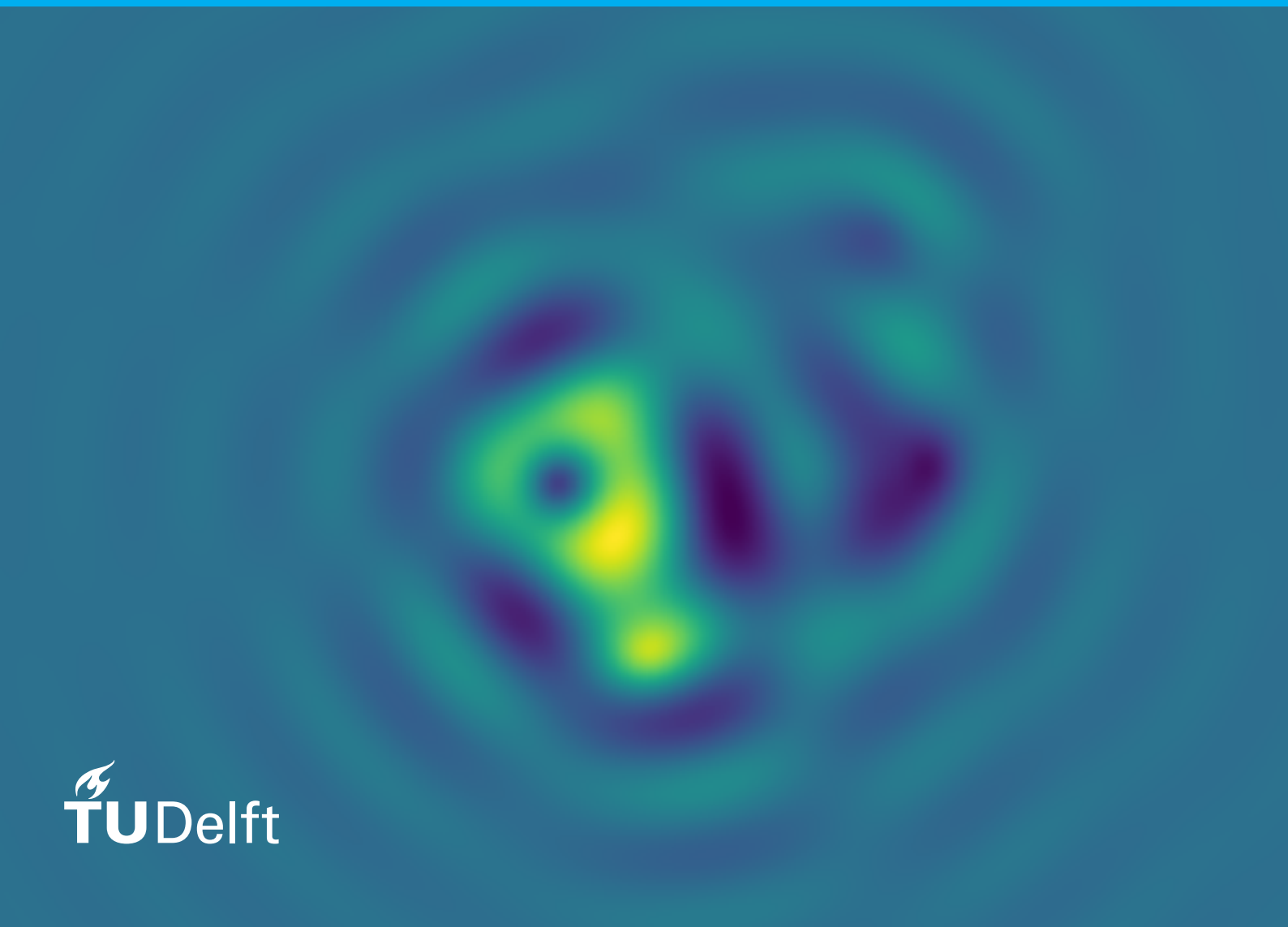
Methods for reducing error in approximations of the Rayleigh integral

Joris Perrenet

Bachelor Thesis 2022



TU Delft

Methods for reducing error in approximations of the Rayleigh integral

by

Joris Perrenet

to obtain the degree of Bachelor of Science in
Applied Mathematics and Applied Physics
at the Delft University of Technology,
to be defended publicly on Friday, 1 July, 2022 at 14:00.

Student number:	5293243	
Project duration:	24 February 2022 – 1 July 2022	
Thesis committee:	Prof. dr. ir. C. Vuik,	TU Delft, EWI supervisor
	Dr. ir. D. J. Verschuur,	TU Delft, TNW supervisor
	Dr. ir. Y. M. Dijkstra,	TU Delft, EWI
	Dr. W. A. van Dongen,	TU Delft, TNW
Additional supervision:	Ir. L.A. Hoogerbrugge,	TU Delft, TNW

An electronic version of this thesis is available at http://repository.tudelft.nl/.
An uncompiled, LaTeX version of this thesis is available at
https://github.com/jorisperrenet/BachelorThesis.

TU Delft Delft University of Technology

Preface

You are currently reading the thesis "Methods for reducing error in approximations of the Rayleigh integral". It has been written in order to obtain the degree of Bachelor of Science in Applied Mathematics and Applied Physics at the Delft University of Technology. I engaged in researching and writing this thesis from the 24th of February to the 1st of July in the year 2022.

From a young age I was captivated by prime numbers. At first, I studied them on paper, but through lack of brain power I started to write computer code. As I got older the efficiency of my computer code increased and from it originated my fascination for programming and optimizing algorithms. Researching and enhancing the calculation of the Rayleigh integral using my knowledge obtained inside and outside this Bachelor degree was therefore an interesting quest and a joyful challenge.

I would like to thank my supervisors, Kees Vuik and Eric Verschuur, as they were of great help during my bachelor thesis. I was especially thankful for the freedom they gave me to explore the topics I found appealing, whilst making sure that the topics remained relevant for present-day applications. I am also grateful for Yoeri Dijkstra and Koen van Dongen, the other members of my thesis committee, for attending the defense of my thesis and assessing this report. In addition, Leo Hoogerbrugge, currently Eric's PhD-student, provided great knowledge and assistance in our weekly meetings.

Further gratitude goes out towards my parents, my brothers, and friends. The feedback I received about the content, code and layout of this thesis was very helpful and more than appreciated.

Joris Perrenet
The Hague, July 2022

The figure at the cover is Figure 5.1a, displaying an artificial integrand of the Rayleigh integral. In this thesis we develop various methods to integrate this function, which we then compare at the end.

Abstract

The goal of seismic imaging is to create a model of the subsurface from samples of a transmitted wavefield that is reflected in soil layers. The created model can be used to locate storage possibilities for CO_2 or H_2 or to find oil and gas reservoirs or other natural resources. Seismic imaging relies on the propagation of wavefields, which can be computed by evaluating the Rayleigh integral, a process that often requires a lot of time and resources. In this thesis we develop methods to reduce error in the computation of the Rayleigh integral whilst remaining time-efficient. To achieve this we first give the derivation of the Rayleigh integral and provide a few methods to evaluate integrals numerically using interpolating polynomials. Afterwards, we adapt the integration methods to the Rayleigh integral and arrive at the main algorithms for its evaluation, at the end we compare these algorithms.

We present the method that is currently used for the computation of the Rayleigh integral and a modification that extends the method to samples on a rectilinear grid. The adjusted version of this method did not perform consistently in our results, although it achieved better results than the original method on average. Secondly we present a combined interpolation method (that is, it interpolates the full integrand without taking advantage that one of the two functions in the integrand is known) named Simpson's rule, which only performed better in the high sample points per wavelength range with minimal noise. An alteration to the method for semi-equidistant grids (see Section 4.2.2 for the definition) also did not perform consistently and is not worth the extra operations. Next was the separate interpolation algorithm (this time taking advantage of the known part of the integrand), this method is difficult to implement and had only slightly better performance than the combined method. The final method was an extension of the separate interpolation algorithm to non-equidistant grids, this method performed undoubtedly the best of all presented methods (often better by a factor of 15 compared to the currently used method), this method is therefore recommended for evaluating the Rayleigh integral. The amount of operations needed for all presented methods is approximately the same and the execution time needed for the calculation depends on the implementation of the problem.

In conclusion, we found that the separate interpolation method for non-equidistant grids delivered the most accurate approximations of the Rayleigh integral. This method can be used in seismic imaging to increase the accuracy of models of the subsurface.

Contents

Chapter 1

Introduction

Imagine you are walking down the beach, and you look at the horizon. An oil drilling platform crosses your sight, and you suddenly start to wonder; how do oil companies know where to drill? How do they know that the drilling platform must be placed at exactly that spot? The answer lies in using (marine) seismic imaging, a technique based on wavefield propagation, to produce an image of the subsurface. Using such images it is possible to find storage possibilities for CO_2 or H_2 or to locate reservoirs that contain oil and gas or other natural resources.

1.1. Problem description

In Figure 1.1 we display a simplified version of marine seismic imaging. A source, typically an air gun at sea, transmits a wavefield close to the surface. A fraction of the transmitted wavefield propagates through the different layers of soil (in the figure this is displayed using arrows) and only a small part of the emitted wavefield is reflected in just the right way to be picked up by the receivers. Data processing then allows us to reconstruct a model of the subsurface by propagating the wavefield backwards, that is, from the collected data at the receivers we try to trace back the path that the waves took. Specifically, if we assume that the layers extend horizontally, the Rayleigh integral describes the propagation of a wavefield from one layer to the next. The assumption of horizontal layers is often an acceptable loss as methods without this assumption are overly complicated. In order to obtain knowledge of the different structural layers of the subsurface it is required to evaluate this integral multiple times for each layer (as waves can get reflected many times).

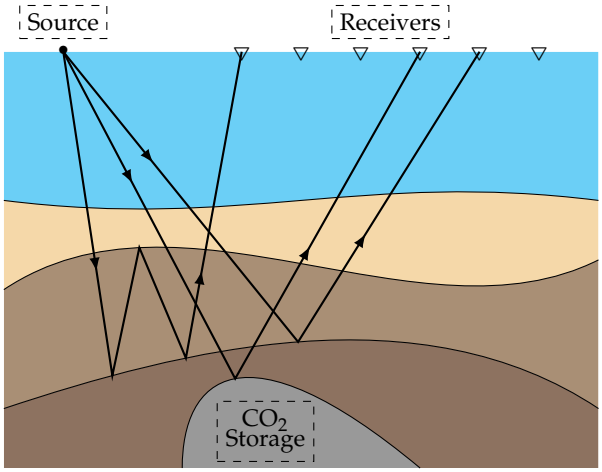


Figure 1.1: Simplified version of marine seismic acquisition to find a potential storage space for CO_2. Different properties of soil layers are denoted by different shades of brown. Often the source and the receivers are towed by a survey ship moving with a constant speed to increase the amount of data, also, the source is usually located in the midst of the receivers.

This brings us to the problem: evaluating the Rayleigh integral takes a lot of time and requires a lot of computing power, as it needs to be done many times and the integral itself is difficult to compute. Also, because the wavefield is propagated numerous times through all the layers of the subsurface, a small mistake in the beginning can amplify errors at the end. This leads us to the main research question of this thesis:

RESEARCH OBJECTIVE
To find an algorithm that reduces error in the evaluation of the Rayleigh integral compared to existing and currently used methods whilst maintaining (or lowering) the time complexity.

1.2. Thesis outline

To achieve our objective and explain our results, this thesis contains the following chapters:

- Chapter 2: *Deriving the Rayleigh integral*. This chapter contains the derivation of the Rayleigh integral. In order to derive the integral we must first derive the acoustical wave equation. This is done by combining relations from Hooke's law and Newton's law. Afterwards we transform the wave equation into the Helmholtz equation, which we then solve by using contour integration. From the solution we derive two main results: the wavefield of a point-source and the wavefield of a point-sink. A seeming paradox that there are multiple solutions to the same equation is explained further in Section 2.2.1. Using our previous results we can then derive the Kirchhoff integral, the predecessor of the Rayleigh integral. By making an assumption and using a mathematical trick developed by Rayleigh we can then derive the Rayleigh integral. The integral describes the propagation from one layer of soil to the next, which is of vital importance for seismic imaging. In following chapters we seek to find methods that can approximate the integral numerically.
- Chapter 3: *Numerical integration using interpolating polynomials*. In this chapter we propose two methods for approximating a simplified version of the Rayleigh integral. Both methods are based on interpolating the integrand by polynomials. The first method makes use of a known part of the integrand (the integrand is a product of two functions, one of which is known) and the second method does not. The first method is however more difficult to implement than the second method as it differs greatly from currently used methods. In the final section of this chapter we provide numerical examples that use both methods to approximate an integral. The corresponding computer code can be found in Appendix D. Both methods lay the groundwork for other methods we derive in the next chapter. The focus of our thesis is to compare both methods (and all methods that originate from them) to the currently used method.
- Chapter 4: *Adapting numerical integration methods for the Rayleigh integral*. In this chapter we get rid of the simplified version of the Rayleigh integral and translate the developed methods to fit the Rayleigh integral. An assumption made in the previous chapter, one that the receivers are placed exactly equidistant, is weakened. In the last section we provide a summary of all the derived methods. This is important because in the next chapter (Chapter 5) we compare the methods to the one currently used. We note that by using all the derived methods we can better approximate the Rayleigh integral, which can in turn be useful for seismic imaging, leading to lower error in models that can support drilling decisions.
- Chapter 5: *Results*. This chapter contains the results of the derived methods applied to synthetically constructed data, similar to the actual Rayleigh integral. At the end of this chapter we give a comparison of the performance of the methods and compare them to the method that is currently used. Inconsistencies in the methods are also discussed by comparing averaged data to single data. This chapter provides the foundation for the conclusion (Chapter 6) and lists the key results from this thesis.

- Chapter 6: *Conclusion and outlook*. In this chapter the main results of our thesis combined with a short introduction of the problem are given. At the end, we lay out subjects for future research, both theoretical and experimental.
- Appendices: In *Appendix A* we discuss a verification of the acoustic wave equation derived in Chapter 2, the verification is more intuitive than the one given and provides support for the correctness of the other derivation. In *Appendix B* we provide proofs for various statements in Subsection 3.2.2. *Appendix C* contains the matrices used in the numerical examples in Section 3.3, whereas *Appendix D* contains the computer code used to generate the examples. This appendix also contains the computer code for the results (Chapter 5).

1.3. Preliminaries and notational conventions

In this section we provide the notational conventions that are used throughout this thesis. Also, we provide a few Fourier transformations (including the definition) that function as lemmas for further derivations.

Notation

In general, vectors are three-dimensional and denoted by a **bold** letter, i.e. $\mathbf{v} = (v_x, v_y, v_z)^\top$ is a vector whereas ω is not. Also, we define $\mathbf{r}$ to be the position vector, thus $\mathbf{r} = (x, y, z)^\top = x\hat{\mathbf{x}} + y\hat{\mathbf{y}} + z\hat{\mathbf{z}}$ where $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ denote the unit vectors in the x, y, and z-directions, respectively.

Using the same unit vectors we define the gradient of a scalar field f as:

$$\nabla f := \hat{\mathbf{x}}\frac{\partial f}{\partial x} + \hat{\mathbf{y}}\frac{\partial f}{\partial y} + \hat{\mathbf{z}}\frac{\partial f}{\partial z}\,.$$

Note that $(\partial/\partial x)$ is used to denote the derivative in the x-directions and similar notations are used for the y and z-directions. The Laplacian of a scalar field f is given as:

$$\nabla^2 f := \hat{\mathbf{x}}\frac{\partial^2 f}{\partial x^2} + \hat{\mathbf{y}}\frac{\partial^2 f}{\partial y^2} + \hat{\mathbf{z}}\frac{\partial^2 f}{\partial z^2}\,.$$

For more information on the gradient or the Laplacian of a scalar field we refer the reader to Griffiths [1, Sections 1.2.3-1.2.7].

The Dirac delta function is defined by the following two equations:

$$\delta(x) := \begin{cases} 0 & \text{if } x \neq 0 \\ \infty & \text{if } x = 0 \end{cases}, \qquad \int_{-\infty}^{\infty} \delta(x)\mathrm{d}x = 1\,.$$

In three dimensions this definition extends to

$$\delta(\mathbf{r}) := \delta(x)\delta(y)\delta(z)\,.$$

Again, for more information the reader is referred to Griffiths [1, Sections 1.5.1-1.5.3].

We use big $\mathcal{O}$ notation (also known as Bachmann-Landau notation or asymptotic notation) to denote the time and space complexity of our algorithms. Below, we give the formal definition, for more information the reader is referred to Goodrich, Tamassia, and Goldwasser [2, Section 4.3].

> Let both $f(n)$ and $g(n)$ be real valued functions. Then $f(n)$ is $\mathcal{O}(g(n))$ if there is a real constant $c > 0$ and a real number n_0 such that:
>
> $$f(n) \leq c \cdot g(n)\,, \quad \text{for} \quad n \geq n_0\,.$$

We note that notations that are not specified above are defined when they are used.

Fourier transformations

In this subsection we provide a few Fourier transformations together with the definition, we note that the spatial variant and the temporal variant are both used in this thesis. In order to signify that we use a spatial Fourier transform we denote the spatial transformation with $\mathcal{F}\{\dots\}(\mathbf{q})$. The three-dimensional spatial Fourier transformation is defined as [3, Section 3.4]:

$$\mathcal{F}\{u\}(\mathbf{q}) = \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty} u(\mathbf{r})e^{-i\mathbf{q}\cdot\mathbf{r}}\mathrm{d}^3\mathbf{r}\,, \tag{1.1}$$

$$u(\mathbf{r}) = \int_{-\infty}^{\infty} U(\mathbf{q})e^{i\mathbf{q}\cdot\mathbf{r}}\mathrm{d}^3\mathbf{q}\,. \tag{1.2}$$

With these integrals we mean to denote the integration over all components of $\mathbf{r}$ (or $\mathbf{q}$) where each integral extends from $-\infty$ to ∞, also, $\mathbf{q}\cdot\mathbf{r}$ represents the dot product between $\mathbf{q}$ and $\mathbf{r}$, i.e. $\mathbf{q}\cdot\mathbf{r} = q_x r_x + q_y r_y + q_z r_z$.

For a temporal Fourier transformation we write a capital function letter, thus

$$x(t) \overset{\mathcal{F}}{\longleftrightarrow} X(\omega)\,,$$

where we used $\overset{\mathcal{F}}{\longleftrightarrow}$ to mark a Fourier transformation. The transformations are then given by [4, equations 4.8 & 4.9]:

$$X(\omega) := \int_{-\infty}^{\infty} x(t)e^{-i\omega t}\mathrm{d}t\,, \tag{1.3}$$

$$x(t) := \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{i\omega t}\mathrm{d}\omega\,. \tag{1.4}$$

We also derive the spatial Fourier transformation of the Dirac delta function centered around $\mathbf{0}$ (the origin), as we will use it in Section 2.2.1:

$$\begin{aligned}\delta(\mathbf{r}) \overset{\mathcal{F}}{\longleftrightarrow} \mathcal{F}\{\delta\}(\mathbf{q}) &= \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty}\delta(\mathbf{r})e^{-i\mathbf{q}\cdot\mathbf{r}}\mathrm{d}^3\mathbf{r} \\ &= \frac{1}{(2\pi)^3}e^{-i\mathbf{q}\cdot\mathbf{0}} = \frac{1}{(2\pi)^3}\,.\end{aligned} \tag{1.5}$$

(anti)causality

For Section 2.2.1 we need to establish the definition of causality. We define a system to be **causal** if the output of the system at any time solely depends on the values of the input at present or past times [4, Section 1.6.3]. Similarly, a system is called **anticausal** if the output of the system at any time solely depends on the values of the input at present or future times. Thus, the system $y(t) = x(t)$ is causal as well as anticausal, $y(t) = x(t - t_0)$ with $t_0 \geq 0$ is causal but not anticausal, and $y(t) = x(t^2)$ is neither causal nor anticausal.

Since Section 2.2.1 also uses the time shifting property of the temporal Fourier transform [4, Section 4.3.2] we derive the property here. We start with equation 1.4 and replace t by $t - t_0$:

$$\begin{aligned}x(t - t_0) &= \int_{-\infty}^{\infty} X(\omega)e^{i\omega(t-t_0)}\mathrm{d}\omega \\ &= \int_{-\infty}^{\infty}\left[e^{-i\omega t_0}X(\omega)\right]e^{i\omega t}\mathrm{d}\omega\,.\end{aligned}$$

We can thus see that

$$x(t - t_0) \overset{\mathcal{F}}{\longleftrightarrow} e^{-i\omega t_0}X(\omega)\,. \tag{1.6}$$

We end this section by noting that in the frequency domain, if we multiply $X(\omega)$ (which in this case denotes the Fourier transform of an input that solely depends on the present) by a negative exponent (i.e. $e^{-i\omega t_0}$ with $t_0 \geq 0$), we get a causal function in the time domain and vice versa. This will be important when we take on solving the Helmholtz equation in Section 2.2.1.

Chapter 2

Deriving the Rayleigh integral

In this chapter we derive the Rayleigh integral. Using this integral we can calculate the pressure wavefield at different depths in the subsurface, which is crucial for constructing a model of the subsurface.

The derivation of the integral is rather long, however important, thus we devote this entire chapter to it. Further information on the derivation can be found in Gisolf and Verschuur [5, Chapter 3-4] and in Kutscha [6, Appendix A], which provided a basis for this chapter.

We start with deriving the three-dimensional acoustic wave equation, which translates to the Helmholtz equation in the frequency domain. The Helmholtz equation is then solved, yielding two cardinal results: the wavefield of a point-source and the wavefield of a point-sink. Afterwards we derive the Kirchhoff integral, which in combination with our previous results and a trick developed by Rayleigh allows us to derive the Rayleigh integral.

2.1. Deriving the acoustic wave equation

In this section we derive the three-dimensional acoustic wave equation, since the derivation is rather algebraic we provide a more intuitive method for deriving the one-dimensional version in Appendix A. This can also be used as a verification of the algebraic approach.

We start off by considering an infinitesimal volume element with $\Delta V = \Delta x \Delta y \Delta z$, displayed in Figure 2.1, experiencing a total pressure $p_{tot}(\mathbf{r}, t)$ depending on position and time on each of its faces.

The total pressure consists of a static part $p_0(\mathbf{r})$ independent of time and a time variant excess pressure part $p(\mathbf{r}, t)$:

$$p_{tot}(\mathbf{r}, t) = p_0(\mathbf{r}) + p(\mathbf{r}, t)\,.$$

From now we only concern ourselves with the excess pressure field.

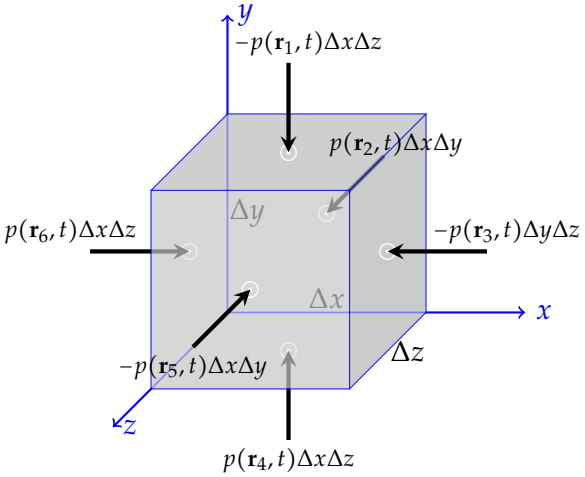


Figure 2.1: Infinitesimal volume element with dimensions Δx, Δy and Δz, experiencing forces on its six faces due to excess pressure $p(\mathbf{r}, t)$.

If there is such an excess pressure the volume element ΔV experiences displacement and deformation. If the pressure on all six faces does not balance, the volume will experience a net force, causing it to accelerate. We describe this by using Newton's (second) law.

However, we first look at the situation where the excess pressure does balance, this causes the volume element ΔV to undergo a change in volume due to confining pressure. This change can be described by using Hooke's law.

After deriving relations describing the volume element by using Hooke's law and Newton's law we can combine the relations into the three-dimensional acoustic wave equation. After solving the equation in the frequency domain we obtain the wavefield of a point-source and the wavefield of a point-sink. These two equations lay the groundwork for the derivation of the Rayleigh integral.

