Department of Applied Mathematics,
Computer Science and Statistics

Faculty of Sciences

Algorithms for time-independent Schrödinger equations

Toon Baeyens

Supervisors
prof. dr. Marnix Van Daele
prof. dr. Joris Van der Jeugt

Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Science: Mathematics

June 2023

GHENT
UNIVERSITY

Table of contents

Table of contents 3

Preface 7

Acknowledgement 8

Summary 11

Summary in English 11

Nederlandse samenvatting 12

Overview of developed software 15

1 Introduction to differential equations 17

1.1 History 18

1.1.1 Calculus 19

1.1.2 Advent of computing 20

1.2 Ordinary differential equations 21

1.2.1 Sturm–Liouville problems 23

1.3 Partial differential equations 24

1.3.1 Time-independent Schrödinger problems 26

1.4 Notation 27

2 One-dimensional time-independent Schrödinger equations 29

2.1 Introduction 29

2.1.1 Properties of the Sturm–Liouville equation 31

2.1.2 Liouville's transformation 34

2.2 Background about Matslise 35

2.2.1 Finite difference scheme for the Sturm–Liouville equation 36

2.2.2 CP-methods 40
2.2.3 Shooting with Prüfer's transformation 56
2.3 Matslise 3.0 62
2.4 Periodic 1D Schrödinger equation 71
2.5 Numerical experiments 77
2.5.1 A first example 77
2.5.2 A potential with jumps 80
2.5.3 Mathieu's equation 82
2.5.4 Coffey–Evans problem 86
2.5.5 Truncated hydrogen potential 90
2.5.6 Periodic problem with an asymmetric potential 91
2.5.7 Sturm–Liouville problems 94
2.6 Matslise 3.0 – A new implementation 95
2.6.1 Implementation challenges 97
2.7 Future work 108

3 A shooting method for 2D time-independent Schrödinger equations 111
3.0.1 A first example 113
3.1 Ixaru's method 116
3.2 Our improvements 121
3.2.1 Automatic optimal sector size 121
3.2.2 Orthonormalization of eigenfunctions 123
3.2.3 Calculation of $\mathbf{V}^{(k)}(y)$ 125
3.2.4 Locating eigenvalues 130
3.3 Determining the index of eigenvalues 137
3.3.1 The main theorem 140
3.3.2 Examples 145
3.4 Numerical experiments 156
3.4.1 The zero potential 158
3.4.2 The harmonic oscillator potential 160
3.4.3 Ixaru's potential 162
3.4.4 The Hénon–Heiles potential 165
3.5 Conclusions 167

4 Strands: a new method for 2D time-independent Schrödinger equations 169
4.1 Inspiration 170
4.1.1 A finite difference scheme 171
4.2 Strands: line-based collocation method 179
4.2.1 Solving the matrix approximation 185

4.2.2 Computing eigenfunctions 196
4.2.3 Numerical experiments 201
4.2.4 Runtime analysis 216
4.2.5 The name Strands 221
4.3 Conclusion 222

Closing remarks **225**
Looking forward 226

Bibliography **229**

Preface

A thesis is a work of long breath[1]. During the past five years, I have worked on this project. In this book you can find my process in learning about and contributing to the numerical study of Schrödinger equations.

This thesis is written from quite a technical perspective and is definitely not suited as an introductory text to the subject. Following along will require quite a lot of background knowledge. The text assumes you are already familiar with (partial) differential equations, numerical methods for solving these and the intricacies when implementing them. If you want to know *all* advances written in this work, then reading the book cover to cover may be the best way. I tried to logically structure the work with sufficient cross-references and citations.

If however you are still interested in reading this book without a deep technical knowledge, then I recommend skipping the more technical discussions. Starting with the summary will provide you with a basic overview about what you can expect from this work. Chapter 1 gives some historical context about mathematical evolution and in particular about the conception of differential equations. Chapters 2, 3 and 4 contain the innovations from this thesis. Each of these chapters starts with some historical background and mathematical motivation, and ends with some numerical experiments to evaluate the performance (both in accuracy as efficiency) of our methods. For simply a cursory examination of my research, these first and last sections may be sufficient.

If you are still interested in this thesis, however not so much in the mathematics, even then I can provide some guidance on how to read it. I assume you are more interested in doing research in general or even in me personally. In this case

[1]This is a literal translation of the Dutch expression: 'Een werk van lange adem'.

you may[2] just skip chapters 2, 3 and 4 entirely. The summary and chapter 1 may provide sufficient context. For some personal notes, I recommend taking a look at the closing remarks at the end of this work as well.

In the introductory paragraph I stated that this research came into fruition within the past five years. In the most literal sense, this is of course true. Practically however, doing research is quite an individual pursuit, and as such it is quite a personal one. Each researcher is different and experiences the world around them uniquely. This difference, in part, relies on the background of the (mathematical) education of the researcher. In my case, I think I can say that my 'mathematical career' started a good fifteen years ago. My interest in and passion for mathematics became abundantly clear[3] throughout high school. My mathematical adventure really took off when I started at Ghent University. Now, ten years later, I can close this chapter with the completion of a PhD in mathematics.

Acknowledgement

Even though doing research is quite a lonely process, I was never alone. There are many people who let me find myself and helped me grow. Here I want to seize the opportunity to thank them wholeheartedly.

First and foremost I want to thank my very best friend. Emilie has always[4] been here for me. Without her, even survival would be difficult. She encourages me to pursue all my dreams and ambitions, she possesses the curiosity to listen to all my ramblings, and she understands me. Maybe not so much when I am saying nonsense, but even then she understands *me*. She has the same wandering mind as mine, only in a uniquely different way. She broadens my horizons and is not afraid to tell me when I am wrong[5]. *Lieve Floe, dank je wel; een eenvoudige, toch zeer diepe en welgemeende "Dank je.".*

Concerning the content of this work, Marnix has been invaluable. As a promotor, he was a true guide. He gave me all the tools necessary during this research.

[2]Of course, as the owner of a book you can read it however you want (or even burn it). So, not that you need it, but you definitely have my permission to skip (large) parts, if they are not of interest to you.

[3]To the detriment of the non-science courses.

[4]At least, as long as I can remember.

[5]I have absolutely no problem admitting when I am wrong. I believe my challenge is to *see* when I am wrong.

He granted me the freedom to explore *all*[6] my ideas and side projects. At times, he even poured gasoline on the fire by encouraging me to create some unimaginably beautiful images. Nonetheless, he always guided me back. Marnix kept motivating me to write down my ideas into articles, and he was rightly critical about my first drafts. As an example, my first draft of chapter 4 (now 54 pages) was a mere two pages of only mathematical notation, without context. As my copromotor, I also thank Joris. When communication stalled, with only a few words, he got everything back on track.

As stated earlier, while researching and learning new things I depend on my background. Thanks to my parents and family, this is such a rich background. *Mama, Papa, dank je wel. Dank je, om me alle ruimte te geven om zelf op zoek te gaan, om in me te geloven en me te steunen. Dank je wel dat ik in Sinaai steeds een plek zal hebben om thuis te komen. Natuurlijk dank ik ook Wannes en Kaat, om mij te leren dat ik nooit alleen ben. Dank je, aan de hele uitgebreide familie voor zoveel adressen waar ik steeds een warme plek kan vinden.*

With much pleasure I want to thank my youngest friend. In all honesty, Ward should be a co-author of this work. We have spent many hours writing together. Most of the time, I was wielding the keyboard. However, he also undoubtedly contributed. If you find some stray letters throughout this text, chances are these were written by a very curious and extremely enthusiastic one-and-a-half year old.

I am fortunate to be able to say: throughout the last years many friends became colleagues, and many colleagues became friends.

There are new friends, who first were colleagues. In particular, I thank Annick, Asmus, Bart, Dieter, Heidi, Jonathan, Louise, Nico, Niko, Oliver, Pieter, Rien, Robbert, Roy, Tibo, and Tom for giving me the joy of going to work, day in and day out. Coming into the office has always been a pleasure by being greeted by the "Good morning!"'s of Camilla, Felix, Jorg, Niels, Pieter, and Wout. There are even a few colleagues, who I now dare to call dear friends. I wholeheartedly thank Alexis for taking the time (during a very busy period) to thoroughly read my whole thesis and providing invaluable comments like 'Pew pew!' every time the 'shooting' technique was mentioned. I also thank Steven for being a companion in discovering new mathematics, solving many problems, and exploring the deep caverns of the `C++` specification. And of

[6]This is not an exaggeration. I am extremely grateful that he never told me "no", he never instructed me to manage my time differently.

course, I thank Charlotte. It still astonishes me how opposite we are, and yet so similar. Thank you for fighting all our battles with me, always side-by-side, never head-to-head.

And there are old friends. People who have grown with me for already a decade and more. In particular, I thank my fellow students Wouter, Frederik, Sam, and David. And I thank Bart and Jorn for their long-lasting friendships, our dates are as valuable as they are scarce. All of you have helped me become the mathematician I am today, however some in particular contributed a great deal to the person I am today. I thank Hadewijch for all the pancakes, all the (late-night) talks, and all the adventurous double-dates which still await us. I thank Simon, for motivating me with chocolate waffles, for letting me sometimes win with Beatsaber, and for the many level-headed much-needed conversations. And, I thank Jens for all his creativity, for first saying yes and then doubting if it may have been too ambitious, and for his courage to do what he loves.

Last but definitely not least I want to thank my jury. Liviu, Guido, Veerle, Kris and Dajana, thank you for taking the time to read my work with such interest. In particular, I thank Liviu for providing many of the building blocks needed in this work, for his hospitality in inviting us to Sinaia, and for his many comments and suggestions.

Closing the opening words

To end this section there is still one person left to thank, and that is you: the reader. Writing this book has been quite the adventure, and I hope reading it will give you a sincere view on the whole affair.

If you have any questions, remarks or comments (about this work, or even more generally about me), I am very eager to hear them. Please do reach out!

Finally, I wish you the very best of luck in reading this thesis.

Toon Baeyens

June 2023

Summary

Summary in English

An n-dimensional time-independent Schrödinger equation is a linear second order partial differential equation given by

$$-\nabla^2\psi(\mathbf{x}) + V(\mathbf{x})\psi(\mathbf{x}) = E\mathbf{x}. \tag{1}$$

In this expression, V is a given potential function defined on a domain $\Omega \subseteq \mathbb{R}^n$. When solving this equation, the goal is to find all functions $\psi : \Omega \to \mathbb{R}$ and values $E \in \mathbb{R}$ that satisfy the Schrödinger equation (1) in combination with appropriate boundary conditions. For such a solution, E is called the eigenvalue with corresponding eigenfunction $\psi(\mathbf{x})$.

For the one-dimensional case, Schrödinger equations are a special case of Sturm–Liouville equations:

$$-(p(x), y'(x))' + q(x)y(x) = \lambda w(x)y(x). \tag{2}$$

In this equation $p(x)$, $q(x)$ and $w(x)$ are given real functions on a possibly unbounded interval. Here, λ is the unknown eigenvalue with corresponding eigenfunction $y(x)$.

In chapter 2, the well-established constant perturbation method for approximating eigenvalues and eigenfunctions of (2) is studied. Here, regular Sturm–Liouville problems on $[a, b]$ are considered with homogeneous Robin boundary conditions:

$$\alpha_a y(a) + \beta_a p(a)y'(a) = 0 \text{ and } \alpha_b y(b) + \beta_b p(b)y'(b) = 0.$$

This constant perturbation method is able to reach extremely accurate results, for small as well as for high eigenvalues. With this method, we build a new

`C++`-implementation which we call `Matslise 3.0` accompanied by a `python`-package named `Pyslise`. The new implementation is up to 100 times faster than `Matslise 2.0` when calculating eigenvalues. For the evaluation of eigenfunctions, our program is even up to a 1000 times faster. These speed-ups are possible due to the development and calculation of novel, more complicated propagation formulae.

We also consider Sturm–Liouville problems with (generalized) periodic boundary conditions. This is the first time, to the best of our knowledge, that a constant perturbation method is used for this type of problem. The main challenge here is to make sure *all* requested eigenvalues are found.

In chapters 3 and 4, two methods for the approximation of eigenvalues and eigenfunctions of the time-independent two-dimensional Schrödinger equation (1) are studied. The first method (discussed in chapter 3) is an existing method developed by Ixaru based on constant perturbation concepts. It is limited to Schrödinger problems on rectangular domains with homogeneous Dirichlet boundary conditions. Throughout that chapter, we introduce some improvements to this method. Most notably, we develop a technique to reliably determine the index of an eigenvalue. This enables us to ensure all requested eigenvalues have been found.

A second method for approximating eigenvalues and eigenfunctions of time-independent two-dimensional Schrödinger equations with homogeneous Dirichlet boundary conditions on (possibly non-rectangular) domains can be found in chapter 4. Here, we are inspired by a simple method based on a finite difference scheme. This leads us to develop `Strands`, a `C++`-program accompanied by a homonymous `python`-package. Through the use of line-based collocation ideas with well-chosen basis functions (inspired by chapter 3), this new method is able to reach much more accurate results, with a similar computational cost.

Nederlandse samenvatting

Een n-dimensionale tijdsonafhankelijke Schrödinger-vergelijking is een lineaire tweede orde partiële differentiaalvergelijking:

$$-\nabla^2\psi(\mathbf{x}) + V(\mathbf{x})\psi(\mathbf{x}) = E\mathbf{x}. \tag{1}$$

In deze uitdrukking is V een gegeven potentiaalfunctie die gedefinieerd is op een domein $\Omega \subseteq \mathbb{R}^n$. Bij het oplossen van deze vergelijking zoekt men alle waarden $E \in \mathbb{R}$ waarvoor functies $\psi : \Omega \to \mathbb{R}$ bestaan die voldoen aan (1)

samen met gegeven randvoorwaarden. In zo een oplossing noemt men E de eigenwaarde met bijhorende eigenfunctie $\psi(\mathbf{x})$.

Een eendimensionale tijdsonafhankelijke Schrödinger-vergelijking is een speciaal geval van een Sturm–Liouville-vergelijking:

$$-(p(x), y'(x))' + q(x)y(x) = \lambda w(x)y(x). \tag{2}$$

Hierbij zijn $p(x)$, $q(x)$ en $w(x)$ gegeven functies op een mogelijk onbegrensd interval. De onbekenden zijn de eigenwaarde λ met bijhorende eigenfunctie $y(x)$.

In hoofdstuk 2 bestuderen we de constante perturbatie methode voor het benaderen van de eigenwaarden en eigenfuncties van vergelijking (2). Meer specifiek behandelen we reguliere Sturm–Liouville-problemen met Robin-randvoorwaarden:

$$\alpha_a y(a) + \beta_a p(a)y'(a) = 0 \text{ and } \alpha_b y(b) + \beta_b p(b)y'(b) = 0.$$

Deze constante perturbatie methode bereikt extreem nauwkeurige resultaten voor zowel kleine als grote eigenwaarden. We gebruiken deze methode om een nieuw `C++`-programma, dat we `Matslise 3.0` noemen, te ontwikkelen. Dit programma wordt vergezeld van de `python`-bibliotheek `Pyslise`. Deze nieuwe implementatie is tot wel 100 keer sneller dan `Matslise 2.0` voor de berekening van eigenwaarden. Wanneer eigenfuncties geëvalueerd worden, is ons programma zelfs tot 1000 keer sneller. Onder meer dankzij de ontwikkeling van nieuwe, meer ingewikkelde formules waren we in staat deze versnellingen te realiseren.

We behandelen ook Sturm–Liouville-problemen met (veralgemeende) periodieke randvoorwaarden. Zover wij weten is dit de eerste keer dat de constante perturbatie methode wordt ingezet voor dit soort problemen. Hierbij bleek het de grootste uitdaging om te kunnen garanderen dat *alle* gevraagde eigenwaarden gevonden worden.

In hoofdstukken 3 en 4 worden twee methoden bestudeerd die de eigenwaarden en eigenfuncties van een tweedimensionale tijdsonafhankelijke Schrödinger-vergelijkingen kunnen benaderen. De eerste methode (onderzocht in hoofdstuk 3) is ontwikkeld door Ixaru met ideeën gebaseerd op de constante perturbatie methoden. Deze methode is beperkt tot Schrödinger-vergelijking op rechthoekige domeinen met homogene Dirichlet-randvoorwaarden. Doorheen hoofdstuk 3 introduceren we enkele uitbreidingen en verbeteringen op die methode. We ontwikkelen onder andere een techniek waarmee we het volgnummer van een eigenwaarde betrouwbaar kunnen bepalen. Met deze techniek kunnen we garanderen dat alle gevraagde eigenwaarden gevonden zijn.

