Ray Methods for Underwater Acoustics

University of Liverpool
Summer Project
Finley Boulton, Jinhui Gong, Yezhang Li
Supervisors: Dr. Daniel Colquitt, Dr. Stewart Haslinger

1 Introduction

In this short project we tackle ray tracing for acoustic waves in oceans with complex sound speed profiles. Our methods will be strictly numerical as the problem at hand is too difficult for exact evaluation. First we consider the derivation of the governing equations which will arrive us at a system of coupled differential equations. From there we will evaluate this system using the simple first order Euler method, then a fourth order Runge-Kutta method (RK4). We then quantitatively validate our methods before calculating and providing many examples for locations around the world.

We follow from Jensen et al. [2011] and Tolstoy et al. [1989] which both provide a comprehensive overview of ocean acoustics. To generate the sound speed profiles we use the work of Dushaw [2022], where on his website there can be found several files under Sound Speed, Temperature, Salinity, and Buoyancy Profiles for the World Ocean from the 2001 World Ocean Atlas. These files let us generate sound speed profiles from arbitrary points around the globe, and they are borrowed from Dushaw [2022] MATLAB acoustic propagation gui package.

Contents

2 Mathematical Derivation

2.1 Derivation of the Eikonal Equation

The governing system of differential equations (which will be solved numerically later on) can be obtained by using the Helmholtz equation which represents a time independent form of the wave equation:

$$\nabla^2 p + \frac{\omega^2}{c^2(\mathbf{x})} p = -\delta(\mathbf{x} - \mathbf{x}_0), \tag{2.1}$$

where $\mathbf{x} = (x, y, z)$, $c(\mathbf{x})$ is the speed of sound, ω is the angular frequency, and $\mathbf{x}_0$ is the location of the source. We seek a solution in the ray series form

$$p(\mathbf{x}) = e^{\mathrm{i}\omega\tau(\mathbf{x})} \sum_{j=0}^{\infty} \frac{A_j(\mathbf{x})}{(\mathrm{i}\omega)^j}. \tag{2.2}$$

Second order derivatives of the ray series $p(\mathbf{x})$ can be easily found:

$$p_{xx} = \mathrm{e}^{\mathrm{i}\omega\tau} \left\{ [-\omega^2(\tau_x)^2 + \mathrm{i}\omega\tau_{xx}] \sum_{j=0}^{\infty} \frac{A_j}{(\mathrm{i}\omega)^j} + 2\mathrm{i}\omega\tau_x \sum_{j=0}^{\infty} \frac{A_{j,x}}{(\mathrm{i}\omega)^j} + \sum_{j=0}^{\infty} \frac{A_{j,xx}}{(\mathrm{i}\omega)^j} \right\}, \tag{2.3}$$

$$p_{yy} = \mathrm{e}^{\mathrm{i}\omega\tau} \left\{ [-\omega^2(\tau_y)^2 + \mathrm{i}\omega\tau_{yy}] \sum_{j=0}^{\infty} \frac{A_j}{(\mathrm{i}\omega)^j} + 2\mathrm{i}\omega\tau_y \sum_{j=0}^{\infty} \frac{A_{j,y}}{(\mathrm{i}\omega)^j} + \sum_{j=0}^{\infty} \frac{A_{j,yy}}{(\mathrm{i}\omega)^j} \right\}, \tag{2.4}$$

$$p_{zz} = \mathrm{e}^{\mathrm{i}\omega\tau} \left\{ [-\omega^2(\tau_z)^2 + \mathrm{i}\omega\tau_{zz}] \sum_{j=0}^{\infty} \frac{A_j}{(\mathrm{i}\omega)^j} + 2\mathrm{i}\omega\tau_z \sum_{j=0}^{\infty} \frac{A_{j,z}}{(\mathrm{i}\omega)^j} + \sum_{j=0}^{\infty} \frac{A_{j,zz}}{(\mathrm{i}\omega)^j} \right\}. \tag{2.5}$$

From $\nabla^2 p = p_{xx} + p_{yy} + p_{zz}$, we have

$$\begin{aligned}\nabla^2 p = \mathrm{e}^{\mathrm{i}\omega\tau}\Bigg\{ & \left[-\omega^2\left((\tau_x)^2 + (\tau_y)^2 + (\tau_z)^2\right) + \mathrm{i}\omega\left(\tau_{xx} + \tau_{yy} + \tau_{zz}\right)\right]\sum_{j=0}^{\infty}\frac{A_j}{(\mathrm{i}\omega)^j} \\ & + 2\mathrm{i}\omega\left(\tau_x\sum_{j=0}^{\infty}\frac{A_{j,x}}{(\mathrm{i}\omega)^j} + \tau_y\sum_{j=0}^{\infty}\frac{A_{j,y}}{(\mathrm{i}\omega)^j} + \tau_z\sum_{j=0}^{\infty}\frac{A_{j,z}}{(\mathrm{i}\omega)^j}\right) \\ & + \left(\sum_{j=0}^{\infty}\frac{A_{j,xx}}{(\mathrm{i}\omega)^j} + \sum_{j=0}^{\infty}\frac{A_{j,yy}}{(\mathrm{i}\omega)^j} + \sum_{j=0}^{\infty}\frac{A_{j,zz}}{(\mathrm{i}\omega)^j}\right)\Bigg\}.\end{aligned} \tag{2.6}$$

Since $\hat{\mathbf{i}}\cdot\hat{\mathbf{j}} = \hat{\mathbf{j}}\cdot\hat{\mathbf{k}} = \hat{\mathbf{i}}\cdot\hat{\mathbf{k}} = 0$ and $\hat{\mathbf{i}}\cdot\hat{\mathbf{i}} = \hat{\mathbf{j}}\cdot\hat{\mathbf{j}} = \hat{\mathbf{k}}\cdot\hat{\mathbf{k}} = 1$ in Cartesian coordinates, we can transform the sum

$$\tau_x\sum_{j=0}^{\infty}\frac{A_{j,x}}{(\mathrm{i}\omega)^j} + \tau_y\sum_{j=0}^{\infty}\frac{A_{j,y}}{(\mathrm{i}\omega)^j} + \tau_z\sum_{j=0}^{\infty}\frac{A_{j,z}}{(\mathrm{i}\omega)^j} \tag{2.7}$$

into

$$\begin{aligned}\tau_x\sum_{j=0}^{\infty}\frac{A_{j,x}}{(\mathrm{i}\omega)^j} + \tau_y\sum_{j=0}^{\infty}\frac{A_{j,y}}{(\mathrm{i}\omega)^j} + \tau_z\sum_{j=0}^{\infty}\frac{A_{j,z}}{(\mathrm{i}\omega)^j} &= \left(\tau_x\hat{\mathbf{i}} + \tau_y\hat{\mathbf{j}} + \tau_z\hat{\mathbf{k}}\right) \\ &\quad \cdot\left(\sum_{j=0}^{\infty}\frac{A_{j,x}}{(\mathrm{i}\omega)^j}\hat{\mathbf{i}} + \sum_{j=0}^{\infty}\frac{A_{j,y}}{(\mathrm{i}\omega)^j}\hat{\mathbf{j}} + \sum_{j=0}^{\infty}\frac{A_{j,z}}{(\mathrm{i}\omega)^j}\hat{\mathbf{k}}\right) \\ &= \left(\tau_x\hat{\mathbf{i}} + \tau_y\hat{\mathbf{j}} + \tau_z\hat{\mathbf{k}}\right)\cdot\sum_{j=0}^{\infty}\frac{(A_{j,x}\hat{\mathbf{i}} + A_{j,y}\hat{\mathbf{j}} + A_{j,z}\hat{\mathbf{k}})}{(\mathrm{i}\omega)^j} \\ &= \nabla\tau\cdot\sum_{j=0}^{\infty}\frac{\nabla A_j}{(\mathrm{i}\omega)^j}.\end{aligned} \tag{2.8}$$

Hence we obtain the equation:

$$\nabla^2 p = \mathrm{e}^{\mathrm{i}\omega\tau}\left\{\left[-\omega^2|\nabla\tau|^2 + \mathrm{i}\omega\nabla^2\tau\right]\sum_{j=0}^{\infty}\frac{A_j}{(\mathrm{i}\omega)^j} + 2\mathrm{i}\omega\nabla\tau\cdot\sum_{j=0}^{\infty}\frac{\nabla A_j}{(\mathrm{i}\omega)^j} + \sum_{j=0}^{\infty}\frac{\nabla^2 A_j}{(\mathrm{i}\omega)^j}\right\}. \tag{2.9}$$

Now substitute $\nabla^2 p$ and p into the Helmholtz equation, we have

$$\begin{aligned}&\mathrm{e}^{\mathrm{i}\omega\tau}\left\{\left[-\omega^2|\nabla\tau|^2 + \mathrm{i}\omega\nabla^2\tau\right]\sum_{j=0}^{\infty}\frac{A_j}{(\mathrm{i}\omega)^j} + 2\mathrm{i}\omega\nabla\tau\cdot\sum_{j=0}^{\infty}\frac{\nabla A_j}{(\mathrm{i}\omega)^j} + \sum_{j=0}^{\infty}\frac{\nabla^2 A_j}{(\mathrm{i}\omega)^j}\right\} \\ &\qquad + \frac{\omega^2}{c^2(\mathbf{x})}e^{i\omega}\tau(\mathbf{x})\sum_{j=0}^{\infty}\frac{A_j(\mathbf{x})}{(i\omega)^j} = -\delta(\mathbf{x} - \mathbf{x}_0).\end{aligned} \tag{2.10}$$

Here we consider the term of $\mathcal{O}(\omega^2)$:

$$
\begin{aligned}
-|\nabla\tau|^2 e^{\mathrm{i}\omega\tau} \cancel{\sum_{j=0}^{\infty} \frac{A_j}{(\mathrm{i}\omega)^j}} + \frac{1}{c^2} e^{\mathrm{i}\omega\tau} \cancel{\sum_{j=0}^{\infty} \frac{A_j}{(\mathrm{i}\omega)^j}} &= 0, && (2.11)\\
-|\nabla\tau|^2 + \frac{1}{c^2} &= 0, && (2.12)\\
|\nabla\tau|^2 &= c^{-2}(\mathbf{x}). && (2.13)
\end{aligned}
$$

$|\nabla\tau|^2 = c^{-2}(\mathbf{x})$ is known as the Eikonal equation. From here we use the method of characteristics to get a system of ODE's so that we can solve it numerically to get ray paths. Terms of $\mathcal{O}(\omega)$ and $\mathcal{O}(\omega^{1-j})$ are the transport equations, which will reveal more details of the waves such as pressure, etc.

2.2 Solving the Eikonal equation

Since the modulus and the sound speed are both positive, we can directly get the positive square roots of both sides of the Eikonal equation. What we are going to do next is to differentiate the both sides of the Eikonal equation with respect of s, separately according to the Cartesian coordinates.