2.1.1. Hooke's law

We start by writing out Hooke's law, which states that (relative) changes in the volume of a volume element due to confining pressure are proportional to that pressure (with a proportionality constant that can be space variant):

$$p(\mathbf{r},t) = -K(\mathbf{r})\frac{\delta\Delta V}{\Delta V}\,, \tag{2.1}$$

where δ indicates a variation with time. To get rid of the factor $\delta\Delta V/\Delta V$ we define the displacement vector field of the mass particles in the medium to be

$$\mathbf{u}(\mathbf{r},t) = \zeta(\mathbf{r},t)\hat{\mathbf{x}} + \eta(\mathbf{r},t)\hat{\mathbf{y}} + \xi(\mathbf{r},t)\hat{\mathbf{z}}\,,$$

remembering that $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ denote the unit vectors in the x, y, and z-directions, respectively. After writing $\Delta\zeta = \zeta(x+\Delta x) - \zeta(x)$ and defining $\Delta\eta$ and $\Delta\xi$ similarly, the increase in volume can now be written as

$$\Delta V + \delta\Delta V = (\Delta x + \Delta\zeta)(\Delta y + \Delta\eta)(\Delta z + \Delta\xi)\,.$$

Bringing ΔV to the other side and writing out the right-hand side whilst omitting negligible factors (since $\Delta\zeta << \Delta x$, $\Delta\eta << \Delta y$, and $\Delta\xi << \Delta z$ for small variations in time we can neglect factors depending on products of these differences) gives us

$$\delta\Delta V = (\Delta x\Delta y\Delta z - \Delta V) + \Delta x\Delta y\Delta\xi + \Delta x\Delta z\Delta\eta + \Delta y\Delta z\Delta\zeta\,.$$

Dividing by ΔV and taking the limit for small volume elements ΔV results in

$$\lim_{\Delta V\to 0}\frac{\delta\Delta V}{\Delta V} = \frac{\partial\zeta(\mathbf{r},t)}{\partial x} + \frac{\partial\eta(\mathbf{r},t)}{\partial y} + \frac{\partial\xi(\mathbf{r},t)}{\partial z} = \nabla\cdot\mathbf{u}(\mathbf{r},t)\,.$$

This allows us to replace the factor $\delta\Delta V/\Delta V$ in Hooke's law (equation 2.1) by a component relating fields. This gives the equation

$$p(\mathbf{r},t) = -K(\mathbf{r})\nabla\cdot\mathbf{u}(\mathbf{r},t)\,.$$

Now, if we have a function $q(\mathbf{r},t)$ describing the "injected" volume into the volume element (we previously did not take this into account) we can rewrite our previous equation into

$$\nabla\cdot\mathbf{u}(\mathbf{r},t) = -\frac{p(\mathbf{r},t)}{K(\mathbf{r})} + q(\mathbf{r},t)\,. \tag{2.2}$$

2.1.2. Newton's law

Before deriving the three-dimensional acoustic wave equation we must acquire another relation using the displacement vector. To do this, we use Newton's second law, it states that "the change of motion of an object is proportional to the force impressed" [7, Section 6.2, 8, Section 1.1]. Thus, if the forces on the volume element ΔV do not balance the volume element will experience acceleration, due to the element having mass and therefore inertia. To put this into formulas, we work out the net force acting on the volume element in Figure 2.1:

$$\begin{aligned}
\Delta F_x &= \Delta y \Delta z (p(x,y,z,t) - p(x+\Delta x,y,z,t)) \approx -\Delta x \Delta y \Delta z \frac{\partial p}{\partial x}, \\
\Delta F_y &= \Delta x \Delta z (p(x,y,z,t) - p(x,y+\Delta y,z,t)) \approx -\Delta x \Delta y \Delta z \frac{\partial p}{\partial y}, \\
\Delta F_z &= \Delta x \Delta y (p(x,y,z,t) - p(x,y,z+\Delta z,t)) \approx -\Delta x \Delta y \Delta z \frac{\partial p}{\partial z}.
\end{aligned}$$

This can be rewritten as

$$\Delta \mathbf{F} = -\Delta V \nabla p(\mathbf{r},t) = \Delta m \mathbf{a},$$

where it can be noted that the last equality follows from Newton's second law and that Δm denotes the mass of the volume element (which can be written as $\rho \Delta V$, where ρ denotes the space variant mass density). After dividing by ΔV, this yields

$$-\nabla p(\mathbf{r},t) = \rho(\mathbf{r}) \frac{\partial^2 \mathbf{u}(\mathbf{r},t)}{\partial t^2}.$$

If we now add an external source $\mathbf{f}(\mathbf{r},t)$ we get extra forces on the volume element, which results in a change of the previous formula, we obtain

$$\mathbf{f}(\mathbf{r},t) - \nabla p(\mathbf{r},t) = \rho(\mathbf{r}) \frac{\partial^2 \mathbf{u}(\mathbf{r},t)}{\partial t^2}. \tag{2.3}$$

2.1.3. Combining the two results

In this subsection we combine our results from Hooke's law and Newton's law into the three-dimensional acoustic wave equation.

To bring this about, we continue with equation 2.3, take the divergence and divide by the mass density:

$$\nabla \cdot \frac{\mathbf{f}(\mathbf{r},t)}{\rho(\mathbf{r})} - \nabla \cdot \left[\frac{\nabla p(\mathbf{r},t)}{\rho(\mathbf{r})} \right] = \nabla \cdot \frac{\partial^2 \mathbf{u}(\mathbf{r},t)}{\partial t^2}.$$

Taking the second time derivative of equation 2.2 results in

$$\nabla \cdot \frac{\partial^2 \mathbf{u}(\mathbf{r},t)}{\partial t^2} = -\frac{1}{K(\mathbf{r})} \frac{\partial^2 p(\mathbf{r},t)}{\partial t^2} + \frac{\partial^2 q(\mathbf{r},t)}{\partial t^2}.$$

If we now equate the previous two equations we obtain

$$\nabla \cdot \frac{\mathbf{f}(\mathbf{r},t)}{\rho(\mathbf{r})} - \nabla \cdot \left[\frac{\nabla p(\mathbf{r},t)}{\rho(\mathbf{r})} \right] = -\frac{1}{K(\mathbf{r})} \frac{\partial^2 p(\mathbf{r},t)}{\partial t^2} + \frac{\partial^2 q(\mathbf{r},t)}{\partial t^2}. \tag{2.4}$$

In order to rewrite this equation we define

$$s(\mathbf{r},t) := \frac{\partial^2 q(\mathbf{r},t)}{\partial t^2} - \nabla \cdot \frac{\mathbf{f}(\mathbf{r},t)}{\rho(\mathbf{r})}.$$

After substituting the new definition into equation 2.4 and reorganizing we get

$$\nabla \cdot \left[\frac{\nabla p(\mathbf{r},t)}{\rho(\mathbf{r})} \right] - \frac{1}{K(\mathbf{r})} \frac{\partial^2 p(\mathbf{r},t)}{\partial t^2} = -s(\mathbf{r},t)\,.$$

If we consider a homogeneous medium, ρ and K will no longer be space variant, yielding the relation

$$\nabla^2 p(\mathbf{r},t) - \frac{1}{c^2} \frac{\partial^2 p(\mathbf{r},t)}{\partial t^2} = -s(\mathbf{r},t)\,, \tag{2.5}$$

with $c = \sqrt{K/\rho}$. This is the three-dimensional acoustic wave equation (for homogeneous media), an important result as we will solve this equation in the frequency domain in the next section. Afterwards, we use the solutions to specify a known part of the Rayleigh integral.

2.2. Wavefield of a single point-source

In this section we continue with the acoustic wave equation derived in the previous section (Section 2.1). First, we transfer it to the frequency domain giving the Helmholtz equation, which we solve by deriving its Green's functions in the coming subsection (Subsection 2.2.1). The result corresponds to a known part of the Rayleigh integral, this part is crucial to derive as some developed integration methods make use of this.

Taking the three-dimensional acoustic wave equation (equation 2.5) and substituting $-s(\mathbf{r},t)$ with a single point-source at $\mathbf{r}_s$, whilst assuming that the point-source sends out one delta pulse, gives

$$\nabla^2 p(\mathbf{r},t) - \frac{1}{c^2} \frac{\partial^2 p(\mathbf{r},t)}{\partial t^2} = \delta(t)\delta(\mathbf{r}-\mathbf{r}_s)\,. \tag{2.6}$$

In order to rewrite this equation, we first note that taking the time derivative on both sides of the inverse Fourier transformation in equation 1.4 results in

$$\frac{\partial x(t)}{\partial t} \overset{\mathcal{F}}{\longleftrightarrow} i\omega X(\omega)\,.$$

That is, temporal differentiation in the time domain results in multiplication with $i\omega$ in the frequency domain. Also, the Fourier transform of a delta function can be derived from equation 1.3:

$$\delta(t) \overset{\mathcal{F}}{\longleftrightarrow} \int_{-\infty}^{\infty} \delta(t) e^{-i\omega t} \mathrm{d}t = 1\,.$$

Using these relations, the Fourier transform of equation 2.6 can be evaluated in order to obtain the Helmholtz equation:

$$\nabla^2 P(\mathbf{r},\omega) + \frac{\omega^2}{c^2} P(\mathbf{r},\omega) = \delta(\mathbf{r}-\mathbf{r}_s)\,. \tag{2.7}$$

In the next subsection (Subsection 2.2.1) we solve the above equation, however, to set an objective, we first give the wavefield of a point-source and the wavefield of a point-sink (both solutions to the equation):

$$P_{src}(\mathbf{r},\omega) = \frac{e^{-i\omega|\mathbf{r}-\mathbf{r}_s|/c}}{4\pi|\mathbf{r}-\mathbf{r}_s|} \quad \text{and} \quad P_{snk}(\mathbf{r},\omega) = \frac{e^{i\omega|\mathbf{r}-\mathbf{r}_s|/c}}{4\pi|\mathbf{r}-\mathbf{r}_s|}\,. \tag{2.8}$$

2.2.1. Solving the Helmholtz equation

In this subsection we prove the validity of equation 2.8 (the equation for the wavefield of a point-source and a point-sink) by solving the Helmholtz equation (equation 2.7). The seeming paradox that both equations are solutions to the same equation is also resolved in this

subsection. The solutions to the equations are used again in Section 2.4 where we derive the Rayleigh integral.

We solve equation 2.7 by deriving its Green's functions. Due to symmetry we know that any Green's function will only depend on its magnitude of the distance between $\mathbf{r}$ and $\mathbf{r}_s$. Thus after denoting $\mathbf{r} - \mathbf{r}_s$ by $\mathbf{d}$, any Green's function will eventually only depend on $|\mathbf{d}|$. For now, we define $k = \omega/c$ and after noting that we assume k to be constant we rewrite the Helmholtz equation (equation 2.7) as (we omit the dependency on ω to avoid confusion)

$$(\nabla^2 + k^2)G(\mathbf{d}) = \delta(\mathbf{d})\,.$$

If we apply the spatial Fourier transform in equation 1.1 to the newly rewritten Helmholtz equation and define $|\mathbf{q}|^2 = q_x^2 + q_y^2 + q_z^2$ (remember that the Fourier transform of a delta function is given by equation 1.5, the differentiation property is given by Čertik [3, Section 3.4.3] and can be easily verified by taking the derivative of equation 1.1), we get

$$(k^2 - |\mathbf{q}|^2)\mathcal{F}\{G\}(\mathbf{q}) = \frac{1}{(2\pi)^3}\,.$$

Rewriting this equation and taking the inverse spatial Fourier transform from equation 1.2 we obtain

$$G(\mathbf{d}) = -\frac{1}{(2\pi)^3}\int_{-\infty}^{\infty}\frac{e^{i\mathbf{q}\cdot\mathbf{d}}}{|\mathbf{q}|^2 - k^2}\mathrm{d}^3\mathbf{q}\,, \tag{2.9}$$

which is of the form

$$\int_{-\infty}^{\infty} f(q)e^{i\mathbf{q}\cdot\mathbf{d}}\mathrm{d}^3\mathbf{q}\,,$$

where q is used to denote the magnitude of $\mathbf{q}$, i.e. $|\mathbf{q}|$. Since $f(q)$ does not depend on the direction of $\mathbf{q}$, this is equivalent to

$$\frac{4\pi}{R}\int_0^{\infty} qf(q)\sin(qR)\mathrm{d}q\,,$$

with $R = |\mathbf{d}| = |\mathbf{r} - \mathbf{r}_s|$. For the proof of this statement the reader is referred to Barton and Barton [9, Appendix F]. Rewriting equation 2.9 by using the equivalence from above gives us

$$G(R) = -\frac{4\pi}{(2\pi)^3 R}\int_0^{\infty}\frac{q}{q^2 - k^2}\sin(qR)\mathrm{d}q\,.$$

Note that we expected that any Green's function would only depend on the magnitude of $\mathbf{r} - \mathbf{r}_s$ (i.e. the newly defined R), which is indeed the case. Since the integrand is even, we write:

$$G(R) = -\frac{4\pi}{(2\pi)^3 2R}\int_{-\infty}^{\infty}\frac{q}{q^2 - k^2}\sin(qR)\mathrm{d}q\,.$$

The sine function can be expressed as a linear combination of exponentials:

$$\sin(qR) = \frac{e^{iqR} - e^{-iqR}}{2i}\,,$$

which after substitution results in the following relation for $G(R)$:

$$G(R) = -\frac{4\pi}{(2\pi)^3 4iR}\left[\underbrace{\int_{-\infty}^{\infty}\frac{qe^{iqR}}{(q-k)(q+k)}\mathrm{d}q}_{I_1} - \underbrace{\int_{-\infty}^{\infty}\frac{qe^{-iqR}}{(q-k)(q+k)}\mathrm{d}q}_{I_2}\right]. \tag{2.10}$$

We wish to find the integrals I_1 and I_2. First we note that after a substitution of $q' = -q$ in I_2 we get $-I_1$ (the intervals also change due to this substitution resulting in the final minus sign). Thus, all Green's function satisfy

$$G(R) = -\frac{4\pi}{(2\pi)^3 2iR} I_1 = -\frac{1}{4\pi^2 iR} I_1 .$$

We note that different Green's functions originate from different values of I_1. Its value is evaluated further by means of contour integration.

We use a contour in the complex q plane. However, due to poles on the real q-axis we first add the term $+i\epsilon$ to k and take the limit $\epsilon \to 0$:

$$I_1 = \lim_{\epsilon \to 0} \int_{-\infty}^{\infty} \underbrace{\frac{z e^{izR}}{(z - (k + i\epsilon))(z + (k + i\epsilon))}}_{f(z)} \mathrm{d}z . \tag{2.11}$$

We can now see that $f(z)$ has poles at $z = k + i\epsilon$ and $z = -k - i\epsilon$, therefore, we take a contour in the upper half-plane, denoted by $\mathcal{C}$. To clarify; the contour consists of the interval $[-A, A]$ and the semicircle $C_A = \{Ae^{i\theta} | \theta \in [0, \pi]\}$. From the residue theorem we get

$$\begin{aligned} \int_{\mathcal{C}} f(z)\mathrm{d}z &= 2\pi i \lim_{\epsilon \to 0} \left(\mathrm{Res}_{z=k+i\epsilon}[f(z)]\right) \\ &= 2\pi i \lim_{\epsilon \to 0} \frac{(k + i\epsilon) e^{i(k+i\epsilon)R}}{2(k + i\epsilon)} \\ &= \pi i e^{ikR} , \end{aligned}$$

yielding the relation

$$I_1 + \int_{C_A} f(z) = \pi i e^{ikR} .$$

We now show that $\int_{C_A} f(z) \to 0$ as $A \to \infty$. From Jordan's lemma [10, Section 74] with $f(z) = e^{izR} g(z)$ where $z \in C_A$ and R is strictly positive, we get

$$\begin{aligned} \left| \int_{C_A} f(z)\mathrm{d}z \right| &\leq \frac{\pi}{R} \max_{\theta \in [0,\pi]} |g(Ae^{i\theta})| \\ &= \frac{\pi}{R} \max_{\theta \in [0,\pi]} \left| \frac{Ae^{i\theta}}{(Ae^{i\theta} - k)(Ae^{i\theta} + k)} \right| \\ &= \frac{\pi}{R} \max_{\theta \in [0,\pi]} \frac{A}{|A^2 e^{2i\theta} - k^2|} \\ &= \frac{\pi}{R} \frac{A}{A^2 - k^2} . \end{aligned}$$

We can thus see that $\int_{C_A} f(z) \to 0$ as $A \to \infty$. Hence we have $I_1 = \pi i e^{ikR}$. Note that if $R = 0$ we get $I_1 = I_2$ in equation 2.10 and due to R being in the denominator we get $G(0) = \delta(0)$, in this case, we did not have to use contour integration.

In addition, the option to add $-i\epsilon$ instead of $+i\epsilon$ to k in equation 2.11 is also possible, for that matter, we could even choose to add $-i\epsilon$ to the first occurrence of k and $+i\epsilon$ to the second or vice versa[1]. All these options correspond to a different propagator, yielding a different result to the integral [11, Section 3.6]. If we add $-i\epsilon$ instead of $+i\epsilon$ we get a pole in $\mathcal{C}$ at $z = -k + i\epsilon$, after

[1] We also could have chosen to evaluate I_2. To achieve this, one can take a contour in the lower half plane and apply an equivalent form of Jordan's lemma. This removes the minus sign in front of $G(R)$.

following the same steps as above we get the result $I_1 = \pi i e^{-ikR}$. Since this results comes from the pole at $-k$ we denote it as $I_1^- = \pi i e^{-ikR}$ and the result from above as $I_1^+ = \pi i e^{ikR}$. The other two results (from adding $-i\epsilon$ and $+i\epsilon$ or $+i\epsilon$ and $-i\epsilon$) are 0 and $2\pi i \cos(kR)$, respectively (note that 0 is a correct solution for I_1 since we divide by R, which gives $G(R) = \delta(R)$, a solution to equation 2.7). The selected option of adding $\pm i\epsilon$'s determines the corresponding physical interpretation [12, Section 11.2.2]. For example, I_1^- represents *outgoing* spherical waves from $\mathbf{r}_s$ (causal), whereas I_1^+ represents *incoming* spherical waves (anticausal). Note that this is not the other way around because we are in the (temporal) frequency domain and an exponential in the frequency domain leads to a time shift in the time domain (equation 1.6, noting that $k = \omega/c$) [5, Section 4.3]. The other two results (or any different linear combination of I_1^+ and I_1^-) are hereafter not used, they do not have an important physical meaning for our purposes. For more information on this subject the reader is referred to [11, Section 3.6, 12, Section 11.2, 5, Section 4.3, 13, 14, 15, 16, 17, 18].

From combining these results we obtain the Green's functions (note that if $R = 0$ we still have $G(0) = \delta(0)$, the same expression as we derived without contour integration)

$$\begin{aligned} G^{\pm}(R) &= -\frac{1}{4\pi^2 iR}(\pi i e^{\pm ikR}) \\ &= -\frac{e^{\pm ikR}}{4\pi R}\,. \end{aligned}$$

We have thus shown that

$$G^{\pm}(\mathbf{r},\omega) = -\frac{e^{\pm i\omega|\mathbf{r}-\mathbf{r}_s|/c}}{4\pi|\mathbf{r}-\mathbf{r}_s|} \tag{2.12}$$

are Green's functions of the Helmholtz equation (equation 2.7). From this we define $P_{src}(\mathbf{r},\omega) = -G^-(\mathbf{r},\omega)$ because it represents outgoing spherical waves and $P_{snk}(\mathbf{r},\omega) = -G^+(\mathbf{r},\omega)$ because it represents incoming spherical waves (note that the minus sign in front of these definitions is chosen because of $-s(\mathbf{r},t)$ in equation 2.5).

2.3. Deriving the Kirchhoff integral

After solving the Helmholtz equation we can derive an independent result; the Kirchhoff integral. This is the last step before deriving the Rayleigh integral in Section 2.4.

If we take the Fourier transform of the three-dimensional acoustic wave equation in the absence of sources, i.e. equation 2.5 with $s(\mathbf{r},t) = 0$, we get

$$\nabla^2 P(\mathbf{r},\omega) + \frac{\omega^2}{c^2} P(\mathbf{r},\omega) = 0\,. \tag{2.13}$$

Writing down the same equation for a point-source at $\mathbf{r}_s$ located inside the volume V displayed in Figure 2.2 (which gives $-s(\mathbf{r},t) = -\delta(t)\delta(\mathbf{r}-\mathbf{r}_s)$) while using G to denote the causal Green's function from equation 2.8 (i.e. P_{src}), we get

$$\nabla^2 G(\mathbf{r},\omega) + \frac{\omega^2}{c^2} G(\mathbf{r},\omega) = -\delta(\mathbf{r}-\mathbf{r}_s)\,. \tag{2.14}$$

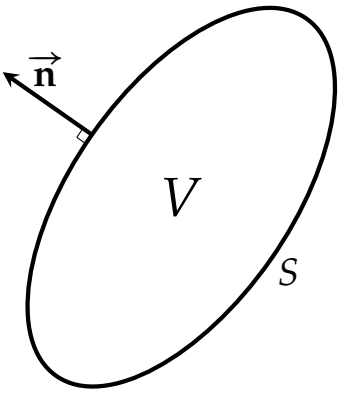


Figure 2.2: Surface S enclosing a volume V with normal vector $\vec{\mathbf{n}}$ pointed outward (enclosing a source placed at $\mathbf{r}_s$).

Now, multiplying equation 2.13 with $-G(\mathbf{r},\omega)$ and adding it to equation 2.14 multiplied by $P(\mathbf{r},\omega)$ and afterwards integrating over the volume V, leads to

$$\int_V P(\mathbf{r},\omega)\nabla^2 G(\mathbf{r},\omega) - G(\mathbf{r},\omega)\nabla^2 P(\mathbf{r},\omega)\mathrm{d}V = \int_V -P(\mathbf{r},\omega)\delta(\mathbf{r}-\mathbf{r}_s)\mathrm{d}V = -P(\mathbf{r}_s,\omega)\,.$$

Now Green's theorem [1, Section 1.3.4] gives the Kirchhoff integral

$$P(\mathbf{r}_s,\omega) = -\oint_S [P(\mathbf{r},\omega)\nabla G(\mathbf{r},\omega) - G(\mathbf{r},\omega)\nabla P(\mathbf{r},\omega)]\cdot\vec{\mathbf{n}}\,\mathrm{d}S\,. \tag{2.15}$$

2.4. Deriving the Rayleigh integral

In this section we finally derive the Rayleigh integral, this is done by using a mathematical trick due to Rayleigh. The integral describes the propagation from one layer of the subsurface to the next assuming that the layers extend horizontally, an important result for (marine) seismic imaging.

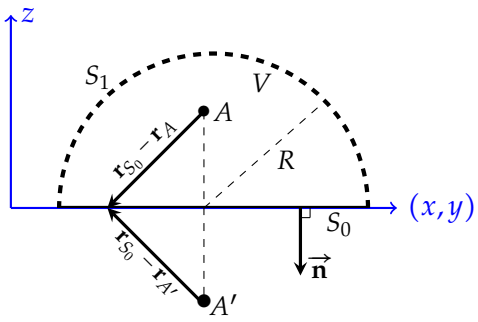


Figure 2.3: All sources generating the wavefield P are below the flat plane S_0 with radius R. The semisphere S_1 also has radius R and we consider the case where $R \to \infty$. The normal vector of S_0, i.e. $\vec{\mathbf{n}}$, is defined outward. The prediction point is denoted by A, which lies inside the volume V. The source point generated by Γ is in A', which is the mirror position of A with respect to the plane S_0.

Rayleigh's main idea was that we can add a homogeneous solution $\Gamma(\mathbf{r},\omega)$ of the partial differential equation 2.7 to the Green's function (an inhomogeneous solution) to obtain another solution of the differential equation. For the remainder of this section we denote P, G and Γ instead of $P(\mathbf{r},\omega)$, $G(\mathbf{r},\omega)$ and $\Gamma(\mathbf{r},\omega)$, respectively, whenever the chance of confusion is low.