Een tweede methode om de eigenwaarden en eigenfuncties van tweedimensionale tijdsonafhankelijke Schrödinger-vergelijking op (mogelijks niet-rechthoekige) domeinen met homogene Dirichlet-randvoorwaarden te benaderen, wordt ontwikkeld in hoofdstuk 4. Hierbij laten we ons inspireren door een eenvoudige techniek gebaseerd op de eindige-differentiemethode en de lessen die we leerden in hoofdstuk 3. Dit stelde ons in staat om `Strands` te ontwikkelen. Dit is een `C++`-programma, begeleid door een gelijknamige `python`-bibliotheek. Dankzij collocatie ideeën toegepast op roosterlijnen met goed gekozen basis functies (zoals bij de methode van Ixaru), bereikt ons nieuw programma een veel hogere nauwkeurigheid met een gelijkaardige uitvoeringstijd.

Developed software

All developed software consists of `C++`-source files together with a `python`-package. This `python`-package provides a user-friendly interface to the efficiently implemented `C++`-code. To build upon this code, `python` is an optional dependency. All Schrödinger problems can also be expressed using only `C++`.

Matslise 3.0 — pyslise

Solving one-dimensional Sturm–Liouville and Schrödinger problems with homogenous Robin or periodic boundary conditions. See chapter 2.

`https://github.com/twist-numerical/matslise`
`https://pypi.org/project/pyslise/`

Matslise 2D — pyslise2d

Approximating eigenvalues and eigenfunctions of two-dimensional Schrödinger problems on rectangular domains with homogenous Dirichlet boundary conditions, using the method from chapter 3.

`https://github.com/twist-numerical/matslise2d`
`https://pypi.org/project/pyslise2d/`

Strands — strands

Approximating eigenvalues and eigenfunctions of two-dimensional Schrödinger problems on general bounded domains with homogenous Dirichlet boundary conditions, using the method from chapter 4.

`https://github.com/twist-numerical/strands`
`https://pypi.org/project/Strands/`

Chapter 1

Introduction to differential equations

Differential equations recall in most mathematicians many feelings. Some only shiver by the mere idea of them, some of my colleagues in the more algebraic, (finite) geometric or discrete fields come to mind. Others, myself included, rejoice at the thought of studying them.

The knowledge about and experience with differential equations vary wildly among even the best of mathematicians. Some only had one, maybe two, introductory courses, others have studied them their whole careers. Important note: there are very few mathematicians that can say they have studied 'differential equations'. Neither can I: I have studied *a* differential equation; maybe two, if you count generously.

Before diving into the required mathematical background, let us take a step back. Mathematics does not live in a vacuum. Modern ideas have grown out of a very rich history, with many influences from the scientific questions of the time. In this introduction we will take the time to appreciate this history, and discover how differential equations have been developed.

1.1 History

In this section, we will walk through an abridged version of the history of differential equations. Before talking about functions, it is important to talk about what makes up these functions.

A good starting point is to take a look at numbers. Counting things is universal and timeless, as such the natural numbers $\mathbb{N}$ are a jumping point to all the other numbers. In modern mathematics, the next logical step is to introduce the integers $\mathbb{Z}$. Historically however, only throughout the Middle Ages became negative numbers and the concept of zero widely accepted as numbers themselves. The first texts about negative numbers are from Chinese mathematicians somewhere between the second and seventh century AD. Later on, Indian and Islamic mathematicians calculated with negative numbers as well.

In historic texts, we find already references to (positive) fractions, as early as Ancient Egypt, around 1000 BC (maybe even earlier). So after $\mathbb{N}$ the next numbers that were 'discovered', were the positive rational numbers $\mathbb{Q}^+$. Soon thereafter, some irrational numbers were found. The most prominent example is probably $\sqrt{2}$, the length of the diagonal of the unit square. But it is a bit too generous to say that this was the discovery of the real numbers. It is more accurate to say that there was a (geometric) notion of algebraic numbers $\overline{\mathbb{Q}}$. These are the numbers that are solutions of polynomial equations, like $x^2 = 2$. However, only around the year 900, the Egyptian mathematician Abū Kāmil Shujā ibn Aslam started to accept these solutions as numbers in and of itself.

The mathematical invention that may have had the most impact in our daily lives may also be unnoticed by many: the Hindu–Arabic numeral system. This is a way of writing down numbers, invented between the first and fourth century. Most numeral systems were not made for large numbers or were cumbersome to work with. Only the best of mathematicians could do computation in those systems; especially multiplication was difficult. The Hindu–Arabic numeral system solved this by being positional based. This means that the last digits is the one's place, the digit before that ten's, then hundred's, and so on. This allows for a compact way to write down large numbers that still allows easy computation through arithmetic manipulations. Our modern numeral system is a direct descendant of the Hindu–Arabic numeral system. Compare for example the Roman numeral MMMDCCCXLVI to our modern equivalent 3846. And to help illustrate this point, try squaring both numbers.

From the sixteenth century onwards, mathematics in Europe started to flourish.

For our purposes, the next stride towards the real numbers was made by Simon Stevin from Bruges. He created a way to write real numbers as a decimal expansion. It was Stevin who introduced the concept of ‘digits after the decimal point’. In this same century the complex numbers were discovered in the context of cubic and quartic polynomials. The computation rules for these numbers were written down by Rafael Bombelli. Later in 1637, it was René Descartes who coined the terms ‘real’ and ‘imaginary’ numbers. [55]

Yet, it still took more than 200 years before we would get a first formal mathematical definition of the real numbers $\mathbb{R}$. It was the work of the great logicians of the late 19th century, spearheaded by Georg Cantor. He provided in 1874 a formal logical construction of the real numbers. It may seem surprising that a rigorous definition of the real numbers came so late in the rich history of mathematics. But, in practice, the development of calculus, and the theory of functions, did not really require strict formal definitions to advance.

1.1.1 Calculus

From a natural sciences view, it is natural to study relations between quantities, for example position in function of time, or air resistance in function of velocity. Yet, the earliest uses of calculus seem to have their origins in geometry: finding the area under a parabola, or the volume of a cone for example.

Throughout Ancient times and the Middle Ages, across mathematical texts of many cultures, diverse examples of what we now call derivatives and integrals can be found. But all these examples were concrete applications to problems. Only in the seventeenth century were the first unifying theories of derivatives and integrals of functions developed. It was by none other than Newton and Leibniz. The history between these two fathers of calculus is much richer than one would expect, it reads almost as a screenplay. In short, it is now accepted that both mathematicians independently developed the theory of calculus; at the time it was not.

The notations

$$\int f(x)\,\mathrm{d}x \quad \text{and} \quad \frac{\mathrm{d}f(x)}{\mathrm{d}x}$$

for the integral and the derivative of a function f were introduced by Leibniz. Newton introduced the notation

$$\dot{f}$$

for the derivative of f. In mathematical texts this notation is less common, but it can be abundantly found in physics. In this work we will mostly use $\frac{\mathrm{d}f}{\mathrm{d}x}$ for the derivative of f. If the context makes the variable to which the derivative is taken clear, then we will abbreviate $\frac{\mathrm{d}f}{\mathrm{d}x}$ to f'.

For many examples of how the theory of calculus and especially differential equations have developed throughout the last 400 years, we refer the interested reader to [5]. One of these examples explains what Newton probably understood by integrals and quadratures, and how these were computed. Another example is concerned with how the works of d'Alembert evolved.

The first differential equations, as we know them now, were studied shortly after Newton's work. Leibniz himself and the Bernoulli brothers came across differential equations in the context of geometry and mechanics. In the eighteenth century the multivariate theory of Leibniz was extended and the first partial differential equations were written down. In the nineteenth century from the theoretical side, more results about the existence of solutions of differential equation were found. From the practical side, many more applications were found. Not only in mechanics, but also in heat transfer, optics, fluid flow and electricity and magnetism were differential equations instrumental. The equations of Navier and Stokes or the laws of Maxwell, to name a few, were developed. We refer to [92] for a more detailed view on the great minds of mathematics who build our modern understanding of calculus.

The developments from the nineteenth century continued into the twentieth. Great new theoretical strides were made, and many new applications were found. Most notably, for this thesis at least, in quantum mechanics, Schrödinger's equation [90] was developed.

1.1.2 Advent of computing

Most[1] differential equations cannot be explicitly solved. For ordinary linear differential equations with constant coefficients, for example, analytical solutions are available. But if one of the coefficients is not a constant, these formulae fail. For this reason, numerical methods for differential equations were developed. These methods do not solve the equation in the strictest sense, but they are able to approximate solutions arbitrarily accurate. The first numerical method is Euler's method[2] from 1768. Albeit simple, in the nineteenth century Cauchy proved it to converge. This convergence assures that, if sufficiently small steps are taken, arbitrarily accurate approximations can be obtained.

[1]Mathematically, we can even say 'almost all'.

[2]Both the forward Euler method as the backward version.

Later, in the middle of the nineteenth century multistep methods were developed. And at the end of that century, Runge published the first Runge–Kutta method. These 'new' methods were of higher order than the simpler methods and therefore could take larger steps to reach sufficiently accurate results.

What strikes me the most is that all these methods were developed long before computers were invented. This implies that the first applications of these methods required an unfathomable amount of manual computation. Now, we are spoiled, and computers do all the numerical heavy lifting for us.[3] Back then, the term 'computer' did exist, but it was a profession, not a device.

Throughout the twentieth century, with the rise of computing devices, the use and applicability of these numerical methods for ordinary differential equations exploded. We were able to reach never-before-seen accuracies in fractions of the time it would have costed only a few decades earlier. This explosion in computational power still continues. Now, the speed of affordable consumer desktop computers can be measured in teraFLOPS[4]. For example, my university computer (Intel i7-8700K and an NVIDIA Quadro P2000) has a combined rated speed of a little over 3 teraFLOPS. Almost all of this speed comes from its graphics processing unit, with the CPU only contributing 0.06 teraFLOPS. This already indicates that FLOPS is an imperfect unit. Writing a program that is able to use *all* these FLOPS is prohibitively difficult.

1.2 Ordinary differential equations

In the most general formulation, an n-dimensional k^{th} order *ordinary differential equation*

$$\mathbf{f}\left(x, \mathbf{y}, \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}, \frac{\mathrm{d}^2\mathbf{y}}{\mathrm{d}x^2}, \dots, \frac{\mathrm{d}^k\mathbf{y}}{\mathrm{d}x^k}\right) = \mathbf{0} \tag{1.1}$$

expresses n constraints on an unknown sufficiently differentiable function $\mathbf{y} : \Omega \to \mathbb{R}^n$ with $\Omega \subseteq \mathbb{R}$ and $\mathbf{f} : \Omega \times (\mathbb{R}^n)^{k+1} \to \mathbb{R}^n$. In most cases, this equation alone is insufficient to determine a unique solution $\mathbf{y}(x)$. For this, some initial or boundary conditions can be imposed on $\mathbf{y}$ (or any of the derivatives). As this abstract description is not intuitive, we provide some examples.

[3]Regularly I feel very fortunate to be a PhD-student now, and not then. I assume that, before computers, many tedious computations must have been assigned to students.

[4]'FLOPS' is a measure of how many *fl*oating point *o*perations *p*er *s*econd a computer can execute. A computer with a speed of 1 teraFLOPS can execute one trillion (10^{12}) floating point operations each second.

Integrals
The problem of determining primitive functions $F(x)$ can be rewritten as a one-dimensional first order ordinary differential equation:

$$F(x) = \int f(x)\,\mathrm{d}x \qquad \Longleftrightarrow \qquad \frac{\mathrm{d}F}{\mathrm{d}x} = f(x).$$

A definite integral can similarly be formulated as a differential equation, however a boundary condition has to be imposed:

$$\int_a^b f(x)\mathrm{d}x = F(b) \quad \text{ with } \frac{\mathrm{d}F}{\mathrm{d}x} = f(x) \text{ and } F(a) = 0.$$

Mathematically, this is a valid example. Intuitively however, this seems rather trivial.

Swinging pendulum
Due to Newton's laws of motion, physical systems often give rise to second order ordinary differential equations. One such example is a pendulum. Here, a mass is suspended such that it can swing freely back and forth. Let $\theta(t)$ be the angle by which the mass is displaced from its hanging resting position at time t. The movement of the pendulum is described by

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{g}{l}\sin(\theta) = 0,$$

with g the gravitational constant, and l the distance between the fixed point and the mass. As initial conditions at time t_0, $\theta(t_0)$ represents the displacement angle at t_0 and $\theta'(t_0)$ represents some initial angular velocity with which the mass is released.

Brachistochrone curve
Mathematicians love challenging each other. This is true today, as much as it was 300 years ago. In 1696, Johann Bernoulli posed the Brachistochrone problem: on which curve slides a frictionless bead the fastest from point A to B? There are many strategies for solving this problem. In one such solution, the following differential equation arises:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \sqrt{\frac{D-y}{x}} \text{ with } y(0) = 0,$$

with D a parameter.

Lotka–Volterra equations
In biological modelling, dynamical systems can describe many real-world situations. The Lotka–Volterra equations make up one such system. This set of equations models the population size of two species. One function $r(t)$ expresses the population size of prey (for example rabbits), the other function $f(t)$ tracks the population size of the predatory species (for example foxes). The Lotka–Volterra equations are

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy \\ \frac{\mathrm{d}y}{\mathrm{d}t} = \delta xy - \gamma y \end{cases} .$$

In this two-dimensional first order ordinary differential system, the constants α, β, γ and δ are parameters which describe the particular characteristics of the situation. Typically, as initial conditions, a researcher supplies the population sizes of both species at the start of the simulation.

1.2.1 Sturm–Liouville problems

These were only a few short examples from a myriad of real-world applications. In chapter 2 we will focus upon the numerical solution of a particular class of second-order linear ordinary differential equations: Sturm–Liouville equations.

A Sturm–Liouville equation is defined by three functions $p(x)$, $q(x)$ and $w(x)$ on a domain $\Omega \subseteq \mathbb{R}$:

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}y}{\mathrm{d}x}\right) + q(x)y = \lambda w(x)y.$$

This problem is only well-defined if appropriate boundary conditions are provided as well. For more details about different types of boundary conditions we refer the reader to the introduction in chapter 2.

One striking particularity of these problems is that, besides $y(x)$, the constant λ is an unknown as well. Solutions for λ are called eigenvalues and the corresponding solution for $y(x)$ is what is called an eigenfunction.

Differential equations can be approached from many angles. Some researchers take a pure theoretical approach, where many powerful results are proven. Other researchers start with real-world applications and consider differential equations as simply a tool in the toolbox to tackle their problems. The approach we have chosen falls somewhere in between. From the beginning we set out to develop and implement numerical methods for particular differential equations.

To build a useful and accurate method, many theoretical results are needed. However, also practical feasibility and computational efficiency are essential.[5]

1.3 Partial differential equations

One of the main characterizing properties of ordinary differential equations is that there is only a single independent variable. In equation (1.1), x is this independent variable. A *partial* differential equation is a generalization which allows multiple independent variables. As these equations are commonly used in physical problems, the independent variables are most commonly the temporal coordinate t and the spatial coordinates $x, y, z, \dots$, or more general $x_1, x_2, \dots, x_n$.

Providing a clean general one-formula definition of partial differential equations is quite cumbersome, and not at all instructive. Therefore, let us immediately take a look at some examples.

The heat equation
Conductive heat transfer (among many other diffusive phenomena) is described by the heat equation. Let $f(t, x)$ be the temperature along an insulated metal rod with $x \in [0, 1]$ and $t \in [0, \infty[$. Physical laws dictate that this function satisfies

$$\frac{\partial f}{\partial t} = \alpha \frac{\partial^2 f}{\partial x^2}.$$

In this expression, α is a parameter. As is the case for ordinary differential equations, to uniquely define a partial differential problem, boundary and or initial conditions have to be imposed. If for example the rod is heated at one side to 500 K, and cooled at the other to 250 K, then this can be expressed as $f(t, 0) = 500$ and $f(t, 1) = 250$ for all $]0, \infty[$. However, this is still insufficient to properly define the problem. Initial conditions have to be provided for $t = 0$. These should express the temperature throughout the rod at the beginning of the simulation. For example, if the experiment is started with the rod at room temperature 300 K, then this is captured as $f(0, x) = 300$ for all $x \in [0, 1]$.

This problem can be easily generalized to higher dimensional situations. Assume a similar experiment is conducted with a metal unit sphere. Let $g : [0, \infty[\times \Omega :$

[5]Personally, I feel this is the true strength of this thesis. Neither the theoretical advancements we made, nor the efficiency of our implementation are perfect. However, I believe the powerful combination of the two to be the real innovation.

$(t, x, y, z) \to g(t, x, y, z)$ be the temperature throughout this sphere $\Omega := \{(x, y, z) \in \mathbb{R}^3 | x^2 + y^2 + z^2 \leq 1\}$. Physics dictates that this g satisfies:

$$\frac{\partial g}{\partial t} = \alpha \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} + \frac{\partial^2 g}{\partial z^2} \right) = \alpha \nabla^2 g.$$

Again, appropriate boundary and initial conditions have to be specified.