For x-component,

$$
\frac{d}{ds}\left(\frac{1}{c}\frac{dx}{ds}\right) = \frac{d}{ds}\left(\frac{\partial\tau}{\partial x}\right) = \frac{\partial^2\tau}{\partial s\partial x} = \frac{\partial^2\tau}{\partial x^2}\frac{\partial x}{\partial s} + \frac{\partial^2\tau}{\partial x\partial y}\frac{\partial y}{\partial s}. \tag{2.14}
$$

Note that $\nabla\tau$ is perpendicular to the wave fronts, so we can define $d\mathbf{x}/ds = c\Delta\tau$, where $\mathbf{x}(s)$ is the ray trajectory. Substitute it into the equation, we have:

$$
\begin{aligned}
\frac{d}{ds}\left(\frac{1}{c}\frac{dx}{ds}\right) &= \frac{\partial^2\tau}{\partial x^2}\frac{dx}{ds} + \frac{\partial^2\tau}{\partial x\partial y}\frac{dy}{ds}\\
&= \frac{\partial^2\tau}{\partial x^2}\left(c\frac{\partial\tau}{\partial x}\right) + \frac{\partial^2\tau}{\partial x\partial y}\left(c\frac{\partial\tau}{\partial y}\right)\\
&= c\left(\frac{\partial^2\tau}{\partial x^2}\frac{\partial\tau}{\partial x} + \frac{\partial^2\tau}{\partial x\partial y}\frac{\partial\tau}{\partial y}\right)\\
&= c\left\{\left[\frac{\partial}{\partial x}\left(\frac{\partial\tau}{\partial x}\right)\right]\cdot\frac{\partial\tau}{\partial x} + \left[\frac{\partial}{\partial x}\left(\frac{\partial\tau}{\partial y}\right)\right]\cdot\frac{\partial\tau}{\partial y}\right\}\\
&= \frac{c}{2}\frac{\partial}{\partial x}\left[\left(\frac{\partial\tau}{\partial x}\right)^2 + \left(\frac{\partial\tau}{\partial y}\right)^2\right]\\
&= \frac{c}{2}\frac{\partial}{\partial x}\left(\frac{1}{c^2}\right) = -\frac{1}{c^2}\frac{\partial c}{\partial x}.
\end{aligned} \tag{2.15}
$$

Similarly, we also have:

$$
\frac{d}{ds}\left(\frac{1}{c}\frac{dy}{ds}\right) = -\frac{1}{c^2}\frac{\partial c}{\partial y},\ \frac{d}{ds}\left(\frac{1}{c}\frac{dz}{ds}\right) = -\frac{1}{c^2}\frac{\partial c}{\partial z}. \tag{2.16}
$$

Hence, for $\mathbf{x} = (x, y, z)$, the eikonal equations can be writen as

$$\frac{d}{ds}\left(\frac{1}{c}\frac{d\mathbf{x}}{ds}\right) = -\frac{1}{c^2}\nabla c. \tag{2.17}$$

In cylindrical coordinates $\mathbf{x} = (r, z)$,

$$\frac{d}{ds}\left(\frac{1}{c}\frac{dr}{ds}\right) = -\frac{1}{c^2}\frac{\partial c}{\partial r},\ \frac{d}{ds}\left(\frac{1}{c}\frac{dz}{ds}\right) = -\frac{1}{c^2}\frac{\partial c}{\partial z}. \tag{2.18}$$

3 Numerical Methods

We will use two different numerical methods for this problem, Euler method and a Runge-Kutta method specifically RK4. The Euler method is only a first order approximation while globally we can expect RK4 to have an error of fourth order, thus we can expect a greater accuracy from our RK4 results. However, RK4 will be more complex to implement correctly.

3.1 Euler Method

For our system

$$\frac{dr}{ds} = c\xi(s),\ \frac{d\xi}{ds} = -\frac{1}{c^2}\frac{\partial c}{\partial r}, \tag{3.1}$$

$$\frac{dz}{ds} = c\zeta(s),\ \frac{d\zeta}{ds} = -\frac{1}{c^2}\frac{\partial c}{\partial z}, \tag{3.2}$$

using simple for finite differences the iteration will be

$$r_{i+1} - r_i = \delta s(c_i \cdot \xi_i), \tag{3.3}$$

$$z_{i+1} - z_i = \delta s(c_i \cdot \zeta_i), \tag{3.4}$$

$$\xi_{i+1} - \xi_i = \delta s\left(-\frac{1}{c_i^2} \cdot \frac{c_{i+1} - c_i}{r_{i+1} - r_i}\right), \tag{3.5}$$

$$\zeta_{i+1} - \zeta_i = \delta s\left(-\frac{1}{c_i^2} \cdot \frac{c_{i+1} - c_i}{z_{i+1} - z_i}\right), \tag{3.6}$$

where δs is referred to as the discretization step or step size, and with the initial conditions:

$$r = r_0,\ \xi_0 = \frac{\cos\theta_0}{c(0)}, \tag{3.7}$$

$$z = z_0,\ \zeta_0 = \frac{\sin\theta_0}{c(0)}, \tag{3.8}$$

where r_0 and z_0 is the location of the source of the acoustic wave, θ_0 is the launch angle of the wave, and $c(0)$ is the speed of sound at the location of the source.

3.2 RK4 Method

RK4 is one of the most well known Runge-Kutta methods, it has fourth order global error scaling. We can define RK4 from its Butcher tableau

$$
\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
1/2 & 0 & 1/2 & & \\
1 & 0 & 0 & 1 & \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

From this we read the general iterative scheme

$$\eta_i^1 = F\left(t_i, \bar{f}_i\right), \tag{3.9}$$

$$\eta_i^2 = F\left(t_i + \frac{dt}{2}, \bar{f}_i + \frac{dt}{2}\eta_i^1\right), \tag{3.10}$$

$$\eta_i^3 = F\left(t_i + \frac{dt}{2}, \bar{f}_i + \frac{dt}{2}\eta_i^2\right), \tag{3.11}$$

$$\eta_i^4 = F\left(t_i + dt, \bar{f}_i + dt\eta_i^3\right), \tag{3.12}$$

$$f_{i+1} = f_i + \frac{dt}{6}\left(\eta_i^1 + 2\eta_i^2 + 2\eta_i^3 + \eta_i^4\right). \tag{3.13}$$

We see that for our system the right hands side have no s dependency so we can eliminate the t terms in the function F, making implementation of this much simpler. However, for our system there are complications involving the partial derivative terms. Firstly, we will still use first order finite difference approximations for these terms this will possibly limit our accuracy. To avoid this higher order finite differences may be used. Secondly, these first order finite difference approximations require the next value of r and z to be known, this will not work if we compute our system in the same manner as the iterative scheme above. There are several options available to us to avoid this.

We can discard the use of functions altogether and directly calculate η and f_{i+1} for r, z, ξ, and ζ independently, see RK4 direct. More details on these approaches will be covered in the computation analysis section. Or one can still use functions but use a backward method with a linear approximation for the first step, see RK4 with backward.

3.2.1 RK4 with backward

Here we consider the numerical method in a single system. However, we have to compute the unknown function $\mathbf{f}$ by order if we adapt the forward step in single system, which means r_{i+1} and z_{i+1} are obtained firstly before we calculate the ξ_{i+1} and ζ_{i+1} via $\partial c/\partial r$ and $\partial c/\partial z$ numerically by forward step. Therefore, we compute the $\partial c/\partial r$ and $\partial c/\partial z$ by backward step to make the iteration in parallel. We adapt the Euler Method to obtain the $\mathbf{f}_2 = [r_2,\ s_2,\ \xi_2,\ \zeta_2]$ in the first step $i = 1$, and then we introduce the system with the unknown function $\mathbf{f}_i = [r_i,\ s_i,\ \xi_i,\ \zeta_i]$: for each iteration:

$$\mathbf{F}(\mathbf{f}_i) = \left[c_i \cdot \xi_i,\ c_i \cdot \zeta_i,\ \frac{1}{c_i^2} \cdot \frac{c_i - c_{i-1}}{r_i - r_{i-1}},\ \frac{1}{c_i^2} \cdot \frac{c_i - c_{i-1}}{z_i - z_{i-1}}\right], \quad i = 2, 3, ... \tag{3.14}$$

Then we obtain the η_i^j, $j = 1, 2, 3, 4$, and then calculate the f_{i+1} by substituting η in the Equation (3.13). More details can be found on this in the Backward RK4 scheme section.

3.2.2 RK4 Direct

Compared with the backward method, this direct method calculate η for each parameter functions (i.e. r, z, ξ, ζ) separately without utilising a single system to perform the iteration. Thus we will not need to use Euler method to perform the first step. Firstly, we calculate the η_r^i for range and η_z^i for depth, $i = 1, \cdots, 4$, following the step from (3.9) to (3.13) where:

$$f_r^i = c_i \cdot \xi_i,\ f_z^i = c_i \cdot \zeta_i. \tag{3.15}$$

Then we use f_z^{i+1} to obtain the new sound speed c_{i+1} via interpolation, and finally calculate the ξ_{i+1} and ζ_{i+1} for the next step, where:

$$f_\xi^i = \frac{1}{c_i^2} \cdot \frac{c_{i+1} - c_i}{r_{i+1} - r_i},\ f_\zeta^i = \frac{1}{c_i^2} \cdot \frac{c_{i+1} - c_i}{z_{i+1} - z_i}. \tag{3.16}$$

However, this implementation will be longer. Functionally both methods perform the same task will insignificant differences in the result. The key to this direct method is that we need to perform the calculations in a uniform order within the iteration. Another idea to keep in mind with this approach is that we have broken down the function and instead are manually performing the function cycle to achieve the forward order. More details can be found on this in the Direct scheme section. More details can be found on this in the Direct scheme section.

4 Computational Analysis

4.1 Iteration of the Euler Method

Using this iteration system we can write up a relatively simple code to solve ray equations. For sound speed profiles there are many options. Later we will demonstrate two for now. The first of the sound speed profiles will come from [Jensen et al., 2011, p. 23], the second is derived from the 2001, 2005, or 2009 World Ocean Atlases.

Our choice of programming language will be `MATLAB` as it is the most convenient for this type of work. However, it is perfectly repeatable in an open source language such as `Python`.

To see what we are working with we first interpolate and plot the sound speed data. Then we can set up things like our solution interval, initial conditions, initialise solution arrays and define other parameters. Next, we can begin our iterative scheme remembering to incorporate things like boundary reflections and re-interpolation of the sound speed profile.

