP'`	inverse-square gradient	profile
`'MUNK'`	Munk	profile
`'TABL'`	tabulated	profile

All but the last value of `cclass` describe analytical profiles, whose derivatives can be calculated explicitely. For the analytical sound speed profiles (`cdist` = `'c(z,z)'` and `cclass` ≠ `'TABL'`) all range derivatives are zero, while depth derivatives are calculated depending on the value of `cclass`. For tabulated sound speed fields (`cdist` = `'c(r,z)'` and `cclass` = `'TABL'`) both range and depth derivatives are calculated using a bi-dimensional barycentric cubic interpolator, on the grid of `nr0`×`nz0` points.

The parameters of the analytical profiles can be specified through a simple list of the form:

```
z0(1)  c0(1)
z0(2)  c0(2)
```

The following is a description of each analytical profile:

- Isovelocity profile:

$$c(z) = c_0 = \text{ constant } , \ \frac{dc}{dz} = 0 \ , \ \frac{d^2c}{dz^2} = 0 \ ;$$

  therefore, only the value `c0(1)` is used during the calculations.

- Linear profile:

$$c(z) = c_0 + k\,(z - z_0) \;\;,\;\; \frac{dc}{dz} = k \;,\;\; \frac{d^2c}{dz^2} = 0 \;;$$

the parameters are calculated as $z_0 =$ `z(1)`, $c_0 =$ `c(1)` and

$$k = \frac{\texttt{c(2)-c(1)}}{\texttt{z(2)-z(1)}} \;.$$

- Parabolic profile:

$$c(z) = c_0 + k\,(z - z_0)^2 \;\;,\;\; \frac{dc}{dz} = 2k\,(z - z_0) \;\;,\;\; \frac{d^2c}{dz^2} = 2k \;;$$

the parameters are calculated as $z_0 =$ `z(1)`, $c_0 =$ `c(1)` and

$$k = \frac{\texttt{c(2)-c(1)}}{\texttt{(z(2)-z(1))}^2} \;.$$

- Exponential profile:

$$c(z) = c_0 e^{-k(z-z_0)} \;,$$

$$\frac{dc}{dz} = -kc_0 e^{-k(z-z_0)} \;,\;\; \frac{d^2c}{dz^2} = k^2 c_0 e^{-k(z-z_0)} \;;$$

the parameters are calculated as $z_0 =$ `z(1)`, $c_0 =$ `c(1)` and

$$k = \frac{1}{\texttt{z(2)-z(1)}} \ln\left[\frac{\texttt{c(1)}}{\texttt{c(2)}}\right] \;.$$

- n^2-linear profile:

$$c(z) = \frac{c_0}{\left[1 + k\,(z - z_0)\right]^{1/2}} \;,$$

$$\frac{dc}{dz} = \frac{-kc_0}{2\left[1 + k\,(z - z_0)\right]^{3/2}} \;,\;\; \frac{d^2c}{dz^2} = \frac{3k^2c_0}{4\left[1 + k\,(z - z_0)\right]^{5/2}} \;;$$

the parameters are calculated as $z_0 =$ `z(1)`, $c_0 =$ `c(1)` and

$$k = \frac{1}{\texttt{z(2)-z(1)}} \left[\left(\frac{\texttt{c(1)}}{\texttt{c(2)}}\right)^2 - 1\right] \;.$$

- Inverse-square gradient profile:

$$c(z) = c_0 \left\{1 + \frac{k\,(z - z_0)}{\left[1 + k^2\,(z - z_0)^2\right]^{1/2}}\right\} \;,$$

$$\frac{dc}{dz} = \frac{kc_0}{2\left[1+k^2\left(z-z_0\right)^2\right]^{3/2}} \ , \ \frac{d^2c}{dz^2} = \frac{-3k^3c_0\left(z-z_0\right)}{\left[1+k^2\left(z-z_0\right)^2\right]^{5/2}} \ ;$$

the parameters are calculated as $z_0 =$ `z(1)`, $c_0 =$ `c(1)` and

$$k = \frac{1}{\texttt{z(2)-z(1)}}\sqrt{\frac{a}{1-a}} \ ,$$

with

$$a = \left[\frac{\texttt{c(1)}}{\texttt{c(2)}} - 1\right]^2 \ .$$

- Munk profile:

$$c(z) = c_0\left[1+\varepsilon(\eta + e^{-\eta} - 1)\right] \ ,$$

$$\frac{dc}{dz} = \frac{2\varepsilon c_1}{B}(1-e^{-\eta}) \ , \ \frac{d^2c}{dz^2} = \frac{4\varepsilon c_1}{B^2}e^{-\eta} \ ,$$

with the parameters $\varepsilon = 7,4 \times 10^{-3}$, $\eta = 2\left(z-z_0\right)/B$, $B =$ 1,3 km, (z_0 represents the depth of the channel axis and c_0 the corresponding value of sound speed); the parameters are calculated as $z_0 =$ `z(1)` and $c_0 =$ `c(1)`. The remaining values are ignored.

A custom sound speed profile can be specified through the combination `cdist = 'c(z,z)'` and `cclass = 'TABL'`. The tabulated sound speed profile is then specified as follows:

```
z0(1)  c0(1)
z0(2)  c0(2)
...
z0(nz0)  c0(nz0)
```

Alternatively, a sound speed field can be specified through the combination of `cdist = 'c(r,z)'` and `cclass = 'TABL'`, to be then specified as follows:

```
r0(1)  r0(2)  r0(3)  ...  r0(nr0)
z0(1)  z0(2)  z0(3)  ...  z0(nz0)
c(1,1)  c(1,2)  ...  c(1,nr0)
c(2,1)  c(2,2)  ...  c(2,nr0)
c(3,1)  c(3,2)  ...  c(3,nr0)
...
c(nz0,1)  c(nz0,2)  ...  c(nz0,nr0)
```

When specifying the sound speed profile or field it is highly recommended to use an evenly spaced grid, avoiding vertical segments where a smooth variation is followed by an isovelocity layer. Including such segments introduces unrealistic artifacts induced by an inaccurate calculation of sound speed gradients.

5.1.5 Object Block

The `Object Block` begins with a single line containing the number of objects present in the waveguide:

```
nobj
```

If `nobj` = 0, no objects are defined and the remainder of the block is empty. All of the remaining lines are only required for `nobj` > 0. When objects are present, the next line specifies the method of interpolation to be applied to the boundaries of *all* objects.

```
oitype
```

Allowable values of `oitype` are `'2P'`or `'4P'` -no other types are allowed.
Next, each object shall be specified through a structure, where the first four lines specify the object's properties, and the following lines describe the object's contour. The object's upper and lower boundaries are sampled along a common range interval. The structure looks like the following:

```
otype                                    object type
obju                                     attenuation units
no                                       number of coordinates
ocp(i) ocs(i) orho(i) oap(i) oas(i)      elastic properties
ro(1) zdn(1) zup(1)
ro(2) zdn(2) zup(2)
...
ro(no) zdn(no) zup(no)
```

`otype` and `obju` may take the same values as those indicated for the `Altimetry Block` in Section 5.1.3.

5.1.6 Bathymetry Block

The Bathymetry Block has the same structure as the `Altimetry Block`, described in Section 5.1.3.

5.1.7 Array Block

This block contains the specifications for the receiver array and is made up of the following four lines:

```
artype                  array type
nra nza                 number of elements in range and depth
r(1) r(2) .... r(nra)   hydrophone ranges
z(1) z(2) .... z(nza)   hydrophone depths
```

Where `artype` can correspond to one of the following strings:

`'RRY'`	Rectangular	arRaY
`'HRY'`	Horizontal	arRaY
`'VRY'`	Vertical	arRaY
`'LRY'`	Linear	arRaY

Note that when `artype` = `'LRY'` it is required that `nra` = `nza`.