Gauss's law for magnetism

Classical electromagnetism states that no magnetic monopoles exist. Mathematically this can be expressed as: at all times, the magnetic field $\mathbf{B} : \mathbb{R}^3 \to \mathbb{R}^3$ is divergence-free. Writing this as a partial differential equation makes this clear:

$$\nabla \cdot \mathbf{B} = 0 \quad \Longleftrightarrow \quad \frac{\partial \mathbf{B}}{\partial x} + \frac{\partial \mathbf{B}}{\partial y} + \frac{\partial \mathbf{B}}{\partial z} = 0. \tag{1.2}$$

This is a first-order three-dimensional linear partial differential equation. It illustrates that many laws and theories in physics can be formulated as (partial) differential equations.

In engineering, computer models of real-world situations are invaluable in making decisions. All of these models have to be based on (our understanding of) the world around us and the physical laws governing it. Partial differential equations are therefore essential in many of these models. If one wants to simulate magnetism, then equation (1.2) may be sufficient. However, if electricity is also involved, then the more general (and much more complicated) equations of Maxwell have to be solved.

Incompressible Navier–Stokes equations

To simulate the flow of incompressible fluids[6], the incompressible Navier–Stokes equations can be used. Let $\mathbf{u} : \mathbb{R} \times \mathbb{R}^3 \to \mathbb{R}^3$ express the flow vector at each moment t in each point (x, y, z). For incompressible flow, $\mathbf{u}$ has to satisfy

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= \nabla^2 \mathbf{u} - (\mathbf{u} \cdot \nabla)\mathbf{u} + \mathbf{f} \\ &\text{while } \nabla \cdot \mathbf{u} = 0. \end{aligned}$$

[6]The flow of water is a straightforward example. Also, these equations can handle slow flowing wind, as pressure can be assumed constant. If pressure cannot be neglected, then the compressible equations should be used.

As one may already suspect, for most[7] partial differential equations, numerical methods need to be developed. In state-of-the-art modelling software, many tools are available. As in many of these programs complicated geometries are supported (such as the blades of a wind turbine, or the CAD-model of a bridge for example), numerical simulation methods based upon finite element techniques are preferred. These methods create a triangular mesh (or tetrahedral mesh for three-dimensional problems) of the domain, and use approximations on each of the cells of this mesh. In general, these techniques can be employed for an extremely diverse set of problems, albeit with an exceptionally high computational cost.

1.3.1 Time-independent Schrödinger problems

In this thesis, and in particular chapters 2 and 3, we will be solving two-dimensional time-independent Schrödinger equations [90]. These are defined by a potential $V(x, y)$ on a domain $\Omega \subseteq \mathbb{R}^2$. The goal is to find all functions $\psi : \Omega \to \mathbb{R}$ and values $E \in \mathbb{R}$ such that[8]

$$-\nabla^2\psi + V(x, y)\psi = E\psi,$$

with appropriate boundary conditions.

As many partial differential equations, the Schrödinger equation has its origins in physics. More specifically, the time-dependent Schrödinger equation describes wave functions in quantum mechanics. As I am a mathematician, I am not at all qualified to give any introduction to such an immensely complicated physics subject. Fortunately, in this thesis we are not concerned with the 'why'[9] of this equation; we are only trying to solve it.

[7]Again, mathematically, we can say 'almost all'.

[8]In principle, ψ and E could be complex valued. In theorem 2.1, we will pose that all solutions are real.

[9]This is an easy-to-make statement as a mathematician. However, it is a dangerous statement for a scientist (which we most definitely are). Being blind to the applications is immoral. Quantum mechanics, and more generally our understanding of physics, has two sides. On the one hand, it obviously improves uncountable many lives immeasurably, directly and maybe even more so indirectly. On the other hand, not all human inventions have been for the better. The most incomprehensibly destructive weapons, for example, have only been possible due to advances in nuclear physics. As we are developing algorithms and methods for solving Schrödinger equations, implicitly (and by acknowledging this, now explicitly as well) we put our trust in the researchers who will be using our tools. So, to any user benefitting from our work, let this footnote be my plea to use it morally and responsibly.

1.4 Notation

Throughout this thesis we try to follow common notations present in the literature. However, different authors prefer different notations. For example from a physics perspective, $\langle f|g\rangle$ is extremely common. In mathematical analysis this notation is rare. There, $\langle f, g\rangle$ or $f \cdot g$ are more frequently used. For clarity, in this section we give an overview of the notational conventions used in this work.

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ $\mathbb{N}^+, \mathbb{Q}^+, \mathbb{R}^+$	The set of the natural numbers, the integers, the rational numbers, the real numbers and the complex numbers respectively. A superscript plus indicates the subset of only strictly positive numbers.
$a, b, c, \dots$ $\alpha, \beta, \gamma, \dots$	Lowercase variables are assumed to be scalars.
$\mathbf{u}, \mathbf{v}, \dots$	Lowercase bold variables are assumed to be vectors.
$\mathbf{B}, \mathbf{M}, \mathbf{\Lambda}, \dots$	Uppercase bold variables are assumed to be matrices.
$\|\mathbf{x}\|_p$	The $L_p(\mathbb{R}^n)$-norm: $(\sum_{i=1}^n \lvert x_i\rvert^p)^{\frac{1}{p}}$.
$\|\mathbf{x}\| = \|\mathbf{x}\|_2$	By default, norms are assumed to be Euclidean.
$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots\right)^\mathsf{T}$	The gradient of a function is computed with respect to the space variables (not time).
$\Delta f = \nabla^2 f = \nabla \cdot \nabla f$	The Laplacian of a function is calculated as $\frac{\partial^2 f}{\partial x_1^2} + \frac{\partial^2 f}{\partial x_2^2} + \dots$.
$\partial\Omega$	For a domain $\Omega \in \mathbb{R}^n$, the boundary $\partial\Omega$ is defined as $\overline{\Omega} \cap \overline{\Omega^c}$ (this is the intersection of the closure of Ω with the closure of the complement of Ω.).
$\langle f \mid g\rangle$	For functions $f, g : \Omega \to \mathbb{C}$ defined on $\Omega \subseteq \mathbb{R}^n$, their inner product is calculated as $\int_\Omega f(\mathbf{x})\overline{g(\mathbf{x})}\, \mathrm{d}\mathbf{x}$.

Chapter 2

One-dimensional time-independent Schrödinger equations

2.1 Introduction

The one-dimensional time-independent Schrödinger equation is a linear ordinary differential equation posed as an eigenvalue problem on a domain $\Omega \subseteq \mathbb{R}$ with specified boundary conditions. A solution is given as an eigenvalue $\lambda \in \mathbb{R}$ with corresponding eigenfunction $y : \Omega \to \mathbb{R}$. Each solution has to satisfy the following equation

$$-y''(x) + V(x)y(x) = \lambda y(x),$$

for each of the values $x \in \Omega$. In this equation, the given function $V : \mathbb{R} \to \mathbb{R}$ is the potential of the problem at hand. Note that if $y(x)$ is an eigenfunction, $c\,y(x)$ will also be an eigenfunction with the same eigenvalue, for each value of $c \in \mathbb{R}_0$. As such, it is not really possible to say "*the* eigenfunction corresponding to a given eigenvalue". Later on we will prove that in many cases the eigenfunction is, up to a constant factor, uniquely defined.

Boundary conditions have to be specified before solutions can be found. These conditions pose restrictions on $y(a)$, $y'(a)$, $y(b)$ and $y'(b)$. Boundary conditions come in many flavors. We provide an overview of the most common ones:

- *Dirichlet boundary conditions* specify which value the solution takes on the boundary of the domain. In our case, eigenfunctions can always be scaled, as such, it is not useful to specify the value of the solution different from zero on the boundary. This type of boundary conditions thus simplifies to $y(a) = 0$ and $y(b) = 0$, and is therefore called the homogeneous Dirichlet boundary conditions.
- *Neumann boundary conditions* specify which value the derivative of a solution takes on the boundary of the domain. In our case, the same remark as given for the Dirichlet boundary conditions applies. This means that homogeneous Neumann boundary conditions imply that $y'(a) = 0$ and $y'(b) = 0$.
- *Robin boundary conditions* are a generalization of both previous boundary conditions. When we impose these conditions on a solution $y(x)$, we imply that a certain weighted average of the function and its derivative are a fixed value. In our case, homogeneous Robin boundary conditions can be imposed. This can be written as

$$\alpha_a y(a) + \beta_a y'(a) = 0 \text{ and } \alpha_b y(b) + \beta_b y'(b) = 0.$$

These three types of boundary conditions are what is called *separated conditions*. The condition on the left side a is decoupled from the one on the right side b. In contrast, there are also boundary conditions that do couple both ends.

- *Periodic boundary conditions* are used to specify that a solution should be periodic. In other words, the solution has to end in the same value as it started, and so should the derivative. Mathematically this can be written as: $y(a) = y(b)$ and $y'(a) = y'(b)$. These condition can be extended to *antiperiodic boundary conditions*: $y(a) = -y(b)$ and $y'(a) = -y'(b)$. Or even generalized with a 2×2 matrix $\mathbf{K}$:

$$\begin{pmatrix} y(a) \\ y'(a) \end{pmatrix} = \mathbf{K} \begin{pmatrix} y(b) \\ y'(b) \end{pmatrix}.$$

Note that homogeneous Dirichlet or Neumann boundary conditions can always be written as homogeneous Robin boundary conditions. So when studying the one-dimensional time-independent Schrödinger equation, it is more general to consider homogeneous Robin boundary conditions. Periodic (or generalized periodic) boundary conditions are less common and problems with these boundary conditions lose some of the nice properties which are guaranteed for separated conditions. This case will be studied later in section 2.4.

2.1.1 Properties of the Sturm–Liouville equation

Before developing numerical methods for solving the one-dimensional time-independent Schrödinger equation, it is important to build a strong theoretical foundation. The goal is to build a thorough understanding of the Schrödinger equation and to use this intuition to develop efficient and accurate numerical algorithms to solve this equation.

In the scientific literature, it is quite rare to find studies about the one-dimensional Schrödinger equation itself. Most, if not all, relevant articles and books cover the more general topic of Sturm–Liouville theory. As Sturm–Liouville equations are a generalization of Schrödinger equations they are more widely applicable, and as such more impactful to study. In this section, we will follow the tradition from the literature and study the Sturm–Liouville equation. Many more details and examples of Sturm–Liouville theory can be found in relevant textbooks, for example [88, chapter 5].

The Sturm–Liouville equation is also a linear ordinary differential equation posed as a boundary value eigenproblem, given by the following equation on the domain $\Omega \subseteq \mathbb{R}$:

$$-(p(x)y'(x))' + q(x)y(x) = \lambda w(x)y(x). \tag{2.1}$$

The functions $p(x)$, $q(x)$ and $w(x)$ are given on the domain Ω. These functions define the Sturm–Liouville problem. If the domain is a bounded interval $\Omega = [a, b]$, and if $p(x)$, $q(x)$ and $w(x)$ are continuous, $p(x)$ is differentiable and $p(x) > 0$ and $w(x) > 0$ over the whole domain $[a, b]$, then the problem is called a *regular* Sturm–Liouville problem.

A solution of a Sturm–Liouville problem consists of an eigenvalue λ with corresponding eigenfunction $y(x)$, which satisfy (2.1) on $[a, b]$. For now, we will study the Sturm–Liouville equation with homogeneous Robin boundary conditions:

$$\alpha_a y(a) + \beta_a p(a)y'(a) = 0 \text{ and } \alpha_b y(b) + \beta_b p(b)y'(b) = 0. \tag{2.2}$$

Note that the Schrödinger equation with homogeneous Robin boundary conditions is a special case of the Sturm–Liouville equation, specifically when $p(x) = 1$, $q(x) = V(x)$ and $w(x) = 1$.

Many theorems can be formulated about self-adjoint linear operators[1]. For now, we provide a few specific theorems about Sturm–Liouville equations. The proofs for some of these theorems are quite extensive and require a thorough understanding of functional analysis. In this work, we will not provide these proofs.

For implementing methods to find solutions, it is valuable to study what a solution would look like. Since the solutions of a Sturm–Liouville problem are eigenvalues with corresponding eigenfunctions, solutions will consist of a (possibly) complex number $\lambda \in \mathbb{C}$ with a corresponding function $y : [a, b] \to \mathbb{C}$. The fact that these could be complex values impacts our study significantly. So, let us first state that solutions can always be real.

Theorem 2.1. *Let $p(x)$, $q(x)$ and $w(x)$ define a regular Sturm–Liouville problem on $[a, b]$.*

All eigenvalues of the Sturm–Liouville problem with real homogeneous Robin boundary condition (2.2) *are real. Corresponding eigenfunctions can always be scaled such that they are real.*

Knowing that we only need to find real solutions simplifies our work. The next theorem allows us to get a grasp on the number of eigenvalues we need to find.

Theorem 2.2. *The number of eigenvalues of a regular Sturm–Liouville problem with homogeneous Robin boundary conditions are countable. All eigenvalues have a lower bound, no upper bound and are unique. This implies that these can be written as:*

$$\lambda_0 < \lambda_1 < \lambda_2 < \cdots \to \infty.$$

We say that the k^{th} eigenvalue λ_k has index *k.*

This last theorem is essential. It enables us to formulate questions such as: "Find all eigenvalues less than 500." or "Find the first 10 eigenvalues." Besides an eigenvalue, a solution also consists of an eigenfunction.

Theorem 2.3. *With each eigenvalue λ_k, an up to scaling unique eigenfunction $y_k : [a, b] \to \mathbb{R}$ is associated. Eigenfunctions corresponding to different*

[1]A self-adjoint linear operator is a concept from functional analysis. A regular Sturm–Liouville problem is equivalent with finding the spectrum of a specific linear self-adjoint operator. The powerful mathematical machinery of functional analysis can thus be valuable for us as well.

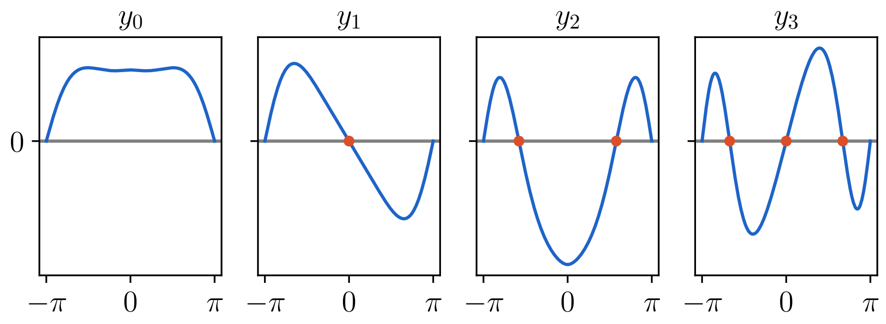


Figure 2.1: The first four eigenfunctions of the Sturm–Liouville problem with $p(x) = 1$, $q = \sqrt{|x|}$, $w(x) = 1 + x^2$ and homogeneous Dirichlet boundary conditions on $[-\pi, \pi]$. The roots of each of these functions are indicated.

eigenvalues are orthogonal with respect to the weight function $w(x)$:

$$\int_a^b y_m(x)y_n(x)w(x)\,\mathrm{d}x = 0 \qquad \textit{if } m \neq n.$$

Even stronger: Sturm–Liouville theory provides a way to characterize the index of an eigenvalue by means of the eigenfunction only.

Theorem 2.4. *The eigenfunction $y_k(x)$ corresponding to the k^{th} eigenvalue λ_k has exactly k zeros in the interior of the domain $[a, b]$.*

This theorem is illustrated in figure 2.1. Here we see the first few eigenfunctions of a Sturm–Liouville problem. Later on we will use the term "highly oscillatory eigenfunction", theorem 2.4 explains why this is justified. If k becomes large, the eigenfunction will have many zeros and therefore will oscillate heavily. This can also already be seen on the last panel of figure 2.1.

With these four theorems in the back of our mind, we will be able to develop numerical methods for solving Sturm–Liouville and Schrödinger problems. As stated in the beginning of this section, we have given a very brief overview of the theorems that are relevant for us. If the reader is interested in more details, or in a rigorous theoretical framework many textbooks are available, for example [88, 35, 103, 34].

Throughout this thesis, and the research preceding it, we have not and will not focus on the applicability of the Sturm–Liouville equation. This is definitely not due to a lack of applications. It is rather the opposite: Sturm–Liouville equations appear in an innumerable diversity of fields. Many real-world problems can be stated as or reduced to differential equations and in particular to the Sturm–Liouville equation.

2.1.2 Liouville's transformation

In [72], Liouville introduced a transformation from a Sturm–Liouville problem to a Schrödinger problem. Here we provide just the transformation, with some remarks on how to implement it. For a derivation the original work by Liouville [72] can be consulted, or a more clear text in the context of `Matslise` can be found in [67]. In this last work, the implementation also gets some attention.

In this subsection, we change the notation a little from (2.1). Let us consider the Sturm–Liouville equation

$$-(p(r)z'(r))' + q(r)z(r) = \lambda w(r)z(r), \tag{2.3}$$

on the domain $[r_{\min}, r_{\max}]$.