Algorithm 1: Euler method with interpolation

Data: Input the sample data array `Depth`, `Sound_speed`

Set the solution interval and discretisation step n and ds

Create array $\mathbf{r}$, $\mathbf{z}$ that will contain the numerical solution, and initial conditions

for *i from* 1 *to* $n-1$ **do**

  $r_{i+1} = r_i + \mathrm{d}s \cdot c_i \cdot \xi_i$;

  $z_{i+1} = z_i + \mathrm{d}s \cdot c_i \cdot \zeta_i$;

  `/* Boundary Reflections */` ;

  if $z_{i+1} > 0$ **then**

    $\cdots$

  else

    if $z_{i+1} <$ *deep bottom* **then**

      $\cdots$

  $c_{i+1} \leftarrow$ **Interpolation**[`Depth`, `Sound_speed`] $\sim z_{i+1}$;

  $\xi_{i+1} = \xi_i - \mathrm{d}s \cdot (c_{i+1} - c_i)/(r_{i+1} - r_i)$;

  $\zeta_{i+1} = \zeta_i - \mathrm{d}s \cdot (c_{i+1} - c_i)/(z_{i+1} - z_i)$;

return $\mathbf{r}$, $\mathbf{z}$

A key thing to note about the iterative scheme here is the we need to recalculate the sound speed at each point. So we need to re-interpolate the data every time. Otherwise, the new position coordinates of the ray may not correspond to any given point of the interpolation of it were just calculated separately at the start. Any interpolation method may be used. Linear interpolation offers a good start as the results can be checked using analytical calculations see for the expected convergence zone. See [Jensen et al., 2011, p. 23] or [Tolstoy et al., 1989, p. 145]. If these seem to be in accordance it is then possible to use higher order interpolation schemes; common ones found in `MATLAB` are `makima`, `pchip`, and `spline`. Akima spline or as found in `MATLAB`. `Makima` is found to be generally preferable as of observed small overshoot.

When performing the coding in Section 3.1, there will be issues arising at some point downrange from the source. What is happening is that the z is reaching points outside of the sound speed data. i.e., at the surface if left to naturally curve back the ray will actually reach $z > 0$, at this point the sound speed can no longer be interpolated as such "`NaN`" related errors will occur. The same will occur at a depth larger than the maximum depth with a corresponding sound speed value. One can avoid this problem by using spline interpolation of the sound speed profile as this will extrapolate the data somewhat reasonably outside the known values. To properly fix this issue one must implement boundary reflections.

4.2 Boundary Reflections

4.2.1 Reflections in Euler Method

When ray reaches to the sea level ($z = 0$), it will be reflected to return to the ocean. Here we need to restart the ray tracing from the $z = 0$ with the take-off angle reflected Jensen et al. [2011]. According to the ideal condition,

$$\overline{r} = r,\ \overline{\xi} = \xi \tag{4.1}$$
$$\overline{z} = z,\ \overline{\zeta} = -\zeta \tag{4.2}$$

It is then also critical to recalculate r_{i+1} with a modified democratisation step such that the new take off point is exactly on the surface where the incoming ray impacted. To find the new democratisation step $\tilde{\delta s}$ we simply use

$$z_{i+1} - z_i = \delta s(c_i \cdot \zeta_i), \tag{4.3}$$

at some maximum depth $z = D$

$$D - z_i = \tilde{\delta s}_D(c_i \cdot \zeta_i), \tag{4.4}$$
$$\tilde{\delta s}_D = \frac{D - z_i}{(c_i \cdot \zeta_i)}, \tag{4.5}$$

or at the surface $z = 0$

$$0 - z_i = \tilde{\delta s}_0(c_i \cdot \zeta_i), \tag{4.6}$$
$$\tilde{\delta s}_0 = -\frac{z_i}{(c_i \cdot \zeta_i)}. \tag{4.7}$$

Then

$$r_{i+1} - r_i = \tilde{\delta s}_D(c_i \cdot \xi_i), \tag{4.8}$$

or if reflecting from the surface

$$r_{i+1} - r_i = \tilde{\delta s}_0(c_i \cdot \xi_i). \tag{4.9}$$

4.2.2 Reflections in RK4

There is a similar approach is taken with implementing boundary reflections in the backward RK4. In this case, we can obtain the formula $r - z$ of the trade from i to $i + 1$.

$$(r - r_i) = \frac{r_i - r_{i+1}}{z_i - z_{i+1}}(z - z_i) \tag{4.10}$$

at some maximum depth $z = D$, we have:

$$\begin{aligned} r &= \frac{r_i - r_{i+1}}{z_i - z_{i+1}}(D - z_i) + r_i \\ &= \frac{D(r_{i+1} - r_i) + r_i z_{i+1} - r_{i+1} z_i}{z_{i+1} - z_i} \end{aligned} \tag{4.11}$$

or at the surface $z = 0$, we obtain:

$$\begin{aligned} r &= \frac{r_i - r_{i+1}}{z_i - z_{i+1}}(0 - z_i) + r_i \\ &= \frac{r_i z_{i+1} - r_{i+1} z_i}{z_{i+1} - z_i} \end{aligned} \tag{4.12}$$

Algorithm 2: Reflection for Euler method

$\cdots$
for i *from* 1 *to* $n-1$ **do**
  $\cdots$
  $z_{i+1} = z_i + \mathrm{d}s \cdot c_i \cdot \zeta_i$;
  if $z_{i+1} > 0$ **then**
    $z_{i+1} = 0$;
    $r_{i+1} = r_i + \tilde{\delta s}_0 (c_i \cdot \xi_i)$;
    $\zeta_i := -\zeta_i$;
  else
    if $z_{i+1} <$ *deep bottom* **then**
      $z_{i+1} = D$; `/* Deep bottom limitation:` D `*/`
      $r_{i+1} = r_i + \tilde{\delta s}_D (c_i \cdot \xi_i)$;
      $\zeta_i := -\zeta_i$;
  $c_{i+1} \leftarrow$ **Interpolation**[`Depth`, `Sound_speed`] $\sim z_{i+1}$;
  $\cdots$

Algorithm 3: Reflection for Backward RK4 method

$\cdots$
for i *from* 1 *to* $n-1$ **do**
  $\cdots$
  if $z_{i+1} > 0$ **then**
    $z_{i+1} = 0$;
    $r_{i+1} = (r_i \cdot z_{i+1} - r_{i+1} \cdot z_i)/(z_{i+1} - z_i)$;
    $\zeta_i := -\zeta_i$;
  else
    if $z_{i+1} <$ *deep bottom* **then**
      $z_{i+1} = D$; `/* Deep bottom limitation:` D `*/`
      $r_{i+1} = \left(D \cdot (r_{i+1} - r_i) + r_i \cdot z_{i+1} - r_{i+1} \cdot z_i\right)/(z_{i+1} - z_i)$;
      $\zeta_i := -\zeta_i$;
  $c_{i+1} \leftarrow$ **Interpolation**[`Depth`, `Sound_speed`] $\sim z_{i+1}$;
  $\cdots$

For the situation of the Direct RK4, we implement the Boundary reflections in a very similar manner to the Euler iterative scheme i.e., we find $\tilde{\delta s}_D$ and $\tilde{\delta s}_0$ when the ray reaches to the surface level and bottom respectively, via our equation for z_{i+1}.

At some maximum depth $z = D$:

$$\tilde{\delta s}_D = -\frac{6 \cdot (D - z_i)}{\eta_z^1 + 2 \cdot \eta_z^2 + 2 \cdot \eta_z^3 + \eta_z^4}, \tag{4.13}$$

and for the surface $z = 0$:

$$\tilde{\delta s}_0 = -\frac{6 \cdot z_i}{\eta_z^1 + 2 \cdot \eta_z^2 + 2 \cdot \eta_z^3 + \eta_z^4}. \tag{4.14}$$

Then we use this modified discretisation step to recalculate r_{i+1} but note we will need to find η_r^j, $j = 1, \cdots, 4$ again. c_{i+1} is found in the exact same way as the Euler scheme.

Algorithm 4: Reflection for Direct RK4 method

$\cdots$
for i *from* 1 *to* $n-1$ **do**
    $\cdots$
    if $z_{i+1} > 0$ **then**
        $z_{i+1} = 0$;
        $\tilde{\delta s}_0 = -6 \cdot z_i / (\eta_z^1 + 2 \cdot \eta_z^2 + 2 \cdot \eta_z^3 + \eta_z^4)$;
        $\eta_r^1 = \tilde{\delta s}_0 \cdot c_i \cdot \xi_i$;
        $\eta_r^2 = \tilde{\delta s}_0 \cdot (c_i \cdot \xi_i + 1/2 \cdot \eta_r^1)$;
        $\eta_r^3 = \tilde{\delta s}_0 \cdot (c_i \cdot \xi_i + 1/2 \cdot \eta_r^2)$;
        $\eta_r^4 = \tilde{\delta s}_0 \cdot (c_i \cdot \xi_i + \eta_r^3)$;
        $r_{i+1} = r_i + \tilde{\delta s}_0 \cdot (\eta_r^1 + 2 \cdot \eta_r^2 + 2 \cdot \eta_r^3 + \eta_r^4)/6$;
        $\zeta_i := -\zeta_i$;
    else
        if $z_{i+1} <$ *deep bottom* **then**
            $z_{i+1} = D$; `/* Deep bottom limitation:` D `*/`
            $\tilde{\delta s}_D = -6 \cdot (D - z_i) / (\eta_z^1 + 2 \cdot \eta_z^2 + 2 \cdot \eta_z^3 + \eta_z^4)$;
            $\eta_r^1 = \tilde{\delta s}_D \cdot c_i \cdot \xi_i$;
            $\eta_r^2 = \tilde{\delta s}_D \cdot (c_i \cdot \xi_i + 1/2 \cdot \eta_r^1)$;
            $\eta_r^3 = \tilde{\delta s}_D \cdot (c_i \cdot \xi_i + 1/2 \cdot \eta_r^2)$;
            $\eta_r^4 = \tilde{\delta s}_D \cdot (c_i \cdot \xi_i + \eta_r^3)$;
            $r_{i+1} = r_i + \tilde{\delta s}_D \cdot (\eta_r^1 + 2 \cdot \eta_r^2 + 2 \cdot \eta_r^3 + \eta_r^4)/6$;
            $\zeta_i := -\zeta_i$;
    $c_{i+1} \leftarrow$ **Interpolation**[`Depth`, `Sound_speed`] $\sim z_{i+1}$;
    $\cdots$

4.3 Iteration of the RK4 Method

4.3.1 Backward scheme

In the backward scheme we calculate η in the single system to remain the dependency of the equation system. In order to keeping the uniform iteration, we have to adapt the Euler Method to obtain c_{i-1} at the first step before performing under the backward order.