5.1.8 Output Block

The Output block is the last block in the `infile` and determines which output the model shall generate and write to the output file. The output block has the following structure

```
outype                  output type
miss                    eigenray parameter
```

The option `outype` defines the type of output and can correspond to one of the following strings:

`'RCO'`	output Ray COordinates;
`'ARI'`	output All Ray information;
`'ERF'`	output Eigenrays (use Regula Falsi);
`'EPR'`	output Eigenrays (use PRoximity method);
`'ADR'`	output Amplitudes and Delays (use Regula falsi);
`'ADP'`	output Amplitudes and Delays (use Proximity method);
`'CPR'`	output Coherent acoustic PRessure;
`'CTL'`	output Coherent Transmission Loss;
`'PVL'`	output coherent Particle VeLocity;
`'PAV'`	output Coherent acoustic Pressure And Particle velocity.

The `miss` parameter is used as a threshold to find eigenrays and to calculate arrivals. It specifies the distance in meters at which a ray passing a hydrophone shall be considered as an eigenray.

5.2 Running The Model

The only command line argument required by cTraceo is the name of the input file, without its extension:

```
$> ctraceo filename
```

The following section will describe the output files generated by cTraceo

5.3 Output Matlab Files

The model outputs the resutls as Matlab ".mat" files. For every input file one output file is written to disk. The name of the output file depends on the type calculation performed by the model, as described in the following subsections.

5.3.1 Ray Coordinates

Ray coordinates are calculated by setting `outype = 'RCO'`. When calculating ray coordinates, only the Eikonal Equation is solved and the resulting ray coordinates are written to "rco.mat", which contains the following variables:

- `rco.mat`
  - `caseTitle` ..... The title given to this case, as defined in the input file.
  - `rays` ..... An array of structures with size *N*, with N being the number of rays calculated. Each element of this array contains the information corresponding to one of the launched rays.
    - `theta` ..... A scalar representing the ray's launching angle in degrees.
    - `r` ..... A vector containing the ray's range coordinates in meters.
    - `z` ..... A vector containing the ray's depth coordinates in meters.
  - `thetas` ..... A vector containing the launching angles of the calculated rays.

5.3.2 All Ray Information

All ray information is calculated by setting `outype = 'ARI'`. When calculating all ray information, the Eikonal as well as the Dynamic Equations are solved. The results are written to "ari.mat", which contains the following variables:

- `ari.mat`
  - `caseTitle` ..... The title given to this case, as defined in the input file.
  - `rays(n)` ..... An array of structures with size N, with N being the number of rays calculated. Each element of this array contains the information corresponding to one of the launched rays.
    - `theta` ..... A scalar representing the ray's launching angle in degrees.
    - `r` ..... A vector containing the ray's range coordinates in meters.
    - `z` ..... A vector containing the ray's depth coordinates in meters.
    - `tau` ..... A vector containing the propagation time in seconds along the ray's path.
    - `amp` ..... A vector of complex values representing the amplitude component of Equation 2.3.
    - `iReturns` ..... A boolean indicating whether the ray has inverted its horizontal direction (for instance by reflection).
    - `nSurRefl` ..... A scalar containing the ray's number of surface reflections.
    - `nBotRefl` ..... A scalar containing the ray's number of bottom reflections.
    - `nObjRefl` ..... A scalar containing the ray's number of object reflections.
    - `nRefrac` ..... A scalar containing the number of refraction points.
    - `refrac_r` ..... A vector containing the range coordinates in meters of the ray's refraction points, if any exist.
    - `refrac_z` ..... A vector containing the depth coordinates in meters of the ray's refraction points, if any exist.
  - `thetas` ..... A vector containing the launching angles of the calculated rays.

5.3.3 Eigenrays

As seen in Section 5.1, there are two methods for determining a hydrophone's eigenrays: by Proximity (`outype = 'EPR'`) and by Regula Falsi (`outype = 'ERF'`). Both method's results are written to a file named "eig.mat" and contain the following variables:

- `eig.mat`
  - `eigenrays(r,z)` ... An array of structures with size `dimR` by `dimZ`, where `dimR` and `dimZ` are the hydrophone's horizontal and vertical index within the hydrophone array. Each element contains the eigenrays arriving at hydrophone (r,z).
    - `nEigenrays` .... A scalar containing the number of eigenrays which arrived at hydrophone (r,z).
    - `rHyd` ..... The range coordinate in meters of hydrophone (r,z).
    - `zHyd` ..... The depth coordinate in meters of hydrophone (r,z).
    - `eigenray(n)` ... An array of structures with size nEigenrays. Each element of this array contains the information corresponding to one of the rays which arrived at hydrophone (r,z).
      - `theta` ..... The ray's launching angle in degrees.
      - `r` ..... A vector containing the eigenray's range coordinates.
      - `z` ..... A vector containing the eigenray's depth coordinates.
      - `tau` ..... A vector containing the propagation time in seconds along the ray's path.
      - `amp` ..... A vector of complex values representing the amplitude component of Equation 2.3.
      - `iReturns` ... A boolean indicating whether the ray has inverted its horizontal direction (for instance by reflection).
      - `nSurRefl` ... A scalar containing the ray's number of surface reflections.
      - `nBotRefl` ... A scalar containing the ray's number of bottom reflections.
      - `nObjRefl` ... A scalar containing the ray's number of object reflections.
      - `nRefrac` .... A scalar containing the number of refraction points.
      - `refrac_r` ... A vector containing the range coordinates in meters of the ray's refraction points, if any exist.
      - `refrac_z` ... A vector containing the depth coordinates in meters of the ray's refraction points, if any exist.

- `arrayR` ..... A vector containing the range coordinates of the hydrophone array's elements.
- `arrayZ` ..... A vector containing the depth coordinates of the hydrophone array's elements.
- `caseTitle` ..... The title given to this case, as defined in the input file.
- `thetas` ..... A vector containing the launching angles of the calculated rays.

5.3.4 Amplitudes And Delays

As seen in Section 5.1, cTraceo provides two methods for determinining amplitudes and arrivals: by proximity (`outype = 'ADR'`) and by Regula Falsi (`outype = 'ADR'`). Both method's results are written to a file named "aad.mat" and contain the following variables:

- `aad.mat`
  - `arrayR` ..... A vector containing the range coordinates of the hydrophone array's elements.
  - `arrayZ` ..... A vector containing the depth coordinates of the hydrophone array's elements.
  - `caseTitle` ..... The title given to this case, as defined in the input file.
  - `arrivals(r,z)` .... An array of structures with size `dimR` by `dimZ`, where `dimR` and `dimZ` are the hydrophone's horizontal and vertical index within the hydrophone array. Each element contains the arrivals at hydrophone (r,z).
    - `nArrivals` ..... A scalar containing the number of arrivals at hydrophone (r,z).
    - `rHyd` ..... The range coordinate in meters of hydrophone (r,z).
    - `zHyd` ..... The depth coordinate in meters of hydrophone (r,z).
    - `arrival(n)` .... An array of strutures with size nArrivals. Each element of this array contains the information corresponding to one of the arrivals at hydrophone (r,z).
      - `theta` ..... The ray's launching angle in degrees.
      - `r` ..... A scalar containing the eigenray's final range coordinate.
      - `z` ..... A scalar containing the eigenray's final depth coordinate.
      - `tau` ..... A scalar containing the eigenray's total propagation time in seconds.
      - `amp` ..... A complex scalar representing the final amplitude component of Equation 2.3.
  - `maxNumArrivals` ... The maximum number of arrivals at any of the hydrophones.
  - `sourceZ` ..... The depth in meters of the acoustic source.
  - `thetas` ..... A vector containing the launching angles of the calculated rays.