With the transformation

$$\begin{aligned} x &= \int_{r_{\min}}^{r} \sqrt{\frac{w(r')}{p(r')}}\,\mathrm{d}r', \\ z &= \sigma y \\ \text{and } V(x) &= \frac{q}{w} + \sigma \frac{\mathrm{d}^2}{\mathrm{d}x^2}\frac{1}{\sigma} \\ \text{with } \sigma(r) &= \frac{1}{\sqrt[4]{p(r)w(r)}}, \end{aligned}$$

equation (2.3) becomes

$$-y''(x) + V(x)y(x) = \lambda y(x)$$

on the domain $[0, x_{\max}]$, with $x_{\max} = x(r_{\max})$. In this transformation, we assumed $p(x)$ and $w(x)$ to be strictly positive and twice differentiable on the domain.

The boundary conditions

$$\begin{aligned} \alpha_{\min} z(r_{\min}) + \beta_{\min} p(r_{\min}) z'(r_{\min}) = 0 \\ \alpha_{\max} z(r_{\max}) + \beta_{\max} p(r_{\max}) z'(r_{\max}) = 0 \end{aligned}$$

are transformed into

$$\begin{aligned}
&A_{\min} y(0) + \beta_{\min} y'(0) = 0\\
&A_{\max} y(x_{\max}) + \beta_{\max} y'(x_{\max}) = 0,\\
\text{with } &A_{\min} = \alpha_{\min}\sigma^2(r_{\min}) + \beta_{\min}\sigma'(r_{\min})\sigma(r_{\max})\\
\text{and } &A_{\max} = \alpha_{\max}\sigma^2(r_{\max}) + \beta_{\max}\sigma'(r_{\max})\sigma(r_{\max}).
\end{aligned}$$

Implementing these formulae is not trivial. In [67], it is proposed to use an adaptive quadrature rule to approximate $x(r)$ up to the full precision available in the `double` datatype. The inverse function $r(x)$ can then be calculated by applying a Newton-Raphson scheme. Again, this computation is executed to get the full precision available in the used floating-point type.

Evaluating these functions up to the maximal precision available may seem overkill. However, we have found[2] this to be essential. The constant perturbation method used to solve the resulting Schrödinger equation builds a piecewise high-order polynomial approximation of the potential function. If these functions are evaluated with relatively low accuracy, the piecewise polynomial approximation algorithm has a hard time to reach the required accuracy and far too many subintervals will be used.

2.2 Background about Matslise

Numerical methods for ordinary differential equations as *initial value* problems are already many centuries old, starting with Euler's method. Numerical methods as a research topic by itself really took off a little more than a century ago. In particular, linear multistep methods and Runge–Kutta methods were described around 1900. First they were applied and calculated by hand, later on "calculating machines" [74] were used. Nowadays, modern computers do all the tedious computations.

For ordinary differential equations as *boundary value* problems, such as the Sturm–Liouville equations, the story is different. There still are only a few general methods for such problems. For linear ordinary differential equations, one of the more popular choices is a method based upon finite differences.

[2]This is an example of things we learned the hard way. See the first paragraph of section 2.6.1.

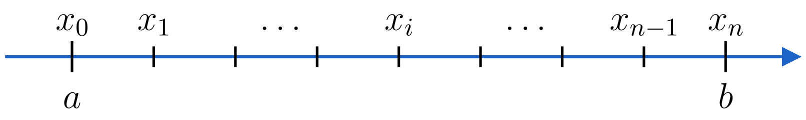


Figure 2.2: An equidistant division of the domain $[a, b]$ in n subintervals.

To illustrate the idea behind methods based upon finite differences, consider the linear second order differential equation

$$y''(x) = \alpha + \beta y(x) + \gamma y'(x)$$

on the interval $[a, b]$ with boundary conditions $y(a) = y_a$ and $y(b) = y_b$. First, we discretize the integration domain $[a, b]$ into an equally spaced grid of points $a = x_0$, x_1, …, x_i, …, x_{n-1}, $x_n = b$, with Δx the distance between two consecutive points. This discretization can be seen in figure 2.2. The differential equation is now approximated by using finite difference expressions of the involved derivatives. For example, the second order approximations:

$$\begin{aligned} y''(x_i) &\approx \frac{y(x_{i-1}) - 2y(x_i) + y(x_{i+1})}{\Delta x^2} \\ \text{and } y'(x_i) &\approx \frac{y(x_{i+1}) - y(x_{i-1})}{2\Delta x}. \end{aligned}$$

This simplification yields a linear system in the variables $y(x_1), \dots, y(x_{n-1})$. In general, for linear ordinary differential equations with linear boundary conditions, this technique leads to a linear matrix-vector reformulation of the differential equation, which can be solved with classical linear algebra tools.

In this general technique, constructing higher order methods is rather straightforward. One only has to use a more accurate finite difference approximation. However, these methods become increasingly computationally expensive as more accuracy is required. In our case, accurately determining higher eigenvalues requires significantly more computation time.

2.2.1 Finite difference scheme for the Sturm–Liouville equation

Until around 1990, the best methods [4, 99] for approximating solutions to the Sturm-Liouville problem looked at the simpler form of the Schrödinger equation and employed a finite difference scheme. After solving the approximating linear algebra problem, some corrections can be applied to improve the accuracy of higher eigenvalues. The authors of [4, 99] experimented with several finite difference approximations. First classical formulae were tried, later on highly tuned exponential-fitted formulae were developed to better handle the oscillatory nature of the eigenfunctions.

To illustrate this class of finite difference methods for Sturm–Liouville equations we will develop a simple version ourselves. This will allow us to appreciate the nuances of these methods more, and it will give us some ideas about the general advantages but also disadvantages of this technique. For completeness, we recall the Sturm–Liouville equation from (2.1):

$$-(p(x)y')' + q(x)y = \lambda w(x)y.$$

In this example, we will approximate the solution to this equation on the interval $[a, b]$ and impose homogeneous Dirichlet boundary conditions $y(a) = y(b) = 0$. As stated earlier, solutions will consist of eigenvalues λ and corresponding eigenfunctions $y(x)$.

As a first step we discretize the domain $[a, b]$ with $n+1$ equidistant points, as in figure 2.2. The eigenfunctions $y(x)$ we are looking for, can now be approximated by values in each of the grid points $y(x_i) \approx y_i$. For translating this problem into a linear matrix-vector equation, we are missing one key component. We need a way to discretize the expression $(p(x)y')'$. For this, we apply the central second order finite difference formula for the first derivative twice, with half the step size. In more detail, the first derivative of a scalar function $f(x)$ can be approximated as:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

Applying this expression once to $(p(x)y')'$ in the point x_i with step size $h = \frac{\Delta x}{2}$ gives:

$$(p(x)y')'(x_i) \approx \frac{1}{\Delta x}\left((py')\left(x_{i+\frac{1}{2}}\right) - (py')\left(x_{i-\frac{1}{2}}\right)\right).$$

Applying the finite difference formula a second time with step size $\frac{\Delta x}{2}$ to approximate $y'(x)$ yields:

$$(p(x)y')'(x_i) \approx \frac{1}{\Delta x^2}\left(p_{i+\frac{1}{2}}y_{i+1} - \left(p_{i-\frac{1}{2}} + p_{i+\frac{1}{2}}\right)y_i + p_{i-\frac{1}{2}}y_{i-1}\right).$$

To ease notation, we have substituted $y(x_i)$ with its approximation y_i, and have denoted $\frac{x_i + x_{i+1}}{2}$ as $x_{i+\frac{1}{2}}$, and $p(x_i)$ as p_i. Also, note that if $p(x) = 1$ this formula simplifies to the classical, well-known, central second order approximation of the second derivative.

If we apply this finite difference approximation to (2.1) in each point x_i, then we get the linear generalized eigenvalue problem:

$$-\frac{1}{\Delta x^2}\left(p_{i+\frac{1}{2}}y_{i+1} - \left(p_{i-\frac{1}{2}} + p_{i+\frac{1}{2}}\right)y_i + p_{i-\frac{1}{2}}y_{i-1}\right) + q(x_i)y_i = \lambda w(x_i)y_i.$$

To emphasize that this is a linear algebra problem we can rewrite this in matrix notation:

$$\left(-\mathbf{T} + \operatorname{diag}(\mathbf{q})\right)\mathbf{y} = \lambda\operatorname{diag}(\mathbf{w})\mathbf{y}. \tag{2.4}$$

The $(n-1)$-dimensional vector $\mathbf{y} = \begin{pmatrix} y_1 & y_2 & \dots & y_{n-1}\end{pmatrix}^\mathsf{T}$ is the unknown approximation of the eigenfunction. The $(n-1)$-dimensional vectors $\mathbf{q} = \begin{pmatrix} q(x_1) & q(x_2) & \dots & q(x_{n-1})\end{pmatrix}^\mathsf{T}$ and $\mathbf{w} = \begin{pmatrix} w(x_1) & w(x_2) & \dots & w(x_{n-1})\end{pmatrix}^\mathsf{T}$ are the values of the coefficient functions q and w. And lastly, the matrix $\mathbf{T}$ is the tridiagonal matrix given by:

$$\mathbf{T} = \frac{1}{\Delta x^2}\begin{pmatrix} -p_{\frac{1}{2}} - p_{\frac{3}{2}} & p_{\frac{3}{2}} & & & \\ p_{\frac{3}{2}} & -p_{\frac{3}{2}} - p_{\frac{5}{2}} & p_{\frac{5}{2}} & & \\ & p_{\frac{5}{2}} & -p_{\frac{5}{2}} - p_{\frac{7}{2}} & p_{\frac{7}{2}} & \\ & & & \ddots & \\ & & & p_{n-\frac{3}{2}} & -p_{n-\frac{3}{2}} - p_{n-\frac{1}{2}} \end{pmatrix}.$$

To find the eigenvalues of (2.4) one can notice that if $w(x_i)$ is never 0 for $0 < i < n$ then $\operatorname{diag}(\mathbf{w})$ is invertible and the problem becomes a simple tridiagonal eigenvalue problem. This can then be solved with any of our favorite tridiagonal eigenvalue solvers. If w happens to be constant, this becomes a symmetric tridiagonal matrix, for which LAPACK [2], for example, contains the specialized routines `?stev` and relatives.

As a numerical experiment of this method we will take a look at the following Sturm–Liouville equation:

$$-\left((1+x)^2y\right)' + (x^2-2)y = \lambda e^x y \tag{2.5}$$

on the domain $[0,1]$ with homogeneous Dirichlet boundary conditions. Figure 2.3 shows the relative errors of the lowest four eigenvalues in function of the chosen number of grid points $n+1$. As n increases, the error decreases,

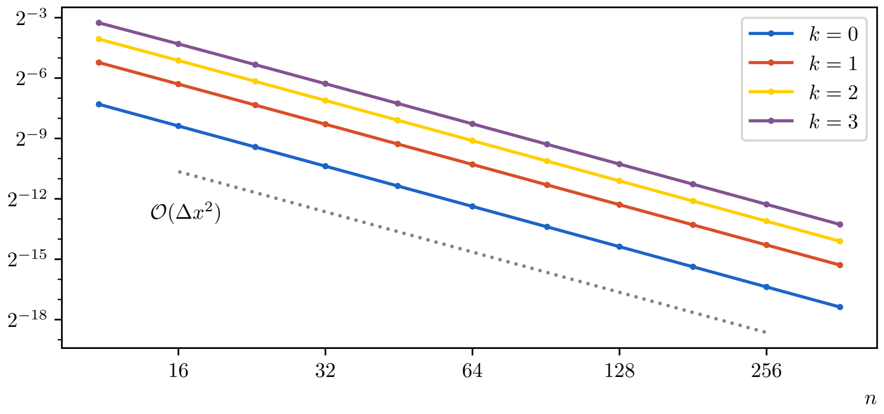


Figure 2.3: Relative error of approximated eigenvalues of problem (2.5) in function of the number of grid points $n + 1$ on the domain.

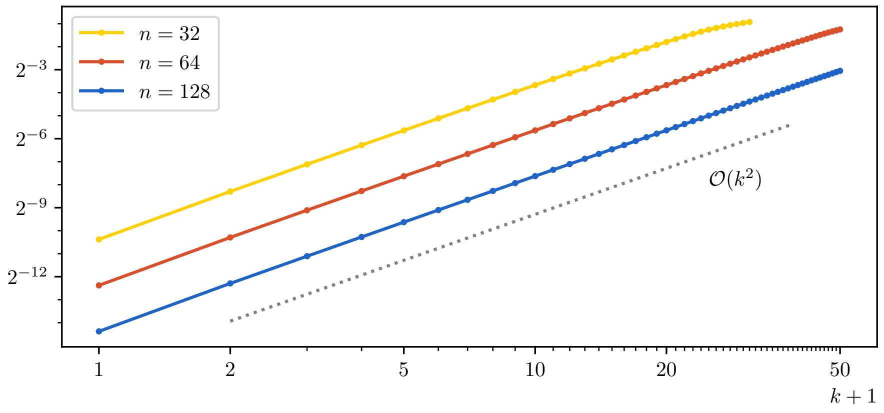


Figure 2.4: Relative error of approximated eigenvalues of problem (2.5) in function of the eigenvalue index k.

as desired. Notice that as the constructed method is based upon second order finite difference formulae, we expect to see the decrease of error be of second order too. This is indicated in the figure with the dotted line. The constructed method is of relatively low order, so to compute very accurate approximations, a dense grid is needed. In the literature, one can find many methods based upon finite difference approximations, many of which use (much) higher order formulae. These better methods can reach high accuracy with relatively little computational work for the first few eigenvalues.

As earlier hinted, methods based upon finite differences struggle with the computation of higher eigenvalues. Figure 2.4 illustrates this. Here, the relative error of the first 50 eigenvalues is plotted, for different values of n. Note that in the case of $n = 32$, only 32 eigenvalues can be computed. On this figure, the line corresponding to $\mathcal{O}(k^2)$ is drawn. Together with the $\mathcal{O}(\Delta x^2)$ from earlier, one expects the relative error of the k^{th} eigenvalue to be

$$\mathcal{O}(\Delta x^2 k^2)$$

for this method applied to problem (2.5). This shows us that the larger the eigenvalue, the more difficult it is to compute accurately. This lesson not only holds for this method in particular, but it is also applicable to most methods of this type. In the literature [4, 98, 99], one can find some techniques to mitigate this effect by applying some corrections. In [4] such a correction is analyzed when applied to Numerov's method [75]. Here, the authors are able to bring the error down from $\mathcal{O}(k^6h^4)$ to $\mathcal{O}(k^3h^4)$. This is remarkably better. But even with this correction technique, computing large eigenvalues is still computationally difficult.

2.2.2 CP-methods

The drawbacks of the methods based upon finite differences are already known for a long time. One of the first[3] works that tries to not only mitigate these troubles but rather fully fix them, was "A new method for the solution of the Schrödinger equation" [19] in 1970. There, Canosa and De Oliveira present a new method to approximate solutions of the one-dimensional time-independent Schrödinger equation. Similar ideas can also be found in internal reports [42, 43] by Ixaru in 1969. These ideas lay the foundations to what would later be

[3]In [61], a brief historical overview is given of the application of CP-methods to Sturm–Liouville problems. In this thesis, we will take the time to take a closer look at the earlier methods. They will provide us with a more intuitive understanding of the algorithms, in preparation of our own advancements within employing these ideas.

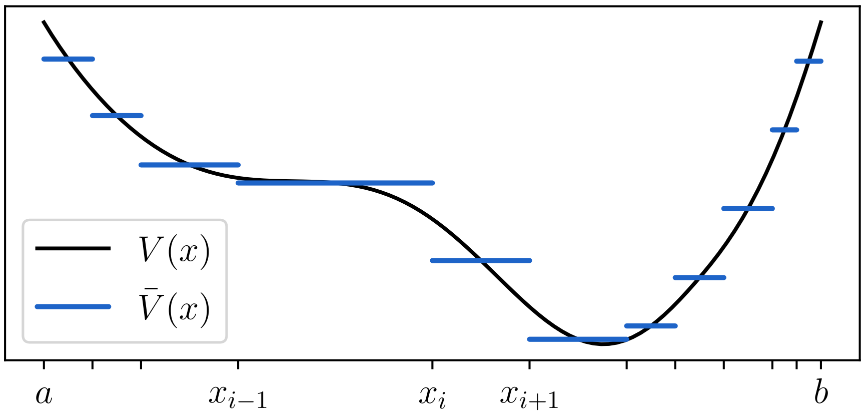


Figure 2.5: A potential function $V(x)$ is approximated as the piecewise constant function $\bar{V}(x)$ along the domain $[a, b]$.

called constant perturbation methods (CPM or CP-methods). To intuitively appreciate these CP-methods it is valuable to study the method from [19].

For this, we will only consider the Schrödinger equation

$$-y'' + V(x)y = \lambda y \tag{2.6}$$

on the domain $[a, b]$ with homogeneous Robin boundary conditions $\alpha_a y(a) + \beta_a y(a) = 0$ and $\alpha_b y(b) + \beta_b y(b) = 0$. However, do note that using Liouville's transformation Sturm-Liouville problems can be transformed into Schrödinger problems.