Causality of the Kirchhoff integral in equation 2.15 tells us that the same integral holds if we go from a source point $\mathbf{r}_s$ to a prediction point $\mathbf{r}_A$ [5, Section 4.3] (that is, instead of transmitting a wavefield in $\mathbf{r}_s$ we can also predict the wavefield in the same point, denoted by the prediction point $\mathbf{r}_A$). Combining the homogeneous solution Γ with the abbreviated notation allows us to rewrite equation 2.15 into

$$P(\mathbf{r}_A,\omega) = -\oint_S [P\nabla(G+\Gamma) - (G+\Gamma)\nabla P]\cdot\vec{\mathbf{n}}\,\mathrm{d}S\,.$$

If we apply this integral to the situation displayed in Figure 2.3, we can replace S by S_0 in the above integral. This can be justified since as $R \to \infty$ the wavefield P generated below S_0 can only propagate outward with respect to the surface S_1 and thus can not (causally) contribute to any knowledge of the wavefield in A.

The trick was to let the field Γ be generated in A'. If we let $G(\mathbf{r}_A,\mathbf{r}_{S_0}) = -\Gamma(\mathbf{r}_{A'},\mathbf{r}_{S_0})$ everywhere on S_0 then G and Γ cancel each other on S_0 (if we assume that S_0 is indeed a flat plane). This can be done by making Γ a point sink whenever G is a point-source. Because the wavefields cancel each other, continuity gives that the normal components of their gradients must be equal on S_0, i.e. $\nabla G(\mathbf{r}_{S_0}) = \nabla\Gamma(\mathbf{r}_{S_0})$, which is perfect, as the integral can now be rewritten into:

$$P(\mathbf{r}_A,\omega) = -2\oint_{S_0} [P\nabla G]\cdot\vec{\mathbf{n}}\,\mathrm{d}S_0\,.$$

If the surface S_0 is assumed to be in the x,y-plane we get $\nabla G\cdot\mathbf{n} = -\partial G/\partial z$ due to an outward pointing normal vector. Hence, the integral becomes:

$$P(\mathbf{r}_A,\omega) = 2\oint_{S_0} P\frac{\partial G}{\partial z}\mathrm{d}S_0\,. \tag{2.16}$$

Remember that the (causal) Green's function is given by equation 2.8:

$$G(\mathbf{r}_A,\mathbf{r};\omega) = \frac{e^{-i\omega|\mathbf{r}-\mathbf{r}_A|/c}}{4\pi|\mathbf{r}-\mathbf{r}_A|}\,.$$

After defining $\Delta r = |\mathbf{r} - \mathbf{r}_A|_{z=0} = \sqrt{(x-x_A)^2 + (y-y_A)^2 + z_A^2}$, we note that

$$\frac{\partial}{\partial z}(|\mathbf{r} - \mathbf{r}_A|)_{z=0} = \left[\frac{(z-z_A)}{\sqrt{(x-x_A)^2 + (y-y_A)^2 + (z-z_A)^2}}\right]_{z=0} = \frac{-z_A}{\Delta r},$$

giving us the ability to compute the derivative with respect to z in the x,y-plane of the (causal) Green's function:

$$\begin{aligned}\left(\frac{\partial G}{\partial z}\right)_{z=0} &= \left[\frac{4\pi|\mathbf{r}-\mathbf{r}_A|\frac{\partial}{\partial z}(e^{-i\omega|\mathbf{r}-\mathbf{r}_A|/c}) - e^{-i\omega|\mathbf{r}-\mathbf{r}_A|/c}\frac{\partial}{\partial z}(4\pi|\mathbf{r}-\mathbf{r}_A|)}{(4\pi|\mathbf{r}-\mathbf{r}_A|)^2}\right]_{z=0} \\ &= \frac{\Delta r \frac{\partial}{\partial z}(-i\omega|\mathbf{r}-\mathbf{r}_A|/c)_{z=0}\, e^{-i\omega\Delta r/c} + e^{-i\omega\Delta r/c}\frac{z_A}{\Delta r}}{4\pi\Delta r^2} \\ &= \frac{\Delta r i\omega \frac{z_A}{\Delta r}/c + \frac{z_A}{\Delta r}}{4\pi\Delta r^2} e^{-i\omega\Delta r/c} \\ &= \frac{z_A(1+i\omega\Delta r/c)}{4\pi\Delta r^3} e^{-i\omega\Delta r/c}.\end{aligned}$$

Plugging this in the integral from equation 2.16 gives us the long-sought Rayleigh integral:

$$P(\mathbf{r}_A,\omega) = \frac{z_A}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P(x,y,0;\omega)\left(1 + i\omega\frac{\Delta r}{c}\right)\frac{e^{-i\omega\Delta r/c}}{\Delta r^3}\mathrm{d}x\mathrm{d}y\,. \tag{2.17}$$

Often, however, the far-field approximation is used (a valid approximation if the prediction point is numerous wavelengths away from the source), which is the result of dropping the 1+ term in the above equation:

$$P(\mathbf{r}_A,\omega) = \frac{i\omega z_A}{2\pi c}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P(x,y,0;\omega)\frac{e^{-i\omega\Delta r/c}}{\Delta r^2}\mathrm{d}x\mathrm{d}y\,.$$

Chapter 3

Numerical integration using interpolating polynomials

In Chapter 2 we derived the Rayleigh integral, whereas the goal of our thesis is to find algorithms that can compute the integral (whilst reducing error and being time-efficient). To find these algorithms we first simplify the integral. Afterwards we interpolate the integrand (or a part of it) by polynomials, which we can then integrate to approximate the simplified integral. In the next chapter (Chapter 4) we extend these simplified methods to the Rayleigh integral and in Chapter 5 we look at results from those methods. This chapter thus forms a basis for the numerical integration methods used throughout this thesis.

First, we present the simplification of the Rayleigh integral (equation 2.17):

We consider an integral on the interval $[x_L, x_R]$, where we define $f : \mathbb{R} \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$ with $f(x) = g(x) = 0$ for $x \notin [x_L, x_R]$[a] and let $a \in \mathbb{R}$, the simplification is to evaluate the integral (which is a function of a)

$$\int_{x_L}^{x_R} f(x)g(x+a)\mathrm{d}x\,. \tag{3.1}$$

[a] We make this approximation (it is an approximation since the wavefield is small for large distances away from the source) due to limitations of our algorithms. Also, in reality, the wavefield for the pressure must be measured/sampled and for this only a finite grid of sensors is used (so for the pressure function this assumption is the best we can do).

Note that the Rayleigh integral (equation 2.17) is also an integral over the product of two functions, that is, the derivative of Green's function and the function for the pressure at $z = 0$, where the Green's function gets shifted after changing the prediction point $\mathbf{r}_A$ (this must be done repeatedly in seismic imaging since we need to propagate the wavefield from each layer to the subsequent, needing roughly the same grid of points where the pressure is known), which translates to changing the constant a in the above integral. To clarify, in the above integral $f(x)$ represents the pressure part of the Rayleigh integral and $g(x)$ represents the derivative of the Green's function. We also note that the derivative of Green's function (and therefore $g(x)$ in our simplification) is known.

The first section of this chapter presents an algorithm based on separately interpolating the two parts of the integral (i.e. the known part $g(x)$ and the unknown part $f(x)$) before integrating, Subsection 4.1.1 will continue on this subject. The second section will focus on the derivation of a formula that evaluates the above integral by interpolating the product of the two parts, Subsection 4.2.2 discusses the extension of this method to the Rayleigh integral.

Specifically, in the second section, we derive the 2nd-order Newton-Cotes equation called Simpson's rule, afterwards we will discuss a more general version of this equation. In the third and final section we present numerical examples for the evaluation of integrals using the methods from the previous two sections, which can be used as an example for future research planning to implement the methods for the evaluation of the Rayleigh integral. Also, this is done to clarify the methods described.

Only basic methods of polynomial interpolation are discussed, mostly due to our main focus to approximate the above integral. For more information on interpolatory methods the reader is referred to Kythe and Schäferkotter [19] and Vuik et al. [20, Chapter 2].

3.1. Separate interpolation

To approximate the simplified integral in equation 3.1, we will interpolate the functions $f(x)$ and $g(x)$ separately.

To achieve this, the interval $[x_L, x_R]$ is first partitioned into ℓ equally large subintervals $[x_0, x_1], [x_1, x_2], \ldots, [x_{\ell-1}, x_\ell]$. That is, $x_1 - x_0 = x_2 - x_1 = \cdots = x_\ell - x_{\ell-1}$ and $x_L = x_0$ and $x_R = x_\ell$. Because we are integrating, closed and open intervals are not distinguished and are therefore always denoted as closed intervals.

For each interval $[x_i, x_{i+1}]$ with $0 \le i \le \ell - 1$ both function are interpolated by polynomials, in Section 3.2.1 there is a simple method to achieve this, but other methods like spline interpolation [20, Section 2.5] can also be used. We write

$$f_i(x) = \sum_{j=0}^{n_f} b_{ij} x^j, \quad g_i(x) = \sum_{k=0}^{n_g} c_{ik} x^k$$

for each interval $[x_i, x_{i+1}]$ where $f_i(x)$ and $g_i(x)$ are polynomials of degree at most n_fth and n_gth, respectively. Using the fact that the intervals are equidistant the interpolation can be achieved with a time complexity of $\mathcal{O}(n_f^2 \ell + n_g^2 \ell)$ [21]. Furthermore, since we need to store all values of b_{ij} and c_{ik} the space complexity is $\mathcal{O}(n_f \ell + n_g \ell)$.

We assume the prediction points (the points corresponding to different values of a in equation 3.1) are spaced equidistantly (this is often the case). Therefore, we let a be a multiple of the interval width $(x_1 - x_0)$, we can assume this because the integral width was previously arbitrary. Provided that we write $a = m(x_1 - x_0)$ with $m \in \mathbb{N}$ and $0 \le m \le s$, where s denotes the number of prediction points (i.e. the number of sensors), the integral in equation 3.1 becomes (remember that $f(x) = g(x) = 0$ for $x \notin [x_L, x_R]$)

$$\begin{aligned}
\int_{x_L}^{x_R} f(x')g(x'+a)\mathrm{d}x' &= \int_{x_L-a}^{x_R-a} f(x-a)g(x)\mathrm{d}x \\
&\approx \sum_{i=0}^{\ell-m} \int_{x_i}^{x_{i+1}} f_i(x-a)g_i(x)\mathrm{d}x \\
&= \sum_{i=0}^{\ell-m} \int_{x_i}^{x_{i+1}} f_{i+m}(x)g_i(x)\mathrm{d}x \\
&= \sum_{i=0}^{\ell-m} \int_{x_i}^{x_{i+1}} \sum_{j=0}^{n_f}\sum_{k=0}^{n_g} b_{i+m,j}c_{ik}x^{j+k}\mathrm{d}x \\
&= \sum_{i=0}^{\ell-m}\sum_{j=0}^{n_f}\sum_{k=0}^{n_g} b_{i+m,j}c_{ik} \int_{x_i}^{x_{i+1}} x^{j+k}\mathrm{d}x \\
&= \sum_{i=0}^{\ell-m}\sum_{j=0}^{n_f} b_{i+m,j} \sum_{k=0}^{n_g} c_{ik} \frac{x_{i+1}^{j+k+1} - x_i^{j+k+1}}{j+k+1}.
\end{aligned} \tag{3.2}$$

Storing the values of

$$d_{ij} = \sum_{k=0}^{n_g} c_{ik} \frac{x_{i+1}^{j+k+1} - x_i^{j+k+1}}{j+k+1}, \tag{3.3}$$

for all i and j then allows us to rewrite equation 3.2, giving

$$\int_{x_L}^{x_R} f(x)g(x+a)\mathrm{d}x \approx \sum_{i=0}^{\ell-m} \sum_{j=0}^{n_f} b_{i+m,j} d_{ij}. \tag{3.4}$$

After storing the values of $(x_{i+1}^{j+k+1} - x_i^{j+k+1})/(j+k+1)$ for all $j+k$ and i with a space and time complexity of $\mathcal{O}(n_f\ell + n_g\ell)$, we can calculate the values of d_{ij} with a time complexity of $\mathcal{O}(n_f n_g \ell)$ and a space complexity of $\mathcal{O}(n_f\ell)$.

To compute the space and time complexity of approximating the integral, we note that initializing this algorithm can be done in $\mathcal{O}((n_f+n_g)^2\ell)$ time and $\mathcal{O}(n_f\ell + n_g\ell)$ space. The algorithm would then step through all the prediction points s, calculating the sum in equation 3.4, having an overall time complexity of $\mathcal{O}(n_f\ell s)$ and a total space complexity of $\mathcal{O}(n_f\ell + n_g\ell + s)$ since we need to store the results as well.

Note that the final time complexity is independent of n_g. Since the (derivative of) Green's function is known, we could interpolate it by using a polynomial of a high degree. This is an important result as we can make use of it to improve the accuracy of our approximation. Also, we can increase the domain on which we interpolate the derivative of Green's function, reducing the approximation made in the simplified version of the Rayleigh integral.

3.2. Combined interpolation

In this section we present another method of finding the integral in equation 3.1, the method relies on interpolating the product of the functions $f(x)$ and $g(x)$.

The method is based on the Newton-Cotes equations and implementing these equations is generally simple. Furthermore, by using this method, we can reduce the space complexity compared to the method in Section 3.1. However, in some situations we present in Section 3.3 and Chapter 5 the accuracy of this method is lacking. Nevertheless, this method still provides a cornerstone for one of the two methods used to evaluate an artificial Rayleigh integral in Chapter 5.

In the first subsection we derive the 2nd-order Newton-Cotes equation commonly known as Simpson's rule, the derivation, however, requires knowledge of interpolating with polynomials, for which Newton polynomials are used. In the next subsection we discuss the generalization of the Newton-Cotes equations, which can be described by the Cotesian numbers. Then, we present a method to go from the (generalized) equations to approximating integrals, where we introduce notation that will also be used in following chapters.

3.2.1. Derivation of Simpson's rule

To derive the 2nd-order Newton-Cotes equation named Simpson's rule, we first need to be able to interpolate 3 points by a parabola (or a polynomial of lower order). Below we present the general method of Newton polynomials that allows us to interpolate $n+1$ points by a polynomial.

Newton Polynomials

Theorem

Let $f : \mathbb{R} \to \mathbb{R}$ denote the function that we want to interpolate on the points $x_0, x_1, \ldots, x_n$. For compacter notation we write f_i instead of $f(x_i)$ from now on[a].

After writing

$$\begin{array}{cccccc} x_0 & f_0 & & & & \\ & & \frac{f_1-f_0}{x_1-x_0} & & & \\ x_1 & f_1 & & \frac{\frac{f_2-f_1}{x_2-x_1}-\frac{f_1-f_0}{x_1-x_0}}{x_2-x_0} & & \\ & & \frac{f_2-f_1}{x_2-x_1} & & \ddots & \\ x_2 & f_2 & & \vdots & & \\ & & \vdots & & \iddots & \\ \vdots & \vdots & & \cdots & & \\ & & \frac{f_n-f_{n-1}}{x_n-x_{n-1}} & & & \\ x_n & f_n & & & & \end{array} \tag{3.5}$$

the top row read from left to right form the constants of the interpolating polynomials. That is, the interpolating polynomial $p(x)$ is given by:

$$p(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x - x_0) + \frac{\frac{f_1-f_0}{x_1-x_0} - \frac{f_2-f_1}{x_2-x_1}}{x_2 - x_0}(x - x_0)(x - x_1) + \cdots. \tag{3.6}$$

Proof: We will prove this result by using induction on the number of points n we interpolate. The induction base with $n = 1$ is easily verified, as the interpolating polynomial is just the horizontal line $p(x) = f_0$. For the induction hypothesis we assume that we can interpolate n points by constructing a Newton polynomial according to the recurrence relation in equation 3.5, we now seek to prove that we can interpolate $n + 1$ points whilst still using the same recurrence to determine all coefficients.
Let p interpolate f at $\{x_0, x_1, \ldots, x_{n-1}\}$ and let q interpolate f at $\{x_1, x_2, \ldots, x_n\}$, which is possible due to the induction hypothesis. Also, denote the coefficient of x^{n-1} in p and q by p_{n-1} and q_{n-1}, respectively. We define

$$r(x) = \frac{(x - x_0)q(x) + (x_n - x)p(x)}{x_n - x_0}$$

and note that $r(x_i) = f(x_i)$ for $i \in \{0, 1, \ldots, n\}$. Also, the coefficient of x^n in r is given by $(q_{n-1} - p_{n-1})/(x_n - x_0)$, following the recurrence relation in equation 3.5 (note that the other coefficients are the same as those of p and thus can be computed using the same relation). Thus, we have shown the induction step, proving the theorem.

Example

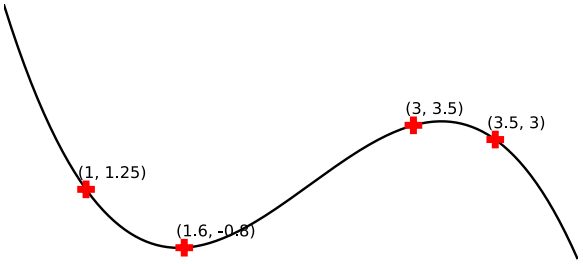
Let us say that we want to interpolate the following points $(1, \frac{5}{4}), (\frac{8}{5}, -\frac{4}{5}), (3, \frac{7}{2})(\frac{7}{2}, 3)$ with a polynomial. We first construct the same triangle as in equation 3.5 with the above coordinates:

$$\begin{array}{cccccc} x_0 = 1 & f_0 = \frac{5}{4} & & & & \\ & & \frac{-\frac{4}{5}-\frac{5}{4}}{\frac{8}{5}-1} = -\frac{41}{12} & & & \\ x_1 = \frac{8}{5} & f_1 = -\frac{4}{5} & & \frac{\frac{43}{14}+\frac{41}{12}}{3-1} = \frac{545}{168} & & \\ & & \frac{\frac{7}{2}+\frac{4}{5}}{3-\frac{8}{5}} = \frac{43}{14} & & \frac{-\frac{15}{7}-\frac{545}{168}}{\frac{7}{2}-1} = -\frac{181}{84}. & \\ x_2 = 3 & f_2 = \frac{7}{2} & & \frac{-1-\frac{43}{14}}{\frac{7}{2}-\frac{8}{5}} = -\frac{15}{7} & & \\ & & \frac{3-\frac{7}{2}}{\frac{7}{2}-3} = -1 & & & \\ x_3 = \frac{7}{2} & f_3 = 3 & & & & \end{array}$$

After inserting the coefficients in equation 3.6 we obtain the formula for the interpolating polynomial:

$$p(x) = \frac{5}{4} - \frac{41}{12}(x-1) + \frac{545}{168}(x-1)(x-\frac{8}{5}) - \frac{181}{84}(x-1)(x-\frac{8}{5})(x-3)\,. \tag{3.7}$$

The figure below illustrates that the above formula (equation 3.7) indeed interpolates the points.



[a]The notation is adapted from Wikipedia contributors [22].

Let us now derive Simpson's rule. We define three equally spaced points (with step size h) and denote them as x_0, $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$ and their corresponding function values by f_0, f_1 and f_2, respectively. To form the Newton polynomial, we first write the points in the form of equation 3.5:

$$\begin{array}{ll} x_0 & f_0 \\ & \frac{f_1-f_0}{h} \\ x_1 & f_1 \qquad \frac{f_0+f_2-2f_1}{2h^2}\,. \\ & \frac{f_2-f_1}{h} \\ x_2 & f_2 \end{array}$$

The interpolating polynomial then becomes

$$p(x) = f_0 + \frac{f_1 - f_0}{h}(x - x_0) + \frac{f_0 + f_2 - 2f_1}{2h^2}(x - x_0)(x - x_1)\,.$$

Integrating from x_0 to x_2 (keeping in mind that $x_1 - x_0 = h$ and $x_2 - x_0 = 2h$) now gives us

Simpson's rule:

$$\begin{aligned}
\int_{x_0}^{x_2} p(x)\mathrm{d}x &= \left[f_0 x + \frac{f_1 - f_0}{2h}(x - x_0)^2 + \frac{f_0 + f_2 - 2f_1}{2h^2}\left(\frac{x^3}{3} - (x_0 + x_1)\frac{x^2}{2} + x_0 x_1 x\right)\right]_{x_0}^{x_2} \\
&= 2hf_0 + 2h(f_1 - f_0) + \frac{f_0 + f_2 - 2f_1}{2h^2}\left(\frac{x_2^3 - x_0^3}{3} - (x_0 + x_1)\frac{x_2^2 - x_0^2}{2} + 2hx_0 x_1\right) \\
&= 2hf_1 + \frac{f_0 + f_2 - 2f_1}{2h^2}\left(\frac{8h^3}{3} + 2x_0^2 h + 4x_0 h^2 - (2h^2 + 2hx_0)(2x_0 + h) + 2hx_0(x_0 + h)\right) \\
&= 2hf_1 + \frac{f_0 + f_2 - 2f_1}{2h^2}\left(\frac{2h^3}{3} + 4x_0^2 h + 2x_0 h^2 - 2hx_0(2x_0 + h)\right) \\
&= 2hf_1 + \frac{f_0 + f_2 - 2f_1}{2h^2}\frac{2h^3}{3} \\
&= 2hf_1 + \frac{h}{3}(f_0 + f_2 - 2f_1) \\
&= \frac{h}{3}(f_0 + 4f_1 + f_2)\,.
\end{aligned}$$

In Subsection 3.2.3 we explain how one can use this result to approximate integrals, together these section form the basis for the combined interpolation method in Chapter 4.

3.2.2. Cotesian numbers

Instead of interpolating 3 points by a parabola, we can also interpolate $n+1$ points by an nth-order polynomial (we note that the order can be lower than n), by doing this, we obtain the Cotesian numbers used in Newton-Cotes equations. Higher order equations can be more accurate in approximating integrals than Simpson's rule.

We will denote the Cotesian numbers as $c_0, c_1, \ldots, c_n$, which are defined such that after defining points $x_0, x_1, \ldots, x_n$ and function values $f_0, f_1, \ldots, f_n$ the integral over the interpolated polynomial can be approximated by $h(c_0 f_0 + c_1 f_1 + \cdots + c_n f_n)$, where h is the distance between two points x_i and x_{i+1}.

To calculate the Cotesian numbers, we can solve the following system of equations (the proof of this statement is in Appendix B as the proof of Theorem 2):

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \cdots & n^n \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} n/1 \\ n^2/2 \\ n^3/3 \\ \vdots \\ n^{n+1}/(n+1) \end{bmatrix},$$

where the matrix on the left-hand side is the Vandermonde matrix. Since the inverse of the Vandermonde matrix exists in closed form (see Lemma 1 in Appendix B), the matrix can be brought to the other side [23]. After denoting the signed Stirling numbers of the first kind as $s(n,k)$ and doing just that we get a non-recursive formula for the coefficients c_i for $0 \le i \le n$ (Theorem 6 in Appendix B):

$$c_i = \frac{1}{(n-1)!}\binom{n}{i}\sum_{j=0}^{n}\sum_{m=0}^{n-j} i^m n^j \frac{(-1)^{i+n} s(n+1, j+m+1)}{j+1}\,.$$

For simplicity, one can also look these numbers up, since they constitute a sequence in the OEIS (the On-Line Encyclopedia of Integer Sequences, http://oeis.org/). In terms of

these sequences[1] we can write the Cotesian numbers as:

$$\frac{nh}{\mathrm{A002176}(n)}\left[\sum_{i=0}^{n}\mathrm{A100642}\left(\frac{n(n+1)}{2}+i\right)f_i\right].$$

The Cotesian numbers then determine the Newton-Cotes equations that can be used for approximating integrals:

$$\begin{gathered}
\frac{h}{2}(f_0+f_1)\\
\frac{h}{3}(f_0+4f_1+f_2)\\
\frac{3h}{8}(f_0+3f_1+3f_2+f_3)\\
\frac{2h}{45}(7f_0+32f_1+12f_2+32f_3+7f_4)\\
\frac{5h}{288}(19f_0+75f_1+50f_2+50f_3+75f_4+19f_5)\\
\frac{h}{140}(41f_0+216f_1+27f_2+272f_3+27f_4+216f_5+41f_6)\\
\frac{7h}{17280}(751f_0+3577f_1+1323f_2+2989f_3+2989f_4+1323f_5+3577f_6+751f_7)\,.\\
\therefore \qquad \vdots \qquad \vdots \qquad \vdots \qquad \ddots
\end{gathered} \tag{3.8}$$

More information on this subject is only referenced [24, 25].