5.3.5 Coherent Acoustic Pressure

Coherent Acoustic Pressure is calculated by setting `outype = 'CPR'`, with the results written to "cpr.mat", which has the following structure:

`cpr.mat`

- `arrayR` ..... A vector containing the range coordinates of the hydrophone array's elements.
- `arrayZ` ..... A vector containing the depth coordinates of the hydrophone array's elements.
- `caseTitle` ..... The title given to this case, as defined in the input file.
- `p` ..... An array of complex scalars with size `dimR` by `dimZ` containing the acoustic pressure at the hydrophones, where `dimR` and `dimZ` are the hydrophone arrays's horizontal and vertical sizes.
- `thetas` ..... A vector containing the launching angles of the calculated rays.

5.3.6 Coherent Transmission Loss

Coherent Transmission Loss is calculated by setting `outype = 'CTL'`, with the results written to "ctl.mat", which has the following structure:

`ctl.mat`

- `arrayR` ..... A vector containing the range coordinates of the hydrophone array's elements.
- `arrayZ` ..... A vector containing the depth coordinates of the hydrophone array's elements.
- `caseTitle` ..... The title given to this case, as defined in the input file.
- `thetas` ..... A vector containing the launching angles of the calculated rays.
- `tl` ..... An array of scalars with size `dimR` by `dimZ` containing the transmission loss at the hydrophones in dB, where `dimR` and `dimZ` are the hydrophone arrays's horizontal and vertical sizes.

5.3.7 Particle Velocity

Particle Velocity is calculated by setting `outype = 'PVL'`, with the results written to "pvl.mat", which has the following structure:

`pvl.mat`

- `arrayR` ..... A vector containing the range coordinates of the hydrophone array's elements.
- `arrayZ` ..... A vector containing the depth coordinates of the hydrophone array's elements.
- `caseTitle` ..... The title given to this case, as defined in the input file.
- `thetas` ..... A vector containing the launching angles of the calculated rays.
- `u` ..... An array of scalars with size `dimR` by `dimZ` containing the horizontal component of particle velocity at the hydrophones in dB, where `dimR` and `dimZ` are the hydrophone arrays's horizontal and vertical sizes.
- `w` ..... An array of scalars with size `dimR` by `dimZ` containing the vertical component of particle velocity at the hydrophones in dB, where `dimR` and `dimZ` are the hydrophone arrays's horizontal and vertical sizes.

5.3.8 Coherent Acoustic Pressure And Particle Velocity

Coherent Acoustic Pressure and Particle Velocity are calculated by setting `outype = 'PAV'`, with the results written to "pav.mat", which has the following structure:

`pav.mat`

- `arrayR` ..... A vector containing the range coordinates of the hydrophone array's elements.
- `arrayZ` ..... A vector containing the depth coordinates of the hydrophone array's elements.
- `caseTitle` ..... The title given to this case, as defined in the input file.
- `p` ..... An array of complex scalars with size `dimR` by `dimZ` containing the acoustic pressure at the hydrophones, where `dimR` and `dimZ` are the hydrophone arrays's horizontal and vertical sizes.
- `thetas` ..... A vector containing the launching angles of the calculated rays.
- `u` ..... An array of scalars with size `dimR` by `dimZ` containing the horizontal component of particle velocity at the hydrophones in dB, where `dimR` and `dimZ` are the hydrophone arrays's horizontal and vertical sizes.
- `w` ..... An array of scalars with size `dimR` by `dimZ` containing the vertical component of particle velocity at the hydrophones in dB, where `dimR` and `dimZ` are the hydrophone arrays's horizontal and vertical sizes.

Chapter 6

Model Results

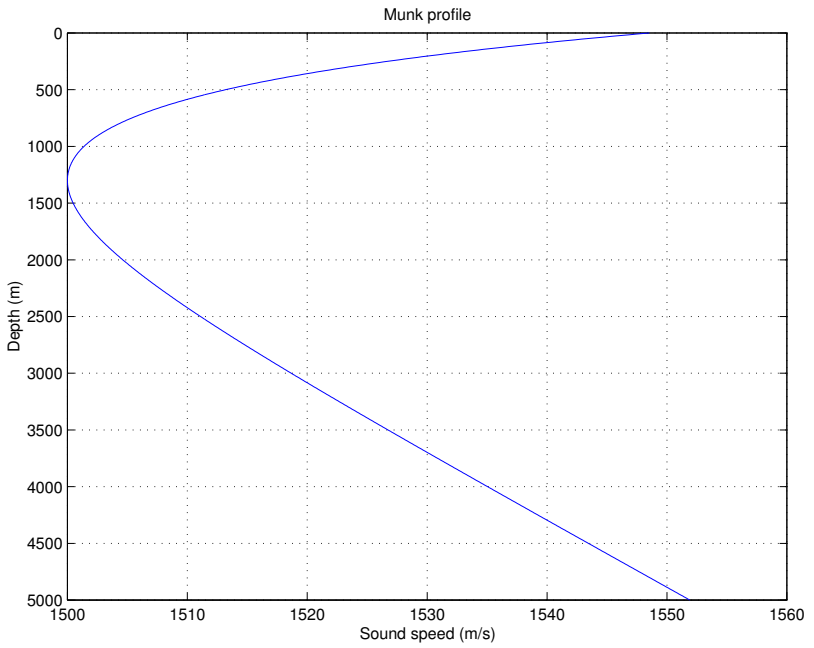
This chapter presents results obtained with the cTraceo model, first with some examples in Section 6.1, comparisons with other models in Section 6.2 and performance issues in Section 6.3. All M-files mentioned in this Chapter are included along with the source code.

6.1 Examples

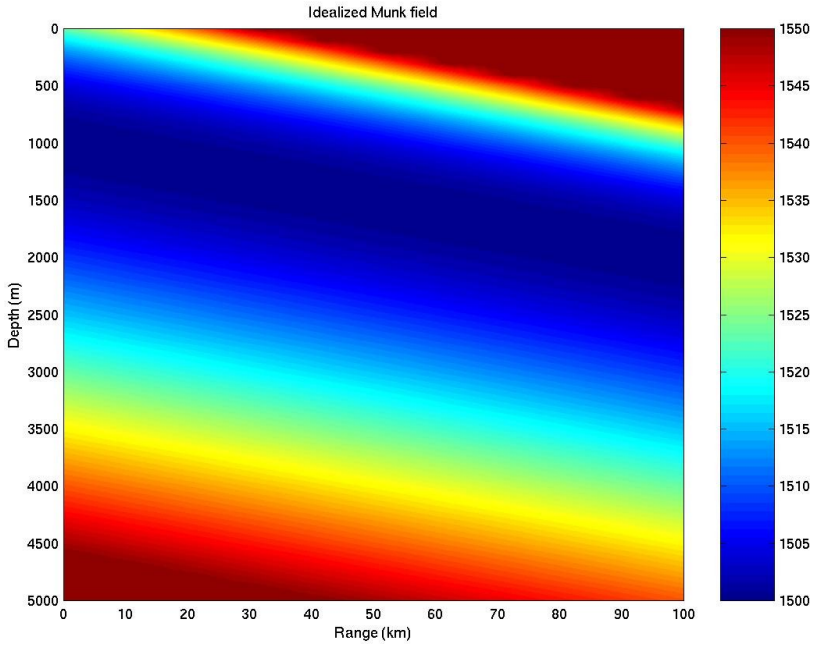
The model is distributed along with a significant number of different examples, located within the `examples/` subdirectory. To showcase some of cTraceo's capabilities, Section 6.1.1 presents deep water calculations for a Munk profile and Munk field. Section 6.1.2 presents shallow water examples, beginning with pekeris waveguide followed by an ultra-shallow range-dependent lake waveguide with an object, caracteristic of the Sletvik experimental site.