2.2.2.1 Piecewise constant approximation of the potential

As a very first step, [19] simplifies the potential $V(x)$ as a piecewise constant approximation $\bar{V}(x)$. This approximation is visualized in figure 2.5. Notice that no restrictions are placed upon the size of the subintervals, and a non-uniform grid is definitely allowed. Upon choosing $\bar{V}(x)$, one should keep in mind that the better the piecewise approximation is, the better the resulting eigenvalues will be. Also, the computational runtime depends linearly on the number of subintervals used.

The next step is to fix the value for λ for now. Then, for each subinterval $[x_i, x_{i+1}]$ the potential is approximated $V_i \approx V(x)$ for $x \in [x_i, x_{i+1}]$. With this approximation, the analytical solution is computed of the problem

$$y'' = (V_i - \lambda)y.$$

These analytical solutions $y_i(x)$ have the following structure:

$$y_i(x) = \begin{cases} A_i + B_i x & \text{if } V_i = \lambda, \\ A_i \cos\bigl(x\sqrt{\lambda - V_i}\bigr) + B_i \sin\bigl(x\sqrt{\lambda - V_i}\bigr) & \text{if } V_i < \lambda, \\ A_i \cosh\bigl(x\sqrt{V_i - \lambda}\bigr) + B_i \sinh\bigl(x\sqrt{V_i - \lambda}\bigr) & \text{if } V_i > \lambda. \end{cases}$$

To determine the appropriate values for A_i and B_i, continuity conditions at the grid points are applied:

$$y_{i-1}(x_i) = y_i(x_i) \text{ and } y'_{i-1}(x_i) = y'_i(x_i).$$

If n subintervals are used, this system of equations has $2n$ variables and $2(n-1)$ equations. Together with the two equations from the boundary conditions, this yields a fully determined linear system. Now, we have translated the problem of finding eigenvalues of the Schrödinger equation (2.6) to finding values for λ such that this system of equations has non-zero solutions. Only these solutions correspond to a non-zero eigenfunction.

Finding values for λ for which the constructed system becomes singular is not as trivial as one may assume. In [19], the authors have provided their own root finding algorithm based upon finding changes in the sign of the determinant. But this algorithm is not without issue. Eigenvalues can be arbitrarily close together, which makes it hard to ensure that one has found all requested values. Choosing the appropriate step sizes when the distance between eigenvalues increases, is a balance between efficiency and not missing any. Later on, we will describe a way to reliably and efficiently determine all required eigenvalues.

One of the main benefits of this “new method for the solution of the Schrödinger equation” [19] is that its accuracy does not depend on the size of the requested eigenvalue. Because only analytical solutions of the piecewise approximated problem are considered, oscillations can be represented exactly. Even the most extreme oscillations are cleanly captured inside the sin en cos of y_i. The idea of using analytical solutions allowed to develop the CP-methods.

When studying this method one can make the observation when looking at figure 2.5 that $\bar{V}(x)$ is a crude approximation of the function $V(x)$. The importance of this remark becomes even more apparent when one realizes that the accuracy of the method solely depends upon the accuracy of this approximation.

Nonetheless, the idea of Canosa and De Oliveira gained traction. In 1971 Ixaru [44] has written a note about the error analysis of this new method. In the same year Pruess [78] has studied this method thoroughly and provided numerical examples of linear piecewise approximations and quadratic piecewise approximations of the potential function. Due to this analysis, this is now known as Pruess's method.

Here we will state the most important theorems from [78], without proof. All details can be found in the original work.

Theorem 2.5 (Pruess 1973)**.** *Let λ_k be the k^{th} eigenvalue of the Schrödinger equation*

$$-y'' + V(x)y = \lambda y$$

on the domain $[a, b]$ with homogeneous Robin boundary conditions. And let $\tilde{\lambda}_k$ be the k^{th} eigenvalue of the approximate Schrödinger equation with potential $\bar{V}(x)$ on the same domain, with identical boundary conditions. Here $\bar{V}(x)$ is a piecewise m^{th} degree polynomial approximation, let h be the width of the largest subinterval in this approximation. For h sufficiently small, we have as $h \to 0$,

$$|\lambda_k - \tilde{\lambda}_k| = \mathcal{O}(h^{2m+2}) \textit{ for each } k.$$

This theorem provides justification for the idea of Canosa and De Oliveira. Even though a constant approximation ($m = 0$) may be crude, it still is a second order method. Furthermore, the theorem suggests that expanding this method up to higher degree piecewise polynomial approximations may be worthwhile.

In [78], the author also studied what happens to the error on the eigenvalues if k is increased. In [19], it was assumed that this error does not get worse as k increases.

Theorem 2.6 (Pruess 1973)**.** *Following the notation from theorem 2.5, assume $\bar{V}(x)$ to be a least squares m^{th} degree piecewise polynomial approximation of $V(x)$, i.e. on each subinterval $[x_i, x_{i+1}]$, $\bar{V}$ is the m^{th} degree polynomial such that the following is minimal*

$$\int_{x_i}^{x_{i+1}} \left(V(x) - \bar{V}(x)\right)^2 \mathrm{d}x.$$

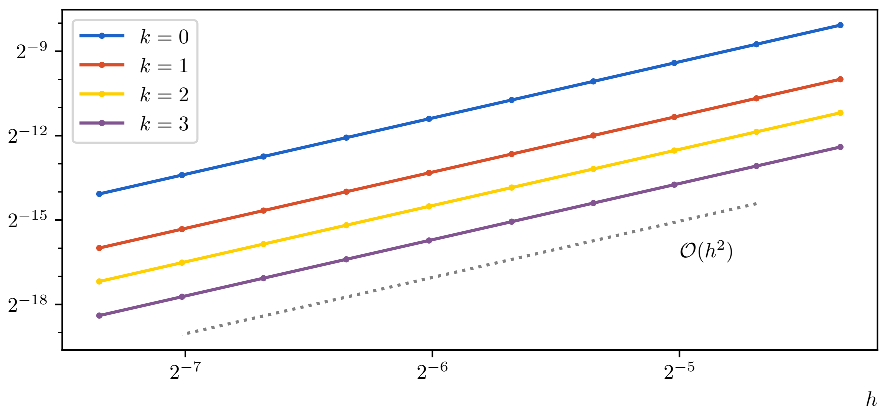


Figure 2.6: Relative error of the found eigenvalues of problem (2.7) by using the method of constant piecewise approximation of the potential on a uniform mesh with step size h. The most accurate calculation used 512 subintervals, the least accurate 64.

Under this assumption, if $k \to \infty$, it holds for the relative error that

$$\frac{\lambda_k - \tilde{\lambda}_k}{\lambda_k} = \mathcal{O}(k^{-4}).$$

This last theorem does not only say that the error does not become larger if k increases, it also tells us that the relative error decreases rapidly when $k \to \infty$. Theorem 2.6 highlights the main advantage of this technique in comparison to other state-of-the-art methods. Where many other methods become less and less accurate for large eigenvalues, this method becomes even more accurate. This makes it a so-called asymptotic method [59, 104].

As a numerical experiment we have applied the method with constant piecewise approximations ($m = 0$) to the problem

$$-y'' + 100\cos^2(x)\, y = \lambda y \tag{2.7}$$

on the domain $[0, \pi]$ with homogeneous Dirichlet boundary conditions. In figure 2.6 the relative error of the application of the method is illustrated. We see that the error follows indeed the predicted line of $\mathcal{O}(h^2)$. In figure 2.7 we see the dramatic increase in accuracy when k becomes larger. The predicted order of $\mathcal{O}(k^{-4})$ seems to be reached once k is sufficiently large.

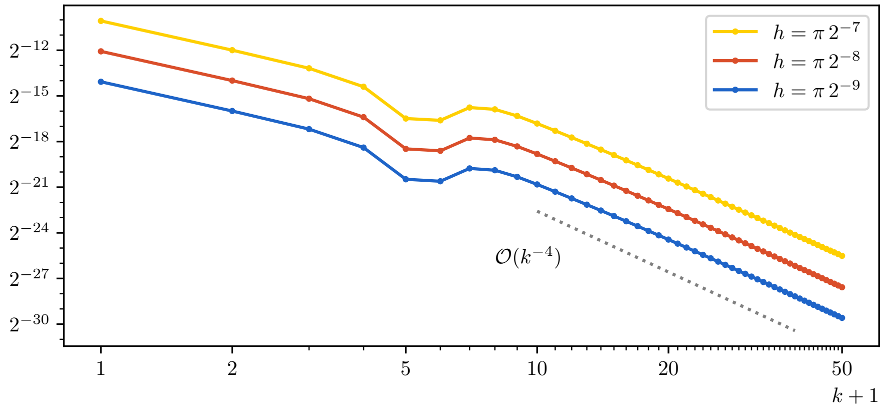


Figure 2.7: Relative error of the found eigenvalues of problem (2.7) by using the method of constant piecewise approximation of the potential on a uniform grid with step size h. The graphs are in function of the index of the eigenvalue. The yellow line used 128 subintervals, the blue line 512.

But still some nuances have to be made. In [78], some numerical examples are given for constant ($m = 0$), linear ($m = 1$) and quadratic ($m = 2$) approximations. But only for the constant approximations analytical solutions are used. For the higher order experiments, the true solution on a single subinterval is approximated by an at least fifteenth order Taylor series expansion. These Taylor series are only polynomial approximations of a possibly highly oscillatory function. This gives problems for higher lying eigenvalues.

2.2.2.2 Constant perturbation methods

In the years since the first article from Canosa and De Oliveira [19], the research into these kinds of methods has flourished. One of the first extensions considered not only piecewise constant approximations but also linear and quadratic approximations, as in [78]. For constant approximations, the exact solution is given by hyperbolic or trigonometric functions. For linear approximations, the Airy functions $\text{Ai}(x)$ and $\text{Bi}(x)$ are appropriate. As these are well-known special functions, software packages are available to evaluate them. But not unexpectedly, these packages are harder to find, and more difficult to use, than sine, cosine, and hyperbolic variants. For quadratic and higher order approximations, no closed form formula exists for the exact solutions.

To improve accuracy and computational efficiency, it is valuable to construct higher order methods, when developing numerical methods. As we are limited to an order of $\mathcal{O}(h^2)$ for constant approximations, or $\mathcal{O}(h^4)$ when using linear approximations, some alternative improvements were required. In the book [51], Ixaru dedicated chapter 3 to the development of the constant perturbation methods. The main idea is to not directly improve the approximation of $V(x)$, but to start with a piecewise constant approximation $\bar{V}(x)$, adding correction terms to capture the difference between the reference solution for $\bar{V}(x)$ and the true solution for $V(x)$.

Before we analyze the constant perturbation methods, let us take the time to formalize which mathematical problem we are solving. For now, we will only look at the differential equation

$$y''(\delta) = (\Delta V(\delta) + \bar{V} - \lambda)y(\delta) \tag{2.8}$$

as initial value problem starting from 0 with $\delta \in [0, h]$. Denote the homogeneous Neumann solution as $u(\delta)$, as such, $u(0) = 1$ and $u'(0) = 0$. And, we will write the homogeneous Dirichlet solution as $v(\delta)$, thus $v(0) = 0$ and $v'(0) = 1$. In (2.8), $\bar{V}$ is a constant (not piecewise) approximation of $V(x)$ on the current subinterval $[x_i, x_{i+1}]$ and $\Delta V(\delta) := V(x_i + \delta) - \bar{V}$. Being able to solve the initial value problem is sufficient to also solve the boundary value eigenproblem. For solving the latter we can employ a shooting procedure to the former, see section 2.2.3 for more details. The procedure we will develop here can then be applied to each of the subintervals in the constant piecewise approximation of $V(x)$.

Following [51, 45], let us denote a solution of (2.8) generally as $p(\delta)$, this thus represents either $u(\delta)$ or $v(\delta)$. Now we write the solution $p(\delta)$ as a perturbation series:

$$p(\delta) = p_0(\delta) + p_1(\delta) + p_2(\delta) + \cdots$$

In this expression, the first order term p_0 is the solution of the reference equation

$$p_0''(\delta) = (\bar{V} - \lambda)p_0(\delta) \tag{2.9}$$

with appropriate initial values. That is, $u_0(0) = 1$ and $u_0'(0) = 0$ for $p = u$, if $p = v$ the initial values are $v_0(0) = 0$ and $v_0'(0) = 1$. The perturbation corrections can now be recursively defined as the solution of

$$p_q''(\delta) = (\bar{V} - \lambda)p_q(\delta) + \Delta V(\delta)p_{q-1} \tag{2.10}$$

with initial conditions $p_q(0) = p'_q(0) = 0$. Notice that this definition implies that p indeed solves (2.8) with the appropriate initial values:

$$\begin{aligned} p &= p_0 + \sum_{q=1}^{\infty} p_q \\ p'' &= (\bar{V} - \lambda)p_0 + \sum_{q=1}^{\infty} \big((\bar{V} - \lambda)p_q + \Delta V(\delta)p_{q-1}\big) \\ &= (\bar{V} - \lambda)\sum_{q=0}^{\infty} p_q + \Delta V(\delta)\sum_{q=0}^{\infty} p_q \\ &= (\Delta V(\delta) + \bar{V} - \lambda)p. \end{aligned}$$

To symbolically compute the expression of p_q Ixaru has introduced some auxiliary functions[4], based upon the analytical solution of equation (2.9).

Definition 2.7 (Ixaru 1984)**.** *The family of η-functions $\eta_m : \mathbb{R} \to \mathbb{R}$ for $m \in \{-1, 0, 1, \dots\}$ is defined recursively as:*

$$\begin{aligned} \eta_{-1}(Z) &= \cosh\Big(\sqrt{Z}\Big) \\ \eta_0(Z) &= \frac{\sinh\Big(\sqrt{Z}\Big)}{\sqrt{Z}} \\ \eta_m(Z) &= \frac{\eta_{m-2}(Z) - (2m-1)\eta_{m-1}(Z)}{Z}. \end{aligned}$$

When $Z < 0$ the definitions of η_{-1} and η_0 should be read as calculations in $\mathbb{C}$. But, notice that the resulting values will always be real. Furthermore, if $Z = 0$, one should take the limit of η_m to zero, this yields[5] $\eta_m(0) = \frac{1}{(2m+1)!!}$.

Much has already been written about these instrumental functions, for example in Ixaru's book [51]. To get a feeling about these functions, we provide figure 2.8. In definition 2.7, we have chosen to simplify the notation by allowing computations in $\mathbb{C}$. But when implementing these formulae, it is much more efficient to keep all calculations in $\mathbb{R}$. For $Z < 0$, we implement $\eta_{-1}(Z) = \cos\big(\sqrt{-Z}\big)$ and $\eta_0(Z) = \sin\big(\sqrt{-Z}\big)/\sqrt{-Z}$.

[4]What we will call η_{-1}, Ixaru has named ξ. The notation of the recursive definition becomes a little easier when the η_m and ξ names are unified.

[5]In this expression $n!!$ is the double factorial: $n!! := n \cdot (n-2) \cdot (n-4) \cdot ...$, with only strictly positive integers as factors. For our purposes we define $0!! = (-1)!! = 1$.

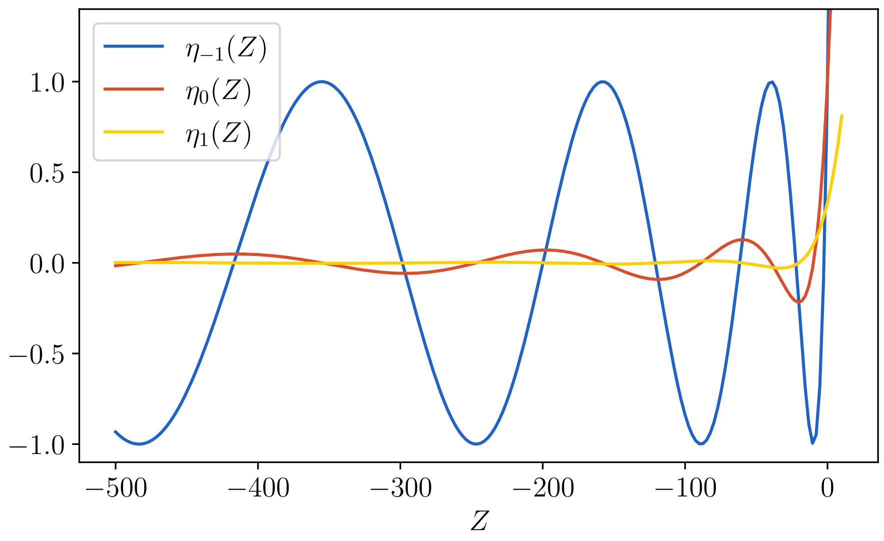


Figure 2.8: A graph of the first three η-functions.