Algorithm 5: RK4 with backward

Data: Input the sample data array `Depth`, `Sound_speed`

Set the solution interval and discretisation step n and ds

Create array **r**, **z** that will contain the numerical solution, and initial conditions

Define the function $F(f_i,\ f_{i-1},\ c_i,\ c_{i-1})$ in the Equation (3.14)

$r_2 = r_1 + \tilde{\delta s} \cdot c_1 \cdot \xi_1$;

$z_2 = z_1 + \tilde{\delta s} \cdot c_1 \cdot \zeta_1$;

$c_2 \leftarrow$ **Interpolation**[`Depth`, `Sound_speed`] $\sim z_2$;

$\xi_2 = \xi_1 - \tilde{\delta s} \cdot (c_2 - c_1)/(r_2 - r_1)$;

$\zeta_2 = \zeta_1 - \tilde{\delta s} \cdot (c_2 - c_1)/(z_2 - z_1)$;

for *i from 2 to $n-1$* **do**

  $f_i = [r_i,\ z_i,\ \xi_i,\ \zeta_i]$;

  $f_{i-1} = [r_{i-1},\ z_{i-1},\ \xi_{i-1},\ \zeta_{i-1}]$;

  $\eta_i^1 = F(f_i,\ f_{i-1},\ c_i,\ c_{i-1})$;

  $\eta_i^2 = F(f_i + \tilde{\delta s}/2 \cdot \eta_i^1,\ f_{i-1} + \tilde{\delta s}/2 \cdot \eta_i^1,\ c_i,\ c_{i-1})$;

  $\eta_i^3 = F(f_i + \tilde{\delta s}/2 \cdot \eta_i^2,\ f_{i-1} + \tilde{\delta s}/2 \cdot \eta_i^2,\ c_i,\ c_{i-1})$;

  $\eta_i^4 = F(f_i + \tilde{\delta s} \cdot \eta_i^3,\ f_{i-1} + \tilde{\delta s} \cdot \eta_i^3,\ c_i,\ c_{i-1})$;

  $f_{i+1} = f_i + \tilde{\delta s}/6 * (\eta_i^1 + 2 * \eta_i^2 + 2 * \eta_i^3 + \eta_i^4)$;

  `/* Boundary Reflections */` ;

  if $z_{i+1} > 0$ **then**

    $\cdots$

  else

    if $z_{i+1} <$ *deep bottom* **then**

      $\cdots$

  $c_{i+1} \leftarrow$ **Interpolation**[`Depth`, `Sound_speed`] $\sim z_{i+1}$;

return r, **z**

4.3.2 Direct scheme

In the direct scheme we will calculate each η for each variable separately, this way we can sequence the variables correctly such to not run into dependency issues. An obvious downside to this is that the code is quite lengthy but, it is very intuitive. Fundamentally this method will achieve the same as the other approach illustrated.

Algorithm 6: RK4 Direct Scheme

Data: Input the sample data array `Depth`, `Sound_speed`

Set the solution interval and discretisation step n and ds

Create array $\mathbf{r}$, $\mathbf{z}$ that will contain the numerical solution, and initial conditions

for i *from* 1 *to* $n-1$ **do**

  `/* Range */` ;

  $\eta_r^1 = \widetilde{\delta s} \cdot c_i \cdot \xi_i$;

  $\eta_r^2 = \widetilde{\delta s} \cdot (c_i \cdot \xi_i + 1/2 \cdot \eta_r^1)$;

  $\eta_r^3 = \widetilde{\delta s} \cdot (c_i \cdot \xi_i + 1/2 \cdot \eta_r^2)$;

  $\eta_r^4 = \widetilde{\delta s} \cdot (c_i \cdot \xi_i + \eta_r^3)$;

  $r_{i+1} = r_i + \widetilde{\delta s} \cdot (\eta_r^1 + 2 \cdot \eta_r^2 + 2 \cdot \eta_r^3 + \eta_r^4)\,/6$;

  `/* Depth */` ;

  $\eta_z^1 = \widetilde{\delta s} \cdot c_i \cdot \zeta_i$;

  $\eta_z^2 = \widetilde{\delta s} \cdot (c_i \cdot \zeta_i + 1/2 \cdot \eta_z^1)$;

  $\eta_z^3 = \widetilde{\delta s} \cdot (c_i \cdot \zeta_i + 1/2 \cdot \eta_z^2)$;

  $\eta_z^4 = \widetilde{\delta s} \cdot (c_i \cdot \zeta_i + \eta_z^3)$;

  $z_{i+1} = z_i + \widetilde{\delta s} \cdot (\eta_z^1 + 2 \cdot \eta_z^2 + 2 \cdot \eta_z^3 + \eta_z^4)\,/6$;

  `/* Boundary Reflections */` ;

  if $z_{i+1} > 0$ **then**

    $\cdots$

  else

    if $z_{i+1} <$ *deep bottom* **then**

      $\cdots$

  `/* Sound Speed */` ;

  $c_{i+1} \leftarrow$ **Interpolation**[`Depth`, `Sound_speed`] $\sim z_{i+1}$;

  `/* Aux variable` ξ `*/` ;

  $\eta_\xi^1 = -\widetilde{\delta s} \cdot (c_{i+1} - c_i)/(r_{i+1} - r_i)$;

  $\eta_\xi^2 = \widetilde{\delta s} \cdot \big(-(c_{i+1} - c_i)/(r_{i+1} - r_i) + 1/2 \cdot \eta_\xi^1\big)$;

  $\eta_\xi^3 = \widetilde{\delta s} \cdot \big(-(c_{i+1} - c_i)/(r_{i+1} - r_i) + 1/2 \cdot \eta_\xi^2\big)$;

  $\eta_\xi^4 = \widetilde{\delta s} \cdot \big(-(c_{i+1} - c_i)/(r_{i+1} - r_i) + \eta_\xi^3\big)$;

  $\xi_{i+1} = \xi_i + \widetilde{\delta s} \cdot \big(\eta_\xi^1 + 2 \cdot \eta_\xi^2 + 2 \cdot \eta_\xi^3 + \eta_\xi^4\big)\,/6$;

  `/* Aux variable` ζ `*/` ;

  $\eta_\zeta^1 = -\widetilde{\delta s} \cdot (c_{i+1} - c_i)/(z_{i+1} - z_i)$;

  $\eta_\zeta^2 = \widetilde{\delta s} \cdot \big(-(c_{i+1} - c_i)/(z_{i+1} - z_i) + 1/2 \cdot \eta_\zeta^1\big)$;

  $\eta_\zeta^3 = \widetilde{\delta s} \cdot \big(-(c_{i+1} - c_i)/(z_{i+1} - z_i) + 1/2 \cdot \eta_\zeta^2\big)$;

  $\eta_\zeta^4 = \widetilde{\delta s} \cdot \big(-(c_{i+1} - c_i)/(z_{i+1} - z_i) + \eta_\zeta^3\big)$;

  $\zeta_{i+1} = \zeta_i + \widetilde{\delta s} \cdot \big(\eta_\zeta^1 + 2 \cdot \eta_\zeta^2 + 2 \cdot \eta_\zeta^3 + \eta_\zeta^4\big)\,/6$;

return $\mathbf{r}$, $\mathbf{z}$

5 Results

First we will concentrate on the example found in [Jensen et al., 2011, p. 23] as seen below.

Depth (m)	Sound Speed (m/s)	Gradient (m/s/m)	θ_i	$\sin\theta_i$
0	1522.0	-	-	-
300	1501.0	-0.0700	0.1663	0.1655
1200	1514.0	0.0144	0.1026	0.1024
2000	1496.0	-0.0225	0.1851	0.1840
5000	1545.0	0.0163	-	-

Then we will use linear sound speed interpolations and expected convergence zone calculations to show that the methods produce accurate results. The convergence zone we will inspect will be the first surface strike for a ray with launch angle of 1°. For a four layer system with a linear sound speed profile the convergence zone is given by Tolstoy et al. [1989]. Following from [Tolstoy et al., 1989, p. 52] and [Tolstoy et al., 1989, p. 147] we can derive the convergence zone for a four layer system with a linear sound speed profile. Starting with the previously derived eikonal equation

$$|\nabla\tau(\mathbf{x})|^2 = \tau_x^2 + \tau_y^2 + \tau_z^2 = c^{-2}(\mathbf{x}), \tag{5.1}$$

we can transform with

$$d\alpha = dx/c, \quad d\beta = dy/c, \quad d\gamma = dz/c, \tag{5.2}$$

thus

$$\tau_\alpha^2 + \tau_\beta^2 + \tau_\gamma^2 = 1. \tag{5.3}$$

i.e., the rays are now straight lines in the α, β and γ space. We know straight lines are given by

$$dl^2 = dx^2 + dy^2 + dz^2, \tag{5.4}$$

$$\implies \quad d\sigma^2 = d\alpha^2 + d\beta^2 + d\gamma^2. \tag{5.5}$$

We require the functional $\sigma \int d\sigma = 0$ satisfying the condition of geodesics. We also see that

$$d\sigma^2 = c^2 dl^2, \tag{5.6}$$

hence

$$\sigma \int d\sigma = \sigma \int \frac{dl}{c} = 0. \tag{5.7}$$

We are only concerned with ray tracing in 2-dimensions and we have $c(\mathbf{x}) = c(z)$ thus

$$dl^2 = dx^2 + \cancel{dy^2} + dz^2, \tag{5.8}$$

$$dl = \sqrt{dx^2 + dz^2}, \tag{5.9}$$

$$= \sqrt{\left(\frac{dx}{dz}\right)^2 + 1} = \sqrt{x_z^2 + 1} \tag{5.10}$$

Now with (4.7)

$$\sigma \int \frac{dl}{c} = \sigma \int \frac{\sqrt{x_z^2+1}}{c} = 0. \tag{5.11}$$

We now apply the Euler-Lagrange equation $\frac{\partial L}{\partial x} - \frac{d}{dz}\left(\frac{\partial L}{\partial x_z}\right) = 0$, note that we can use $\frac{\partial L}{\partial x_z} = a$ where a is constant,

$$\frac{x_z}{c\sqrt{x_z^2+1}} = a. \tag{5.12}$$

Take $x_z = \tan\theta$ hence

$$a = \frac{\tan\theta}{c\sqrt{\tan^2\theta+1}}, \tag{5.13}$$

$$= \frac{\tan\theta}{c\sec\theta}, \tag{5.14}$$

$$= \frac{\tan\theta\cos\theta}{c} = \frac{\sin\theta}{c}, \tag{5.15}$$

which is just snells law. To find x we simply integrate $x_z = ac\sqrt{x_z^2+1} = \tan\theta$

$$x = \pm \int_0^z x_z dz = \pm \int_0^z \tan\theta dz = \pm \int_0^z ac\sqrt{x_z^2+1}dz, \tag{5.16}$$

$$= \pm\, a \int_0^z c\sqrt{\tan^2\theta+1}dz, \tag{5.17}$$

$$= \pm\, a \int_0^z c\sqrt{\tan^2\theta+1}dz, \tag{5.18}$$

$$= \pm\, a \int_0^z c\left(\frac{1}{\cos^2\theta}\right)^{1/2} dz, \tag{5.19}$$

$$= \pm\, a \int_0^z c\left(\frac{1}{1-\sin^2\theta}\right)^{1/2} dz, \tag{5.20}$$

$$= \pm\, a \int_0^z c\left(\frac{1}{1-a^2c^2}\right)^{1/2} dz, \tag{5.21}$$

$$= \pm\, a \int_0^z \frac{cdz}{(1-a^2c^2)^{1/2}}. \tag{5.22}$$

$$\tag{5.23}$$

For this approximation we assume linear sound speed $c(z) = gz$ where g is the gradient of the line. Hence

$$x = \pm a \int_0^z \frac{gzdz}{(1-a^2(gz)^2)^{1/2}} = \text{Const} + \pm\frac{(1-a^2(gz)^2)^{1/2}}{a}. \tag{5.24}$$