6.1.1 Deep Water

The deep water examples consider a waveguide with variable boundaries, namely a sinusoidal surface and a Gaussian sea mountain. Following the classical case described in [4], the acoustic source is located at a depth of 1000 m, and the maximum range corresponds to 100 km. The first example considers the Munk sound speed profile shown in Fig. 6.1(a), while the second exapmple considers a Munk sound speed field, with a channel depth deepening from 1000 m to 2000 as shown in Fig. 6.1(b). Ray paths for the first case are obtained by running the M-File `varbounds_ssp_rco.m` and the corresponding result is shown in Fig. 6.2(a). Similarly, Fig. 6.2(b) shows the ray paths for the second example, as obtained from `varbounds_ssf_rco.m`. The obtained figures showcase the model's capacity to handle range-dependent boundaries in combination with sound speed profiles and/or sound speed fields.

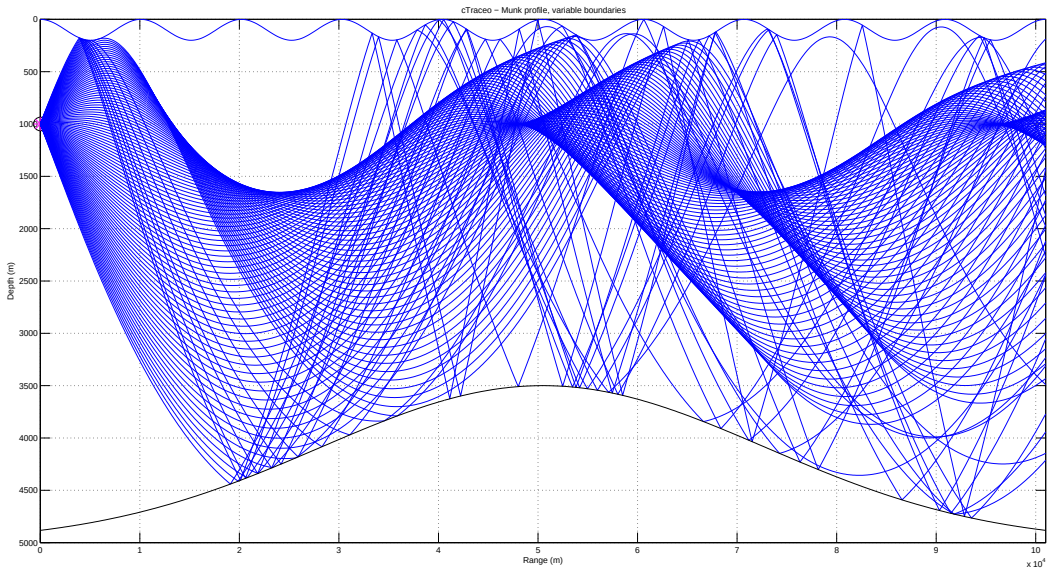


(a)

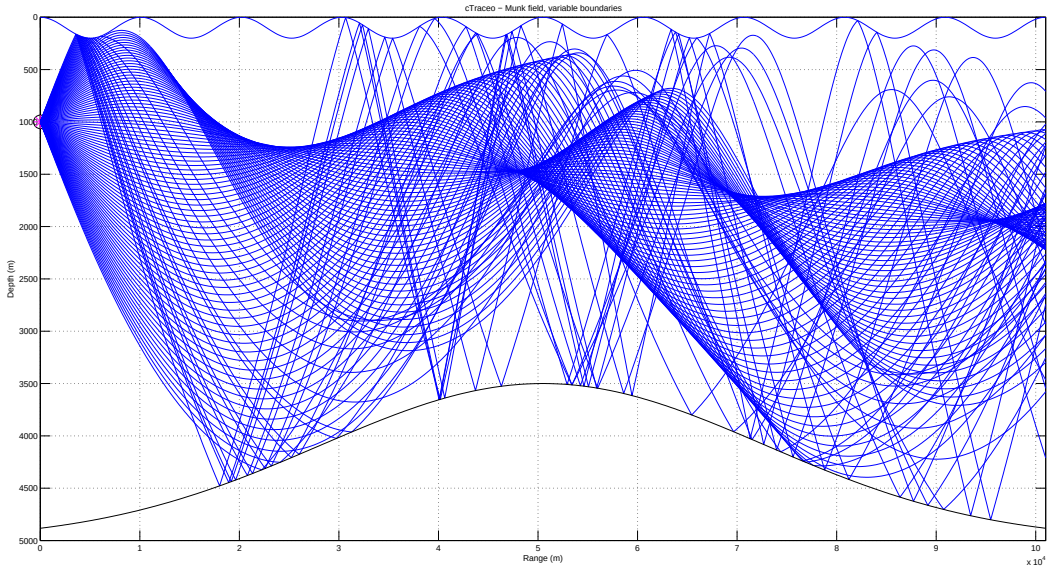


(b)

Figure 6.1: (a) Munk sound speed profile as used in `varbounds_ssp_rco.m`; (b) idealized Munk sound speed field as used in `varbounds_ssf_rco.m`.



(a)



(b)

Figure 6.2: cTraceo model result obtained from: (a) `varbounds_ssp_rco.m`; (b) `varbounds_ssf_rco.m`.

6.1.2 Shallow Water

The first shallow water example is a pekeris waveguide with flat boundaries, as shown in Fig. 6.3. The result shown in Fig. 6.4 ilustrates eigenrays which where found by the proximity method. The figure reveals that this method generates groups of eigenrays which propagate along similar paths. The Regula Falsi method for eigenray search may be used to avoid such grouping of rays. For instance, the results shown in Fig. 6.5, calculated by running `pekeris_erf.m` produced the expected eigenrays without such groups. The Regula Falsi method was also used in `pekeris_adr.m` to generate the amplitudes and delays shown in Fig. 6.6, which reveals the formation of quadruplet groups.

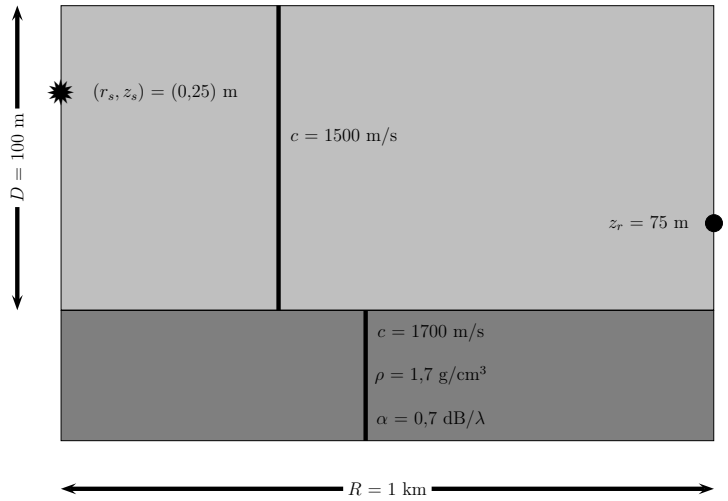


Figure 6.3: The flat pekeris waveguide used for shallow water examples.