Another interesting property of these η-functions is their series expansions:

$$\eta_{-1}(Z) = \sum_{q=0}^{\infty} \frac{Z^q}{(2q)!}$$

$$\eta_m(Z) = 2^m \sum_{q=0}^{\infty} \frac{(q+m)!Z^q}{q!(2q+2m+1)!} \qquad \text{if } m \geq 0.$$

When $|Z|$ is small, the recursion from the definition becomes numerically unstable. It is more accurate to use this series expansion on η_m and η_{m+1}, for m sufficiently large, and work our way back to $m = 0$ and $m = -1$ with the inverted recursion

$$\eta_{m-1}(Z) = Z\eta_{m+1}(Z) + (2m+1)\eta_m(Z).$$

Other interesting properties of these special functions are already studied. Here we provide some results, without proofs.

Theorem 2.8 (Ixaru 1984)**.** *For the family of functions defined in definition 2.7, the following properties hold for $m \in \{-1, 0, 1, 2, \dots\}$ and $Z \in \mathbb{R}$.*

- *Asymptotic behavior for* $|Z| \to \infty$*:*

$$\eta_m(Z) \approx \begin{cases} \dfrac{\eta_{-1}(Z)}{Z^{(m+1)/2}} & \text{if } m \text{ is odd,} \\ \dfrac{\eta_0(Z)}{Z^{m/2}} & \text{if } m \text{ is even.} \end{cases}$$

- *Differentiation with respect to* Z*:*

$$\eta'_m(Z) = \frac{1}{2}\eta_{m+1}(Z).$$

- *Differentiation with respect to* δ *if* $Z = F\delta^2$*:*

$$\begin{aligned} \frac{\partial \eta_{-1}(Z)}{\partial \delta} &= \frac{Z}{\delta}\eta_0(Z) \\ \frac{\partial \delta^{2m+1}\eta_m(Z)}{\partial \delta} &= \delta^{2m}\eta_{m-1}(Z) \qquad \text{if } m \geq 0. \end{aligned}$$

Later on, we will use these η-functions as part of terms in infinite sums. Convergence of these sums is aided by the fact that $\eta_m(Z) \to 0$ for $m \to +\infty$ for any fixed Z.

Surprisingly, during the writing of this thesis my copromotor mentioned he recognized the series expansion of the η-functions to be those of a specific generalized hypergeometric function. In the literature we have not found this relation to hypergeometric functions. As such, we believe the following theorem to be new.

Theorem 2.9. *For all* $m \in \{-1, 0, 1, \dots\}$ *is* $\eta_m(Z)$ *a special case of generalized hypergeometric function*[6]*:*

$$\eta_m(Z) = 2^{m+1}(m+1)!\,{}_0F_1\left(\begin{matrix} - \\ m+\frac{3}{2} \end{matrix}; \frac{Z}{4}\right) \tag{2.11}$$

Consequently if $m \geq 0$*, this can also be expressed with modified Bessel functions*[7] *of the first kind* I_α*:*

$$\eta_m(Z) = \sqrt{2\pi}\,(2m+1)!\,Z^{-\frac{m}{2}-\frac{1}{4}}\,I_{m+\frac{1}{2}}\left(\sqrt{Z}\right).$$

[6]Chapter 16 of [27] is dedicated to these functions, with many known properties.

[7]The relation to Bessel functions has already been documented in [52].

Proof. Following chapter 16 of [27], the hypergeometric function ${}_0F_1$ is defined as[8]:

$$
{}_0F_1\left(\begin{matrix} - \\ m+\frac{3}{2} \end{matrix};\frac{Z}{4}\right) = \sum_{k=0}^{\infty} \frac{Z^k}{4^k\left(m+\frac{3}{2}\right)_k k!}. \tag{2.12}
$$

Expanding Pochhammer's symbol yields:

$$
\begin{aligned}
\left(m+\frac{3}{2}\right)_k &= \left(m+\frac{3}{2}\right)\left(m+\frac{5}{2}\right)\cdots\left(m+\frac{2k+1}{2}\right) \\
&= 2^{-k}\left(2m+3\right)\left(2m+5\right)\cdots\left(2m+2k+1\right) \\
&= 2^{1-2k}\frac{(2m+2k+1)!(m+1)!}{(m+k)!}.
\end{aligned}
$$

Inserting this into (2.12) allows us to write

$$
\begin{aligned}
{}_0F_1\left(\begin{matrix} - \\ m+\frac{3}{2} \end{matrix};\frac{Z}{4}\right) &= \frac{1}{2(m+1)!}\sum_{k=0}^{\infty}\frac{(m+k)!Z^k}{k!(2m+2k+1)!} \\
&= \frac{1}{2^{m+1}(m+1)!}\eta_m(Z).
\end{aligned}
$$

For the second expression, equation 10.39.9 of [27] gives:

$$
I_\alpha(x) = \left(\frac{x}{2}\right)^{\alpha}\frac{1}{\Gamma(\alpha+1)}\,{}_0F_1\left(\begin{matrix} - \\ \alpha+1 \end{matrix};\frac{x^2}{4}\right).
$$

Substituting this into (2.11) with $\alpha = m+\frac{1}{2}$ and $x=\sqrt{Z}$ yields:

$$
\eta_m(Z) = 2^{2m+\frac{3}{2}}\,(m+1)!\,Z^{-\frac{m}{2}-\frac{1}{4}}\,\Gamma\left(m+\frac{3}{2}\right)\,I_{m+\frac{1}{2}}\left(\sqrt{Z}\right).
$$

Using equation 5.5.5 in [27]:

$$
\sqrt{\pi}\Gamma(2z) = 2^{2z-1}\Gamma(z)\Gamma\left(z+\frac{1}{2}\right)
$$

with $z=m+1>0$ we get the required expression. □

Although this relation between the η-functions and other special functions has never (as far as we can tell) been published before, it only has limited added

[8]Pochhammer's symbol is used in this expression: $(\alpha)_n := \alpha(\alpha+1)(\alpha+2)\cdots(\alpha+n-1)$.

value. The developed theory of the last forty years is still equally valid and valuable.

With the theory about the family of η-functions in hand, we are able to solve equations (2.9) and (2.10) symbolically. The following theorem from [51] captures these symbolic calculations for $p_q(\delta)$. As these formulae have been instrumental to our work (especially for section 2.3), we will provide a proof.

Theorem 2.10 (Ixaru 1984)**.** *Let $p(\delta)$ be a general solution of*

$$p''(\delta) = \left(\Delta V(\delta) + \bar{V} - \lambda\right) p(\delta)$$

over the interval $[0, h]$. In this expression $V = \bar{V} + \Delta V(\delta)$ is a given potential function, with $\bar{V}$ a constant approximation and $\Delta V(\delta)$ the residual term. The value λ is fixed. We denote the solution $p(\delta)$ with initial conditions $p(0) = 1$ and $p'(0) = 0$ as $u(\delta)$, and the solution with initial conditions $p(0) = 0$ and $p'(0) = 1$ will be denoted with $v(\delta)$. These two functions $u(\delta)$ and $v(\delta)$ are called the propagators*. For ease of notation we will write $Z(\delta) = \left(\bar{V} - \lambda\right)\delta^2$. Let $p(\delta) = p_0(\delta) + p_1(\delta) + p_2(\delta) + \ldots$ with:*

$$\begin{aligned} p_0''(\delta) &= (\bar{V} - \lambda)p_0(\delta) \\ p_q''(\delta) &= (\bar{V} - \lambda)p_q(\delta) + \Delta V(\delta)p_{q-1}(\delta) \quad \text{for } q > 0. \end{aligned}$$

The function p_0 inherits the initial conditions of p. If $q > 0$, p_q has as initial conditions $p_q(0) = p_q'(0) = 0$.

If $\Delta V(\delta)$ is a polynomial in δ then each term p_q is given by:

$$p_q(\delta) = \sum_{i=-1}^{\infty} \delta^{2i+1} C_i^{(q)}(\delta) \eta_i(Z(\delta)) \tag{2.13}$$

with derivative:

$$\begin{aligned} p_q'(\delta) = {} & \frac{C_{-1}^{(q)}(\delta)}{\delta^2} \left(\eta_{-1}(Z(\delta)) + Z(\delta)\eta_0(Z(\delta))\right) \\ & + \sum_{i=-1}^{\infty} \left(C_i^{(q)\prime}(\delta) + \delta C_{i+1}^{(q)}(\delta)\right) \delta^{2i+1} \eta_i(Z(\delta)). \end{aligned} \tag{2.14}$$

The functions $C_i^{(q)}(\delta)$ are polynomials and satisfy the following recursive rela-

tion:

$$
\begin{aligned}
C_i^{(q)}(\delta) &= \frac{\delta^{-i}}{2}\int_0^\delta \varepsilon^{i-1}\left(C_{i-1}^{(q-1)}(\varepsilon)\Delta V(\varepsilon) - C_{i-1}^{(q)\prime\prime}(\varepsilon)\right)\mathrm{d}\varepsilon \\
C_i^{(0)}(\delta) &= \begin{cases} \delta & \textit{if } p = u \textit{ and } i = -1 \\ 1 & \textit{if } p = v \textit{ and } i = 0 \\ 0 & \textit{otherwise} \end{cases} \\
C_{-1}^{(q)}(\delta) &= 0 \quad \textit{if } q > 0.
\end{aligned} \tag{2.15}
$$

Proof. As these formulae are recursively defined, a proof by induction is most natural. For $q = 0$ the exact solutions can be calculated. If $\bar{V} \geq \lambda$, then $u_0(\delta) = \cosh\left(\sqrt{Z(\delta)}\right)$ and $v_0(\delta) = \sinh\left(\sqrt{Z(\delta)}\right)/\sqrt{\bar{V} - \lambda}$. If $\bar{V} < \lambda$ on the other hand, then $u_0(\delta) = \cos\left(\sqrt{-Z(\delta)}\right)$ and $v_0(\delta) = \sin\left(\sqrt{-Z(\delta)}\right)/\sqrt{\lambda - \bar{V}}$. By using the η-functions both these cases can be summarized as:

$$u_0 = \eta_{-1}(Z(\delta)) \text{ and } v_0 = \delta\eta_0(Z(\delta)).$$

Substituting the values of $C_i^{(0)}$ into (2.13) for $q = 0$, yields exactly the same expressions. This proves the induction basis.

Assume, as induction hypothesis, that the theorem holds for any value for q less than Q. First we prove that for each i, $C_i^{(Q)}$ is a polynomial. For this we apply induction with respect to i. For $i = -1$, $C_{-1}^{(Q)} = 0$ is a polynomial. Since $\Delta V(\delta)$ is assumed to be polynomial, we notice that for any other i

$$C_{i-1}^{(Q-1)}(\varepsilon)\Delta V(\varepsilon) - C_{i-1}^{(Q)\prime\prime}(\varepsilon)$$

is a polynomial, as a consequence of the induction hypothesis in Q, and the induction hypothesis in i. This implies that the integrand in the definition of $C_i^{(Q)}(\delta)$ is polynomial with no terms of degree less than $i - 1$. This means that the integral in that definition will be divisible by δ^i, which proves that all $C_i^{(Q)}$ are polynomials.

Let us then prove that p_Q satisfies the initial conditions $p_Q(0) = p'_Q(0) = 0$. That $p_Q(0) = 0$ can be seen in (2.13), using the facts that $C_{-1}^{(Q)} = 0$ for $Q > 0$ and that $C_i^{(Q)}$ are polynomials. Using the properties from theorem 2.8, we can compute $p'_Q(\delta)$ to be as in (2.14). That this expression is zero for $\delta = 0$ is less apparent. First, notice that most terms are zero because $C_i^{(Q)}$ are polynomials and $C_{-1}^{(Q)} = 0$ for $Q > 0$. The only term for which this is not clear is with $i = -1$: $C_0^{(Q)}(0)\eta_0(Z(0))$. For this term, we remark that the polynomial $C_{-1}^{(q)}(\delta)$ never contains a constant term. For most values of q, it is zero, and for $q = 0$ it can only be zero or δ. This means, by the recursive construction, that also $C_i^{(q)}(\delta)$ does not contain a constant term. This in turn means $C_i^{(q)}(0) = 0$, and thus $p'_Q(0) = 0$, for $Q > 0$. The last thing that we still have to prove, is that this p_Q indeed solves its defining equation.

Next we prove that $p_Q(\delta)$ is a solution of

$$p''_Q(\delta) = (\bar{V} - \lambda)p_Q(\delta) + \Delta V(\delta)p_{Q-1}(\delta).$$

For this, we compute p''_Q. For the sake of brevity, we will omit the argument δ from most functions, and remember that all derivatives are with respect to δ. But first, we know from the definition of C_i^q that

$$C_i^{(q)\prime} = -i\delta^{-1}C_i^{(q)} + \frac{\delta^{-1}}{2}\left(C_{i-1}^{(q-1)}\Delta V - C_{i-1}^{(q)\prime\prime}\right),$$

but also that

$$\begin{aligned} \frac{\mathrm{d}\delta^{2i+1}\eta_i(Z)}{\mathrm{d}\delta} &= \delta^{2i}\eta_{i-1} \\ \text{and } \frac{\mathrm{d}^2\delta^{2i+1}\eta_i(Z)}{\mathrm{d}\delta^2} &= \delta^{2i-1}\left(2i\eta_{i-1}(Z) + Z\eta_i(Z)\right). \end{aligned}$$

These expressions, together with $C_{-1}^{(Q)} = 0$, now can be used to simplify p''_Q:

$$\begin{aligned} p_Q =& \sum_{i=0}^{+\infty} C_i^{(Q)}\delta^{2i+1}\eta_i(Z) \\ p''_Q =& \sum_{i=0}^{+\infty} C_i^{(Q)}\delta^{2i-1}\left(2i\eta_{i-1}(Z) + Z\eta_i(Z)\right) \\ &+ \sum_{i=-1}^{+\infty} 2C_{i+1}^{(Q)\prime}\delta^{2i+2}\eta_i(Z) + \sum_{i=0}^{+\infty} C_i^{(Q)\prime\prime}\delta^{2i+1}\eta_i(Z) \\ =& \sum_{i=0}^{+\infty} C_i^{(Q)}\delta^{2i-1}\left(2i\eta_{i-1}(Z) + Z\eta_i(Z)\right) \\ &- \sum_{i=-1}^{+\infty} 2(i+1)C_{i+1}^{(Q)}\delta^{2i+1}\eta_i(Z) + p_{Q-1}\Delta V \end{aligned}$$

$$= \frac{Z}{\delta^2} p_Q + p_{Q-1} \Delta V.$$

Due to the definition of $Z = \delta^2(\bar{V} - \lambda)$, this expresses exactly that p_Q is a solution to $p_Q'' = (\bar{V} - \lambda)p_Q + \delta V p_{Q-1}$. □

At first glance, it may not be clear why theorem 2.10 is that important. Still, these rather tedious computations and relatively complicated recursive relation allows us to analytically construct formulae of any order. To calculate such a formula, we choose the number of perturbation corrections Q and the number of η-functions to use N and compute the values of $C_i^{(q)}$ with the recursive relations (2.15) for $q \leq Q$ and $i \leq N$. If ΔV is assumed to be a polynomial in δ, then $C_i^{(q)}$ will be as well. The solution $p(\delta)$ will now have the following form with finite sums:

$$p(\delta) = \sum_{q=0}^{Q} p_q(\delta) \text{ with } p_q(\delta) = \sum_{i=-1}^{N} C_i^{(q)}(\delta)\delta^{2i+1}\eta_i(Z).$$

However, these formulae only can be calculated if $V(x)$ is a polynomial. For this, the very first step in the CP-algorithm is to approximate $V(x)$ on each sector by a polynomial $V^N(x)$ of sufficient high degree N. One possible polynomial representation is by expressing it in terms of the orthogonal family of shifted Legendre polynomials $\widetilde{P}_n(x)$. This family of polynomials satisfies[9]

$$\int_0^1 \widetilde{P}_m(x)\widetilde{P}_n(x)\mathrm{d}x = \frac{\delta_{mn}}{2n+1} \qquad \text{for } m \neq n,$$

and are increasing in degree, together with $\widetilde{P}_n(x) = 1$ for any n. As a reference we provide the first four:

$$\begin{aligned} \widetilde{P}_0(x) &= 1, & \widetilde{P}_2(x) &= 6x^2 - 6x + 1, \\ \widetilde{P}_1(x) &= 2x - 1, & \widetilde{P}_3(x) &= 20x^3 - 30x^2 + 12x - 1. \end{aligned}$$

Applying a least squares approximation of $V(\delta)$ with $N + 1$ shifted Legendre polynomials yields

$$V(\delta) \approx \sum_{n=0}^{N} V_n h^n \widetilde{P}_n\left(\frac{\delta}{h}\right)$$

[9] Here δ_{mn} is the Kronecker δ, defined as $\delta_{mn} = 1$ if $m = n$ else $\delta_{mn} = 0$.

where

$$V_n = \frac{2n+1}{h^{n+1}} \int_0^h V(\delta)\widetilde{P}_n\left(\frac{\delta}{h}\right) d\delta, \qquad \text{with } n \in \{0, 1, \ldots, N\}. \tag{2.16}$$

To implement this, the numerical value of the integral can be computed with Gauss quadrature rules of sufficiently high degree.