Let Const $= 0$ this implies

$$x^2 = a^{-2} - g^2 z^2, \tag{5.25}$$

$$x^2 + z^2 = a^{-2} g^{-2} = R^2. \tag{5.26}$$

i.e., the rays are circles of radius $1/ag$ centered on the line $c = 0$. The integration constant a defines a particular ray in terms of its angle of incidence at some depth. Now consider a two layer linear system, we define

$$R_1 = \frac{c_0}{g_1}, \tag{5.27}$$

$$R_1 = \frac{c_0}{g_2}, \tag{5.28}$$

and

$$\Delta_1 = 2R_1 \sin\theta_1, \tag{5.29}$$

$$\Delta_2 = 2R_2 \sin\theta_1. \tag{5.30}$$

Hence

$$R_{CZ} = \Delta_1 + \Delta_2 = 2\sin\theta_1(R_1 + R_2) = 2c_0 \sin\theta_1 \left(\frac{1}{|g_1|} + \frac{1}{|g_2|}\right). \tag{5.31}$$

Expanding this out to three and four layer systems

$$R_{CZ} = 2c_0 \left(\frac{\sin\theta_1}{|g_1|} + \frac{\sin\theta_1 - \sin\theta_2}{|g_2|} + \frac{\sin\theta_2}{|g_3|}\right), \tag{5.32}$$

and

$$R_{CZ} = 2c_0 \left(\frac{\sin\theta_1}{|g_1|} + \frac{\sin\theta_1 - \sin\theta_2}{|g_2|} + \frac{\sin\theta_3 - \sin\theta_2}{|g_3|} + \frac{\sin\theta_3}{|g_4|}\right), \tag{5.33}$$

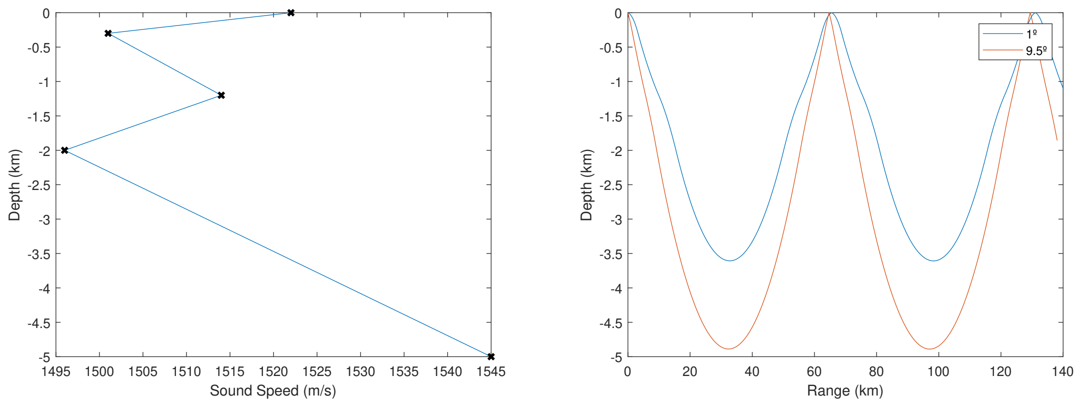
where g_i is the gradient of c_i, and the ray angle at the interface is given by

$$\theta_i = \arccos\left(\frac{c_i}{c_0}\right). \tag{5.34}$$

Visually the results are all very similar and are in line with what we would expect given this sound speed profile and what is presented in [Jensen et al., 2011, p. 23]. Quantitatively for the convergence zone of the first surface strike for a ray with launch angle of 1° we see

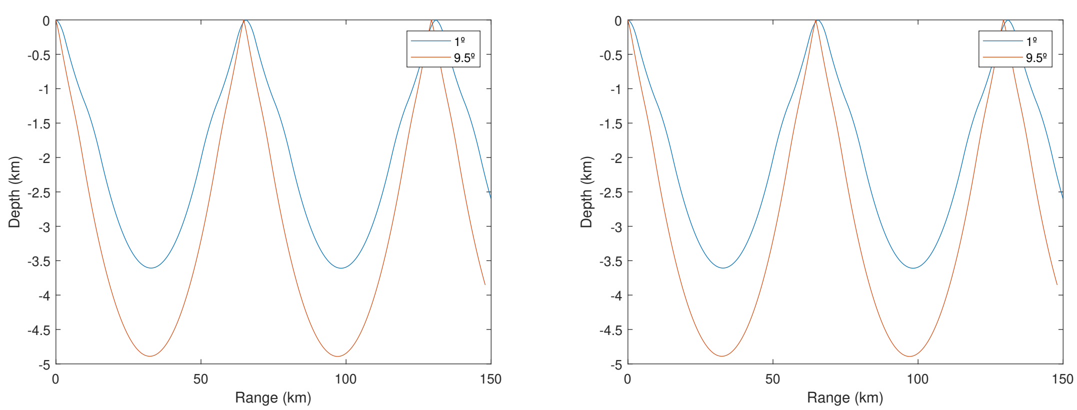
Method	R_{CZ}	Euler	RK4 Backward	RK4 Direct
Range (m)	65854	65537	65521	65527
Difference to R_{CZ} (m)	0	317	333	327

It should be noted that this R_{CZ} calculation is the loop length for a 0°-ray for a simple



(a) (b)

Figure 1: (a) [Jensen et al., 2011, p. 23] Sound speed profile with linear interpolation. (b) Corresponding Euler method ray trace of angles 1° and 9.5°.



(a) (b)

Figure 2: (a) RK4 Backward Method (b) RK4 Direct Method

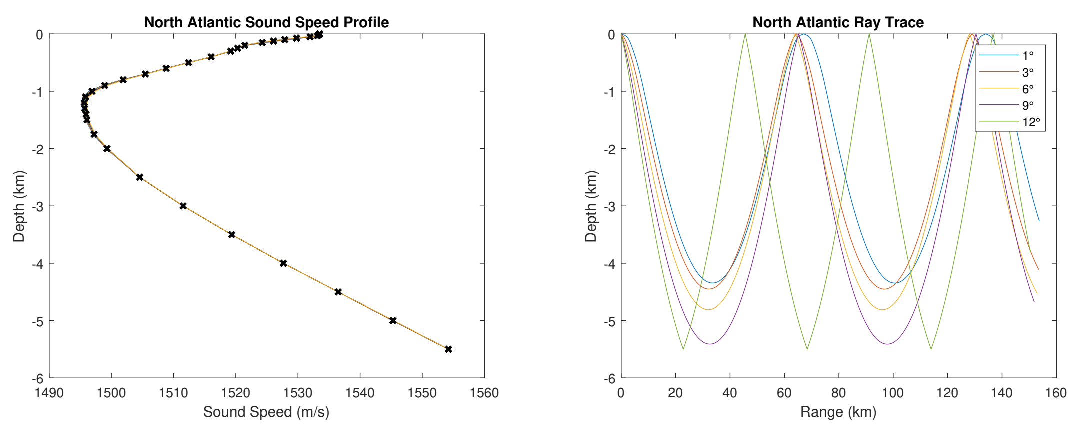
linear four layer profile. Running the code for ray angles of 0.1° to 1° achieves convergence zones ranging from 66.2km to 65.5km. [Jensen et al., 2011, p. 21] computes transmission loss convergence zone of 65km. This goes to show we have achieved a reasonable and expected accuracy from our methods; environmental errors in measuring and interpolating the sound speed profile will be larger than those caused by numerical integration. For quick and easy implementation of ray tracing code Euler's method is good and in the limit $\delta s \to 0$ it will converge to an exact solution. However, having a global error of order one may not be sufficient in this case using higher methods such as the fourth order RK4 methods demonstrated will improve accuracy at the cost of efficiency. This efficiency cost can be somewhat negated by using an adaptive step size Runge-Kutta scheme such as the Runge–Kutta–Fehlberg method, further details on this can be found in Fehlberg [1969]. Further improvements may be brought about by using better interpolation techniques such as cubic and spline methods.

6 Examples

In this section example locations are selected then we find the corresponding sound speed profile and then corresponding ray trace with a source located at the origin. A few buried sources will also be presented. Following that we present example codes of the Euler method and two implementations of the RK4 method.

6.1 Example profiles

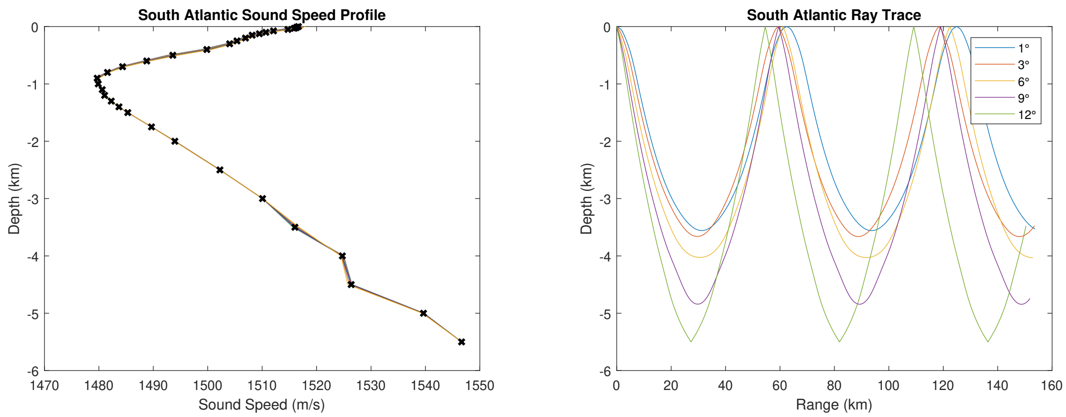
We have chosen varying launch angles to try and best illustrate the nature of the sound speed profile at various points across the globe. The locations were selected based on trying to get a wide variety of locations including all the worlds oceans. A submerged source location is also included.