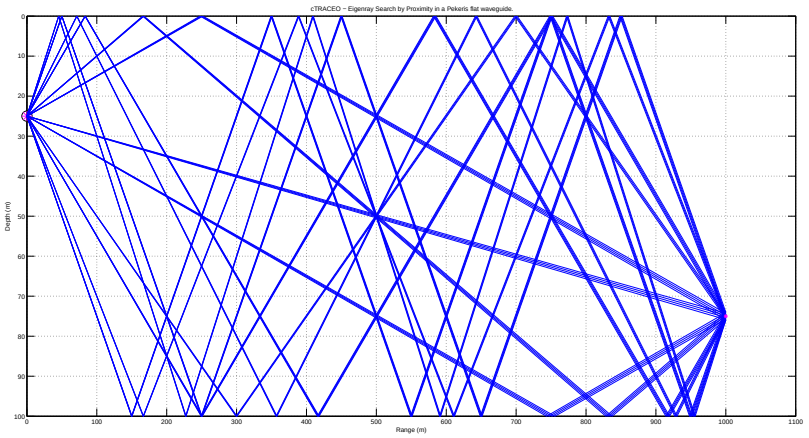


Figure 6.4: Eigenrays obtained using the Proximity Method from `pekeris_epr.m`.

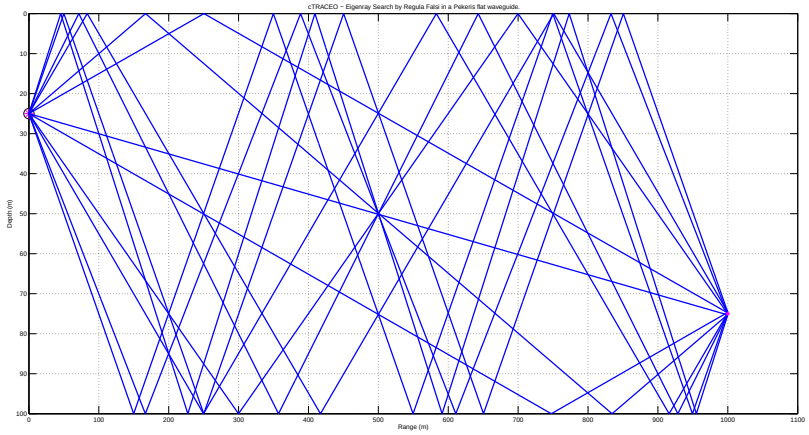


Figure 6.5: Eigenrays obtained using the Regula Falsi Method from `pekeris_erf.m`.

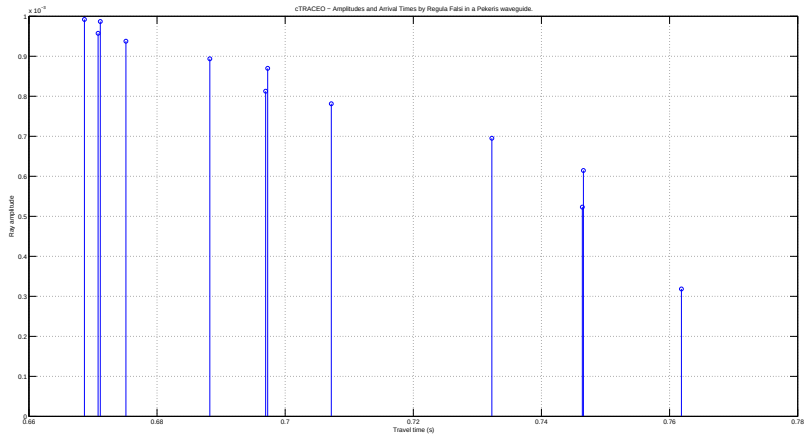
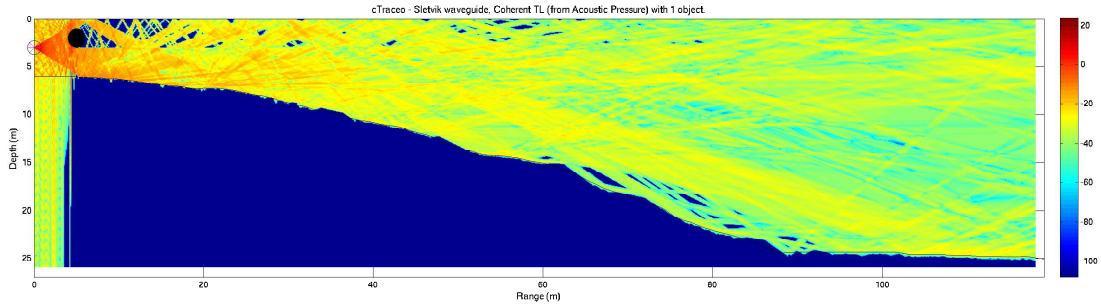


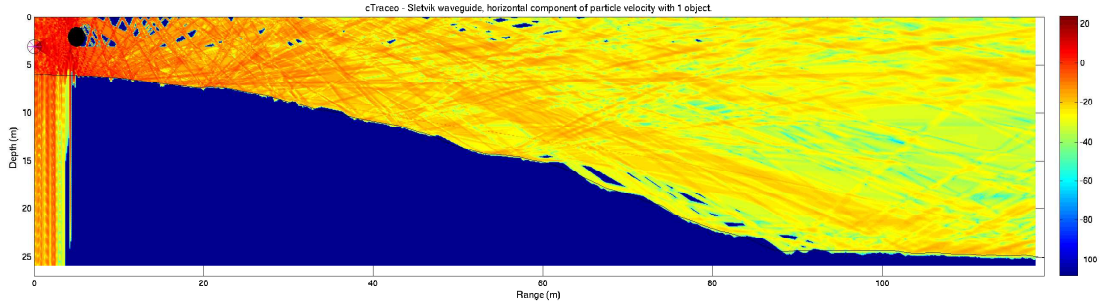
Figure 6.6: Arrival pattern obtained using the Proximity Method from `pekeris_adr.m`.

Results for the Sletvik waveguide[1] are shown in Fig. 6.7, as obtained from `sletvik_pav.m`. This showcases cTraceo's capability to do field calculations of acoustic pressure and particle velocity components when an object is included in the waveguide.

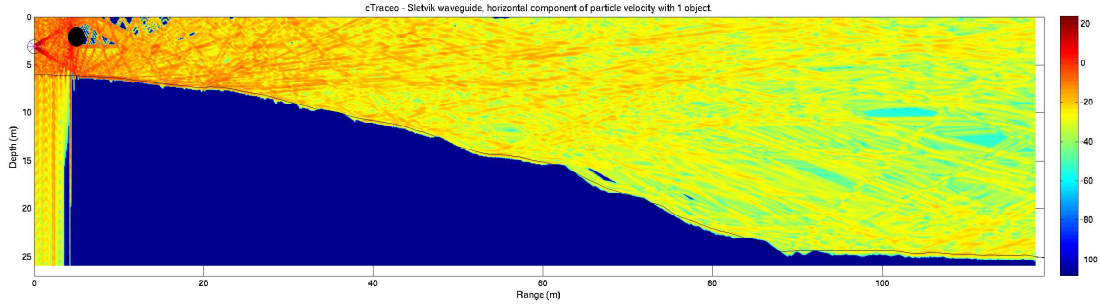
[1]Norwegian University of Science and Technology, Sletvik Field Station: http://www.ntnu.edu/biology/sletvik-field-station



(a)



(b)



(c)

Figure 6.7: cTraceo model resultsobtained from `sletvik_pav.m`: (a) Acoustic pressure; (b) horizontal component of particle velocity; (c) vertical compoent of particle velocity.