Let us denote the constant perturbation method with given choices for the degree N and the number of correction terms Q as CPM$[N, Q]$, then we get the following result [49].

Theorem 2.11. *If CPM$[N, Q]$ is applied to propagate the solution on an equidistant partition with mesh size h then*

- *for small values of E (i.e. if Z remains sufficiently small), the error in the mesh points is bounded by $C_N h^{2N+2}$ (for some constant C_N) provided that $Q \geq \left\lfloor \frac{2N}{3} \right\rfloor + 1$ if $N \geq 1$ and $Q = 0$ if $N = 0$.*
- *if E is such that $Z(h) \ll 0$ in all intervals, the error in the mesh points is bounded by $C_N^* h^N / \sqrt{E}$ (for some constant C_N^*) provided that $Q \geq 1$ if $N \geq 1$ and $Q = 0$ if $N = 0$.*

Finally, one can simplify the expressions for the coefficients of the perturbation terms $C_i^{(q)}$ by taking into account the orders P_0 (for small Z) and P_{ass} (for negative Z with $|Z| \gg 0$) that can be attained by the CPM$[N, Q]$ method. Some terms will contribute to the solution in higher orders of h, and so they do not need to be computed. This pruning then finally results in a method which was generically denoted as CPM$\{P_0, P_{ass}\}$. Later in theorem 2.13, we will prove that $P_{ass} = P_0$, thus we will denote our method as CPM$\{P_0\}$.

The original SLCPM12-code implemented the CPM$\{12, 10\}$ algorithm. In the original `Matslise` package, a user could choose between several algorithms: CPM$\{12, 10\}$, CPM$\{14, 12\}$, CPM$\{16, 14\}$, CPM$\{18, 16\}$. In `Matslise 2.0` only CPM$\{18, 16\}$ and CPM$\{16, 14\}$ are implemented. The CPM$\{16, 14\}$ method is used for propagation and the difference between the two methods is used for error estimation.

The CP methods have also been successfully applied to other types of problems, such as the so-called coupled channel Schrödinger-equations problem. Research in the past lead to the `Fortran` code `LILIX` [46] and the MATLAB code `MatSCS` [63]. These programs made it then possible to construct CP-based methods for solving the two-dimensional Schrödinger problem [47] and the time-dependent one-dimensional Schrödinger problem [62].

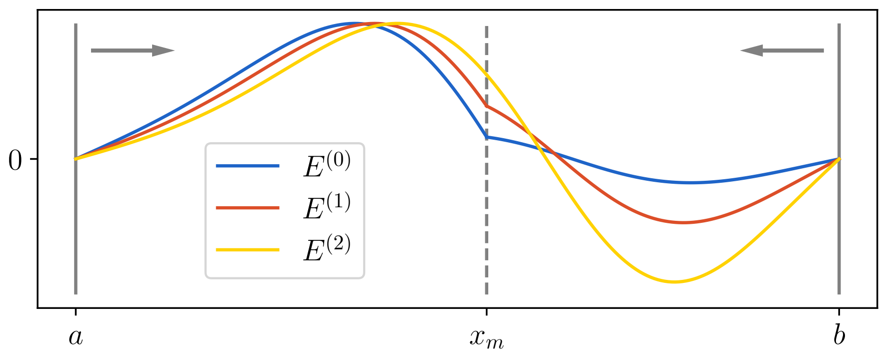


Figure 2.9: Multiple shooting procedures are applied to a Sturm–Liouville equation with homogeneous Dirichlet boundary conditions. $E^{(0)}$ is a possible first eigenvalue guess, $E^{(1)}$ and $E^{(2)}$ are consecutive adjustments. Notice that for $E^{(2)}$ a differentiable function in the matching point x_m is found.

In chapters 3 and 4 we will develop methods for the two-dimensional time-independent Schrödinger equation. These new methods are only possible, thanks to the accurate and efficient CP-methods for the one-dimensional problem.

2.2.3 Shooting with Prüfer's transformation

We now return to the ideas of Canosa and De Oliveira [19]. The first thing they did to find analytical solutions of the piecewise constant approximated problem was fix the eigenvalue λ. In the developments following these ideas λ was always assumed to be a constant. But in reality, this λ is unknown.

In [19] the eigenvalue differential equation was translated into the question: for which values of λ does a given linear system of equations has non-zero solutions? For the CP-methods no such translation is available. In [51] Ixaru applies a shooting procedure to solve boundary value problems using constant perturbation techniques.

The main idea of shooting is to 'guess' a value $\lambda = E$. With this guess, the forward initial value problem is solved from the left side of the domain up to a point x_m, and also, a backward initial value problem is solved from the right side to the same point x_m. If the forward solution and the backward solution emit a global continuously differentiable solution, then $\lambda = E$ is a valid eigenvalue. If this is not the case, E is adjusted, and the procedure is repeated. In figure 2.9 this is demonstrated schematically.

More concretely, to apply this technique to the Schrödinger equation (2.6) we again assume E is fixed and convert it to a forward (left) initial value problem:

$$-y_{\mathrm{L}}''(x) + V(x)y_{\mathrm{L}}(x) = Ey_{\mathrm{L}}(x),$$

with the starting condition

$$y_{\mathrm{L}}(a) = \beta_a \text{ and } y_{\mathrm{L}}'(a) = -\alpha_a.$$

Analogously, we define a backward (right) initial value problem:

$$-y_{\mathrm{R}}''(x) + V(x)y_{\mathrm{R}}(x) = Ey_{\mathrm{R}}(x),$$

with the starting condition

$$y_{\mathrm{R}}(b) = \beta_b \text{ and } y_{\mathrm{R}}'(b) = -\alpha_b.$$

Now, the domain $[x_{\min}, x_{\max}]$ is partitioned in K intervals. The i^{th} interval can be written as $I_i = [x_{i-1}, x_i]$, with $a = x_0, x_1, \dots, x_{K-1}, x_K = b$ the grid points. One of these grid points can now be chosen as the matching point x_m. Note that numerically it is a little easier if this matching point is not in the exact middle of the domain. If the potential happens to be zero, some eigenfunctions will have roots in this midpoint. And, if the matching points coincide with a root of an eigenfunction, care has to be taken to determine the scaling of the left and right solutions.

Generically we can write such an interval as $I = [X, X+h]$. On I, the solution y can now be expressed as:

$$\begin{pmatrix} y(X+\delta) \\ y'(X+\delta) \end{pmatrix} = \begin{pmatrix} u(\delta) & v(\delta) \\ u'(\delta) & v'(\delta) \end{pmatrix} \begin{pmatrix} y(X) \\ y'(X) \end{pmatrix}, \qquad 0 \leq \delta \leq h\,. \tag{2.17}$$

The functions $u(\delta)$ and $v(\delta)$ are the propagators from theorem 2.10 with the initial conditions $u(0) = 1, u'(0) = 0$ and $v(0) = 0, v'(0) = 1$. This matrix is called the one-step propagation matrix:

$$\mathbf{P}(\delta) = \begin{pmatrix} u(\delta) & v(\delta) \\ u'(\delta) & v'(\delta) \end{pmatrix}. \tag{2.18}$$

To execute backward propagation, its inverse can be determined. For $\delta = 0$, due to the initial conditions, the determinant of $\mathbf{P}(0)$ is one. To determine the determinant throughout the domain we compute its derivative with respect to δ.

$$\begin{aligned}\frac{\mathrm{d}\det\mathbf{P}}{\mathrm{d}\delta} &= (u(\delta)v'(\delta) - u'(\delta)v(\delta))' \\ &= u''v + u'v' - u''v - u'v' \\ &= (V(X+\delta) - E)uv - u(V(X+\delta)-E)v \\ &= 0.\end{aligned}$$

This implies that $\det\mathbf{P}(\delta) = 1$, for each $0 \le \delta \le h$, which allows us to write down $\mathbf{P}^{-1}(\delta)$:

$$\mathbf{P}^{-1}(\delta) = \begin{pmatrix} v'(\delta) & -v(\delta) \\ -u'(\delta) & u(\delta)\end{pmatrix}.$$

This matrix can now be used for the backward initial value problem.

If we compute the matrix $\mathbf{P}(h)$ on each interval left of x_m and $\mathbf{P}^{-1}(h)$ on each interval to the right of x_m, then the solution of $y_\mathrm{L}(x_m)$, $y'_\mathrm{L}(x_m)$ and $y_\mathrm{R}(x_m)$, $y'_\mathrm{R}(x_m)$ can be determined.

To know if the guess for E was correct, we need to find a scaling of y_L and y_R such that $y_\mathrm{L}(x_m) = y_\mathrm{R}(x_m)$ and $y'_\mathrm{L}(x_m) = y'_\mathrm{R}(x_m)$. This can be expressed via a matching error function:

$$e(E) = y_\mathrm{L}(x_m)y'_\mathrm{R}(x_m) - y'_\mathrm{L}(x_m)y_\mathrm{R}(x_m). \tag{2.19}$$

The goal now is to find all zero-values of this function. For this, we use the Newton–Raphson method. Computing the derivative $\frac{\mathrm{d}e(E)}{\mathrm{d}E}$ is far from trivial. But in theorem 2.10 we see the dependence on E of each $u(h)$ and $v(h)$ through $Z(\delta) = \eta_i\left((V(x) - E)\delta^2\right)$. In theorem 2.8, we provided the derivatives needed. These can be used to also compute the derivative of $e(E)$ with respect to E. This allows for fast convergence of E to true eigenvalues λ, for which $e(\lambda) = 0$.

As an example, we consider one of Paine's problems from [76]. On the domain $[0, \pi]$, we will solve the Schrödinger equation

$$-y''(x) + e^x y(x) = \lambda y(x), \tag{2.20}$$

with homogeneous Dirichlet boundary conditions. In [76], the eigenvalues are reported with five decimal digits; our new implementation `Matslise 3.0` (see section 2.3) agrees for all provided digits. The first twelve eigenvalues with 24 fractional digits are reported here.

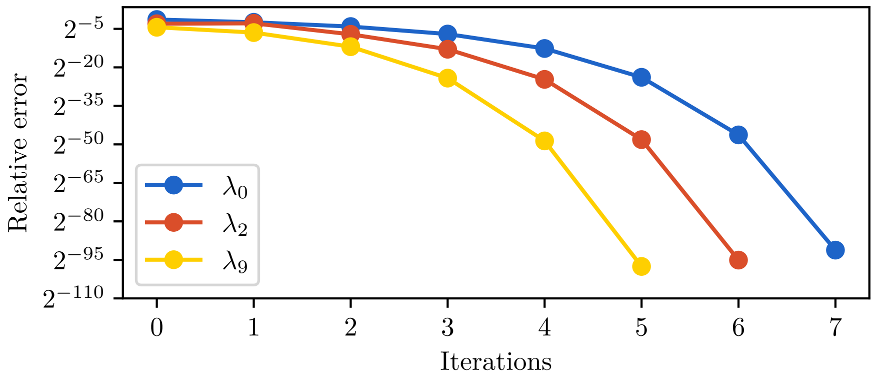


Figure 2.10: Using shooting, the relative error of estimates for different eigenvalues of (2.20) is plotted with respect to the number of iterations of the Newton–Raphson procedure.

4.896 669 379 967 691 490 474 902	56.181 594 022 847 580 296 684 220
10.045 189 893 253 741 994 613 494	71.152 997 537 057 822 221 278 311
16.019 267 250 492 220 805 222 091	88.132 119 191 546 181 466 153 432
23.266 270 940 022 341 911 164 117	107.116 676 138 267 797 713 195 715
32.263 707 045 804 466 735 963 540	128.105 021 273 333 317 509 823 750
43.220 019 640 534 137 263 517 478	151.096 043 745 596 921 632 322 296

In figure 2.10, the convergence of the Newton–Raphson procedure is visualized. We have started with the guesses 3, 18 and 102 to find the respective eigenvalues $\lambda_0 \approx 4.896669$, $\lambda_2 \approx 16.019267$ and $\lambda_9 \approx 107.116676$. From the graphs, we see that the convergence is definitely faster than linear. After at most merely six iteration steps, we have reached the maximal precision of the `double` datatype ($10^{-16} \approx 2^{-53}$) and with one more iteration we reach close to the maximal precision of the `float128` datatype ($10^{-34} \approx 2^{-112}$). In a table, it becomes clear that the convergence has a quadratic behavior. Here the relative errors are tabulated from the second iteration onward.

	2	3	4	5	6	7
λ_0	$6 \cdot 10^{-2}$	$8 \cdot 10^{-3}$	$2 \cdot 10^{-4}$	$6 \cdot 10^{-8}$	$1 \cdot 10^{-14}$	$4 \cdot 10^{-28}$
λ_2	$7 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	$4 \cdot 10^{-8}$	$3 \cdot 10^{-15}$	$2 \cdot 10^{-29}$	
λ_9	$3 \cdot 10^{-4}$	$5 \cdot 10^{-8}$	$2 \cdot 10^{-15}$	$4 \cdot 10^{-30}$		

Note that this example has been performed in quadruple floating point precision.

This is only possible because of our improvements from section 2.6.1.2.

There still remains one problem: how do we ensure that we have found *all* eigenvalues?

For this we remember theorem 2.4. To know the index of the eigenvalue we can 'simply' count the number of roots of the corresponding eigenfunction. In practice this is quite hard, especially for highly oscillatory eigenfunctions. For these, it is very easy to miss a few roots.

To combat this issue and to provide a reliable way to even count the roots of highly oscillatory eigenfunctions, Prüfer's scaled transformation can be used [79].

Theorem 2.12 (Prüfer 1926)**.** *For a fixed value E define $\theta(x)$ and $\rho(x)$ as the continuous functions which satisfy*

$$y(x) = \frac{\rho(x)}{\sqrt{S(x)}}\sin(\theta(x)) \quad \textit{and} \quad p(x)y'(x) = \sqrt{S(x)}\rho(x)\cos(\theta(x)) \tag{2.21}$$

with $\tan\theta(a) = -\frac{S(a)\beta_a}{\alpha_a}$, $\theta(a) \in [0,\pi[$ *and $S(x)$ a strictly positive scaling function to ensure numerical stability. Here $y(x)$ is a solution of the Sturm–Liouville equation*

$$-\frac{\mathrm{d}p(x)y'(x)}{\mathrm{d}x} + q(x)y(x) = Ew(x)y(x)$$

on the interval $[a,b]$ with boundary conditions $\alpha_a y(a) + \beta_a p(a)y(a) = 0$ and $\alpha_b y(b) + \beta_b p(b)y(b) = 0$.

Define k to be

$$k = \frac{1}{\pi}\left(\theta(b) - \theta_b\right),$$

with $\tan\theta_b = -\frac{S(b)\beta_b}{\alpha_b}$ *and $\theta_b \in]0,\pi]$. If k is an integer, then $y(x)$ has k roots in the interior of the domain and satisfies the Sturm–Liouville problem. In other words, $\lambda_k = E$ is the k^{th} eigenvalue.*

Imposing that $\theta(x)$ and $\rho(x)$ are continuous functions, ensures that (2.21) uniquely defines these functions. $\theta(x)$ can be calculated as

$$\theta(x) = \operatorname{atan}\frac{S(x)y(x)}{p(x)y'(x)} + j\pi, \tag{2.22}$$

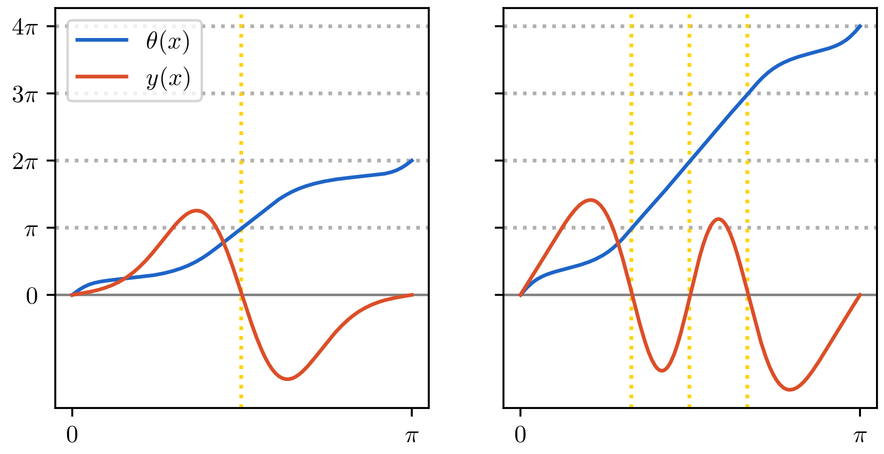


Figure 2.11: Eigenfunction $y_1(x)$ and $y_3(x)$ for the Mathieu problem with $q = 10$ from section 2.5.3. The scaled Prüfer angle $\theta(x)$ is plotted as well.

with j integer. If $\theta(x)$ has to be continuous we can uniquely determine which value should be taken for j. To compute $\theta(x)$ efficiently and reliably, we substitute (2.21) into the Sturm–Liouville equation. This yields:

$$\theta' = \frac{S}{p}\cos^2(\theta) + \frac{Ew - q}{S}\sin^2\theta + \frac{S'}{S}\sin(\theta)\cos(\theta). \tag{2.23}$$

In the implementation while propagating solutions with equation (2.17), we also solve (2.23). Luckily, this second ordinary differential equation does not need to be solved accurately, because the very accurate value of y can be used to compute $\theta(x)$ with equation (2.22) up to a multiple of π. The value computed with (2.23) can then be rounded to the nearest value found with (2.22).