(a)

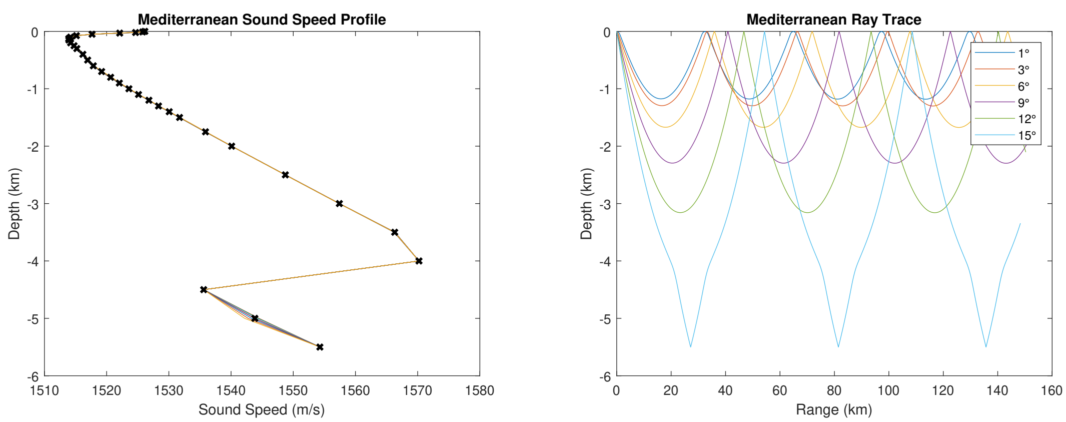
(b)

Figure 3: (a) Location (27, -40) (Lat, Long) (b) corresponding ray trace SD=0



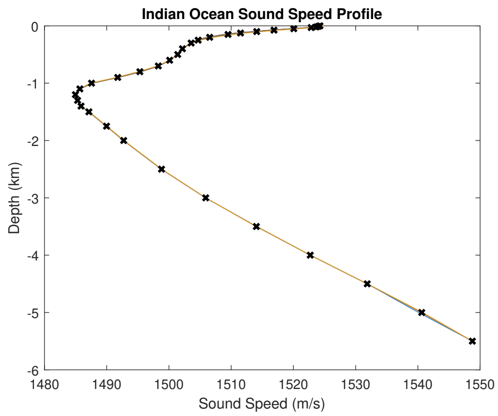
(a) (b)

Figure 4: (a) Location (-35, -16) (Lat, Long) (b) corresponding ray trace SD=0

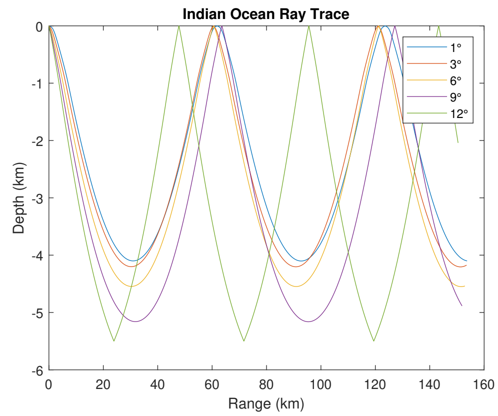


(a) (b)

Figure 5: (a) Location (35, 19) (Lat, Long) (b) corresponding ray trace SD=0

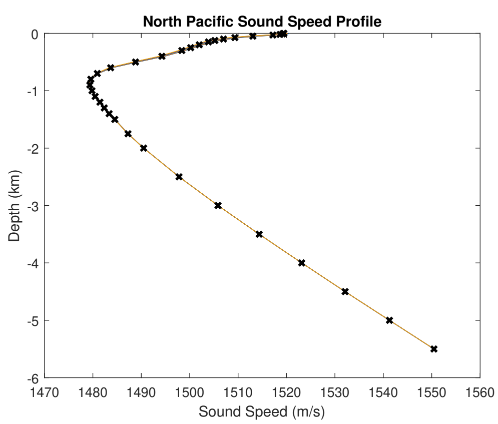


(a)

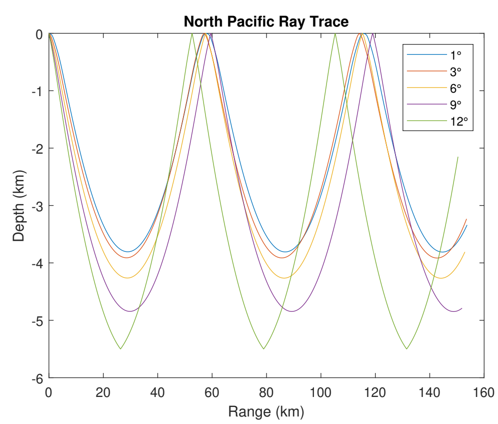


(b)

Figure 6: (a) Location (-30, 80) (Lat, Long) (b) corresponding ray trace SD=0

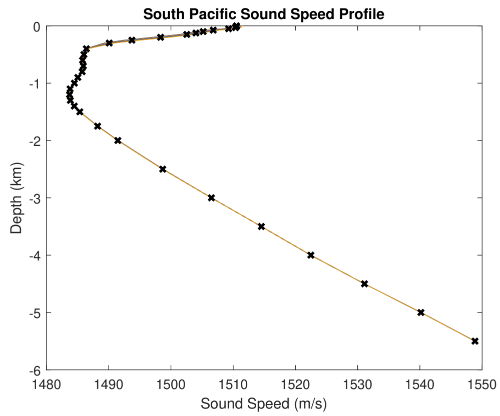


(a)

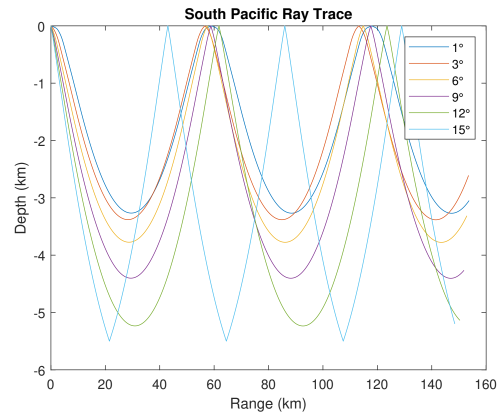


(b)

Figure 7: (a) Location (33, -167) (Lat, Long) (b) corresponding ray trace SD=0

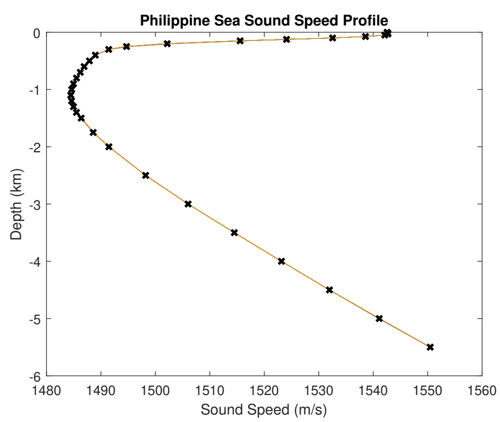


(a)

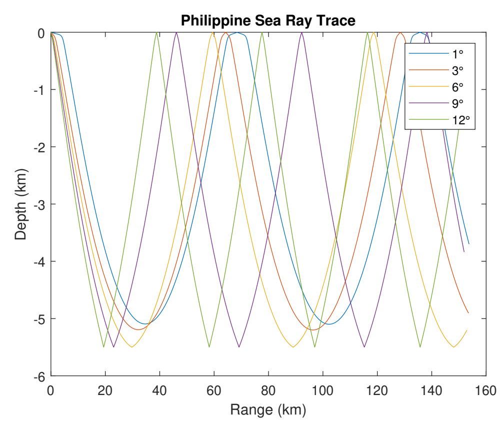


(b)

Figure 8: (a) Location (-37, -134) (Lat, Long) (b) corresponding ray trace SD=0

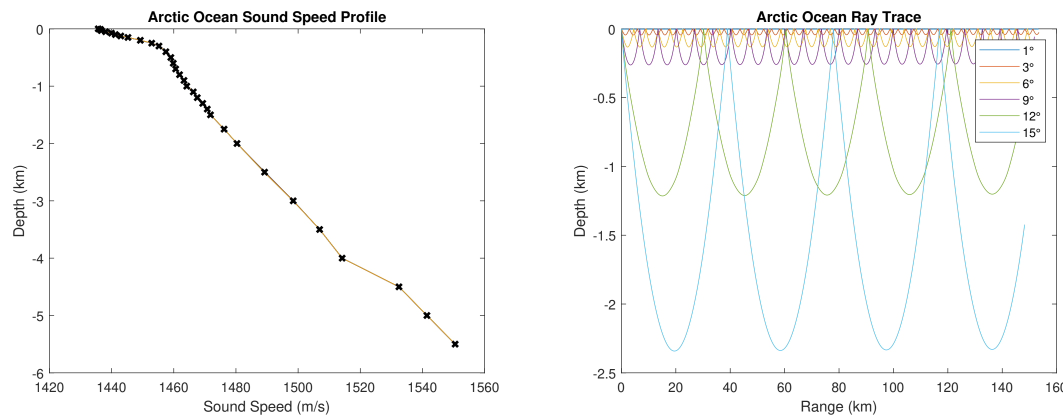


(a)



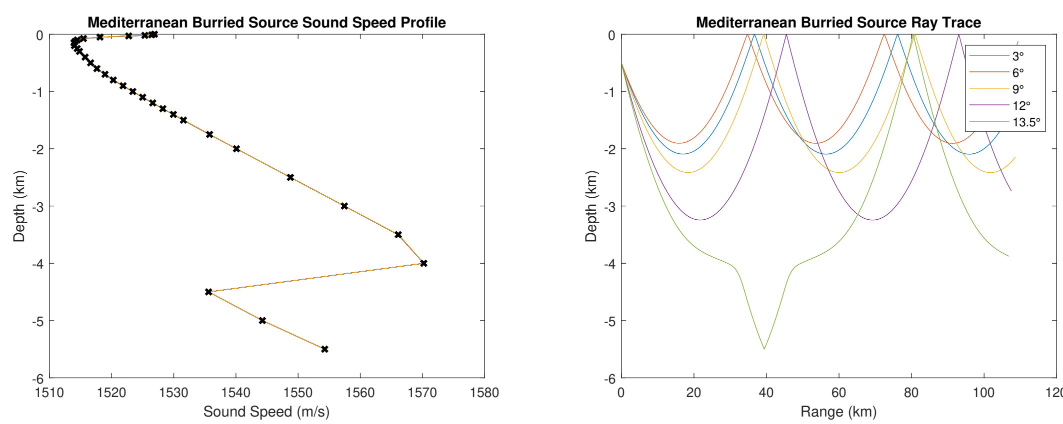
(b)

Figure 9: (a) Location (6, 136) (Lat, Long) (b) corresponding ray trace SD=0



(a) (b)

Figure 10: (a) Location (83, -179) (Lat, Long) (b) corresponding ray trace SD=0



(a) (b)

Figure 11: (a) Location (34, -17.5) (Lat, Long) (b) corresponding ray trace SD=-500

6.2 Example Codes

Included here is example MATLAB scripts for the Euler method, RK4 backward method, and the RK4 direct method. We have also included two methods for sound speed profile generation in each method. The first is using Dushaw [2022] code to extract sound speed data from any desired location around the globe. The second is a manual method of specifying a discrete sound speed profile which is then interpolated. In all cases ds is kept small ($ds \sim 1$m) this is to ensure numerical stability of the solution. In all cases the Depth limit parameter should be set to the depth of the last sound speed measurement this is done so we aren't then extrapolating the sound speed; this can cause large and sometimes fatal errors depending on the type of interpolation used. Whilst not super computationally expensive code it is still not recommended to run them on low end hardware.