6.2 Model Comparisons

This section presents some comparisons between cTraceo results and results obtained by other models. First a comparison with the Bellhop raytracing model and the KRAKEN normal mode model is presented in Section 6.2.1, followed by a comparison of cTraceo's results with those of the UAN project in Section 6.2.2.

6.2.1 Comparison with Bellhop and KRAKEN

The discussion presented in [9, 10] and [4] share in common a calculation of transmission loss for the canonical Munk profile shown in Fig.6.1. The source frequency is 50 Hz, and the source is located at a depth of 1000 m. 51 rays were traced with cTraceo between -14° and 14°, restricting the ray fan exclusively to waterborne rays (see Fig. 6.8). The ray plot indicates the existence of three large shadow zones, two of them (10-50 km and 70-100 km) in the upper part of the waveguide and the other one (40-80 km) in the lower part.

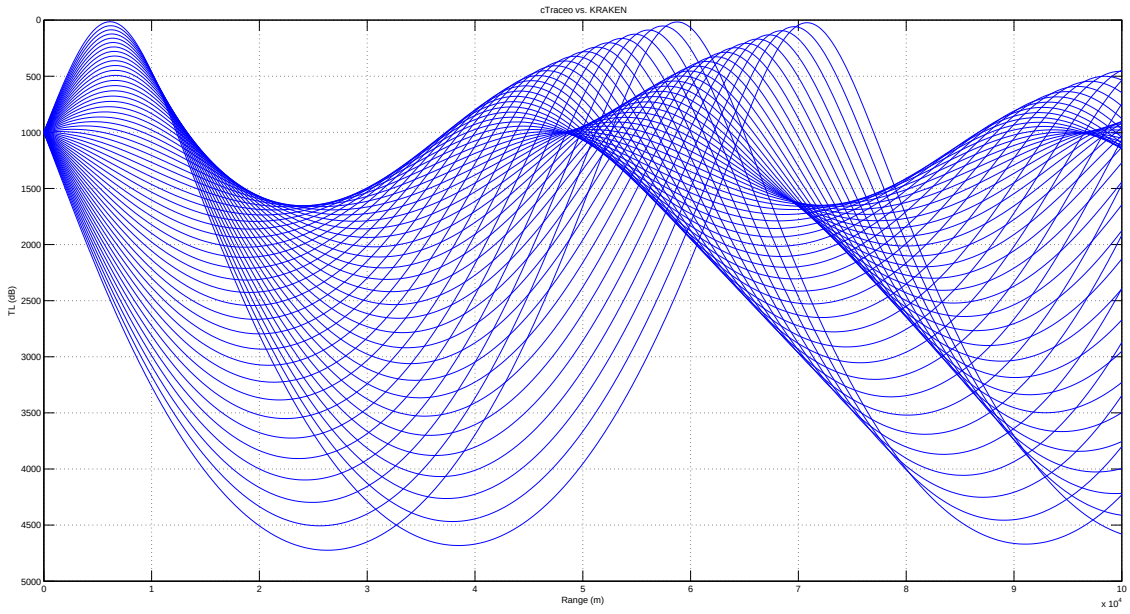
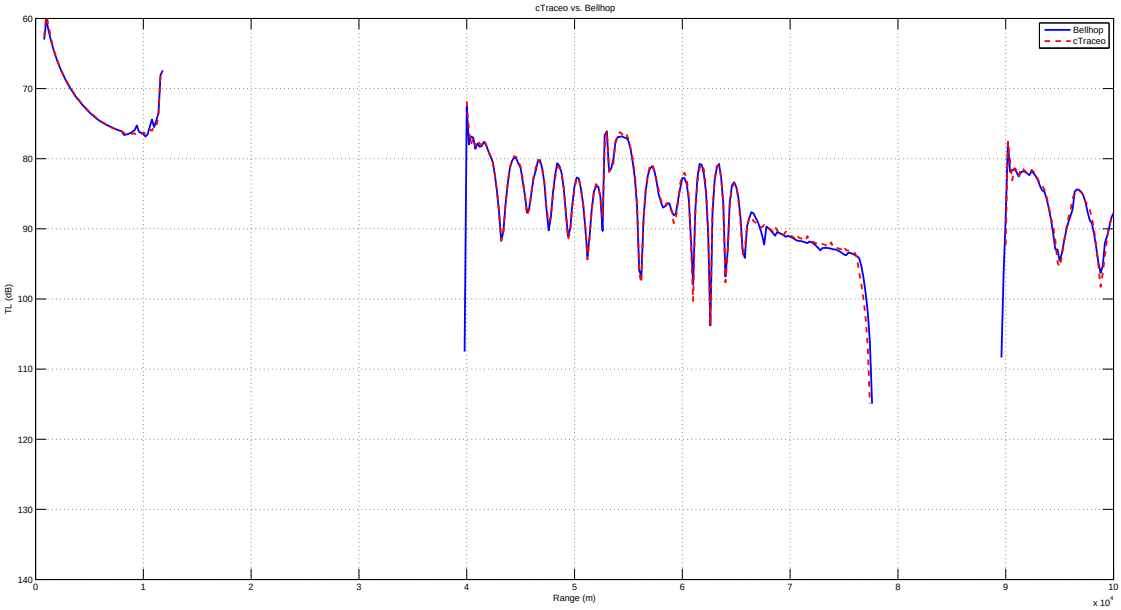
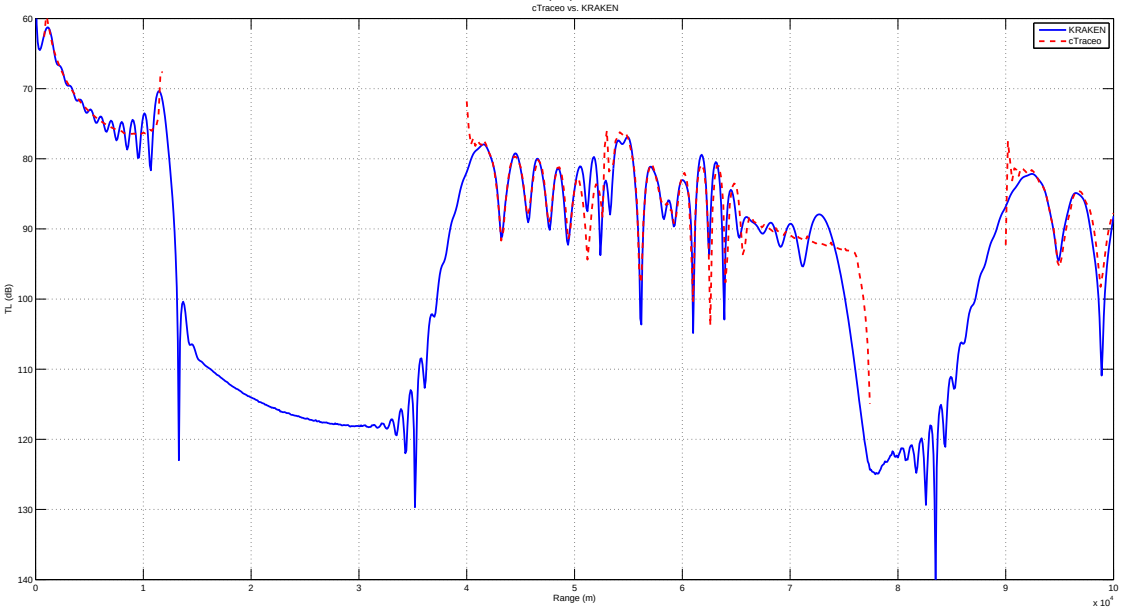


Figure 6.8: Ray trace obtained with cTraceo.