3.2.3. From Newton-Cotes equations to integrals

We now give the method for extending the Newton-Cotes equations to evaluate integrals. Let us integrate over the interval $[x_L, x_R]$. Partitioning this interval into the subintervals $[x_0, x_1]$, $\ldots$, $[x_i, x_{i+1}]$, $\ldots [x_{\ell-1}, x_\ell]$ with $x_0 = x_L$ and $x_\ell = x_R$, whilst noting that the subintervals do not have to be equidistant, allows us to approximate the integral over each subinterval by applying an nth-order Newton-Cotes equation (note that the order is allowed change for different subintervals) to that subinterval by defining the function values to be $f_0 = f(x_i)$, $f_1 = f(x_i + \frac{x_{i+1}-x_i}{n}), \ldots, f_k = f(x_i + k\frac{x_{i+1}-x_i}{n}) \ldots, f_n = f(x_{i+1})$.

If we partition the original interval $[x_L, x_R]$ into equidistant subintervals and always apply Simpson's rule to the subinterval, we get the same effect as the following equation (where we evaluated the function at points $x_0, \ldots, x_n$ with $x_0 = x_L$ and $x_n = x_R$)

$$\int_{x_L}^{x_R} f(x)\mathrm{d}x \approx \frac{h}{3}(f_0+4f_1+2f_2+4f_3+\cdots+2f_{n-2}+4f_{n-1}+f_n)\,.$$

This way, between every 3 points $f_{2i}, f_{2i+1}, f_{2i+2}$ we interpolate the points by a parabola, integrate the parabola from x_{2i} to x_{2i+2}, and we add up the results to approximate our integral.

Provided that we can split up the subintervals into the number of equidistant parts needed for the Newton-Cotes equations, we can also use vector notation to approximate the integral, for the above example this works out to be

$$\int_{x_L}^{x_R} f(x)\mathrm{d}x \approx \mathbf{c}\mathbf{f}^\top\,,$$

[1] For the corresponding sequences we refer the reader to OEIS Foundation Inc. (2022), Denominators of Cotesian numbers, Entry A002176 and Numerators of Cotesian numbers, Entry A100642 in The On-Line Encyclopedia of Integer Sequences, http://oeis.org/A002176 and http://oeis.org/A100642.

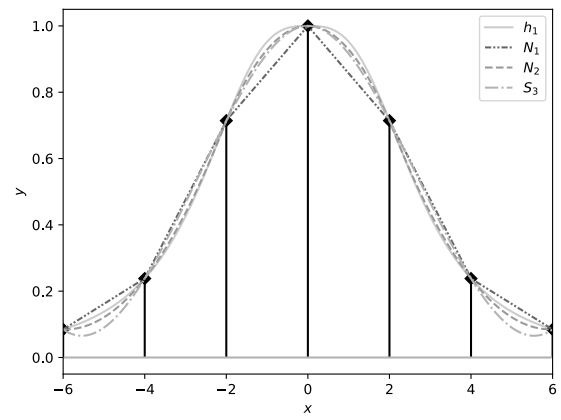
with

$$\mathbf{c} = \frac{h}{3}\begin{bmatrix}1 & 4 & 2 & 4 & \dots & 2 & 4 & 1\end{bmatrix} \quad \text{and} \quad \mathbf{f} = \begin{bmatrix}f_0 & f_1 & \dots & f_n\end{bmatrix}. \tag{3.9}$$

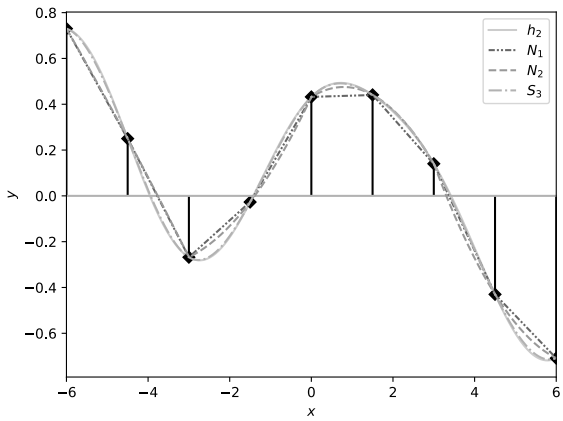
3.3. Examples

In this section we give numerical examples of approximating integrals using the methods described in previous sections of this chapter. The examples provide an overview of the described methods and give an idea of what their accuracy is and where we can find room for improvement. This section can also function as a tool to check the implementation of other researchers wanting to implement these methods themselves (note that the code to our implementation can be found in Appendix D).

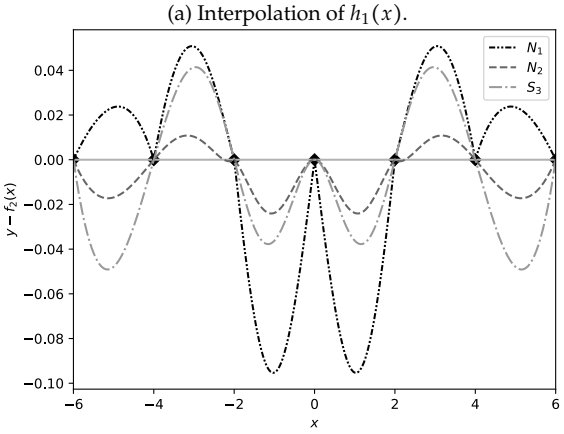
We start by interpolating functions using various types of polynomials. The first type is N_1, the 1st-order Newton-Cotes equation known as the trapezoidal rule (this uses linear interpolation) for which the integral can be found at the top in equation 3.8. This type is currently most commonly used for evaluating the Rayleigh integral and is therefore added as a benchmark. The second type is N_2, the 2nd-order Newton-Cotes equation known as Simpson's rule. The third type is S_3, a cubic spline interpolation. Results of the interpolation can be seen in Figure 3.1.



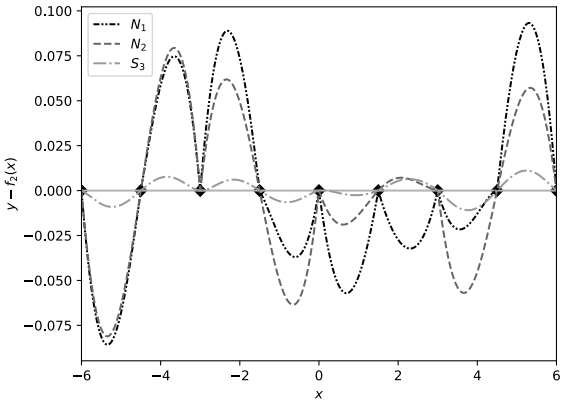
(a) Interpolation of $h_1(x)$.



(b) Interpolation of $h_2(x)$.



(c) Difference between interpolated functions and $h_1(x)$.



(d) Difference between interpolated functions and $h_2(x)$.

Figure 3.1: Interpolation of $h_1(x) = 1/(1+|x|^3/20)$ and $h_2(x) = \cos(\sqrt{(x-1)^2/4+1}) \cdot \sin(\sqrt{(x-2)^2/4+1})$ using 7 and 9 sample points of the function, respectively.

The integrals over the various interpolation types are listed in Table 3.1, where a few are calculated explicitly later on in this section. To illustrate how polynomials of higher order behave we have added a fourth type: N_6 (and the type N_{1-6-1}, which is the same as N_6 enclosed

by two N_1's to account for the number of sample points), being the 6th-order Newton-Cotes equation.

From Table 3.1 and Figure 3.1 we can see that N_1 got lucky with good approximation for the integral over function $h_1(x)$, since although $N_1(x) - h_1(x)$ (Figure 3.1c) has the greatest deflections the resulting integral is very close to the actual integral because the average deflection is coincidentally close to zero. However, with the other function ($h_2(x)$) the interpolation is less lucky, as other methods now give significantly better approximation to the integral.

Func.	Integral	Abs. diff.	Rel. diff.
$h_1(x)$	6.02845	-	-
N_1	5.97902	0.04943	0.820%
N_2	5.95426	0.07419	1.23%
N_6	6.00540	0.02305	0.382%
S_3	5.92113	0.10732	1.78%

Func.	Integral	Abs. diff.	Rel. diff.
$h_2(x)$	0.79728	-	-
N_1	0.82342	0.02615	3.28%
N_2	0.78280	0.01448	1.82%
N_{1-6-1}	0.88535	0.08807	11.0%
S_3	0.79930	0.00203	0.254%

Table 3.1: Various interpolations of function $h_1(x)$ on the left side and $h_2(x)$ on the right side together with the absolute difference and the relative difference between the integral and the correct integral.

The large error in the approximation using N_{1-6-1} is due to the boundaries of N_6. These have great deflections causing an inaccurate result. A method to remove the error would be to derive the coefficients using the integral, but instead of integrating over the full interval we can integrate over a subsection of it. For the 6th-order Newton-Cotes equation we can take the integral from x_1 to x_5, resulting in the following formula (the original one can be found in equation 3.8):

$$\frac{2h}{945}(-4f_0 + 171f_1 + 612f_2 + 332f_3 + 612f_4 + 171f_5 - 4f_6)\,. \tag{3.10}$$

We can then evaluate the integral of $h_2(x)$ again with an N_{2-6^*-2} interpolation (we need to pad with two N_2's since the new N_{6^*} is integrated over fewer points), also we note that in the above formula the endpoints (f_0 and f_6) are weighted less since they do not directly contribute to the integral, only indirectly by interpolation. The approximation for the integral becomes 0.77928, with an absolute difference of 0.01800 and a relative difference of 2.26%. This is a huge improvement compared to the relative error of 11.0% from using N_{1-6-1}. However, we do not investigate these methods further as the method still performes worse than N_2 and S_3 (this is instead left to future research in Section 6.2) and only polynomials up to the 2nd-order are used hereafter.

The best methods from above are therefore N_2 and S_3, however, since this is still combined interpolation (Section 3.2) the integrals can also be approximated using separate interpolation (Section 3.1).

We compare the combined interpolation using S_3 (thus we interpolate $f(x) \cdot g(x)$ by a cubic spline) and the separate interpolation using S_3 (thus we interpolate $f(x)$ and $g(x)$ separately where we interpolate $f(x)$ by a cubic spline and $g(x)$ to a high degree since it is known and combine the results by the method described in Section 3.1). The difference between the functions and their interpolations can be seen in Figure 3.2, where we note that $h_2(x) = f_2(x) \cdot g_2(x)$ but that $h_1(x) \neq f_1(x) \cdot g_1(x)$ due to the function that was chosen. The corresponding differences are listed in Table 3.2, and the calculation of $S_3 - \mathrm{S}$ can be found at the end of this section.

Method	Abs. diff.	Rel. diff.
$S_3 - C$	0.001156	0.0251%
$S_3 - S$	0.0001215	0.00264%

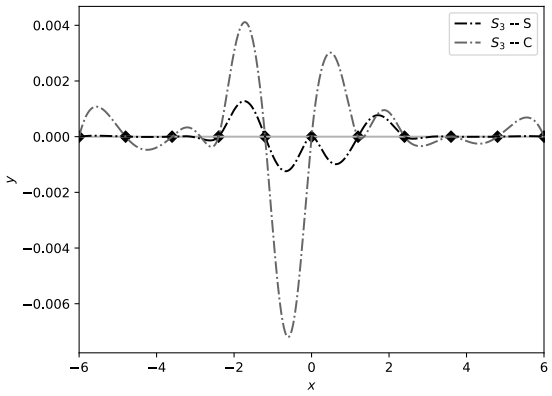
(a) The integral is approximately 4.60931999.

Method	Abs. diff.	Rel. diff.
$S_3 - C$	0.00202819	0.254%
$S_3 - S$	0.00174655	0.219%

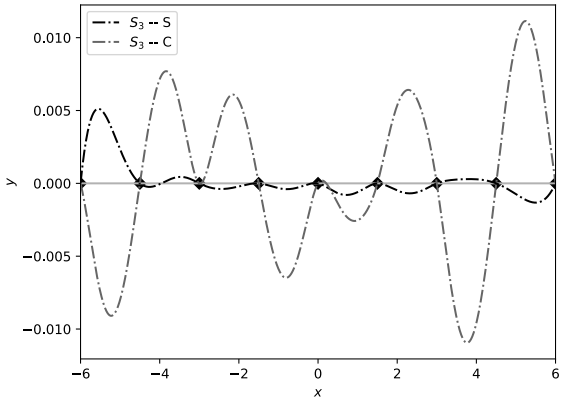
(b) The integral is approximately 0.79727674.

Table 3.2: The absolute difference and the relative difference between the integral and the correct integral using combined and separate interpolation techniques. For the functions we used $f_1(x) \cdot g_1(x)$ on the left-hand side and $f_2(x) \cdot g_2(x)$ on the right-hand side, see the caption of Figure 3.2 for the corresponding functions.

The separate interpolation method is better than the combined method as can be seen in Figure 3.2 and Table 3.2. However, the large difference from the second function on the left side of the interval was not compensated, resulting in a high overall difference that does not do the method justice.



(a) Difference between interpolation and $f_1(x) \cdot g_1(x)$.



(b) Difference between interpolation and $f_2(x) \cdot g_2(x)$.

Figure 3.2: In these figures $f_1(x) = 1/(1 + x^2/10)$, $g_1(x) = 1/(1 + (x+1)^2/10)$ and $f_2(x) = \cos(\sqrt{(x-1)^2/4 + 1})$, $g_2(x) = \sin(\sqrt{(x-2)^2/4 + 1})$. Interpolation was done on 11 and 9 samples points, respectively, by means of a separate interpolation method denoted by $S_3 - S$ and a combined interpolation method denoted by $S_3 - C$.

Calculating the integrals using the Newton-Cotes equations by interpolating $h_2(x)$

We provide example calculations of using various Newton-Cotes equations to approximate the integral over $h_2(x)$, the general methods are presented in Section 3.2.

The function values of $h_2(x)$ at the 9 samples points from Figure 3.1b written in vector notation are:

$$\mathbf{f} = \begin{bmatrix} 0.73018 & 0.24998 & -0.26794 & -0.02706 & 0.4321 & 0.44099 & 0.14023 & -0.43006 & -0.70876 \end{bmatrix}.$$

The value for h can be calculated by dividing the total width of the interval by the total number of subintervals (which is one less than the number of sample points):

$$h = \frac{12}{9-1} = \frac{3}{2}.$$

Now, for all different interpolating polynomials N_1, N_2, N_{1-6-1}, and N_{2-6^*-2} we determine the corresponding vector from equation 3.9 (note that the coefficients can be found in equations 3.8 and 3.10):

$$\mathbf{c}_1 = \frac{3}{4}\begin{bmatrix} 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \end{bmatrix}, \quad \mathbf{c}_2 = \frac{1}{2}\begin{bmatrix} 1 & 4 & 2 & 4 & 2 & 4 & 2 & 4 & 1 \end{bmatrix},$$

$$\mathbf{c}_{1-6-1} = \begin{bmatrix} \frac{3}{4} & \frac{3}{4} + \frac{3 \cdot 41}{280} & \frac{3 \cdot 216}{280} & \frac{3 \cdot 27}{280} & \frac{3 \cdot 272}{280} & \frac{3 \cdot 27}{280} & \frac{3 \cdot 216}{280} & \frac{3 \cdot 41}{280} + \frac{3}{4} & \frac{3}{4} \end{bmatrix},$$

$$\mathbf{c}_{2-6^*-2} = \begin{bmatrix} \frac{1}{2} & 2 - \frac{3\cdot 4}{945} & \frac{1}{2} + \frac{3\cdot 171}{945} & \frac{3\cdot 612}{945} & \frac{3\cdot 332}{945} & \frac{3\cdot 612}{945} & \frac{3\cdot 171}{945} + \frac{1}{2} & -\frac{3\cdot 4}{945} + 2 & \frac{1}{2} \end{bmatrix}.$$

Now, we can finally approximate the integral by determining the integral over N_1, N_2, N_{1-6-1}, and N_{2-6^*-2}, this can thus be done without ever having to interpolate the function:

$$\mathbf{c}_1\mathbf{f}^\top \approx 0.823425\,, \quad \mathbf{c}_2\mathbf{f}^\top \approx 0.782800\,, \quad \mathbf{c}_{1-6-1}\mathbf{f}^\top \approx 0.885348\,, \quad \mathbf{c}_{2-6^*-2}\mathbf{f}^\top \approx 0.779280\,.$$

Verifying the results in Table 3.1 and below.

Calculating the integral over $f_2(x) \cdot g_2(x)$ using separate spline interpolation

To replicate the results from Table 3.2 we first write $f_2(x)$ and $g_2(x)$ without subscripts, i.e. $f(x)$ and $g(x)$, to avoid confusion later on. In this example we have used $n_f = 3$ and $n_g = 5$.

To avoid rounding errors we modify the method in Section 3.1, avoiding any high powers of x_i when calculating the various values of d_{ij}. The modification of the method is to shift all the subintervals $[x_i, x_{i+1}]$ to $[x_4, x_5] = [0, \frac{3}{2}]$ and carry out the interpolation for all the subintervals on this new domain. Using this method we only need to calculate powers of $x_4 = 0$ and $x_5 = \frac{3}{2}$ instead of powers of x_i and x_{i+1}, this speeds up the process and avoids multiplying by large numbers resulting in less rounding errors in our answer. For the shifted interpolation the coefficients of the interpolating polynomial of $f_3(x)$ (i.e. $\mathbf{b}_{3j}$) are now the coefficients after interpolating $f_3(x - \frac{3}{2})$ on the interval $[0, \frac{3}{2}]$. Similarly, the coefficients for $\mathbf{b}_{1j}$ are the coefficients from interpolating $f_1(x - 3 \cdot \frac{3}{2})$ on the interval $[0, \frac{3}{2}]$. To obtain the correct answer using these shifted interpolation coefficients we need to make a second adjustment to our method. This is for calculating the values of d_{ij} (equation 3.3). We need to use the values x_4 and x_5 repeatedly instead of x_i and x_{i+1}, respectively. Since $x_4 = 0$ this results in calculating and storing the values of $x_5^{j+k+1}/(j+k+1)$ for all values of $j+k$ before calculating (and storing if we have multiple sensors) the values of

$$d_{ij} = \sum_{k=0}^{n_g} c_{ik} \frac{x_5^{j+k+1}}{j+k+1}\,.$$

All things considered, these modifications improve the accuracy of our results as they avoid rounding errors.

The function $f_3(x - \frac{3}{2})$ interpolated on $[x_4, x_5] = [0, \frac{3}{2}]$ is given as:

$$f_3\left(x - \frac{3}{2}\right) \approx -0.02998 + 0.39005x - 0.03051x^2 - 0.014518x^3\,,$$

resulting in row 3 of matrix B,

$$\mathbf{b}_{3j} = \begin{bmatrix} -0.02998 & 0.39005 & -0.03051 & -0.014518 \end{bmatrix}.$$

Continuing this process to calculate all elements of B and C we can calculate the elements of D and use equation 3.4 to approximate the integral. In Appendix C we listed all values of the matrices. We note that the method seems like a lot of work, but after we initialize it we can easily calculate the sum in equation 3.4 again with a shift to approximate the integral for a different sensor.

Taking everything into account, using this method we get a value of 0.79902 for the evaluation of the integral, which corresponds to an absolute difference of 0.00175 and a relative difference of 0.219%. Without the method of shifting the intervals we would have obtained the value 0.79958 with a relative difference of 0.289%, due to rounding errors resulting from multiplying by a factor of $(x_{i+1}^{j+k+1} - x_i^{j+k+1})/(j+k+1)$.

The obtained value can be improved upon (by a tiny fraction) by increasing the order of the spline (or polynomial) used to interpolate $g(x)$, i.e. increasing n_g.

Chapter 4

Adapting numerical integration methods for the Rayleigh integral

In this chapter we adapt the numerical methods from Chapter 3 to better fit the Rayleigh integral (equation 2.17). The methods we develop will be compared in Chapter 5, the results of this thesis.

The first section will generalize both methods derived in the previous chapter (Section 3.1 for separate interpolation and Section 3.2 for combined interpolation) to complex, double integrals. In the next section we present a variation to the combined interpolation method (Section 3.2), in specific Simpson's rule, to evaluate integrals on non-equidistant intervals (single integrals) and semi-equidistant grids (double integrals), weakening the assumption made in Chapter 3 of an equidistant grid. In the last section we summarize all the derived methods and their qualities for the next chapter, where we present the results of evaluating an artificial Rayleigh integral using those methods.

4.1. From single to double integrals

The Rayleigh integral is not a real-valued, single integral. It is a complex valued, double integral. Therefore, in this section, we present methods that extend the algorithms from Sections 3.1 and 3.2 to complex-valued, double integrals. Using these new methods we can approximate the Rayleigh integral in three dimensions, giving us the ability to propagate a wavefield from one layer of soil to the next.

4.1.1. Separate interpolation

This argument is very similar to the one in Section 3.1. However, since this is a key element of the implementation of this thesis we will write it out. This extension does not include the complex-valued integral. We note that to calculate it one could use the method described below for the real-real values of f and g, the imaginary-real, the real-imaginary and the imaginary-imaginary (as a consequence the initialization of the algorithm only takes twice as long whereas the evaluation takes 4 times as long), after combining the results one finds the integral.

The interval $[x_L, x_R] \times [y_L, y_R]$ is partitioned into $\ell_x \cdot \ell_y$ equally large subintervals

$$\begin{matrix} [x_0, x_1] \times [y_0, y_1] & [x_0, x_1] \times [y_1, y_2] & \cdots & [x_0, x_1] \times [y_{\ell_y - 1}, y_{\ell_y}] \\ [x_1, x_2] \times [y_0, y_1] & [x_1, x_2] \times [y_1, y_2] & \cdots & [x_1, x_2] \times [y_{\ell_y - 1}, y_{\ell_y}] \\ \vdots & \vdots & \ddots & \vdots \\ [x_{\ell_x - 1}, x_{\ell_x}] \times [y_0, y_1] & [x_{\ell_x - 1}, x_{\ell_x}] \times [y_1, y_2] & \cdots & [x_{\ell_x - 1}, x_{\ell_x}] \times [y_{\ell_y - 1}, y_{\ell_y}] \end{matrix},$$

that is, $x_1 - x_0 = \cdots = x_{\ell_x} - x_{\ell_x - 1}$, $x_L = x_0$, and $x_R = x_{\ell_x}$ and $y_1 - y_0 = \cdots = y_{\ell_y} - y_{\ell_y - 1}$, $y_L = y_0$, and $y_R = y_{\ell_y}$.

For each interval $[x_{i_x}, x_{i_x+1}] \times [y_{i_y}, y_{i_y+1}]$ with $0 \le i_x \le \ell_x - 1$ and $0 \le i_y \le \ell_y - 1$ both functions are interpolated by polynomials [26]:

$$f_{i_x,i_y}(x,y) = \sum_{j_x=0}^{n_f} \sum_{j_y=0}^{m_f} b_{i_x i_y j_x j_y} x^{j_x} y^{j_y}, \quad g_{i_x,i_y}(x,y) = \sum_{k_x=0}^{n_g} \sum_{k_y=0}^{m_g} c_{i_x i_y k_x k_y} x^{k_x} y^{k_y}.$$

Using the fact that the intervals are equidistant the interpolation can be achieved with a time complexity of $\mathcal{O}(n_f^2 m_f^2 \ell_x \ell_y + n_g^2 m_g^2 \ell_x \ell_y)$, we also need $\mathcal{O}(n_f m_f \ell_x \ell_y + n_g m_g \ell_x \ell_y)$ space.