Listing 1: Euler Method

```
% Euler Method
%% Custom location

lat=[30 31];
lon=[175 176];

[r, glat, glon]=dist(lat,lon,10);
[P1,z]=get_lev(glat,glon,'c');

figure
plot(P1,-z/1000); xlabel('Sound Speed (m/s)'); ylabel('Depth (km)')
meanSSP = mean(P1,2);
hold on
plot(meanSSP,-z/1000,'kx','LineWidth',2)

%SSP
depth = z;
%disp(depth)
sspd  = transpose(meanSSP);
%disp(sspd)

%interp
zq = 0:1:5499 ;
c = interp1(depth, sspd, zq, 'linear') ;

%for bottom reflections
depth_lim = 5500.0 ;


```

```
%% Jensen data

%Manual data input
depth = [0 300 1200 2000 5000] ;
sspd  = [1522.0 1501.0 1514.0 1496.0 1545.0] ;

%for bottom reflections
depth_lim = 5000.0 ;

%Interpolation
zq = 0:1:4999 ;
c = interp1(depth, sspd, zq, 'linear') ;

%RCZ calculation
g = zeros(1,length(depth));
a = zeros(1,length(depth));
sine = zeros(1,length(depth));
for k = 1:length(depth)-1
    g(k) = (sspd(k+1)-sspd(k))./(depth(k+1)-depth(k));
    a(k) = acos(sspd(k)./sspd(1));
    sine(k) = sin(a(k)) ;
end
%disp('RCZ')
%RCZ = 2*sspd(1)*((sine(2)./abs(g(1))) + (sine(2) - sine(3))./abs(g(2)) + (
   sine(4) - sine(3))./abs(g(3)) + (sine(4)./abs(g(4))) );
%disp(g)
%disp(a)
%disp(sin(a))
%disp(RCZ)

%SSP plot
figure
plot(c, -zq/1000) ;
hold on
plot(sspd,-depth/1000,'kx','LineWidth',2)
xlabel('Sound Speed (m/s)'); ylabel('Depth (km)')
hold off
%% Euler method

%IC
r0 = 0.0 ; z0 = 0.0 ; %Source point

```

```
%Solution interval and discretization step
ds = 1 ;
n=140000;

%for many angle
angle_arr = [1, 3, 6, 9, 13.5] ;

figure
for j = 1:length(angle_arr)

    %aux varibles IC
    xi0   = cosd(angle_arr(j)) ./ sspd(1) ;
    zeta0 = sind(angle_arr(j)) ./ sspd(1) ;

    %Create array that will contain the numerical solution
    r_tst = zeros(1, n) ; %range
    z_tst = zeros(1, n) ; %depth
    s     = zeros(1, n) ; %arclength parameter
    xi    = zeros(1, n) ; %aux
    zeta  = zeros(1, n) ;
    c_arr = zeros(1, n) ; %sound speed

    %Set the initial values
    r_tst(1) = r0 ;
    z_tst(1) = z0 ;
    s(1)     = ds ;
    xi(1)    = xi0 ;
    zeta(1)  = zeta0 ;
    c_arr(1) = sspd(1) ;


    % Perform the iteration
    for i = 1:n-1
        r_tst(i+1) = r_tst(i) + ds * c_arr(i) * xi(i) ;
        z_tst(i+1) = z_tst(i) + ds * c_arr(i) * zeta(i) ;

        %BR surface
        if z_tst(i+1) < 0.0
            z_tst(i+1) = 0.0;
            r_tst(i+1) = r_tst(i) - (z_tst(i) ./ zeta(i)) * xi(i);
            zeta(i) = -zeta(i);
            %disp('Euler')
```

```
            %disp(r_tst(i+1))
        end

        %BR bottom
        if z_tst(i+1) > depth_lim
            z_tst(i+1) = depth_lim;
            r_tst(i+1) = r_tst(i) + ((depth_lim - z_tst(i)) ./ zeta(i)) * xi(i);
            zeta(i) = -zeta(i);
        end

        c_arr(i+1) = interp1(zq, c, z_tst(i+1), 'linear') ;

        xi(i+1)    = xi(i)    - ds * (1./c_arr(i)^2 * (c_arr(i+1)-c_arr(i))./(r_tst(i+1)-r_tst(i))) ;
        zeta(i+1)  = zeta(i)  - ds * (1./c_arr(i)^2 * (c_arr(i+1)-c_arr(i))./(z_tst(i+1)-z_tst(i))) ;
        s(i+1)     = (i+1) * ds ;
    end

    %plot
    plot(r_tst/1000, -z_tst/1000) ; hold on
    xlabel('Range (km)') ;
    ylabel('Depth (km)') ;
    legend('1  ','3  ' ,'6  ','9  ', '13.5  ');

end
%plot([0, n/1000], [0, 0], 'k-') ;
```

Listing 2: RK4 Backward Method

```
% RK4 Backward
%% Custom location

lat=[34 35];
lon=[17.5 18.5];

[r, glat, glon]=dist(lat,lon,10);
[P1,z]=get_lev(glat,glon,'c');

figure
plot(P1,-z/1000); xlabel('Sound Speed (m/s)'); ylabel('Depth (km)')
meanSSP = mean(P1,2);
```

```
hold on
plot(meanSSP,-z/1000,'kx','LineWidth',2)

%SSP
depth = z;
%disp(depth)
sspd  = transpose(meanSSP);
%disp(sspd)

%interp
zq = 0:1:5499 ;
c = interp1(depth, sspd, zq, 'linear') ;

%for bottom reflections
depth_lim = 5500.0 ;


%% Jensen data

%Manual data input
depth = [0 300 1200 2000 5000] ;
sspd  = [1522.0 1501.0 1514.0 1496.0 1545.0] ;

%for bottom reflections
depth_lim = 5000.0 ;

%Interpolation
zq = 0:1:4999 ;
c = interp1(depth, sspd, zq, 'linear') ;

%RCZ calculation
g = zeros(1,length(depth));
a = zeros(1,length(depth));
sine = zeros(1,length(depth));
for k = 1:length(depth)-1
    g(k) = (sspd(k+1)-sspd(k))./(depth(k+1)-depth(k));
    a(k) = acos(sspd(k)./sspd(1));
    sine(k) = sin(a(k)) ;
end
%disp('RCZ')
%RCZ = 2*sspd(1)*((sine(2)./abs(g(1))) + (sine(2) - sine(3))./abs(g(2)) + (sine(4) - sine(3))./abs(g(3)) + (sine(4)./abs(g(4))) );
```

```
%disp(g)
%disp(a)
%disp(sin(a))
%disp(RCZ)

%SSP plot
figure
plot(c, -zq/1000) ;
hold on
plot(sspd,-depth/1000,'kx','LineWidth',2)
xlabel('Sound Speed (m/s)'); ylabel('Depth (km)')
hold off

%% Perform the angle iteration

%for many angle
angle = [1, 3, 6, 9, 13.5] ;


for j = 1:length(angle)

    % Set initial conditions
    r0 = 0.0 ;
    z0 = 0.0 ;
    xi0   = cosd(angle(j)) / sspd(1) ;
    zeta0 = sind(angle(j)) / sspd(1) ;

    f = [r0, z0, xi0, zeta0] ;

    % Arrays to save solution for plotting
    r_arr    = [f(1)] ; z_arr    = [f(2)] ;
    xi_arr   = [f(3)] ; zeta_arr = [f(4)] ;

    c_arr    = zeros(1, n) ;
    c_arr(1) = sspd(1) ;

    % Calculate for i = 2 via Euler Method
    r_arr(2)    = r_arr(1) + ds * c_arr(1) * xi_arr(1) ;
    z_arr(2)    = z_arr(1) + ds * c_arr(1) * zeta_arr(1) ;

    c_arr(2)    = interp1(depth, sspd, z_arr(2), 'makima') ;

```

```
    xi_arr(2)   = xi_arr(1)   - ds * (1/c_arr(1)^2 * (c_arr(2)-c_arr(1))/(r_arr(2)-r_arr(1))) ;
    zeta_arr(2) = zeta_arr(1) - ds * (1/c_arr(1)^2 * (c_arr(2)-c_arr(1))/(z_arr(2)-z_arr(1))) ;

    %f = [r_arr(2), z_arr(2), xi_arr(2), zeta_arr(2)] ;

    % Backward RK4 iteration
    for i = 2:n-1

        f  = [r_arr(i)  , z_arr(i)  , xi_arr(i)  , zeta_arr(i)] ;
        fi = [r_arr(i-1), z_arr(i-1), xi_arr(i-1), zeta_arr(i-1)] ;

        eta1 = F(f            , fi            , c_arr(i), c_arr(i-1)) ;
        eta2 = F(f + ds/2*eta1, fi + ds/2*eta1, c_arr(i), c_arr(i-1)) ;
        eta3 = F(f + ds/2*eta2, fi + ds/2*eta2, c_arr(i), c_arr(i-1)) ;
        eta4 = F(f + ds  *eta3, fi + ds  *eta3, c_arr(i), c_arr(i-1)) ;

        % f = [r(i+1), z(i+1), xi(i+1), z(i+1)]
        f = f + ds/6*(eta1 + 2*eta2 + 2*eta3 + eta4) ;

        r_arr(i+1)    = f(1) ;
        z_arr(i+1)    = f(2) ;
        xi_arr(i+1)   = f(3) ;
        zeta_arr(i+1) = f(4) ;

        % If the ray reaches to the sea level z = 0.
        if z_arr(i+1) < 0
            r_arr(i+1) = (r_arr(i)*z_arr(i+1) - r_arr(i+1)*z_arr(i)) / (z_arr(i+1) - z_arr(i)) ;
            z_arr(i+1) = 0 ;
            zeta_arr(i+1) = -zeta_arr(i+1) ;
            %disp('Func')
            %disp(r_arr(i+1))
            %c_arr(i+1) = interp1(depth, sspd, z_arr(i+1), 'makima') ;
        end

        % If the ray penetrated the deep bottom.
        if z_arr(i+1) > depth_lim
            r_arr(i+1) = (depth_lim*(r_arr(i+1)-r_arr(i)) + r_arr(i)*z_arr(i+1) - r_arr(i+1)*z_arr(i)) / (z_arr(i+1) - z_arr(i)) ;
            z_arr(i+1) = depth_lim ;
```

```
            zeta_arr(i+1) = -zeta_arr(i+1) ;
            %c_arr(i+1) = interp1(depth, sspd, z_arr(i+1), 'makima') ;
        end

        c_arr(i+1) = interp1(depth, sspd, z_arr(i+1), 'makima') ;
    end

    plot(r_arr/1000, -z_arr/1000) ;
    hold on

end

hold on
%plot([0, n/1000], [0, 0], 'k--') ;
xlabel('Range (km)') ;
ylabel('Depth (km)') ;
legend('1  ','3  ','6  ','9  ','13.5  ') ;

%%
function res = F(fi, fi1, ci, ci1)
    res = [ci*fi(3), ci*fi(4), -1/ci^2 * (ci-ci1)/(fi(1)-fi1(1)), -1/ci^2 * (ci-ci1)/(fi(2)-fi1(2))] ;
end
```