Those rays were used to calculate a transmission loss curve for a receiver at 800 m depth. The comparison between cTraceo, Bellhop and KRAKEN as seen Fig. 6.9, reveals a good agreement in the general trend of transmission loss calculated by all models. The differences in amplitude are negligible, except in the shadow zones, where rays contribute poorly to the field. These comparison results where obtained with the `cTraceo_vs_Bellhop.m` and `cTraceo_vs_Kraken.m` M-Files contained in the "examples/" subdirectory.



(a)



(b)

Figure 6.9: Transmission loss comparison between results obtained with: (a) Bellhop (continuous line) and cTraceo (dashed line); (b) KRAKEN (continuous line) and cTraceo (dashed line).

6.2.2 Comparison with UAN models

The last comparison is shown in Fig. 6.10, which exhibits the transmission loss calculated at 25,6 kHz by cTraceo and the acoustic models JEPE, REV3D and XRAY, used by the partners of the UAN project[2]; the figure reveals that all the models exhibit the same trend.

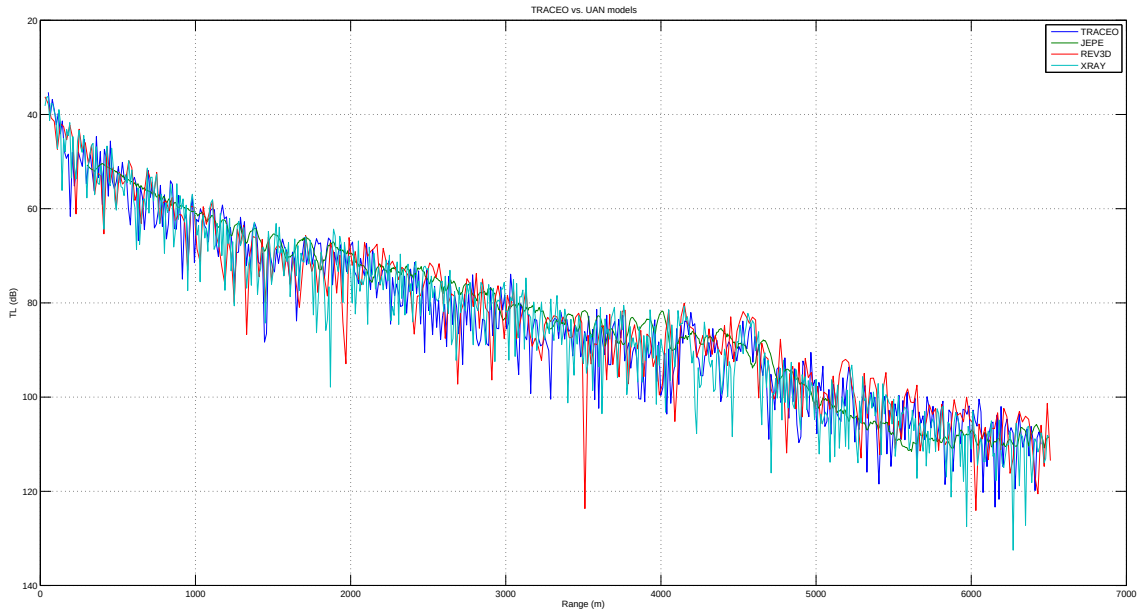


Figure 6.10: Transmission loss comparison: **TRACEO** vs. JEPE, REV3D and XRAY.

[2]http://www.ua-net.eu/

6.3 Performance Comparisons

For the performance comparisons, the average processing times for three executions of each of the comparison cases was determined. Instead of the absolute timing results, the ratio

$$\text{Ratio} = \frac{T_{[Model]}}{T_{[cTraceo]}}$$

between the processing times for the compared model and cTraceo is used as a way of assessing the performance differences. Thus, the higher the value, the faster cTraceo is in relation to the compared model.
The performance results from the comparison between cTraceo, Bellhop and KRAKEN in Section 6.2.1, are summed up in Table 6.1:

	Ratio
Bellhop	1.25
KRAKEN	0.92
TRACEO	1.45

Table 6.1: Comparison between cTraceo, Bellhop and KRAKEN.

Another set of comparisons compares cTraceo's processing times against those of TRACEO for several of the examples distributed along with the model, as shown in Table 6.2:

Test Case			
Filename	N. Rays	Array Size (r×z)	Ratio
pekeris_epr.m	5001	1×1	1.73
pekeris_erf.m	5001	1×1	1.75
munk_epr.m	1001	1×1	3.20
munk_erf.m	1001	1×1	3.20
isovelocity_pvl_rry.m	101	501×201	1.69
isovelocity_pvl_rry_2o.m	101	501×201	3.99

Table 6.2: Processing time comparison between TRACEO and cTraceo.

The `pekeris_epr.m` and `pekeris_erf.m` examples perform an eigenray search by Proximity and by Regula Falsi, respectively, in a pekeris waveguide with vacuum above the surface, a rigid bottom and 2001 launched rays.
The `munk_epr.m` and `munk_erf.m` examples also perform an eigenray search by Proximity and by Regula Falsi, respectively. The waveguide has a flat rigid bottom and a flat vacuum surface interface. 1001 rays are launched and a tabulated Munk sound speed profile is ued.
The `isovelocity_pvl_rry.m` example models particle velocity in a waveguide with an isovelocity sound speed profile, flat vacuum surface and flat rigid bottom. The previous case is extended in the `isovelocity_pvl_rry_2o.m` example by including two objects in the waveguide. In both cases 101 rays are launched and a rectangular array of receivers with a dimension of 501×201 is considered.

Chapter 7

Conclusions And Future Work

TRACEO's conversion to the C language has led to a model which is able to generate the same results as its predecessor, but with a significant performance and usability gain. Also, cTraceo features improved memory handling, which results in greatly reduced memory usage and the removal of hard limits to variable's dimensions.
From the performance comparisons shown in Section 6.3, it can be observed that cTraceo's performance advantage relative to it's predecessor increases with the complexity of the modeled environment. This is due to code improvements at many stages. Even so, several possibilities for code optimization still remain, and it would be desirable to explore them in the future.
Another desirable option for future performance improvement is the implementation of a GPU version of the model. Due to the high computional throughput of GPUs, a GPU version would also be a good starting point for the model's extension to the three-dimensional case.

Bibliography

[1] Orlando C. Rodriguez. *The TRACEO ray tracing program*. Universidade do Algarve - Signal Processing Laboratory, 2010.

[2] Finn B. Jensen, William A. Kuperman, Michael B. Porter, and Henrik Schmidt. *Computational Ocean Acoustics*. American institute of Physics, Woodbury, New York, 1994.

[3] Michael J. Buckingham. Ocean-acoustic propagation models. technical report EUR 13810. Technical report, 1991.

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

[5] Michael Porter. The kraken normal mode program. Technical report, SACLANT UNDERSEA RESEARCH CENTRE, 1994.

[6] P.J. Papadakis, M.I. Taroudakis, and J.S. Papadakis. Recovery of the properties of an elastic bottom using reflection coefficient measurements. In *Proceedings of the 2nd European Conference on Underwater Acoustics*, volume II, pages 943–948, Copenhagen, Denmark, 1994.

[7] H Schmidt. Safari, seismo-acoustic fast field algorithm for range-independent environments. user's guide. Technical report, SACLANT UNDERSEA RESEARCH CENTRE, La Spezia, Italy, 1987.