We assume the prediction points (the points corresponding to different values of a_x, a_y) are spaced in an equidistant grid (this is often the case). We write $a_x = m_x(x_1 - x_0)$ and $a_y = m_y(y_1 - y_0)$ with $m_x \in \mathbb{N}$, $m_y \in \mathbb{N}$, $0 \le m_x \le s_x$ and $0 \le m_y \le s_y$, where s_x denotes the number of prediction points in the x-direction and s_y those in the y-direction, such that the integral becomes (we make the same assumption that $f(x,y) = g(x,y) = 0$ for $(x,y) \notin [x_L, x_R] \times [y_L, y_R]$):

$$\begin{aligned}
&\int_{y_L}^{y_R} \int_{x_L}^{x_R} f(x',y') g(x'+a_x, y'+a_y) \mathrm{d}x' \mathrm{d}y' \\
&\quad = \int_{y_L - a_y}^{y_R - a_y} \int_{x_L - a_x}^{x_R - a_x} f(x - a_x, y - a_y) g(x,y) \mathrm{d}x \mathrm{d}y \\
&\quad \approx \sum_{i_x=0}^{\ell_x - m_x} \sum_{i_y=0}^{\ell_y - m_y} \int_{y_{i_y}}^{y_{i_y+1}} \int_{x_{i_x}}^{x_{i_x+1}} f_{i_x,i_y}(x - a_x, y - a_y) g_{i_x,i_y}(x,y) \mathrm{d}x \mathrm{d}y \\
&\quad = \sum_{i_x=0}^{\ell_x - m_x} \sum_{i_y=0}^{\ell_y - m_y} \int_{y_{i_y}}^{y_{i_y+1}} \int_{x_{i_x}}^{x_{i_x+1}} f_{i_x+m_x,i_y+m_y}(x) g_{i_x,i_y}(x) \mathrm{d}x \\
&\quad = \sum_{i_x=0}^{\ell_x - m_x} \sum_{i_y=0}^{\ell_y - m_y} \int_{y_{i_y}}^{y_{i_y+1}} \int_{x_{i_x}}^{x_{i_x+1}} \sum_{j_x=0}^{n_f} \sum_{j_y=0}^{m_f} \sum_{k_x=0}^{n_g} \sum_{k_y=0}^{m_g} b_{i_x+m_x,i_y+m_y,j_x,j_y} c_{i_x i_y k_x k_y} x^{j_x+k_x} y^{j_y+k_y} \mathrm{d}x \mathrm{d}y \\
&\quad = \sum_{i_x=0}^{\ell_x - m_x} \sum_{i_y=0}^{\ell_y - m_y} \sum_{j_x=0}^{n_f} \sum_{j_y=0}^{m_f} \sum_{k_x=0}^{n_g} \sum_{k_y=0}^{m_g} b_{i_x+m_x,i_y+m_y,j_x,j_y} c_{i_x i_y k_x k_y} \int_{y_{i_y}}^{y_{i_y+1}} \int_{x_{i_x}}^{x_{i_x+1}} x^{j_x+k_x} y^{j_y+k_y} \mathrm{d}x \mathrm{d}y \\
&\quad = \sum_{i_x=0}^{\ell_x - m_x} \sum_{i_y=0}^{\ell_y - m_y} \sum_{j_x=0}^{n_f} \sum_{j_y=0}^{m_f} b_{i_x+m_x,i_y+m_y,j_x,j_y} \sum_{k_x=0}^{n_g} \sum_{k_y=0}^{m_g} c_{i_x i_y k_x k_y} \frac{x_{i_x+1}^{j_x+k_x+1} - x_{i_x}^{j_x+k_x+1}}{j_x + k_x + 1} \frac{y_{i_y+1}^{j_y+k_y+1} - y_{i_y}^{j_y+k_y+1}}{j_y + k_y + 1}.
\end{aligned}$$

Storing the values of

$$d_{i_x i_y j_x j_y} = \sum_{k_x=0}^{n_g} \sum_{k_y=0}^{m_g} c_{i_x i_y k_x k_y} \frac{x_{i_x+1}^{j_x+k_x+1} - x_{i_x}^{j_x+k_x+1}}{j_x + k_x + 1} \frac{y_{i_y+1}^{j_y+k_y+1} - y_{i_y}^{j_y+k_y+1}}{j_y + k_y + 1},$$

for all i_x, i_y and j_x, j_y, then allows us to rewrite the equation, giving:

$$\int_{y_L}^{y_R} \int_{x_L}^{x_R} f(x,y) g(x+a_x, y+a_y) \mathrm{d}x \mathrm{d}y \approx \sum_{i_x=0}^{\ell_x - m_x} \sum_{i_y=0}^{\ell_y - m_y} \sum_{j_x=0}^{n_f} \sum_{j_y=0}^{m_f} b_{i_x+m_x,i_y+m_y,j_x,j_y} d_{i_x i_y j_x j_y}.$$

The final space and time complexity are given by the initialization, which uses $\mathcal{O}((n_f + n_g)^2 (m_f + m_g)^2 \ell_x \ell_y)$ time and $\mathcal{O}(n_f m_f \ell_x \ell_y + n_g m_g \ell_x \ell_y)$ space and by the computation, using $\mathcal{O}(n_f m_f \ell_x \ell_y s_x s_y)$ time and $\mathcal{O}(n_f m_f \ell_x \ell_y + n_g m_g \ell_x \ell_y + s_x s_y)$ space.

Note that the final time complexity is again independent of n_g and m_g.

4.1.2. Combined interpolation

In this section we extend the combined interpolation method from Section 3.2 (the majority of the used notation is adapted from this section as well) to complex-valued double integrals. Luckily, the Newton-Cotes equations are easier to extend.

For the 2nd-order Newton-Cotes equation we can write

$$C = \frac{h_x h_y}{9}\begin{bmatrix} 1 & 4 & 2 & 4 & \cdots & 2 & 4 & 1 \\ 4 & 16 & 8 & 16 & \cdots & 8 & 16 & 4 \\ 2 & 8 & 4 & 8 & \cdots & 4 & 8 & 2 \\ 4 & 16 & 8 & 16 & \cdots & 8 & 16 & 4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 2 & 8 & 4 & 8 & \cdots & 4 & 8 & 2 \\ 4 & 16 & 8 & 16 & \cdots & 8 & 16 & 4 \\ 1 & 4 & 2 & 4 & \cdots & 2 & 4 & 1 \end{bmatrix},$$

which is the same as

$$C = \mathbf{c}_x^\top \mathbf{c}_y \,.$$

If we let the matrix F be the matrix of function values, i.e. $F_{ij} = f(x_i, y_j)$ for all i, j, we can approximate a complex-valued, double integral by

$$\int_{x_L}^{x_R}\int_{y_L}^{y_R} f(x,y)\mathrm{d}x\mathrm{d}y \approx \langle C, F\rangle_\mathrm{F} = \mathbf{c}_x F \mathbf{c}_y^\top \,.$$

Note that $\langle A, B\rangle_\mathrm{F}$ denotes the Frobenius inner product of A and B, i.e. the sum over the elements of the pointwise (or Hadamard) product of both matrices.

4.2. Uncertainty in gridpoints

In all previous described methods we have made an assumption: the gridpoints are spaced equidistantly. This assumption speeds up the calculation, but it is still only approximately true; if we measure the pressure wavefield on certain gridpoints we will never space the sensors *exactly* the same distance apart each time. However, in general the corrections to the gridpoints are often known. In this section we thus present a revision to Simpson's rule to interpolate the data. The first subsection (Subsection 4.2.1) does this for single integrals with non-equidistant intervals. The second subsection (Subsection 4.2.2) does this for double integrals with semi-equidistant gridpoints (in the section we also define semi-equidistant). We note that these adjustments are also possible for higher-order Newton-Cotes equations, but these will not be derived and are left to future research (Section 6.2).

4.2.1. Single integral alterations

Like in our previous derivation of Simpson's rule in Section 3.2.1, we start by interpolating just 3 points. The modification is that we now interpolate the points x_0, $x_1 + \delta$ and x_2 in order to account for non-equidistant intervals. The corresponding function values are f_0, f_1 and f_2, respectively. The interpolating polynomial can be derived from the top row of:

$$\begin{array}{cc} x_0 & f_0 \\ & \frac{f_1-f_0}{h+\delta} \\ x_1+\delta & f_1 \\ & \frac{f_2-f_1}{h-\delta} \\ x_2 & f_2 \end{array} \qquad \frac{h(f_2+f_0-2f_1)+\delta(f_2-f_0)}{2h(h^2-\delta^2)} \,,$$

where the fraction on the right can be seen to hold from observing that

$$\begin{aligned}\frac{\frac{f_2-f_1}{h-\delta}-\frac{f_1-f_0}{h+\delta}}{2h} &= \frac{\frac{f_2-f_1}{h-\delta}+\frac{f_0-f_1}{h+\delta}}{2h}\\ &= \frac{\frac{(h+\delta)(f_2-f_1)+(h-\delta)(f_0-f_1)}{(h-\delta)(h+\delta)}}{2h}\\ &= \frac{(h+\delta)(f_2-f_1)+(h-\delta)(f_0-f_1)}{2h(h^2-\delta^2)}\\ &= \frac{h(f_2+f_0-2f_1)+\delta(f_2-f_0)}{2h(h^2-\delta^2)}.\end{aligned}$$

The interpolating polynomial thus becomes

$$p(x) = f_0 + \frac{f_1-f_0}{h+\delta}(x-x_0) + \frac{h(f_2+f_0-2f_1)+\delta(f_2-f_0)}{2h(h^2-\delta^2)}(x-x_0)(x-x_1).$$

Again, integrating from x_0 to x_2 whilst keeping in mind that $x_1 - x_0 = h + \delta$ and $x_2 - x_0 = 2h$ now gives us our desired result:

$$\begin{aligned}&\int_{x_0}^{x_2} p(x)\mathrm{d}x\\ &= \left[f_0 x + \frac{f_1-f_0}{2(h+\delta)}(x-x_0)^2 + \frac{h(f_2+f_0-2f_1)+\delta(f_2-f_0)}{2h(h^2-\delta^2)}\left(\frac{x^3}{3}-(x_0+x_1)\frac{x^2}{2}+x_0x_1x\right)\right]_{x_0}^{x_2}\\ &= 2hf_0 + \frac{f_1-f_0}{2(h+\delta)}(2h)^2 + \frac{h(f_2+f_0-2f_1)+\delta(f_2-f_0)}{2h(h^2-\delta^2)}\left(\frac{x_2^3-x_0^3}{3}-(x_0+x_1)\frac{x_2^2-x_0^2}{2}+2hx_0x_1\right)\\ &= \frac{2h(h^2-\delta^2)f_0}{h^2-\delta^2} + \frac{2h^2(h-\delta)(f_1-f_0)}{h^2-\delta^2}\\ &\qquad + \frac{h(f_2+f_0-2f_1)+\delta(f_2-f_0)}{2h(h^2-\delta^2)}\left(\frac{2h^3}{3}+4x_0^2h+2x_0h^2-2hx_0(2x_0+h)-2\delta h^2\right)\\ &= \frac{2h(h^2-\delta^2)f_0}{h^2-\delta^2} + \frac{2h^2(h-\delta)(f_1-f_0)}{h^2-\delta^2} + \frac{h(f_2+f_0-2f_1)+\delta(f_2-f_0)}{2h(h^2-\delta^2)}\left(\frac{2h^3}{3}-2\delta h^2\right)\\ &= \frac{6h(h^2-\delta^2)f_0 + 6h^2(h-\delta)(f_1-f_0) + \{h(f_2+f_0-2f_1)+\delta(f_2-f_0)\}(h^2-3\delta h)}{3(h^2-\delta^2)}\\ &= \frac{\delta^2h\{-6f_0-3(f_2-f_0)\}}{3(h^2-\delta^2)} + \frac{\delta h^2\{-6(f_1-f_0)-3(f_2+f_0-2f_1)+(f_2-f_0)\}}{3(h^2-\delta^2)}\\ &\qquad + \frac{h^3\{6f_0+6(f_1-f_0)+(f_2+f_0-2f_1)\}}{3(h^2-\delta^2)}\\ &= \frac{\delta^2h\{-3(f_0+f_2)\}}{3(h^2-\delta^2)} + \frac{\delta h^2\{(2f_0-2f_2)\}}{3(h^2-\delta^2)} + \frac{h^3\{f_0+4f_1+f_2\}}{3(h^2-\delta^2)}\\ &= \frac{h(-3\delta^2(f_0+f_2)+2\delta(f_0-f_2)h+h^2(f_0+4f_1+f_2))}{3(h^2-\delta^2)}.\end{aligned} \tag{4.1}$$

This results in Simpson's rule for a single integral on a non-equidistant interval (consisting of 3 points). We note that filling in $\delta = 0$ gives the regular Simpson's rule derived in Section 3.2.1.

This method can be extended to approximate integrals similarly to Section 3.2.3, whilst only noting that this time all h and all δ are dependent on the x coordinate, so they can not be placed in front of the vector. Because for full integrals we cannot fuse Simpson's rule on different intervals, this method requires more operations than before (a factor $\frac{3}{2}$ more).

4.2.2. Double integral alterations

We now derive similar modifications to the method for evaluating double integrals. First we define semi-equidistant gridpoints. Note that this method will yield the exact result for 2nd-order polynomials in two dimensions if the sample points are semi-equidistant. To define semi-equidistant gridpoints we refer to Figure 4.1. The restrictions compared to a non-equidistant grid are that f_{00}, f_{20}, f_{22} and f_{02} are forced to lie on a rectangular grid and f_{01}, f_{11} and f_{21} are restricted to the same height (which is shifted δ_y from the middle of the rectangle). Also, f_{10} and f_{12} are restricted to lie on the rectangle, although the former can have a shift δ_{x0} in the x-direction and the latter can have a shift of δ_{x2} in the x-direction. The middle point f_{11} can also have a shift of δ_{x1} in the x-direction. We note that this grid has a "preferred" direction; the most uncertainty in the gridpoints can be in the x-direction, therefore we advise orienting the axis so that the direction with the most uncertainty is oriented in the x-direction.

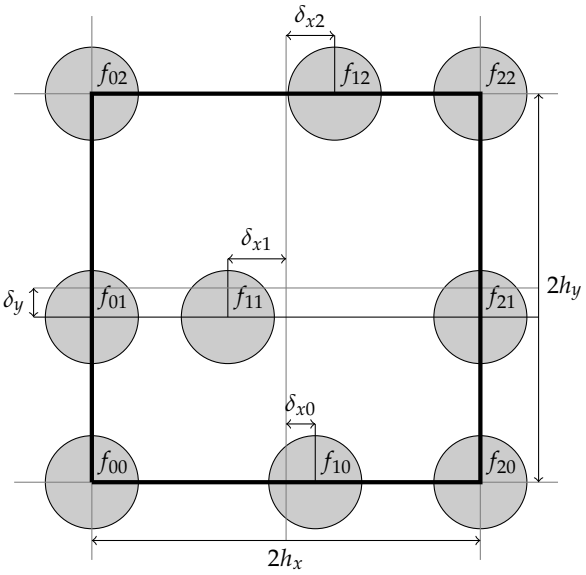


Figure 4.1: A semi-equidistant grid, we note that in this case δ_{y0} and δ_{x1} are both negative.

Now, to derive the adjustment for double integrals on semi-equidistant grids we first interpolate the function. We do this in two steps, first in the x-direction whilst fixing y and afterwards in the y-direction for all values of x. We denote $f_{ij} = f(x_i, y_j)$ as the function value at the point (x_i, y_j) (for clarity we omit δ in this notation) for all i, j and $f_{xi} = f(x, y_i)$ as the function values whilst fixing y_i for all i and list the interpolating polynomials:

$$f(x,y_0) = f_{x0} = f_{00} + \frac{f_{10}-f_{00}}{h_x+\delta_{x0}}(x-x_0) + \frac{h_x(f_{20}+f_{00}-2f_{10})+\delta_{x0}(f_{20}-f_{00})}{2h_x(h_x^2-\delta_{x0}^2)}(x-x_0)(x-x_1),$$
$$f(x,y_1) = f_{x1} = f_{01} + \frac{f_{11}-f_{01}}{h_x+\delta_{x1}}(x-x_0) + \frac{h_x(f_{21}+f_{01}-2f_{11})+\delta_{x1}(f_{21}-f_{01})}{2h_x(h_x^2-\delta_{x1}^2)}(x-x_0)(x-x_1),$$
$$f(x,y_2) = f_{x2} = f_{02} + \frac{f_{12}-f_{02}}{h_x+\delta_{x2}}(x-x_0) + \frac{h_x(f_{22}+f_{02}-2f_{12})+\delta_{x2}(f_{22}-f_{02})}{2h_x(h_x^2-\delta_{x2}^2)}(x-x_0)(x-x_1).$$

The final interpolating polynomial can now be written as (note that we essentially interpolate $f(x,y_0)$, $f(x,y_1)$, and $f(x,y_2)$ for all values of x with known function values at y_0, $y_1+\delta_y$, and y_2, resulting in interpolation similar to a single integral)

$$f(x,y) = f_{x0} + \frac{f_{x1}-f_{x0}}{h_y+\delta_y}(y-y_0) + \frac{h_y(f_{x2}+f_{x0}-2f_{x1})+\delta_y(f_{x2}-f_{x0})}{2h_y(h_y^2-\delta_y^2)}(y-y_0)(y-y_1).$$

After defining

$$a_i := \int_{x_0}^{x_2} f(x,y_i)\mathrm{d}x = \int_{x_0}^{x_2} f_{xi}\mathrm{d}x$$
$$= \frac{h_x(-3\delta_{xi}^2(f_{0i}+f_{2i})+2\delta_{xi}(f_{0i}-f_{2i})h_x+h_x^2(f_{0i}+4f_{1i}+f_{2i}))}{3(h_x^2-\delta_{xi}^2)},$$

the final integral becomes (the second step can be seen from equation 4.1)

$$\begin{aligned}
&\int_{x_0}^{x_2}\int_{y_0}^{y_2} f(x,y)\mathrm{d}y\mathrm{d}x \\
&= \int_{x_0}^{x_2} \frac{h_y(-3\delta_y^2(f_{x0}+f_{x2})+2\delta_y(f_{x0}-f_{x2})h_y+h_y^2(f_{x0}+4f_{x1}+f_{x2}))}{3(h_y^2-\delta_y^2)}\mathrm{d}x \\
&= \frac{h_y(-3\delta_y^2(a_0+a_2)+2\delta_y(a_0-a_2)h_y+h_y^2(a_0+4a_1+a_2))}{3(h_y^2-\delta_y^2)}.
\end{aligned}$$

We note that the rectangle in Figure 4.1 only encloses the area of 4 full circles, that is, in the case where we can connect Simpson's rule in an infinite grid, we only need 4 operations per grid to approximate the integral. Since we cannot join the grids any longer we now require (at least) 9 operations per grid to approximate the integral.

4.3. Summary of used methods

In this section we recapitulate all derived methods, in the next chapter we then use those methods to evaluate an artificial Rayleigh integral.

The implementation of these methods is in Appendix D, for the implementation we made use of Python3.10. Specifically the packages matplotlib (visualizing), NumPy (mathematical computation) and SciPy (interpolating functions) were used.

For notation we use s_x to denote the number of prediction points in the x-direction and likewise we define s_y to denote the number of prediction points in the y-direction. We consider approximating the Rayleigh integral on $\ell_x\ell_y$ sample points.

Basic method

The current method to evaluate the Rayleigh integral is a combined interpolation method where the trapezoidal rule is used and uncertainty in gridpoints is ignored. Also, the time complexity of this method is $\mathcal{O}(\ell_x\ell_y s_x s_y)$, whereas the space complexity is $\mathcal{O}(s_x s_y)$. Complex integrals will require twice as many operations since we need to do this for the real part and the imaginary part separately (if we do not so explicitly the computer will do this internally resulting in the same factor).

Altered basic method

Although this variation was not previously discussed it is quite simple. In this alteration we weaken the assumption that the gridpoints are regularly spaced. Instead, we assume that the gridpoints are spaced in rectangles (see the outer points of Figure 4.1, this is often called rectilinear) where h_x and h_y can vary along the grid.

This method is implemented by calculating $\mathbf{c}_x$, $\mathbf{c}_y$ (equation 3.9) without the common factors h_x or h_y in front, as they can vary at different positions. Again, the time complexity of this method is $\mathcal{O}(\ell_x\ell_y s_x s_y)$ and the space complexity is $\mathcal{O}(s_x s_y)$. Also, for complex integrals the same argument holds and this will take twice as long.

Separate interpolation method

For this method we interpolate the pressure function by 2 dimensional cubic splines (i.e. $n_f = m_f = 3$ in Section 4.1.1). In our implementation of this method (computer code in Appendix D) we assume that the derivative of the Green's function is interpolated to such a high degree that it corresponds to the exact function[1].

[1] This is generally a good approximation to compute results, but it definitely does not correspond to a fast implementation in our case since integrals over subintervals are computed by dividing them into subsubintervals. Normally, taking $n_g = m_g = 5$ or $n_g = m_g = 6$ results in approximately the same answer with much faster computation.

We differentiate two methods:

- We assume that the pressure wavefield is measured on an equidistant grid and discard any uncertainty. The method is now completely described in Section 4.1.1 with an initialization time complexity of $\mathcal{O}((n_f + n_g)^2(m_f + m_g)^2\ell_x\ell_y)$ and an initialization space complexity of $\mathcal{O}(n_f m_f \ell_x \ell_y + n_g m_g \ell_x \ell_y)$. The computation then takes $\mathcal{O}(n_f m_f \ell_x \ell_y s_x s_y)$ time and $\mathcal{O}(n_f m_f \ell_x \ell_y + n_g m_g \ell_x \ell_y + s_x s_y)$ space. Also, for complex integrals the initialization will take twice as long and the computation will take 4 times as long.
- We do not assume that the wavefield is sampled on an equidistant grid. First, we interpolate the non-equidistant grid by cubic splines (for this Clough-Tocher interpolation can be used), then, we determine the function values for an equidistant grid by evaluating the interpolated splines. We then approximate the integral by the method described above. Note that this method has to do the interpolation on a non-equidistant grid for the initialization phase. Although the time complexity for the interpolation is the same for an equidistant grid the number of operations increase significantly [27, **scipy**/scipy/interpolate/interpnd.pyx], therefore this method is only advisable for large numbers of s_x and s_y, so initialization costs can be neglected. The space complexity is the same as the other separate interpolation method because the first interpolation on a non-equidistant grid can be discarded when starting with the interpolation on an equidistant grid.

Combined interpolation method

We use the method described in Subsection 4.1.2 to approximate the Rayleigh integral. Due to large deflections at the boundaries we do not use higher-order Newton-Cotes equations to approximate the integral, this is instead left to future research in Section 6.2. Also, the time complexity of this algorithm is the same as the basic method (i.e. a time complexity of $\mathcal{O}(\ell_x \ell_y s_x s_y)$ and a space complexity of $\mathcal{O}(s_x s_y)$), since this method only uses different weighting factors in front of the function values.

We assume that the grid is spaced equidistantly in this method. We do not use the method to first interpolate the data by splines after determining the function values on an equidistant grid as this supersedes the advantages of the method, it is fast and implementation is easy. Complex integrals can be evaluated in twice the time.

Altered combined interpolation method

This method is described in Subsection 4.2.2; the grid is now assumed to be spaced semi-equidistantly (see Figure 4.1) and other uncertainties are discarded. As discussed in Subsection 4.2.2 we need at least $\frac{9}{4}$ times as many operations to approximate the integral since we cannot fuse adjacent grids (since we need to calculate fractions and use function values multiple times the extra amount of operations will be much larger in our implementation). The time and space complexity are, however, still the same as the combined interpolation method (i.e. a time complexity of $\mathcal{O}(\ell_x \ell_y s_x s_y)$ and a space complexity of $\mathcal{O}(s_x s_y)$). Complex integrals will take twice as long.

Chapter 5
Results

In this chapter we approximate an artificial Rayleigh integral. The settings to synthetically construct the integral are listed in Section 5.1, and the integrand is displayed in Figure 5.1. Table 5.1 contains performance data of the evaluation of a single integral, whereas Table 5.2 contains averaged data of the computation of multiple integrals. We compare various methods to approximate the integral in Section 5.2 (the methods are listed in Section 4.3). Also, the computer code used to generate these results can be found in Appendix D.