Listing 3: RK4 Direct Method

```
% RK4 Direct
%% Custom location

%Location of choice in lat long. Note two neighboring locations required.
lat=[34 35];
lon=[17.5 18.5];

%use of pre defined functions to extract SSP
[r, glat, glon]=dist(lat,lon,10);
[P1,z]=get_lev(glat,glon,'c');

%SSP plot
figure
plot(P1,-z/1000); xlabel('Sound Speed (m/s)'); ylabel('Depth (km)')
meanSSP = mean(P1,2);
hold on
plot(meanSSP,-z/1000,'kx','LineWidth',2)
```

```
hold off
title('Mediterranean Burried Source Sound Speed Profile');

%SSP
depth = z;
%disp(depth)
sspd  = transpose(meanSSP);
%disp(sspd)

%interp
zq = 0:1:5499 ;
c = interp1(depth, sspd, zq, 'makima') ;

%for bottom reflections
depth_lim = 5500.0 ;

%alternate SSP plot
%plot
%figure
%plot(c, -zq/1000) ; %new interpolated profile
%hold on
%plot(sspd,-depth/1000,'kx','LineWidth',2) %old profile
%xlabel('Sound Speed (m/s)'); ylabel('Depth (km)')
%hold off


%% Jensen data

%Manual data input
depth = [0 300 1200 2000 5000] ;
sspd  = [1522.0 1501.0 1514.0 1496.0 1545.0] ;

%for bottom reflections
depth_lim = 5000.0 ;

%Interpolation
zq = 0:1:4999 ;
c = interp1(depth, sspd, zq, 'linear') ;

%RCZ calculation
g = zeros(1,length(depth));
a = zeros(1,length(depth));
```

```
sine = zeros(1,length(depth));
for k = 1:length(depth)-1
    g(k) = (sspd(k+1)-sspd(k))./(depth(k+1)-depth(k));
    a(k) = acos(sspd(k)./sspd(1));
    sine(k) = sin(a(k)) ;
end
%disp('RCZ')
%RCZ = 2*sspd(1)*((sine(2)./abs(g(1))) + (sine(2) - sine(3))./abs(g(2)) + (
   sine(4) - sine(3))./abs(g(3)) + (sine(4)./abs(g(4))) );
%disp(g)
%disp(a)
%disp(sin(a))
%disp(RCZ)

%SSP plot
figure
plot(c, -zq/1000) ;
hold on
plot(sspd,-depth/1000,'kx','LineWidth',2)
xlabel('Sound Speed (m/s)'); ylabel('Depth (km)')
hold off

%% RK4 iteration

%IC
r0 = 0.0 ; z0 = 0.0 ; %Source point

%Solution interval and discretization step
ds = 0.6 ;
n=90000;

%for many angle
angle_arr = [1, 3, 6, 9, 13.5] ;

figure
for j = 1:length(angle_arr)

    %aux varibles IC
    xi0   = cosd(angle_arr(j)) ./ sspd(1) ;
    zeta0 = sind(angle_arr(j)) ./ sspd(1) ;

    % Create array that will contain the numerical solution
```

```
    r_arr = zeros(1, n) ; %range
    z_arr = zeros(1, n) ; %depth
    s     = zeros(1, n) ; %arclength parameter
    xi    = zeros(1, n) ; %aux
    zeta  = zeros(1, n) ;
    c_arr = zeros(1, n) ; %sound speed

    %Set the initial values
    r_arr(1) = r0 ;
    z_arr(1) = z0 ;
    s(1)     = ds ;
    xi(1)    = xi0 ;
    zeta(1)  = zeta0 ;
    c_arr(1) = sspd(1) ;


    for i=1:n-1

        %r
        etar1 = c_arr(i) * xi(i);
        etar2 = (c_arr(i)*xi(i) + ds/2 * etar1);
        etar3 = (c_arr(i)*xi(i) + ds/2 * etar2);
        etar4 = (c_arr(i)*xi(i) + ds*etar3);
        r_arr(i+1) = r_arr(i) + ds*(etar1 + 2*etar2 + 2*etar3 + etar4)./6;

        %z
        etaz1 = c_arr(i) * zeta(i);
        etaz2 = (c_arr(i)*zeta(i) + ds/2 * etaz1);
        etaz3 = (c_arr(i)*zeta(i) + ds/2 * etaz2);
        etaz4 = (c_arr(i)*zeta(i) + ds*etaz3);
        z_arr(i+1) = z_arr(i) + ds*(etaz1 + 2*etaz2 + 2*etaz3 + etaz4)./6;

        %BR surface
        if z_arr(i+1) < 0.0
            z_arr(i+1) = 0.0;
            ds_0 = - (6 * z_arr(i)) ./ (etaz1 + 2*etaz2 + 2*etaz3 + etaz4) ;
            etar1 = c_arr(i) * xi(i);
            etar2 = (c_arr(i)*xi(i) + ds_0/2 * etar1);
            etar3 = (c_arr(i)*xi(i) + ds_0/2 * etar2);
            etar4 = (c_arr(i)*xi(i) + ds_0*etar3);
            r_arr(i+1) = r_arr(i) + ds_0*(etar1 + 2*etar2 + 2*etar3 + etar4)
               ./6;
```

```
            zeta(i) = -zeta(i);
            %disp('RK4')
            disp(r_arr(i+1))
        end

        %BR bottom
        if z_arr(i+1) > depth_lim
            z_arr(i+1) = depth_lim;
            ds_d = 6*(depth_lim - z_arr(i))./(etaz1 + 2*etaz2 + 2*etaz3 + etaz4);
            etar1 = c_arr(i) * xi(i);
            etar2 = (c_arr(i)*xi(i) + ds_d/2 * etar1);
            etar3 = (c_arr(i)*xi(i) + ds_d/2 * etar2);
            etar4 = (c_arr(i)*xi(i) + ds_d*etar3);
            r_arr(i+1) = r_arr(i) + ds_d*(etar1 + 2*etar2 + 2*etar3 + etar4)./6;
            zeta(i) = -zeta(i);
            disp(z_arr(i+1))
        end

        %sound speed
        c_arr(i+1) = interp1(zq, c, z_arr(i+1), 'linear') ;

        %xi
        etaxi1 = - 1./c_arr(i)^2 * (c_arr(i+1)-c_arr(i))./(r_arr(i+1)-r_arr(i));
        etaxi2 = ( -1./c_arr(i)^2 *(c_arr(i+1)-c_arr(i))./(r_arr(i+1)-r_arr(i)) + ds/2 *etaxi1 );
        etaxi3 = ( -1./c_arr(i)^2 *(c_arr(i+1)-c_arr(i))./(r_arr(i+1)-r_arr(i)) + ds/2 *etaxi2 );
        etaxi4 = ( -1./c_arr(i)^2 *(c_arr(i+1)-c_arr(i))./(r_arr(i+1)-r_arr(i)) + ds*etaxi3 );
        xi(i+1) = xi(i) + ds*(etaxi1 + 2*etaxi2 + 2*etaxi3 + etaxi4)./6;

        %zeta
        etazeta1 = - 1./c_arr(i)^2 * (c_arr(i+1)-c_arr(i))./(z_arr(i+1)-z_arr(i));
        etazeta2 = ( -1./c_arr(i)^2 * (c_arr(i+1)-c_arr(i))./(z_arr(i+1)-z_arr(i)) + ds/2 * etazeta1 );
        etazeta3 = ( -1./c_arr(i)^2 * (c_arr(i+1)-c_arr(i))./(z_arr(i+1)-z_arr(i)) + ds/2 * etazeta2 );
        etazeta4 = ( -1./c_arr(i)^2 * (c_arr(i+1)-c_arr(i))./(z_arr(i+1)-
```

```
            z_arr(i)) + ds*etazeta3 );
        zeta(i+1) = zeta(i) + ds*(etazeta1 + 2*etazeta2 + 2*etazeta3 +
            etazeta4)./6;

        %s (not technically needed here)
        s(i+1) = (i+1) * ds ;
    end

    %plot
    plot(r_arr/1000, -z_arr/1000) ; hold on
    xlabel('Range (km)') ;
    ylabel('Depth (km)') ;
    legend('1  ','3  ' ,'6  ','9  ', '13.5  ');

end
%plot([0, 2*n/1000], [0, 0], 'k-') ;
```

7 Summary

Key further improvements to the ray methods we developed here would be implementation of an adaptive Runge-Kutta method, this allows for a variable step size. Whilst this would add an extra computation to each step the net efficiency gain is still positive. This results from when the solution isn't changing rapidly we can keep a large step size and still have numerical accuracy, then at rapidly changing regions in the solution, such as turning points, we can decrease the step size to hit some target accuracy. Overall this would decrease the number of steps required to perform a ray trace of a given distance. Whilst the code developed here is somewhat sophisticated, addition of an adaptive Runge-Kutta method would take it closer in line with many professional applications.

Another interesting problem is that of eigen rays. These are a set of rays which connect two points in the ocean, a source and a receiver. due to the nature of acoustics in oceans there will be many such rays with different paths and hence different launch angles. One can also solve for the critical angle, the angle at which a ray will strike the bottom.

In our ray methods we neglected to solve terms of the Helmholtz equation of order $\mathcal{O}(\omega)$ and $\mathcal{O}(\omega^{1-j})$, these are known as the transport equations. The solutions of which let us associate phase and intensity with the ray paths, hence we can compute the pressure field. From here we can find what is known as transmission loss. There are numerous approaches to this such as: coherent transmission loss, semi-coherent transmission loss, incoherent transmission loss, and geometric beams. This is discussed in detail in [Jensen et al., 2011, p. 168].

References

Brain Dushaw. Brain dushaw's homepage. http://staff.washington.edu/dushaw/, 2022. Accessed: 2022-08-18.

Erwin Fehlberg. *Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems*, volume 315. National aeronautics and space administration, 1969.

Finn B Jensen, William A Kuperman, Michael B Porter, Henrik Schmidt, and Alexandra Tolstoy. *Computational ocean acoustics*, volume 794. Springer, 2011.

Ivan Tolstoy, CS Clay, and David H Berman. Ocean acoustics: Theory and experiment in underwater acoustics, 1989.