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

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

[10] Baxley P.A., Bucker H., and Porter M.B. Comparison of beam tracing algorithms. In *Proceedings of the 5th. European Conference on Underwater Acoustics*, Lyon, France, July 2000.

Appendix A

Documentation for `wtraceoinfil.m`

The `wtraceoinfil.m` is included in the `M-Files/` subdirectory and is used to easily write an `infile` from within Matlab, as described in Section 5.1. The following is a description of the input structures for the M-File:

```
source_info         :A structure containing the source configuration
    |
    |---.ds         :The ray step [m].
    |
    |---.position   :A vector containing the source coodinates [m].
    |               Format:
    |                   [rx, zx]
    |
    |---.rbox       :A vector containing the "range box"; lower and
    |               upper boundaries for the ray range coordinates [m].
    |               Format:
    |                   [rbox1, rbox2]
    |
    |---.f          :Source frequency [Hz]
    |
    '---.thetas     :A vector containing the launching angles.
                    The length of the vector, as well as the first
                    and the last elements will be used.

surface_info        :A structure containing the surface (altimetry)
    |               configuration.
    |
    |---.type       :A string defining the type of interface.
    |               Allowed values are:
    |                   '''A'''     Absorvent surface
    |                   '''E'''     Elastic surface
    |                   '''R'''     Rigid surface
    |                   '''V'''     Vacuum beyond surface
    |
    |---.ptype      :A string defining the type of interface properties
    |               Allowed values are:
```

```
    |                   '''H'''     Homogeneous surface properties
    |                   '''N'''     Non-Homogeneous surface properties
    |
    |---.properties :A vector if interface properties are homogeneous;
    |               An array if properties are non-homogeneous, each line
    |               representing the properties at a point of the surface.
    |               Format:
    |                   [cp,  cs,  rho,  ap,  as]
    |                or:
    |                   [cp1, cs1, rho1, ap1, as1;
    |                    cp2, cs2, rho2, ap2, as2; ... ]
    |               Where:
    |                   cp  :Compressional speed                [m/s]
    |                   cs  :Shear speed                        [m/s]
    |                   rho :Density of medium beyond interface [kg/m3]
    |                   ap  :Compressional attenuation   [see ".units"]
    |                   as  :Shear attenuation           [see ".units"]
    |
    |---.units      :A string defining the units of the attenuation
    |               values given in the interface properties.
    |               Allowed valueas are:
    |                   '''F'''     dB/kHz
    |                   '''M'''     dB/meter
    |                   '''N'''     dB/neper
    |                   '''Q'''     Q factor
    |                   '''W'''     dB/<wavelength in meter>
    |
    |---.x          :An array containing the surface coordinates.
    |               Each line contains the range and depth coordinate
    |               of a surface point.
    |               Format:
    |                   [r1, z1;
    |                    r2, z2; ... ]
    |
    '---.itype      :A string defining the interpolation type used
                    for the surface.
                    Allowed values are:
                        '''FL'''    Flat surface
                        '''SL'''    Surface with a Slope
                        '''2P'''    Piecewise linear interpolation
                        '''3P'''    Piecewise parabolic interpolation
                        '''4P'''    Piecewise cubic interpolaton

ssp_info            :A structure containing the sound speed
    |               configuration.
    |
    |---.cdist      :A string defining the type of sound speed distribution.
    |               Allowed values are:
```

```
    |                   '''c(z,z)'''   Sound speed profile (range-independent)
    |                   '''c(r,z)'''   Sound speed field (range-dependent)
    |
    |---.cclass     :A string defining what type of sound speed profile
    |               to use. Allowed values are:
    |                   '''ISOV'''     Isovelocity
    |                   '''LINP'''     Linear
    |                   '''PARP'''     Parabolic
    |                   '''EXPP'''     Exponential
    |                   '''N2LP'''     n2-linear
    |                   '''ISQP'''     Inverse square gradient
    |                   '''MUNK'''     Munk
    |                   '''TABL'''     Tabulated
    |
    |---.r          :A vector containing the range coordinates.
    |
    |---.z          :A vector containing the depth coordinates.
    |
    '---.c          :A vector for sound speed profiles, or an array
                    for sound speed fields.


object_info         :A structure containing optional object
    |               configuration.
    |
    |---.nobjects   :A scalar containinng the number of objects.
    |
    |---.itype      :A string determining what type of interpolation to
    |               use for all objects. Allowed values are:
    |                   '''2P'''    Piecewise Linear
    |                   '''3P'''    Piecewise Parabolic
    |                   '''4P'''    Piecewise Cubic
    |
    |---.npobjects  :A vector, with each element containing the number
    |               of points of an object.
    |
    |
    |---.x          :A 3D array containing the coordinates of all
    |               objects. Objects are composed of an upper a lower
    |               surface, defined in pairs for a range coordinate.
    |               Format:
    |                   x(<object_number>;
    |                     <r>, <z_lower_surface>, <z_upper_surface>;
    |                     <coord_index>)
    |
    |---.type       :Determines the object interface type.
    |                   '''A'''     Absorvent surface
    |                   '''E'''     Elastic surface
```

```
    |                   '''R'''     Rigid surface
    |                   '''V'''     Vacuum beyond surface
    |
    |---.units      :A vector of strings, with each string defining
    |               which attenuation untis to use for the
    |               corresponding object properties.
    |               Allowed valueas are:
    |                   '''F'''     dB/kHz
    |                   '''M'''     dB/meter
    |                   '''N'''     dB/neper
    |                   '''Q'''     Q factor
    |                   '''W'''     dB/<wavelength in meter>
    |
    '---.properties :An Array, with each line representing the
                    interface properties of the corresponding object.
                    Format:
                        [cp1, cs1, rho1, ap1, as1;
                         cp2, cs2, rho2, ap2, as2; ... ]
                    Where:
                        cp  :Compressional speed                [m/s]
                        cs  :Shear speed                        [m/s]
                        rho :Density of medium beyond interface [kg/m3]
                        ap  :Compressional attenuation   [see ".units"]
                        as  :Shear attenuation           [see ".units"]

bathymetry_info     :Identical to surface_info (see above)

output_info         :A structure containing the desired output
    |               configuration.
    |
    |---.ctype      :A string defining what calculation to perform.
    |               Allowed values are:
    |                   '''RCO'''   Ray Coordinates
    |                   '''ARI'''   All Ray Information
    |                               (RCO + ray amplitudes)
    |                   '''ERF'''   Eigenrays, using Regula
    |                               Falsi method
    |                   '''EPR'''   Eigenrays, using
    |                               Proximity method
    |                   '''ADR'''   Amplitudes and Delays,
    |                               using Regula Falsi method
    |                   '''ADP'''   Amplitudes and Delays,
    |                               using Proximity method
    |                   '''CPR'''   Coherent Acoustic Pressure
    |                   '''CTL'''   Coherent Transmission Loss
    |                   '''PVL'''   Coherent Particle Velocity
    |                   '''PAV'''   Coherent Acoustic Pressure
    |                               and Particle Velocity
```

```
|
|---.array_shape:A string defining the shape of the hydrophone array
|                Allowed values are:
|                    '''RRY'''   Rectangular Array
|                    '''HRY'''   Horizontal Array
|                    '''VRY'''   Vertical Array
|                    '''LRY'''   Linear Array
|
|---.r           :Range coordinates of hydrophone array
|
|---.z           :Depth coordinates of hydrophone array
|
'---.miss        :Determines distance threshhold for eigenray search.
```