5.1. Settings

The settings used for generating the results and constructing the artificial integral are as follows:

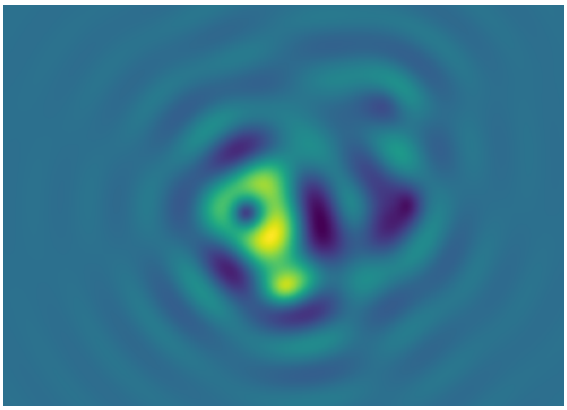
- We integrate on the interval $[-25,25] \times [-35,35]$, the integrand can be seen in Figure 5.1.
- To simulate a wavefield we used multiple sources placed at $(x_s, y_s, z_s) = (10,0,2)$, $(0,15,1.5)$, $(1,-5,1.7)$ and $(-13,13,2.3)$.
- In all results we used $k = \omega/c = 2\pi/\lambda = 0.5$, thus $\lambda = 4\pi$ (note that λ denotes wavelength).
- In Table 5.1 we used a prediction point located at $(x_A, y_A, z_A) = (0.9,-1.4,-10)$, in Table 5.2 we used multiple prediction points located at $(x_A, y_A, z_A) = (0.9,-1.4,10)$, $(2,-1,9)$, $(-1,2,11)$, $(-3,4,11.5)$, $(0.1,1.4,9.5)$, $(0.5,-1.5,10)$, $(2.1,-1.1,9)$, $(-1.1,2.1,11)$, $(-3.1,4.1,11.5)$, and $(0.2,1.5,9.5)$. The motivation to use multiple prediction points and average them was that the generated table (Table 5.1) has coincidences where the calculated integral is really close to the actual integral because deflections cancelled each other (thus a method could get lucky and have a better result than a method that is better in general), taking an average ironed (some of) these coincidences out.
- We calculated the points per wavelength from the number of sample points, the following numbers of sample points were used: $(\text{samples}_x, \text{samples}_y) = (19, 9)$, $(29, 29)$, $(49, 49)$, $(69, 69)$, $(99, 99)$, $(199, 199)$ and $(499, 499)$, respectively. We then used the following formula (dependent on direction) to calculate the points per wavelength (note that we let i denote the direction and h_i the width of the interval in that direction):

$$(\text{points}/\lambda)_i = \frac{\lambda(\text{samples}_i - 1)}{h_i}.$$

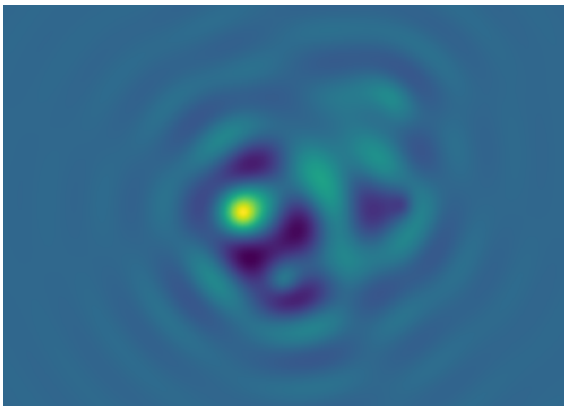
5.2. Discussion

Upon inspecting Figure 5.2a with approximately 1.44 points per wavelength we can see that there is no hope left to approximate the integral. Whereas using approximately 5 sample points

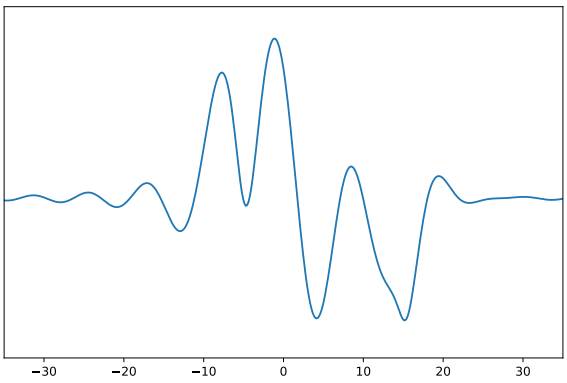
per wavelength in Figure 5.2b seems doable, resulting in relatively close integrals (see Table 5.1 and Table 5.2).



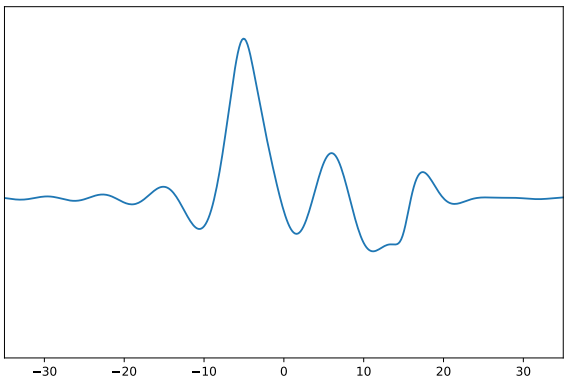
(a) Real part of $P \nabla G$ (the integrand)



(b) Imaginary part of $P \nabla G$ (the integrand)

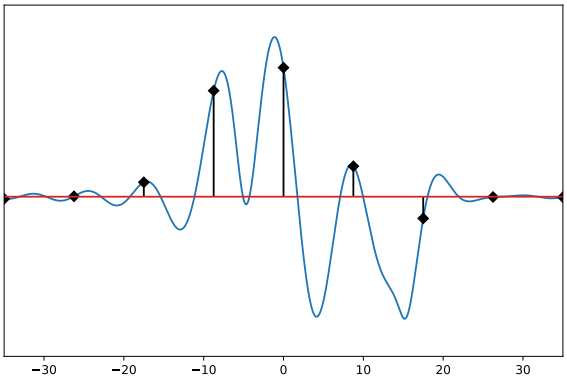


(c) Real part of $P \nabla G$ (the integrand) at $x = 0$

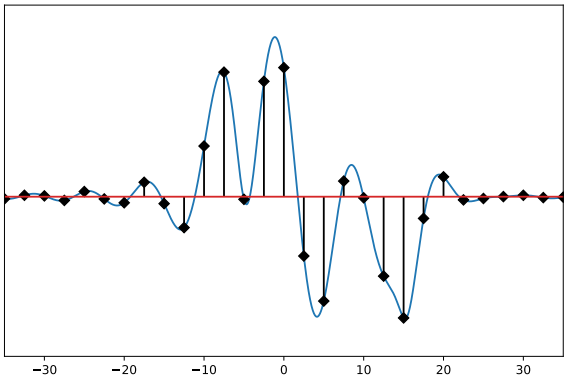


(d) Imaginary part of $P \nabla G$ (the integrand) at $x = 0$

Figure 5.1: The integrand of the artificial Rayleigh integral for a prediction point at $(x_A, y_A, z_A) = (0.9, -1.4, -10)$ and sources at locations described in Section 5.1. The Figures (a) and (b) are rotated such that the y-direction is horizontal and the x-direction is vertical. The Figures (c) and (d) can be extracted from Figures (a) and (b) by looking at the intensity of the wavefield in the middle, horizontal line (where $x = 0$).



(a) Real part of $P \nabla G$ (the integrand) at $x = 0$ with 9 sample points, i.e. approximately 1.44 sample points per wavelength.



(b) Real part of $P \nabla G$ (the integrand) at $x = 0$ with 29 sample points, i.e. approximately 5.03 sample points per wavelength.

Figure 5.2: Figure 5.1c, displayed with varying amounts of sample points (per wavelength).

Noise	$(\text{points}/\lambda)_x$ $(\text{points}/\lambda)_y$	4.52 1.44	7.04 5.03	12.1 8.62	17.1 12.2	24.6 17.6	49.8 35.5	125 89.4
0%	basic	1.5	$6.6{\cdot}10^{-3}$	$1.1{\cdot}10^{-3}$	$5.5{\cdot}10^{-4}$	$2.7{\cdot}10^{-4}$	$6.5{\cdot}10^{-5}$	$1.0{\cdot}10^{-5}$
	combined	1.60	0.305	0.298	0.827	7.06	406.	63.5
	separate	1.03	1.88	4.21	28.0	44.8	183.	$1.14{\cdot}10^{3}$
0.05%	basic	1.5	$6.6{\cdot}10^{-3}$	$1.1{\cdot}10^{-3}$	$4.7{\cdot}10^{-4}$	$2.5{\cdot}10^{-4}$	$6.0{\cdot}10^{-5}$	$1.0{\cdot}10^{-5}$
	basic_δ	1.00	1.01	1.03	0.864	0.942	0.920	1.01
	combined	1.60	0.302	0.311	0.696	5.06	15.7	11.5
	combined_δ	1.60	0.301	0.322	0.638	3.65	46.2	3.71
	separate	1.03	1.89	5.05	17.4	20.8	8.82	16.4
	separate_δ	1.02	1.88	4.42	25.1	49.4	282.	356.
0.2%	basic	1.5	$6.6{\cdot}10^{-3}$	$1.3{\cdot}10^{-3}$	$2.5{\cdot}10^{-4}$	$2.0{\cdot}10^{-4}$	$4.7{\cdot}10^{-5}$	$1.1{\cdot}10^{-5}$
	basic_δ	0.999	1.05	1.13	0.458	0.771	0.698	1.07
	combined	1.60	0.296	0.348	0.306	1.56	3.16	3.29
	combined_δ	1.61	0.296	0.404	0.242	1.25	9.89	0.996
	separate	1.02	1.90	7.32	1.77	4.72	1.76	4.29
	separate_δ	1.02	1.89	5.04	14.9	65.1	89.4	81.2
1%	basic	1.5	$8.4{\cdot}10^{-3}$	$2.1{\cdot}10^{-3}$	$1.0{\cdot}10^{-3}$	$1.6{\cdot}10^{-4}$	$6.4{\cdot}10^{-5}$	$2.1{\cdot}10^{-5}$
	basic_δ	0.995	1.60	1.51	2.06	0.764	0.712	1.76
	combined	1.62	0.343	0.518	0.464	0.271	0.861	1.32
	combined_δ	1.64	0.339	1.36	0.361	0.251	2.79	0.390
	separate	1.01	1.67	2.75	1.36	0.783	0.479	1.69
	separate_δ	1.00	2.44	8.50	105.	11.7	19.4	33.0
5%	basic	1.2	$3.0{\cdot}10^{-2}$	$6.2{\cdot}10^{-3}$	$7.3{\cdot}10^{-3}$	$1.6{\cdot}10^{-3}$	$5.0{\cdot}10^{-4}$	$9.4{\cdot}10^{-5}$
	basic_δ	0.967	2.17	2.91	10.0	3.38	0.933	0.593
	combined	1.80	0.798	0.864	0.707	0.542	1.34	1.18
	combined_δ	1.90	0.788	0.900	0.587	0.507	4.38	0.333
	separate	0.925	1.46	1.39	1.94	1.60	0.754	1.54
	separate_δ	0.818	7.85	54.6	123.	27.8	46.5	48.0
20%	basic	$7.5{\cdot}10^{-2}$	$1.3{\cdot}10^{-1}$	$1.9{\cdot}10^{-2}$	$3.0{\cdot}10^{-2}$	$7.1{\cdot}10^{-3}$	$2.2{\cdot}10^{-3}$	$3.7{\cdot}10^{-4}$
	basic_δ	0.548	1.40	1.56	3.70	0.898	0.302	0.147
	combined	0.129	1.19	1.11	0.765	0.588	1.40	1.18
	combined_δ	0.117	1.53	0.545	0.560	0.538	0.659	0.174
	separate	0.109	1.58	1.06	2.21	1.90	0.818	1.71
	separate_δ	0.0474	12.4	37.9	43.6	33.5	265.	128.

Table 5.1: For different amounts of noise (normal distributed, with a standard deviation of 0%, 0.05%, 0.2%, 1%, 5% and 20% of the length between equidistant points) and different amounts of samples points per wavelength varying per direction ((4.52, 1.44), (7.04, 5.03), (12.1, 8.62), (17.1, 12.2), (24.6, 17.6), (49.8, 35.5), (125, 89.4)) we listed the relative error (the mean-squared error was used due to working with complex numbers) of the approximation of the simulated Rayleigh integral (with settings listed in Section 5.1) using the basic method described in Section 4.3. Also, the other methods described in Section 4.3 are listed (where the adapted version of the method on a non-equidistant grid is denoted with a subscript δ) with their relative error, divided by the relative error of the basic method. We thus have that the numbers represent the number of times that the used method is better than the basic method. We note that for a noise of 0% the adapted versions of each method are the same as the regular methods, since we have no uncertainty in gridpoints. Furthermore, to give an example: for a noise of 0.2% (thus each gridpoints is shifted in the x-direction with $\mathcal{N}(0.2\% \cdot h_x)$ and in the y-direction with $\mathcal{N}(0.2\% \cdot h_y)$, where h_x, h_y denote the distance between the equidistant gridpoints in the x and y-directions respectively and $\mathcal{N}(\sigma)$ denotes normal distributed noise with standard deviation σ) and 12.1 samples per wavelength in the x-direction and 8.62 samples per wavelength in the y-direction, the basic method has a relative error of $1.3 \cdot 10^{-3}$, whereas the altered separate method (separate_δ) has a relative error that is 5.04 times as low, i.e. $1.3/5.04 \cdot 10^{-3} \approx 2.6 \cdot 10^{-4}$ and the combined method has a relative error that is larger than the error of the basic method with a factor of $1/0.348 \approx 2.87$.

Noise	$(\text{points}/\lambda)_x$	4.52	7.04	12.1	17.1	24.6	49.8	125
	$(\text{points}/\lambda)_y$	1.44	5.03	8.62	12.2	17.6	35.5	89.4
0%	basic	1.2	$3.9{\cdot}10^{-3}$	$1.2{\cdot}10^{-3}$	$5.8{\cdot}10^{-4}$	$2.8{\cdot}10^{-4}$	$6.8{\cdot}10^{-5}$	$1.1{\cdot}10^{-5}$
	combined	1.51	0.173	0.323	3.31	13.5	219.	62.0
	separate	1.01	1.01	13.0	63.3	104.	433.	$2.66{\cdot}10^{3}$
0.05%	basic	1.2	$3.9{\cdot}10^{-3}$	$1.2{\cdot}10^{-3}$	$5.4{\cdot}10^{-4}$	$2.8{\cdot}10^{-4}$	$6.6{\cdot}10^{-5}$	$1.0{\cdot}10^{-5}$
	basic_δ	1.00	1.02	1.01	0.932	0.985	0.958	0.953
	combined	1.51	0.173	0.321	2.30	5.41	19.8	9.10
	combined_δ	1.51	0.172	0.326	3.44	6.42	22.5	4.14
	separate	1.01	1.02	13.4	13.1	26.3	19.0	12.5
	separate_δ	1.01	1.01	14.0	76.0	136.	890.	$1.22{\cdot}10^{3}$
0.2%	basic	1.1	$4.1{\cdot}10^{-3}$	$1.2{\cdot}10^{-3}$	$4.8{\cdot}10^{-4}$	$2.7{\cdot}10^{-4}$	$6.0{\cdot}10^{-5}$	$9.1{\cdot}10^{-6}$
	basic_δ	1.00	1.07	1.03	0.845	0.955	0.869	0.886
	combined	1.51	0.176	0.315	1.78	1.88	5.05	2.30
	combined_δ	1.52	0.173	0.338	3.42	2.48	5.05	0.903
	separate	1.01	1.08	7.09	2.63	6.75	4.68	2.87
	separate_δ	1.01	1.05	18.7	130.	265.	159.	188.
1%	basic	1.1	$5.9{\cdot}10^{-3}$	$1.7{\cdot}10^{-3}$	$1.2{\cdot}10^{-3}$	$4.4{\cdot}10^{-4}$	$6.5{\cdot}10^{-5}$	$1.4{\cdot}10^{-5}$
	basic_δ	0.999	1.52	1.40	2.74	2.25	1.15	1.58
	combined	1.52	0.229	0.409	0.808	0.733	1.18	0.832
	combined_δ	1.54	0.211	0.727	1.72	0.918	1.69	0.311
	separate	1.00	1.23	1.91	1.43	2.24	1.03	1.08
	separate_δ	0.993	1.50	16.4	62.1	58.1	27.6	48.1
5%	basic	$9.3{\cdot}10^{-1}$	$2.1{\cdot}10^{-2}$	$5.2{\cdot}10^{-3}$	$6.5{\cdot}10^{-3}$	$1.9{\cdot}10^{-3}$	$2.4{\cdot}10^{-4}$	$8.6{\cdot}10^{-5}$
	basic_δ	0.991	1.57	3.11	10.9	7.19	0.741	0.858
	combined	1.61	0.578	0.712	0.844	0.697	0.842	0.988
	combined_δ	1.73	0.457	1.82	1.74	0.799	1.76	0.369
	separate	0.936	1.37	1.09	1.60	1.98	0.628	1.24
	separate_δ	0.828	5.25	37.9	102.	108.	27.0	86.4
20%	basic	$7.2{\cdot}10^{-2}$	$9.0{\cdot}10^{-2}$	$1.8{\cdot}10^{-2}$	$2.7{\cdot}10^{-2}$	$7.4{\cdot}10^{-3}$	$1.1{\cdot}10^{-3}$	$3.6{\cdot}10^{-4}$
	basic_δ	0.886	1.16	3.94	3.89	1.04	0.224	0.223
	combined	0.159	1.00	0.884	0.858	0.694	0.923	1.04
	combined_δ	0.130	0.821	1.09	3.32	0.429	0.550	0.216
	separate	0.145	1.38	0.903	1.72	1.94	0.675	1.33
	separate_δ	0.0687	8.67	21.4	85.6	48.3	112.	231.

Table 5.2: The average of Table 5.1 for the 10 prediction points listed in Section 5.1. For further information on the meaning of the values the reader is referred to the caption of Table 5.1.

Let us first have a look at Table 5.2. Note that the cells using the basic method have different units (relative error) than the cells using other methods (relative error/relative error = times better compared to basic method).

Generally, we expect that fewer points per wavelength result in higher relative errors and more noise also results in higher relative errors. This relation does not hold in the first column (with 4.52, 1.44 points per wavelength) of the table as we go from a relative error in the basic method of 1.2 at a noise of 0% to a relative error of $7.2 \cdot 10^{-2}$ at a noise of 20%. This is most likely a coincidence (that did not smooth out in the average) as other methods did not have such sudden improvements. If we go from 17.1, 12.2 sample points per wavelength to 12.1, 8.62 sample points per wavelength (thus from the fourth column to the third) and compare

the relative errors in the basic methods with 5% and 20% noise, the error decreases. Also, this effect is likely to be caused by chance, with such a high noise in the gridpoints the decrease in sample points does not outrank the coincidences caused by the noise. All in all, we can see that apart from a few exceptions the relation holds (as expected).

The altered basic method (basic$_\delta$) performs better than the basic method with a noise of 1% and not much worse for less noise. However, for more noise (5% and 20%) and numerous points per wavelength it performs worse than the basic method. This is probably due to the method not being exact for completely randomized grids, but only for rectilinear grids. This limitation of the method combined with chance caused it to perform worse than the basic method.

If the integrand is well sampled and has a low amount of noise the combined method overtakes the basic method. If noise increases the method is only a little worse than the basic method, likely due to inability to average the result as good as the basic method (it is more prone to errors and noise). Since this method is relatively easy to implement it is recommended to use only with a noise below 1% and more than 10 sample points per wavelength (in both directions). If the noise increases, we can see no reason to choose this method over the basic method and if the number of sample points decreases we mainly have worse results.

The adaptation of the combined interpolation method (combined$_\delta$) often exceeds the regular combined method except for a large amount of sample points per wavelength. Again, the reason for this is probably the fact that the randomized grid is not a semi-equidistant grid. The trade-off for more operations hardly seems worth it.

The separate method performs really well with less noise and many sample points per wavelength, like the combined method. This method, if compared to the combined method is generally better (except for the first column with the fewest sample points per wavelength). Also, this method extends better to a low sample points per wavelength range (except the lowest). The cost of using this method is that it is more difficult to implement and depending on the implementation (although it has the same time complexity as can be seen in Section 4.3) it can take longer to compute the integral.

The final method, the modified separate method (separate$_\delta$), has the highest accuracy compared to all other described methods from the third column on (that is with more than 12.1, 8.62 sample points). Especially with a high amount of noise (e.g. 1%) this method really exceeds the others (by a factor of at least 15 if compared to the basic method). This method also has peak performance with a lot of sample points per wavelength, but the remarkable thing is that with less sample points it still outperforms most of the other methods. The cost of using this method is the implementation as it is the most difficult of all methods, nonetheless, the method makes up for this by accuracy.

Now, we provide a note on the time complexity of the methods from Section 4.3. The basic method, the altered basic method, the combined method and its modification all have the same time complexity (although for the alterations we do need more operations). Compared to the separate methods the time complexity is better by a factor of $n_f m_f$ (the orders of the polynomials for the interpolation). This is, however, only true when using splines. In that case we need to process $n_f m_f$ coefficients per subinterval, compared to only 1 using combined methods. If instead polynomials are used for this separate method (this is left to future research in Section 6.2) we get the same time complexity (apart from initialization costs, which can be neglected for many sensors). The space complexity of the separate method does not have a serious limitation and is therefore only mentioned for completeness.

After comparing the values in Table 5.2 with Table 5.1 we can say something about the consistency of the method and see if there are outliers (remember that the former table took an average over multiple prediction points and the latter did not).

The basic method still has approximately the same error in this table.

Before the average was taken over multiple prediction points the basic$_\delta$ method is more unpredictable. Therefore, we do not think it is worth the extra processing time.

Except for situations with noise below 0.2% and more than 24.6, 17.6 sample points per wavelength in the x and y-direction respectively the combined method is generally worse than the basic method. We can also see this effect in Table 5.2, although, the break-even point happened at 17.1, 12.2 sample points per wavelength, we thus conclude that this method cannot be relied upon for consistent results with moderate sampling amounts.

The adaptation of the combined method is also more unpredictable in this situation, instead of being (somewhat) consistently better than the unmodified method it is now at odds with it.

The separate method performed about the same, the numbers just varied more in Table 5.1, whereas the sudden improvements were smoothed out in Table 5.2. The same effect occurred for the modified separate method, the factor that this method is better than the basic method does not always increase as the amount of sample points increase, nevertheless, the general trend is upward (since if we take the average we get the smoother version in Table 5.2).

In conclusion:

- The modified basic method is inconsistent.
- The combined method only performs well in the high sample points per wavelength range with hardly any noise.
- The adapted combined method is not worth the extra computation time compared to the original version.
- The separate method often performs better than the combined method and extends better to fewer sample points but takes more time to implement.
- The altered separate method performs the best, exceeding the performance of all other methods in most situations.

Chapter 6

Conclusion and outlook

6.1. Conclusion

In this thesis we started with deriving the Rayleigh integral and subsequently developed polynomial interpolation methods for numerical integration in order to determine whether modifying the integration methods would reduce the error in approximations of the Rayleigh integral. The presented methods are:

- The basic method; this is the one that is currently used for approximating the Rayleigh integral.
- An adaptation of the basic method; the basic method for a rectilinear grid.
- The combined interpolation method; Simpson's rule in our case.
- The modified combined interpolation method; an adaptation for a semi-equidistant grid.
- The separate interpolation method; this method separately interpolates the unknown part of the integrand (to a low degree) and the known part of the integrand (to a higher degree) and combines the results.
- The altered separate interpolation method; this is an adaptation of the separate interpolation method to non-equidistant grids.

These different integration methods were then used to approximate a synthetic Rayleigh integral with varying amounts of noise in the sample points.

The best performing method is the altered version of the separate interpolation method. It outperforms all other methods in accuracy and is on average, in comparison with the currently used method, at least 15 times better in the moderate to well sampled range[1].

Our adjustment to the basic method only performed marginally better and the cost of more operations is not beneficial.

The combined interpolation approach performed well with relatively small noise (up to about 1% of the distance between gridpoints) and more than 10 sample points per wavelength (in both directions). However, the improvement on the approximation fluctuates too much to be relied upon if the conditions are not met. An advantage of this method is that it is relatively easy to implement.

The modified version of the method also has a lot of fluctuations, giving inconsistent results that are on average only marginally better than the original version. Therefore, we do not think that the extra computation time is worth it in comparison to the combined method.

[1] This is the achieved with more than 12 sample points per wavelength in the x-direction and more than 8 sample points per wavelength in the y-direction.

The separate interpolation method scores only slightly better than the combined method, it had peak-performance with almost no noise and numerous sample points per wavelength. However, this method is more difficult to implement and will likely take longer to evaluate than the combined method, hence, this method is not recommended.

6.2. Future Research

In future research we recommend to look at making an "accuracy vs speed" analysis. In seismic imaging it is crucial that the first propagations of the wavefield are accurate, if the error in these propagations are too large one has no hope to arrive at a decent answer after propagating it hundreds of times more. Since we presented methods to increase accuracy a researcher could look at using different methods for different propagations, to get a better result in the first few propagations and to limit the error in the end result. Our presented methods likely take more time to evaluate, therefore, an optimum can be found if we compare accuracy to speed and use various methods.

Placing the sensors is important, the methods presented could yield better results if the grid is not placed approximately equidistant, i.e. some the methods could perform better on a different grid. An example of this could be a triangular grid, which could also result in time advantages using the modified separate interpolation method as this relies on triangulating the input data. In addition, one could look at systematic errors in gridpoints and the impact of those on the accuracy of the different methods.

Also, an implementation of some algorithms would be needed to perform a speed analysis of the methods. Even though the time complexity of the algorithms is the same, it depends on the computer used and the implementation of the method.

Another combined interpolation method of using higher-order polynomials was discussed shortly in this thesis. The methods were not developed further because of great deflections on the boundaries of the interpolation. In Section 3.3 we provided a way to circumvent this problem, giving adequate results in that example. For these methods there are a lot of possible variations, as one could go to arbitrary orders of polynomials. In future research one could look at the accuracy of those methods as they would be easy to implement.

The separate interpolation method could also be made easier to implement by using polynomial interpolation instead of spline interpolation. We did not compute the results for this new method. This would promote the separate interpolation even more as the downside of difficult implementations would reduce.

Chapter A

Verification of the wave equation

This is an intuitive verification of the one-dimensional acoustic wave equation[1] (the three-dimensional version is derived in Section 2.1). To start off with this approach, let us assume that there is no sound. Then, no vibrations in the air can occur which, in turn, results in equal pressure in the air not varying with position (denoted p_0).

However, if there is sound we do get vibrations and therefore some variation of pressure depending on the position you are measuring the sound as can be seen in Figure A.1.

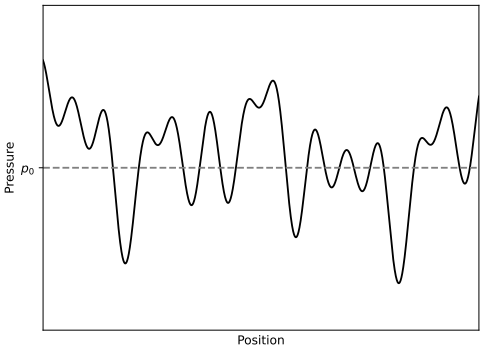


Figure A.1: Some function of pressure depending on position that will change with time.

To gain intuition for what happens over time let us discretize the pressure function and pick only 3 "adjacent" points from the function. This can also be thought of as zooming in on Figure A.1 until you see only 3 points. An example of this can be seen in Figure A.2a. We take look at how the pressure of the middle point, p_2, changes as a function of its differences with p_1 and p_3. After rewriting this function we look at the limit and get the partial differential equation named the one-dimensional acoustic wave equation.

So, to begin we start at $t = 0$ with Figure A.2a, the three points are denoted x_1, x_2, and x_3 with corresponding pressures p_1, p_2, and p_3, respectively. We assume that p_1 and p_3 are fixed and investigate the function p_2 describes (note that the values 1 for p_1, 4 for p_2 and -1 for p_3 are chosen arbitrarily and can be scaled appropriately). We start with the situation where the air at x_1, x_2, and x_3 has no velocity. Because there is a high pressure at x_2 and lower pressures at x_1 and x_3, the system wants to "equalize" the pressure, and we get a large acceleration of air from x_2 towards x_1 and x_3 (from now on we say that p_2 has a large acceleration towards p_0, the static pressure), but since the system needs time to "start up" the velocity of the individual particles stays small but will increase rapidly. The total change in pressure will thus be little, giving the situation in Figure A.2b.

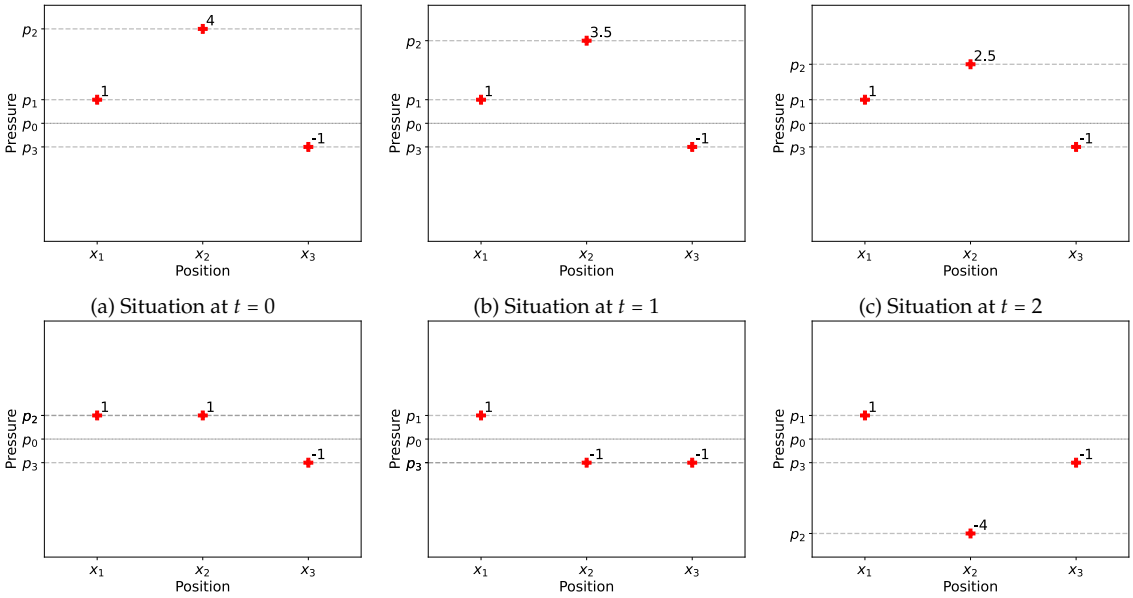
Now the velocity of the individual particles, i.e. the change in pressure, has started to build up, although the acceleration of p_2 towards p_0 has started to decrease due to the decreased difference to p_1 and p_3 (the system does not want to "equalize" the pressure as much) thus the velocity of p_2 towards p_0 will not increase as much as last time. We obtain the situation in Figure A.2c.

[1] Notation and ideas used in this appendix are adapted from 3Blue1Brown [28].

The velocity still increases due to a positive acceleration but with each timestep the acceleration decreases. Hence, the velocity will increase less and less, resulting in the situation in Figure A.2d.

At this point we have $p_1 = p_2$ meaning that x_1 and x_2 are in equilibrium, however, x_2 and x_3 are not. So there will still be a positive acceleration, due to a high velocity, the pressure in x_2, p_2, shoots past the static pressure p_0. At the next timestep we see something like the situation in Figure A.2e.

The velocity starts to decrease because of the difference in pressure at x_1 and x_2, thus we have negative acceleration. But the velocity is still positive thus we still get a decreasing pressure in x_2. Due to the negative acceleration that keeps on increasing as p_3 decreases, the velocity will eventually be 0 again. This happens in the situation in Figure A.2f, at $t = 7$.



(a) Situation at $t = 0$ (b) Situation at $t = 1$ (c) Situation at $t = 2$

(d) Situation at $t = 3$ (e) Situation at $t = 4$ (f) Situation at $t = 7$

Figure A.2: Zoomed in version of Figure A.1, here $p_1 = p(x_1)$, $p_2 = p(x_2)$ and $p_3 = p(x_3)$. Note that the timesteps can be scaled appropriately.

We have gotten a more intuitive feel of how the differences in pressure increase the acceleration of the pressure. To put that thought in formulas, we get that the acceleration of the pressure (its 2nd time derivative) is a function of the sum of differences between p_1 and p_2 and between p_2 and p_3, thus after defining some constant α and noting that positive acceleration defined is upwards we get (note that Δ denotes the spatial difference):

$$\frac{\partial^2 p}{\partial t^2} = -\alpha(\underbrace{(p_2 - p_1)}_{\Delta p_1} + \underbrace{(p_2 - p_3)}_{-\Delta p_2}) = \alpha \underbrace{(\Delta p_2 - \Delta p_1)}_{\Delta\Delta p_1}. \tag{A.1}$$

Thus, $\frac{\partial^2 p}{\partial t^2}$ is a function of the spatial difference of Δp_1 and Δp_2, thus it can be denoted by the second spatial difference $\Delta\Delta p_1$. Now, if we take the limit so that the difference in positions goes to zero ($(x_2 - x_1) \to 0$ and $(x_3 - x_2) \to 0$ as well). We get that these differences turn into spatial derivates. Then A.1 becomes

$$\frac{\partial^2 p}{\partial t^2} = \alpha \frac{\partial^2 p}{\partial x^2},$$

which is the one-dimensional acoustic wave equation.

Chapter B

Proofs concerning Cotesian numbers

Here we provide proofs for statements from Subsection 3.2.2.

Lemma 1. *The Vandermonde matrix*

$$V = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \cdots & n^n \end{bmatrix}$$

is invertible.

Proof. We show by contradiction that the columns of V are linearly independent.

Denote the j-column of matrix V as $\mathbf{v}_j$, we thus have $\mathbf{v}_j = \begin{bmatrix} 0^j & 1^j & \cdots & n^j \end{bmatrix}^\top \in \mathbb{R}^{n+1}$. Assume that there exist coefficients $a_0, a_1, \ldots, a_n \in \mathbb{R}$ such that $a_0\mathbf{v}_0 + a_1\mathbf{v}_1 + \cdots + c_n\mathbf{v}_n = \mathbf{0}$, where $\mathbf{0} = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}^\top \in \mathbb{R}^{n+1}$. Then, for each $k \in \mathbb{N}$ with $0 \le k \le n$ we get

$$a_0 + a_1 k + a_2 k^2 + \cdots + a_n k^n = 0\,.$$

Hence, k is a root of the polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$. This means that the polynomial $f(x)$ has $n+1$ different roots. Since $f(x)$ is at most an nth-order polynomial we must have $a_0 = a_1 = \cdots = a_n = 0$. Proving that the columns of matrix V are indeed linearly independent, therefore, the matrix is invertible. □

Theorem 2. *Define $f : \mathbb{R} \to \mathbb{R}$, and let $x_0, x_1, \ldots, x_n \in \mathbb{R}$ denote equidistant real numbers. Denote their function values as $f_0, f_1, \ldots f_n \in \mathbb{R}$, respectively (thus $f_0 = f(x_0)$, $f_1 = f(x_1)$, $\ldots$, $f_n = f(x_n)$). Furthermore, let h denote the distance between the equidistant numbers, that is, $h = \frac{x_n - x_0}{n}$. Also, let $c_0, c_1, \ldots c_n$ denote the Cotesian numbers, i.e. the numbers such that after interpolating the function values by a polynomial of degree at most n the integral over the polynomial can be approximated by*

$$\int_{x_0}^{x_n} f(x)\mathrm{d}x \approx h(c_0 f_0 + c_1 f_1 + \cdots + c_n f_n)\,.$$

Then, $c_0, c_1, \ldots, c_n$ satisfy

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \cdots & n^n \end{bmatrix}^\top \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} n/1 \\ n^2/2 \\ n^3/3 \\ \vdots \\ n^{n+1}/(n+1) \end{bmatrix}.$$

Proof. Without loss of generality we let $x_0 = 0$ (this can be verified by defining $x_0' = x_0 - x_0$, $x_1' = x_1 - x_0, \ldots, x_n' = x_n - x_0$ and noting that $\int_{x_0}^{x_n} f(x)\mathrm{d}x = \int_0^{x_n'} f(x')\mathrm{d}x'$). Employing this new definition allows us to write $x_i = ih$ for $0 \le i \le n$.

All interpolating polynomials $p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n$ must satisfy $p(x_0) = f_0$, $p(x_1) = f_1, \ldots, p(x_n) = f_n$, hence, the coefficients $p_0, p_1, \ldots, p_n$ satisfy

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}}_{\mathbf{p}} = \underbrace{\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}}_{\mathbf{f}} . \tag{B.1}$$

Also, we know that integrating $p(x)$ yields the approximation of the integral, equalling $h(c_0 f_0 + c_1 f_1 + \cdots + c_n f_n)$, giving the relation

$$\begin{aligned} \int_{x_0}^{x_n} p(x)\mathrm{d}x = \int_0^{nh} p(x)\mathrm{d}x &= \left[p_0 x + p_1 \frac{x^2}{2} + p_2 \frac{x^3}{3} + \cdots + p_n \frac{x^{n+1}}{n+1} \right]_0^{nh} \\ &= p_0 \frac{nh}{1} + p_1 \frac{(nh)^2}{2} + p_2 \frac{(nh)^3}{3} + \cdots + p_n \frac{(nh)^{n+1}}{n+1} = h(c_0 f_0 + c_1 f_2 + \cdots + c_n f_n) . \end{aligned}$$

After removing a factor h from both sides of the equal sign we use vector notation to rewrite the equation into

$$\underbrace{\begin{bmatrix} n/1 \\ n^2 h/2 \\ n^3 h^2/3 \\ \vdots \\ n^{n+1} h^n/(n+1) \end{bmatrix}^\top}_{\mathbf{n}^\top} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} = \underbrace{\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}^\top}_{\mathbf{c}^\top} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} . \tag{B.2}$$

This allows for shorter notation; equation B.1 can be written as

$$A\mathbf{p} = \mathbf{f} ,$$

and equation B.2 can be written as

$$\mathbf{n}^\top \mathbf{p} = \mathbf{c}^\top \mathbf{f} .$$

Using these equalities we can deduce that (note that A is invertible due to Lemma 1):

$$\begin{aligned} \mathbf{p} &= A^{-1}\mathbf{f} \\ \mathbf{n}^\top \mathbf{p} = \mathbf{n}^\top A^{-1}\mathbf{f} &= \mathbf{c}^\top \mathbf{f} \\ \mathbf{f}^\top (A^{-1})^\top \mathbf{n} &= \mathbf{f}^\top \mathbf{c}, \quad \text{for arbitrary } \mathbf{f} \\ (A^{-1})^\top \mathbf{n} &= \mathbf{c} \\ \mathbf{n} &= A^\top \mathbf{c} . \end{aligned}$$

Writing this out and substituting $x_i = ih$ gives us:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & h & h^2 & \cdots & h^n \\ 1 & 2h & (2h)^2 & \cdots & (2h)^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & nh & (nh)^2 & \cdots & (nh)^n \end{bmatrix}^\top \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} n/1 \\ n^2 h/2 \\ n^3 h^2/3 \\ \vdots \\ n^{n+1} h^n/(n+1) \end{bmatrix} .$$

Removing h (we can this do due to the transposition of the matrix) results in

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1^2 & \cdots & 1^n \\ 1 & 2 & 2^2 & \cdots & 2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \cdots & n^n \end{bmatrix}^\top \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} n/1 \\ n^2/2 \\ n^3/3 \\ \vdots \\ n^{n+1}/(n+1) \end{bmatrix},$$

concluding our proof. □

Definition 3. *Let $X = \{X_1, X_2, \ldots, X_n\}$ with $X_1, X_2, \ldots, X_n \in \mathbb{R}$ and let $k \in \mathbb{N}_{\geq 0}$. The elementary symmetric polynomial is then defined as:*

$$e_k(X) = \begin{cases} 1 & \text{if } k = 0\,, \\ \displaystyle\sum_{1 \leq m_1 < \cdots < m_k \leq n} X_{m_1} \cdots X_{m_k} & \text{if } 1 \leq k \leq n\,, \\ 0 & \text{if } k > n\,. \end{cases}$$

Lemma 4. *Let $s(n,k)$, with $n, k \in \mathbb{N}_{\geq 0}$ denote the (un)signed Stirling numbers of the first kind, then, if $0 \leq j \leq \ell$ with $j, \ell \in \mathbb{N}$, the following equality holds:*

$$e_j(\{1, 2, \ldots, \ell\}) = |s(\ell + 1, \ell + 1 - j)|\,. \tag{B.3}$$

Proof. We use induction twice to complete the proof.

First, we use induction on j. For the induction base with $j = 0$ we note that by definition: $e_j(\{1, 2, \ldots, \ell\}) = 1 = |s(\ell + 1, \ell + 1)|$. For the induction hypothesis we assume that $e_{j-1}(\{1, 2, \ldots, \ell\}) = |s(\ell + 1, \ell + 2 - j)|$. For the induction step we then need to show that equation B.3 holds.

We do this using induction on ℓ. For the induction base $\ell = 0$ and thus $j = 0$, we get $e_0(\varnothing) = 1 = |s(1,1)|$. For the induction hypothesis we assume $e_j(\{1, 2, \ldots, \ell - 1\}) = |s(\ell, \ell - j)|$, applying this to our previous assumption yields $e_{j-1}(\{1, 2, \ldots, \ell - 1\}) = |s(\ell, \ell + 1 - j)|$. For the induction step we need to prove equation B.3. Note that by the definition of the Stirling numbers we have [29, equation 15]:

$$|s(n,k)| = |s(n-1, k-1)| + (n-1)|s(n-1, k)|\,.$$

Using our induction hypotheses we now show that equation B.3 holds

$$\begin{aligned} e_j(\{1, 2, \ldots, \ell\}) &= \sum_{1 \leq m_1 < \cdots < m_j \leq \ell} m_1 \cdots m_j \\ &= \sum_{\substack{1 \leq m_1 < \cdots < m_{j-1} \leq \ell - 1 \\ m_{j-1} < m_j < \ell}} m_1 \cdots m_j + \sum_{\substack{1 \leq m_1 < \cdots < m_{j-1} \leq \ell - 1 \\ m_{j-1} < m_j = \ell}} m_1 \cdots m_j \\ &= \sum_{1 \leq m_1 < \cdots < m_j \leq \ell - 1} m_1 \cdots m_j + \ell \sum_{1 \leq m_1 < \cdots < m_{j-1} \leq \ell - 1} m_1 \cdots m_{j-1} \\ &= e_j(\{1, 2, \ldots, \ell - 1\}) + \ell e_{j-1}(\{1, 2, \ldots, \ell - 1\}) \\ &= |s(\ell, \ell - j)| + \ell |s(\ell, \ell + 1 - j)| \\ &= |s(\ell + 1, \ell + 1 - j)|\,. \end{aligned}$$

Hence, the induction step, equation B.3, holds, concluding our proof. □

Lemma 5. *Let $0 \leq i \leq n$ with $i \in \mathbb{N}$, also, let $0 \leq k \leq n$ with $k, n \in \mathbb{N}$, then the following relation holds:*

$$e_k(\{1,\dots,n\} \setminus \{i\}) = \sum_{m=0}^{k} (-i)^m e_{k-m}(\{1,\dots,n\}) .$$

Proof. This relation can easily be seen to hold for $1 \leq i \leq n$ since

$$e_k(\{1,\dots,n\} \setminus \{i\}) = e_k(\{1,\dots,n\}) - i e_{k-1}(\{1,\dots,n\} \setminus \{i\}) ,$$

and $e_0(\{1,\dots,n\} \setminus \{i\}) = 1 = e_0(\{1,\dots,n\})$.

For $i = 0$ the relation also holds since $e_k(\{1,\dots,n\} \setminus \{0\}) = e_k(\{1,\dots,n\}) - 0$. □

Theorem 6. *We can calculate the coefficients of c_i for $i \in \{0,\dots,n\}$ in the following equation*

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1^2 & \cdots & 1^n \\ 1 & 2 & 2^2 & \cdots & 2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \cdots & n^n \end{bmatrix}^\top \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} n/1 \\ n^2/2 \\ n^3/3 \\ \vdots \\ n^{n+1}/(n+1) \end{bmatrix} \tag{B.4}$$

by computing

$$c_i = \frac{1}{(n-1)!} \binom{n}{i} \sum_{j=0}^{n} \sum_{m=0}^{n-j} i^m n^j \frac{(-1)^{i+n} s(n+1, j+m+1)}{j+1} .$$

Proof. We first rewrite equation by transposing the matrix in equation B.4 and bringing it to the other side, yielding

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 2 & \cdots & n \\ 0 & 1^2 & 2^2 & \cdots & n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1^n & 2^n & \cdots & n^n \end{bmatrix}^{-1}}_{(W_{n+1}^{-1})^\top} \begin{bmatrix} n/1 \\ n^2/2 \\ n^3/3 \\ \vdots \\ n^{n+1}/(n+1) \end{bmatrix} ,$$

where the matrix W_{n+1} is the Vandermonde matrix defined according to Pr∞fWiki [30]. The inverse of the matrix (without transposition) is also given by Pr∞fWiki and turns out to be

$$[W_{n+1}^{-1}]_{ij} = \frac{(-1)^{n-i} e_{n-i}(\{0,1,\dots,n\} \setminus \{j\})}{\prod_{m=0, m\neq j}^{n+1} (j-m)} ,$$

note that in the citation $i, j \in \{1,\dots,n\}$, whereas here we define $i, j \in \{0,\dots,n\}$. Transposing this matrix and noting that we can safely omit 0 in the set yields

$$[(W_{n+1}^{-1})^\top]_{ij} = \frac{(-1)^{n-j} e_{n-j}(\{1,\dots,n\} \setminus \{i\})}{\prod_{m=0, m\neq i}^{n} (i-m)} .$$

Using this matrix we can then compute the coefficients of c_i by using the following relation

$$\begin{aligned} c_i &= \sum_{j=0}^{n} \left[(W_{n+1}^{-1})^\top\right]_{ij} n^{j+1}/(j+1) \\ &= \sum_{j=0}^{n} n^{j+1} \frac{(-1)^{n-j} e_{n-j}(\{1,\dots,n\} \setminus \{i\})}{(j+1) \prod_{m'=0, m'\neq i}^{n} (i-m')} . \end{aligned} \tag{B.5}$$

Since $0 \leq i \leq n$ we can simplify the product in the denominator to

$$\prod_{m=0, m\neq i}^{n} (i-m) = i \cdot (i-1) \cdots 2 \cdot 1 \cdot -1 \cdot -2 \cdots (i-n-1) \cdot (i-n) = i!(n-i)!(-1)^{n-i}.$$

Also, from Lemma 4 in combination with Lemma 5 we have that

$$e_{n-j}(\{1,\ldots,n\} \setminus \{i\}) = \sum_{m=0}^{n-j} (-i)^m |s(n+1, n+1-(n-j-m))|.$$

Substituting these relations into equation B.5 yields

$$\begin{aligned}
c_i &= \sum_{j=0}^{n} n^{j+1} \frac{(-1)^{n-j} e_{n-j}(\{1,\ldots,n\} \setminus \{i\})}{(j+1) i!(n-i)!(-1)^{n-i}} \\
&= \sum_{j=0}^{n} n^{j+1} \frac{(-1)^{n-j} \sum_{m=0}^{n-j} (-i)^m |s(n+1, n+1-(n-j)+m)|}{(j+1) i!(n-i)!(-1)^{n-i}} \\
&= \frac{1}{(n-1)!} \binom{n}{i} \sum_{j=0}^{n} n^j \frac{(-1)^{i-j} \sum_{m=0}^{n-j} (-i)^m |s(n+1, j+m+1)|}{j+1} \\
&= \frac{1}{(n-1)!} \binom{n}{i} \sum_{j=0}^{n} \sum_{m=0}^{n-j} i^m n^j \frac{(-1)^{i-j+m} (-1)^{n-j-m} s(n+1, j+m+1)}{j+1} \\
&= \frac{1}{(n-1)!} \binom{n}{i} \sum_{j=0}^{n} \sum_{m=0}^{n-j} i^m n^j \frac{(-1)^{i+n} s(n+1, j+m+1)}{j+1}.
\end{aligned}$$

Proving the theorem. □

Chapter C

Matrices from the example calculation

These are the matrices from the example calculation in Section 3.3 for the separate interpolation method described in Section 3.1. The computer code for generating these matrices can be found in Appendix D.

$$B = \begin{bmatrix} -0.87832 & -0.26219 & 0.14557 & -0.0097228 \\ -0.97689 & 0.10889 & 0.10182 & -0.0097228 \\ -0.61727 & 0.34872 & 0.058066 & -0.019683 \\ -0.02998 & 0.39005 & -0.03051 & -0.014518 \\ 0.43745 & 0.20053 & -0.095841 & -0.0025022 \\ 0.51415 & -0.10389 & -0.1071 & 0.011436 \\ 0.15594 & -0.348 & -0.055639 & 0.018071 \\ -0.43025 & -0.39293 & 0.025681 & 0.018071 \end{bmatrix},$$

$$C = \begin{bmatrix} -0.83134 & 0.26959 & 0.096787 & -0.011336 & -0.0019914 & 0.00021437 \\ -0.2559 & 0.46198 & 0.026152 & -0.018416 & -0.00030821 & 0.00024732 \\ 0.43407 & 0.41822 & -0.052521 & -0.014786 & 0.0015849 & 0.0001794 \\ 0.90271 & 0.18678 & -0.091602 & -0.0013727 & 0.0029417 & 0.000096965 \\ 0.98777 & -0.055268 & -0.053976 & 0.016086 & 0.0059934 & -0.001363 \\ 0.85769 & -0.062041 & 0.05026 & 0.024694 & -0.014291 & 0.0013866 \\ 0.89925 & 0.097732 & 0.017243 & -0.028331 & 0.0014721 & 0.00040516 \\ 0.99955 & -0.011708 & -0.077081 & -0.011424 & 0.0035001 & -0.00012811 \end{bmatrix},$$

$$D = \begin{bmatrix} -0.8518 & -0.52995 & -0.47276 & -0.49211 \\ 0.14199 & 0.23699 & 0.3016 & 0.38253 \\ 1.0466 & 0.87334 & 0.91425 & 1.0546 \\ 1.4641 & 1.1135 & 1.1177 & 1.2588 \\ 1.3856 & 1.0132 & 1.0001 & 1.1164 \\ 1.2855 & 0.97247 & 0.97915 & 1.1071 \\ 1.4454 & 1.1042 & 1.1126 & 1.2566 \\ 1.3901 & 1.0027 & 0.97905 & 1.0837 \end{bmatrix}.$$

Chapter D

Computer code

This computer code is for the example calculation conducted in Section 3.3. The implemented method is from Section 3.1 and can function as a guideline for future research.

```
'''
In this script we approximate the integral of f(x)*g(x) over the
interval [-x_L, x_R]. The used method is described in chapter 3.1
and the used alteration to the method is given in chapter 3.3.
'''
import numpy as np
import numpy.polynomial.polynomial as np_pol
from scipy.interpolate import interp1d


### We first define both functions
def f(x):
   return np.cos(np.sqrt((x-1)**2/4+1))
   # return 1/(1+(abs(x)**2)/10)

def g(x):
   return np.sin(np.sqrt((x-2)**2/4+1))
   # return 1/(1+(abs(x+1)**2)/10)


### Here we plot the difference between separate and combined spline interpolation
x_L, x_R = -6, 6
n_f, n_g = 3, 5
num_samples = 9

### Defining the samples points and the distances/inner_widths
### of the subintervals, i.e. x_1-x_0
ell = num_samples-1
samples = np.linspace(x_L, x_R, num_samples)
inner_width = (x_R-x_L)/(ell)

### Interpolating the not known polynomial with (cubic) splines at the
### sample points
splines = interp1d(samples, f(samples), kind='cubic')

### Calculating the coefficients of interpolation, i.e. matrices B and C
# Note: The function interp1d from scipy does not give the coefficients of the
#     cubic splines therefore we use numpy for a polynomial fit on the cubic
#     splines on all the subintervals from the fit we can extrapolate the
#     coefficients of the polynomials. However, this method is not recommended
```

```
#     as it requires a subdivision of the interval, instead implement a spline
#     interpolation that does return the interpolating coefficients.
b, c = [], []
for idx, i in enumerate(samples[:-1]):
   # with shifted intervals
   interval = np.linspace(i, i+inner_width, 1000)
   mapped_interval = np.linspace(0, inner_width, 1000)
   pol_f = np_pol.Polynomial.fit(mapped_interval, splines(interval), n_f)
   pol_g = np_pol.Polynomial.fit(mapped_interval, g(interval), n_g)

   b.append(list(pol_f.convert().coef))
   c.append(list(pol_g.convert().coef))

### Calculating and storing the powers of x, note that the intervals were shifted
powers_x = [
      [(inner_width)**(jk+1) / (jk+1) for jk in range(n_f+n_g+2)]
      for i in range(len(c))
]

### Calculating and storing all values of the matrix D
# Note: We do not need to store the values of the matrix D if we only have
#     one sensor because we then do not need to shift the matrix B to calculate
#     its frobenius product with matrix D, however for clarity this is
#     implemented nonetheless.
d = [[0 for _ in range(n_f+1)] for _ in range(ell)]
for i in range(ell):
   for j in range(n_f+1):
      d[i][j] = sum(c[i][k] * powers_x[i][j+k] for k in range(n_g+1))

### Calculating the final integral
som = 0
for i in range(ell):
   for j in range(n_f+1):
      som += b[i][j] * d[i][j]

print(f'Our answer for the integral is {som}')
```

This is the computer code for generating the results from Chapter 5.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from scipy.interpolate import interp2d, griddata

def p(x, y):
   """Returns the [real, imaginary] part of the pressure function P(x,y,0,omega)
   """
   ans = np.zeros((2,) + x.shape)
   for x_S, y_S, z_S in sources:
      dr = np.sqrt((x-x_S)**2 + (y-y_S)**2 + z_S**2)
      ans[0] += np.cos(k * dr) / dr
      ans[1] += -np.sin(k * dr) / dr
   return ans

def dG(x, y, x_A, y_A):
   """Returns the [real, imaginary] part of the Green's function (dG/dz)_{z=0}
   """
   dr = np.sqrt((x-x_A)**2 + (y-y_A)**2 + z_A**2)
   real_part1 = z_A /(dr**3) * np.cos(k*dr)
   real_part2 = z_A*k/(dr**2) * np.sin(k*dr)
   imag_part1 = -z_A /(dr**3) * np.sin(k*dr)
   imag_part2 = z_A*k/(dr**2) * np.cos(k*dr)
```

```
    return [real_part1+real_part2, imag_part1+imag_part2]

def prod(x, y, x_A, y_A):
    """Returns the [real, imaginary] part of the product of p and dG
    """
    pressure = p(x, y)
    greens = dG(x, y, x_A, y_A)
    return [pressure[0]*greens[0] - pressure[1]*greens[1],
          pressure[0]*greens[1] + pressure[1]*greens[0]]


def integral(x, y, res, method='trapezeoid'):
    """Calculate the integral using various methods
    """
    if method == 'trapezeoid': # equidistant is assumed
        h_x = inter_x/(x.shape[1]-1) # average
        h_y = inter_y/(x.shape[0]-1) # average
        c_x = np.array([1] + [2]*(x.shape[0]-2) + [1]) * h_x/2
        c_y = np.array([1] + [2]*(x.shape[1]-2) + [1]) * h_y/2
        C = c_x.reshape((len(x), 1)) * c_y
        real = np.multiply(C, res[0]).sum()
        imag = np.multiply(C, res[1]).sum()
        return [real, imag]

    elif method == 'altered_trapezeoid': # rectilinear is assumed
        C = np.zeros(x.shape)
        for i_x in range(x.shape[1]-1):
            for i_y in range(x.shape[0]-1):
                h_x = (x[i_y][i_x+1]-x[i_y][i_x])
                h_y = (y[i_y+1][i_x]-y[i_y][i_x])
                h = h_x * h_y / 4
                C[i_y:i_y+2, i_x:i_x+2] += h

        real = np.multiply(C, res[0]).sum()
        imag = np.multiply(C, res[1]).sum()
        return [real, imag]

    elif method == 'simpson': # equidistant is assumed
        lst = [1, 4, 1]
        frac = 1/3

        ### check if dimensions add up, this would not be necessary
        #  if using appropriate boundary interpolations
        if ((x.shape[0]-len(lst)) % (len(lst)-1) != 0
            or (x.shape[1]-len(lst)) % (len(lst)-1) != 0):
            print('change #ticks to', (x.shape[0]-len(lst)) % (len(lst)-1)
                                    (x.shape[1]-len(lst)) % (len(lst)-1))
            raise BaseException

        ### do the calculation
        h_x = inter_x/(samples_x-1) # average
        h_y = inter_y/(samples_y-1) # average

        c_x = np.array(
            [lst[0]] +
            (lst[1:-1] + [lst[-1]+lst[0]]) * ((x.shape[0]-len(lst))//(len(lst)-1)) +
            lst[1:]
        ) * (h_x*frac)

        c_y = np.array(
            [lst[0]] +
```

```
        (lst[1:-1] + [lst[-1]+lst[0]]) * ((x.shape[1]-len(lst))//(len(lst)-1)) +
        lst[1:]
    ) * (h_y*frac)

    C = c_x.reshape((x.shape[0], 1)) * c_y
    real = np.multiply(C, res[0]).sum()
    imag = np.multiply(C, res[1]).sum()
    return [real, imag]

elif method == 'altered_simpson': # semi-equidistant is assumed
    ### do the calculation
    tot_real = 0
    tot_imag = 0
    # looping through all possible starting points
    for i_x in range(0, len(x[0])-2, 2):
        for i_y in range(0, len(x)-2, 2):
            for is_complex in range(0, 2): # real, complex
                f = {(i%3, i//3):
                      res[is_complex][i_y+i%3, i_x+i//3] for i in range(9)}

                h_x = sum((x[i][i_x+2] - x[i][i_x])/2
                          for i in range(i_y, i_y+3)) / 3 # average
                h_y = sum((y[i_y+2][i] - y[i_y][i])/2
                          for i in range(i_x, i_x+3)) / 3 # average

                d_x0 = x[i_y+0][i_x+1] - (x[i_y+0][i_x+2]+x[i_y+0][i_x])/2
                d_x1 = x[i_y+1][i_x+1] - (x[i_y+1][i_x+2]+x[i_y+1][i_x])/2
                d_x2 = x[i_y+2][i_x+1] - (x[i_y+2][i_x+2]+x[i_y+2][i_x])/2

                d_y0 = y[i_y+1][i_x+0] - (y[i_y+2][i_x+0]+y[i_y][i_x+0])/2
                d_y1 = y[i_y+1][i_x+1] - (y[i_y+2][i_x+1]+y[i_y][i_x+1])/2
                d_y2 = y[i_y+1][i_x+2] - (y[i_y+2][i_x+2]+y[i_y][i_x+2])/2
                d_y = (d_y0+d_y1+d_y2)/3 # no rotation of axis for better result


                a_0 = h_x * (
                      -3*d_x0**2*(f[(0,0)]+f[(2,0)]) +
                       2*d_x0 *(f[(0,0)]-f[(2,0)])*h_x +
                         h_x**2 *(f[(0,0)]+4*f[(1,0)]+f[(2,0)])
                ) / (3*(h_x**2-d_x0**2))

                a_1 = h_x * (
                      -3*d_x1**2*(f[(0,1)]+f[(2,1)]) +
                       2*d_x1 *(f[(0,1)]-f[(2,1)])*h_x +
                         h_x**2 *(f[(0,1)]+4*f[(1,1)]+f[(2,1)])
                ) / (3*(h_x**2-d_x1**2))

                a_2 = h_x * (
                      -3*d_x2**2*(f[(0,2)]+f[(2,2)]) +
                       2*d_x2 *(f[(0,2)]-f[(2,2)])*h_x +
                         h_x**2 *(f[(0,2)]+4*f[(1,2)]+f[(2,2)])
                ) / (3*(h_x**2-d_x2**2))


                if is_complex == 0:
                    tot_real += h_y * (
                          -3*d_y**2*(a_0+a_2) +
                           2*d_y *(a_0-a_2)*h_y +
                             h_y**2*(a_0+4*a_1+a_2)
                    ) / (3*(h_y**2-d_y**2))
                elif is_complex == 1:
```

```
                tot_imag += h_y * (
                      -3*d_y**2*(a_0+a_2) +
                       2*d_y *(a_0-a_2)*h_y +
                         h_y**2*(a_0+4*a_1+a_2)
                ) / (3*(h_y**2-d_y**2))

    return [tot_real, tot_imag]

elif method == 'separate': # equidistant is assumed
    '''
    Instead of dealing with the summation over the coefficients, in this
    implementation the pressure function is interpolated by a cubic spline.
    Since the derivative of the Green's function is known analytically,
    we do not interpolate it in this implementation, however, this does
    need to be done for faster evaluation. Afterwards we calculate the integral
    over the subinterval by means of a regular sum (thus diving it into
    subsubintervals). Once again, we do not use a summation over coefficients,
    since this was easier to implement and produces approximately the same
    results.
    '''
    ### First we interpolate the pressure graph
    real_pressure = interp2d(meas_x, meas_y,
                    p(x, y)[0], kind='cubic')(plot_x, plot_y)
    imag_pressure = interp2d(meas_x, meas_y,
                    p(x, y)[1], kind='cubic')(plot_x, plot_y)
    ### Then for the greens function we use a finer grid
    real_greens = interp2d(plot_x, plot_y,
                dG(plot_X, plot_Y, x_A, y_A)[0], kind='cubic')(plot_x, plot_y)
    imag_greens = interp2d(plot_x, plot_y,
                dG(plot_X, plot_Y, x_A, y_A)[1], kind='cubic')(plot_x, plot_y)

    real = real_pressure*real_greens - imag_pressure*imag_greens
    imag = real_pressure*imag_greens + imag_pressure*real_greens
    res = [real, imag]

    return integral(plot_X, plot_Y, res, method='trapezeoid')

elif method == 'altered_separate': # non-equidistant is assumed
    '''
    See "separate" method
    '''
    ### First we interpolate the pressure graph
    points = np.array([x.flatten(), y.flatten()]).transpose()

    # We get an error if values to interpolate to not lie inside the convex
    # hull of points, thus the outer points are assumed to be equidistant
    # else, a nearest value approach is recommended.
    xmeas = x.copy()
    xmeas[1:-1,1:-1] = meas_X[1:-1,1:-1].copy()
    ymeas = y.copy()
    ymeas[1:-1,1:-1] = meas_Y[1:-1,1:-1].copy()

    interpol = np.array([xmeas.flatten(), ymeas.flatten()]).transpose()
    a = griddata(points, p(x, y)[0].flatten(), interpol, method='cubic')
    b = griddata(points, p(x, y)[1].flatten(), interpol, method='cubic')
    a = a.reshape((samples_y, samples_x))
    b = b.reshape((samples_y, samples_x))

    real_pressure = interp2d(meas_x, meas_y, a, kind='cubic')(plot_x, plot_y)
    imag_pressure = interp2d(meas_x, meas_y, b, kind='cubic')(plot_x, plot_y)
    ### Then for the greens function we use a finer grid
```

```
      real_greens = interp2d(plot_x, plot_y,
            dG(plot_X, plot_Y, x_A, y_A)[0], kind='cubic')(plot_x, plot_y)
      imag_greens = interp2d(plot_x, plot_y,
            dG(plot_X, plot_Y, x_A, y_A)[1], kind='cubic')(plot_x, plot_y)

      real = real_pressure*real_greens - imag_pressure*imag_greens
      imag = real_pressure*imag_greens + imag_pressure*real_greens
      res = [real, imag]

      return integral(plot_X, plot_Y, res, method='trapezeoid')


### defining variables
inter_x, inter_y = 50, 70
samples_x, samples_y = 29, 29
plot_samples_x, plot_samples_y = 4001, 4001
sources = [(10, 0, 2), (0, 15, 1.5), (1, -5, 1.7), (-13, 13, 2.3)]
k = 0.5 # omega/c, constant here
x_A, y_A, z_A = 0.9, -1.4, 10


### making measurement ticks
meas_x = np.linspace(-inter_x/2, inter_x/2, samples_x)
meas_y = np.linspace(-inter_y/2, inter_y/2, samples_y)
meas_X, meas_Y = np.meshgrid(meas_x, meas_y)
# meas_res = prod(meas_X, meas_Y, x_A, y_A)

## with noise
noise = 0.0005 # relative noise
np.random.seed(1)
meas_X2, meas_Y2 = meas_X.copy(), meas_Y.copy()
x_noise = np.random.normal(size=(samples_y, samples_x))*(inter_x*noise/samples_x)
y_noise = np.random.normal(size=(samples_y, samples_x))*(inter_y*noise/samples_y)
meas_X2 += x_noise
meas_Y2 += y_noise
meas_res = prod(meas_X2, meas_Y2, x_A, y_A)

### printing values
h_x = meas_x[1]-meas_x[0]
h_y = meas_y[1]-meas_y[0]
amount_noise_x = np.abs(x_noise).sum()/(samples_x*samples_y)
amount_noise_y = np.abs(y_noise).sum()/(samples_x*samples_y)
print(f'relative x noise of {amount_noise_x/h_x*100:.2g}%')
print(f'relative y noise of {amount_noise_y/h_y*100:.2g}%')

### making plot ticks
plot_x = np.linspace(-inter_x/2, inter_x/2, plot_samples_x)
plot_y = np.linspace(-inter_y/2, inter_y/2, plot_samples_y)
plot_X, plot_Y = np.meshgrid(plot_x, plot_y)
plot_res = prod(plot_X, plot_Y, x_A, y_A)

# calculate the answer for the integral on a finer grid
ans = integral(plot_X, plot_Y, plot_res)

# approximate the integral using the basic method
compare = 'trapezeoid'
val = integral(meas_X2, meas_Y2, meas_res, method=compare)
compare_rel_mse = np.sqrt(
   1/2*(abs(val[0]-ans[0]))**2 + abs(val[1]-ans[1])**2) / (ans[0]**2 + ans[1]**2)
)
print(f'{compare.ljust(20)} method, with a MSE of: {compare_rel_mse:#.4e}')
```

```
268
269 # approximate the integral using various methods
270 for method in ['altered_trapezeoid', 'simpson',
271             'altered_simpson', 'separate', 'altered_separate']:
272    val = integral(meas_X2, meas_Y2, meas_res, method=method)
273    rel_mse = np.sqrt(
274       1/2*(abs(val[0]-ans[0])**2 + abs(val[1]-ans[1])**2) / (ans[0]**2+ans[1]**2)
275    )
276    print(f'{method.ljust(20)} method, with a MSE of: {rel_mse:#.4e} '
277          f'this is a factor {compare_rel_mse/rel_mse:#.3g} better')
```

Bibliography

[1] D.J. Griffiths. *Introduction to Electrodynamics*. Cambridge University Press, 2017. ISBN: 9781108420419. URL: https://books.google.nl/books?id=ndAoDwAAQBAJ.

[2] M.T. Goodrich, R. Tamassia, and M.H. Goldwasser. *Data Structures and Algorithms in Java, 6th Edition*. Wiley Global Education, 2014. ISBN: 9781118803165. URL: https://books.google.nl/books?id=HTxDAwAAQBAJ.

[3] O. Čertik. *Theoretical Physics Reference 0.5 documentation*. https://www.theoretical-physics.net/dev/index.html. [Online; accessed 28-May-2022]. 2009-2011.

[4] A.V. Oppenheim et al. *Signals & Systems*. Prentice-Hall signal processing series. Prentice Hall, 1997. ISBN: 9780138147570. URL: https://books.google.nl/books?id=LwQqAQAAMAAJ.

[5] D. Gisolf and E. Verschuur. *The principles of quantitative acoustical imaging*. EAGE Publ., 2010. ISBN: 9789073781931. URL: https://books.google.nl/books?id=MAiTkQEACAAJ.

[6] H. Kutscha. "The double focal transformation and its application to data reconstruction". In: (2014). URL: https://doi.org/10.4233/uuid:14259a8a-8e72-462b-b284-c28e23b3a095.

[7] S.C. Frautschi et al. *The Mechanical Universe: Mechanics and Heat, Advanced Edition*. Cambridge University Press, 2007. ISBN: 9780521715904. URL: https://books.google.nl/books?id=ZTnxQGJ1fHMC.

[8] W.D. McComb. *Dynamics and Relativity*. Oxford University Press, 1999. ISBN: 9780198501121. URL: https://books.google.nl/books?id=REeqQgAACAAJ.

[9] G. Barton and G. Barton. *Elements of Green's Functions and Propagation: Potentials, Diffusion, and Waves*. Oxford science publications. Clarendon Press, 1989. ISBN: 9780198519980. URL: https://books.google.nl/books?id=-iPwVGfDtecC.

[10] J.W. Brown and R.V. Churchill. *Complex Variables and Applications*. Brown-Churchill series. McGraw-Hill Higher Education, 2004. ISBN: 9780072878349. URL: https://books.google.nl/books?id=4vfuAAAAMAAJ.

[11] D.H. Griffel. *Applied Functional Analysis*. Dover Books on Mathematics. Dover Publications, 2002. ISBN: 9780486422589. URL: https://books.google.nl/books?id=iptmm7OJqwAC.

[12] N. Zettili. *Quantum Mechanics: Concepts and Applications*. Wiley, 2009. ISBN: 9780470026786. URL: https://books.google.nl/books?id=6jXlpJCSz98C.

[13] ROBIN RAJ (https://physics.stackexchange.com/users/235379/robin-raj). *Method of pole shifting (Feynman's trick) in Scattering theory vs contour deformation trick*. Physics Stack Exchange. URL: https://physics.stackexchange.com/q/511172.

[14] *Green's Functions Tutorial: Introduction to Green's Functions*. URL: https://web.archive.org/web/20110905015156/http://www.boulder.nist.gov/div853/greenfn/tutorial.html.

[15] DanielSank (https://physics.stackexchange.com/users/31790/danielsank). *Complex integration by shifting the contour*. Physics Stack Exchange. URL: https://physics.stackexchange.com/q/138221.

[16] Mikkel Rev (https://math.stackexchange.com/users/128323/mikkel-rev). *Contour integration and Green's function*. Mathematics Stack Exchange. URL: https://math.stackexchange.com/q/3908399.

[17] mike stone (https://physics.stackexchange.com/users/80687/mike-stone). *Greens function for Helmholtz equation*. Physics Stack Exchange. URL: https://physics.stackexchange.com/q/316893.

[18] Wikipedia contributors. *Propagator — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Propagator&oldid=1082410447. [Online; accessed 28-May-2022]. 2022.

[19] P.K. Kythe and M.R. Schäferkotter. *Handbook of Computational Methods for Integration*. CRC Press, 2004. ISBN: 9780203490303. URL: https://books.google.nl/books?id=Gwygh5csmV4C.

[20] C. Vuik et al. *Numerical Methods for Ordinary Differential Equations*. DAP, Delft Academic Press, 2015. ISBN: 9789065623737. URL: https://books.google.nl/books?id=-g2TrgEACAAJ.

[21] Wikipedia contributors. *Polynomial interpolation — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Polynomial_interpolation&oldid=1088961459. [Online; accessed 28-May-2022]. 2022.

[22] Wikipedia contributors. *Newton polynomial — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Newton_polynomial&oldid=1090154211. [Online; accessed 28-May-2022]. 2022.

[23] Carl Bender, Dorje Brody, and Bernhard Meister. "Inverse of a Vandermonde matrix". In: (Jan. 2002). URL: https://www.researchgate.net/publication/240061354_Inverse_of_a_Vandermonde_matrix.

[24] W.M. Johnson. "On Cotesian numbers: their history, computation and values to $n = 20$". In: (2002). URL: http://oeis.org/A002176/a002176.pdf.

[25] jorisperrenet (https://math.stackexchange.com/users/1049661/jorisperrenet). *Integration of a function approximated by a nth order polynomial*. Mathematics Stack Exchange. URL: https://math.stackexchange.com/q/4431435.

[26] Wikipedia contributors. *Bilinear interpolation — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Bilinear_interpolation&oldid=1081914622. [Online; accessed 28-May-2022]. 2022.

[27] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[28] 3Blue1Brown. *But what is a partial differential equation? | DE2*. YouTube. 2019. URL: https://youtube.com/watch?v=ly4S0oi3Yz8.

[29] E.W. Weisstein. *Stirling Number of the First Kind*. MathWorld – A Wolfram Web Resource. URL: https://mathworld.wolfram.com/StirlingNumberoftheFirstKind.html.

[30] Pr∞fWiki. [Online; accessed 10-Jun-2022]. URL: https://proofwiki.org/wiki/Inverse_of_Vandermonde_Matrix.

This page intentionally left blank.

(in order to achieve a total page count of 73; a prime number)