Much has been written about the best choices for the scaling function $S(x)$. In our implementation we follow `Matslise` as written in [67].

As an illustration of this scaled Prüfer transformation with equation (2.23) we present figure 2.11. Here two eigenfunctions from the Mathieu problem from section 2.5.3 with $q = 10$ are plotted. The scaled Prüfer angles can be seen in blue. Notice that the Prüfer angle crosses a multiple of π if and only if the corresponding x value is a zero of $y(x)$. The final value of $\theta(b)/\pi - 1$ on the rightmost point of the domain tells us the index of the relevant eigenvalue.

2.3 Matslise 3.0

In the previous section, we have studied the background of constant perturbation methods. From here on we develop our additions to these well-established methods. Some of these results were published in [9], particularly this section and some numerical experiments from sections 2.5.3, 2.5.4 and 2.5.5. The other numerical experiments from 2.5 take a closer look at our improvements to the algorithm since [9]. In section 2.4, we present how the constant perturbation method can be used to solve periodic boundary conditions as well. And, later on in section 2.6 we take a deep dive into how our new efficient implementation is built.

In 2005, the first version of `Matslise` [64, 66], a MATLAB package for solving Sturm–Liouville problems (SLP), was published. This program was a modern take on the successful constant perturbation (CP) method based code SLCPM12 [50]. It was the first implementation of CP methods in the user-friendly, and then modern, numerical computing environment MATLAB. Up to that point most, if not all, of these programs for solving Sturm–Liouville problems (SLCPM12, SLEIGN, SLEDGE, …) [50, 10, 28] were written in `Fortran`.

`Matslise` provided a graphical user interface that made it easy for all researchers to effectively solve the Sturm–Liouville equation without any knowledge of a particular programming language. Before its release, if one wanted to solve a particular SLP, sufficient knowledge of `Fortran` was needed to implement the problem at hand.

In 2016 a new version, `Matslise 2.0` [68], was released. In this whole new version of `Matslise`, generalized CP methods for SLP were implemented. These new methods enabled one to solve the SLP without explicitly transforming the equation to the Liouville normal form. Avoiding this normalization also made it possible to solve new types of Sturm–Liouville problems. As of this writing, `Matslise 2.0` has been downloaded over a thousand times, according to SourceForge. This number proves the need of researchers to solve the Sturm–Liouville problem.

As the original `Matslise` modernized the implementation of SLCPM12, the need to remodernize this proven package has become apparent in our attempt to use the `Matslise` routines to solve the multidimensional Schrödinger equation, following the ideas of Ixaru in [47]. This approach requires the fast and accurate computation of both the eigenvalues and the corresponding eigenfunctions of several 1D Schrödinger-problems. The `Matslise 2.0` version however mainly focuses on the fast and accurate computation of eigenvalues and the graphical representation of the eigenfunction. In `Matslise 2.0` the eigenfunctions can be computed quite accurately and fast, but only in a limited set of points (depending on the partition of the integration interval). The accurate computation of the eigenfunction over the whole integration interval however is time-consuming in the implementation of `Matslise 2.0`.

In fact, this illustrates that `Matslise 2.0` has not been built as a library of functions, but as a set of functions around a central GUI to solve SLP (and Schrödinger problems in particular). Several other features illustrate this, such as the algorithm that is used to detect whether a given problem is singular and the automatic computation of the error estimates for the eigenvalues. The algorithms are very useful for solving a particular SLP, but valuable computation time is lost if this detection or the error estimation is not needed.

These challenges were the main motivator for a more efficient implementation of `Matslise`. We have considered different ways to optimize `Matslise`, keeping in mind its main features, in particular its user-friendliness, one of the main reasons for the development of the first version of `Matslise`.

In this section, we give an overview of our theoretical advancements [9] in building a very efficient implementation of the constant perturbation method. Later on, in section 2.5, we will perform some numerical experiments and some runtime analysis. In section 2.6, we will take the time to take a look at the challenges we faced and the choices we made when building our new implementation.

As a high-level overview, the constant perturbation method for solving one-dimensional Schrödinger problems can be summarized in the following steps:

- Split the domain $[a, b]$ in intervals $(a = x_0, x_1, x_2, \dots, x_k, \dots, x_n = b)$.
- For each interval k write $X = x_{k-1}$ and $h = x_k - x_{k-1}$.
  - Construct propagators $u(\delta)$ and $v(\delta)$ for $\delta \in [0, h]$, according to the formulae from theorem 2.10.
  - A solution $y(x)$ can be calculated on the k^{th} interval if $y(X)$ is known:
    $$\begin{aligned} y(X+\delta) &= u(\delta)y(X) + v(\delta)y'(X) \\ y'(X+\delta) &= u'(\delta)y(X) + v'(\delta)y'(X). \end{aligned}$$
- Employ multiple shooting to find solutions to the boundary value problem.

Until now the perturbation terms for $u(\delta)$ and $v(\delta)$ given in theorem 2.10 were only calculated and implemented with $\delta = h$. In that case, one not only obtains superconvergence as formulated in theorem 2.11, but there is also a major simplification in symbolic complexity. Nevertheless, for higher orders even these 'simplified' formulae are daunting to work with.

For the efficient calculation of eigenfunctions in arbitrary points of the domain it would however be beneficial if the propagation terms were also implemented in function of δ. To prove the corresponding mathematical results, we reformulate the expressions of the propagators in terms of $\theta = \delta/h$.

Functions of θ will be denoted with a bar, like $\bar{C}(\theta) = C(\theta h) = C(\delta)$ and $\bar{u}_q(\theta) = u_q(\theta h) = u_q(\delta)$. Again $\bar{p}$ generically denotes the functions $\bar{u}$ or $\bar{v}$. The CP-correction terms are now given by:

$$\bar{p}_q(\theta) = \sum_{i=-1}^{\infty} h^{2i+1}\theta^{2i+1}\bar{C}_i^{(q)}(\theta)\eta_i(Z(h\theta)) \tag{2.24}$$

with derivative:

$$\begin{aligned}\bar{p}_q'(\theta) = \sum_{i=-1}^{\infty} &\left(\bar{C}_i^{(q)\prime}(\theta) + h^2\theta\bar{C}_{i+1}^{(q)}(\theta)\right) h^{2i+1}\theta^{2i+1}\eta_i(Z) \\ &+ \frac{\bar{C}_{-1}^{(q)}(\theta)}{\theta^2 h}\left(\eta_{-1}(Z) + Z\eta_0(Z)\right).\end{aligned} \tag{2.25}$$

The functions $\bar{C}_i^{(q)}$ satisfy the following recursive relation:

$$\begin{aligned}&\bar{C}_i^{(q)}(\theta) = \frac{\theta^{-i}}{2}\int_0^\theta \sigma^{i-1}\left(\bar{C}_{i-1}^{(q-1)}(\sigma)\Delta V(X+h\sigma) - h^{-2}\bar{C}_{i-1}^{(q)\prime\prime}(\sigma)\right)\mathrm{d}\sigma \\ &\bar{C}_i^{(0)}(\theta) = \begin{cases} h\theta & \text{if } p = u \text{ and } i = -1 \\ 1 & \text{if } p = v \text{ and } i = 0 \\ 0 & \text{otherwise}\end{cases} \\ &\bar{C}_{-1}^{(q)}(\theta) = 0 \quad \text{if } q > 0\,.\end{aligned} \tag{2.26}$$

Denoting $\bar{y}(\theta) = y(X + \theta\, h)$, the propagation relation (2.17) can be rewritten as

$$\begin{pmatrix}\bar{y}(\theta)\\ \bar{y}'(\theta)\end{pmatrix} = \begin{pmatrix}\bar{u}(\theta) & \bar{v}(\theta)/h\\ \bar{u}'(\theta) & \bar{v}'(\theta)/h\end{pmatrix}\begin{pmatrix}\bar{y}(0)\\ \bar{y}'(0)\end{pmatrix}, \qquad 0 \le \theta \le 1. \tag{2.27}$$

Theorem 2.13 (Baeyens and Van Daele 2021)**.** *Assume, for h sufficiently small, that an approximation of the propagation matrix*

$$\begin{pmatrix} \bar{u}(\theta) & \bar{v}(\theta)/h \\ \bar{u}'(\theta) & \bar{v}'(\theta)/h \end{pmatrix}$$

is desired to be accurate to at least $\mathcal{O}(h^r)$. Then the number of correction terms Q has to be at least $\lfloor \frac{r}{3} \rfloor$ and V can be approximated by a polynomial of degree $N = r - 2$.

Proof. This theorem is heavily inspired by [49]. There an error estimate is calculated and theorem 2.10 and theorem 2.11 are proved. These proofs give a useful framework for providing a result for the δ-dependent formulae.

First we show by induction on q and i that $\bar{C}_i^{(q)}$ is a polynomial in both θ and h where for each term the power in h is not smaller than the power in θ. This indeed holds for all i when $q = 0$ and for all q when $i = -1$. Assuming that this also holds for $\bar{C}_{i-1}^{(q-1)}$ and $\bar{C}_i^{(q-1)}$, we will show that it also holds for $\bar{C}_i^{(q)}$. Indeed, $h^{-2}\bar{C}_{i-1}^{(q)\prime\prime}(\sigma)$ is a polynomial satisfying the same property and

$$\sigma^{i-1}\left(\bar{C}_{i-1}^{(q-1)}(\sigma)\Delta V(X + h\sigma) - h^{-2}\bar{C}_{i-1}^{(q)\prime\prime}(\sigma)\right)$$

is a polynomial of degree at least $i - 1$ in σ. After integrating with respect to σ between 0 and θ and multiplying by θ^{-i}, it follows that $\bar{C}_i^{(q)}(\theta)$ is a new polynomial in which no terms have gained factors in θ nor has any lost factors of h. Thus, the result is a polynomial in θ and h with in each term at least as many factors in h as in θ.

Further, one notices that if both $\bar{C}_{i-1}^{(q-1)}$ and $\bar{C}_{i-1}^{(q)}$ are zero, so is $\bar{C}_i^{(q)}$. With the given conditions, it is easy to verify that

$$\bar{C}_i^{(q)}(\theta) = \begin{cases} 0 \text{ for } i < q - 1, & \text{if } \bar{p} = \bar{u}, \\ 0 \text{ for } i < q, & \text{if } \bar{p} = \bar{v}. \end{cases}$$

In case $\bar{C}_i^{(q)}(\theta)$ is not zero, we investigate the common degree in h of all its terms. Let us denote this common degree by $d_h(\bar{C}_i^{(q)})$. Since ΔV does not contain a constant term, $d_h\left(\bar{C}_{q-1}^{(q)}\right) = d_h\left(\bar{C}_{q-2}^{(q-1)}\right) + 1$. On the other hand $d_h\left(\bar{C}_i^{(q)}\right) = d_h\left(\bar{C}_{i-1}^{(q)}\right) - 2$.

For $\bar{p}=\bar{u}$, we have $d_h\left(\bar{C}_{-1}^{(0)}\right)=1$ such that $d_h\left(\bar{C}_{q-1}^{(q)}\right)=q+1$ and $d_h\left(\bar{C}_{i}^{(q)}\right)=q+1-2(i-(q-1))=3q-2i-1$ for $i\geq q-1$.

For $\bar{p}=\bar{v}$, we similarly have $d_h\left(\bar{C}_{0}^{(0)}\right)=0$ such that $d_h\left(\bar{C}_{q}^{(q)}\right)=q$ and $d_h\left(\bar{C}_{i}^{(q)}\right)=q-2(i-q)=3q-2i$ for $i\geq q$.

Combining these results and assuming all $\eta_i(Z(h\theta))$ are bounded we get:

$$\bar{u}_q(\theta)=\sum_{i=q-1}h^{2i+1}\theta^{2i+1}\bar{C}_i^{(q)}(\theta)\eta_i(Z(h\theta))=\mathcal{O}(h^{3q})$$
$$\bar{v}_q(\theta)/h=h^{-1}\sum_{i=q}h^{2i+1}\theta^{2i+1}\bar{C}_i^{(q)}(\theta)\eta_i(Z(h\theta))=\mathcal{O}(h^{3q}).$$

Using the expression for the derivative $\bar{p}_q'(\theta)$, one can similarly prove that

$$\bar{u}_q'(\theta)=\mathcal{O}(h^{3q})$$
$$\bar{v}_q'(\theta)/h=\mathcal{O}(h^{3q})$$

We can thus indeed conclude that, in order to be accurate up to order r in h, we need to consider all terms $\bar{p}_q(\theta)$, $q=0,1,2,\dots$ for which $3q\leq r$, i.e. $Q=\lfloor\frac{r}{3}\rfloor$.

Next we determine the minimal value of N, the degree of the polynomial approximation $V^N(x)$ of V, to obtain a given accuracy.

We first consider the case $p=u$, and we will show that for $i\geq q-1$ each term in $\bar{C}_i^{(q)}$ contains a factor of the form $V_{j_1}V_{j_2}\cdots V_{j_q}h^{j_1+j_2+\ldots+j_q+1-2(i-(q-1))}$. For $\bar{C}_0^{(1)}$, all terms indeed contain an expression V_jh^{j+1} and by induction we find for $i>0$ that all terms in $\bar{C}_i^{(1)}$ have a factor of the form V_jh^{j+1-2i}, since $\bar{C}_i^{(0)}(\theta)=0$. Assuming this result holds for all $\bar{C}_i^{(q-1)}$ with $i\geq q-1$, one verifies that the result also holds for $\bar{C}_{q-1}^{(q)}$ and by induction on i, also for $\bar{C}_i^{(q)}$ with $i>q-1$.

This property of $\bar{C}_i^{(q)}$ can now be used to determine the lowest degree N that leads to the $\mathcal{O}(h^r)$ approximation. Since $\bar{C}_i^{(q)}$ is multiplied by h^{2i+1}, it follows that $\bar{u}_q(\theta)$ contains terms with factors $V_{j_1}V_{j_2}\cdots V_{j_q}h^{j_1+j_2+\ldots+j_q+2q}$ with $1\leq j_k\leq N$, $k=1,\dots,q$. For $q=1$ we have V_jh^{j+2}, which can only be discarded when $j>r-2$. Thus choosing $N=r-2$ will only discard these terms. The choice $N=r-2$ is also sufficient when $q>1$. We can thus conclude that, in order to obtain accuracy up to order h^r in $\bar{u}(\theta)$ it suffices to take $N = r - 2$.

For $p = v$, one similarly shows that all terms in $\bar{C}_i^{(q)}$ contain expressions of the form $V_{j_1} V_{j_2} \cdots V_{j_q} h^{j_1+j_2+\ldots+j_q-2(i-q)}$ for $i \geq q$. Then $\bar{v}_q(\theta)/h$ contains terms with factors $V_{j_1} V_{j_2} \cdots V_{j_q} h^{j_1+j_2+\ldots+j_q+2q}$ and all necessary terms will be generated when $N = r - 2$.

For the expressions of the derivatives $\bar{u}'(\theta)$ and $\bar{v}'(\theta)/h$, no extra term is needed. So in summary, ΔV can be truncated to a degree of $r - 2$.

□

To illustrate this theorem, we give the first terms in the error $\Delta\bar{p}_{(Q)} = \bar{p}(\theta) - \sum_{i=0}^{Q} \bar{p}_i(\theta)$ of the propagation formulae.

Zero correction terms:

$$\begin{aligned}
\Delta\bar{u}_{(Q=0)} &= h^3 \left(\left(\frac{1}{2}\theta^3 - \frac{1}{2}\theta^2 \right) V_1\eta_0 - \frac{1}{2}\theta^3 V_1\eta_1 \right) + \mathcal{O}(h^4) \\
\Delta\bar{v}_{(Q=0)}/h &= h^3 \left(\left(\frac{1}{2}\theta^4 - \frac{1}{2}\theta^3 \right) V_1\eta_1 \right) + \mathcal{O}(h^4) \\
\Delta\bar{u}'_{(Q=0)} &= h^3 \left(\left(\frac{1}{2}\theta^2 - \frac{1}{2}\theta \right) V_1\eta_{-1} + \left(\frac{1}{2}\theta^2 - \frac{1}{2}\theta \right) V_1\eta_0 \right) + \mathcal{O}(h^4) \\
\Delta\bar{v}'_{(Q=0)}/h &= h^3 \left(\left(\frac{1}{2}\theta^3 - \frac{1}{2}\theta^2 \right) V_1\eta_0 + \frac{1}{2}\theta^3 V_1\eta_1 \right) + \mathcal{O}(h^4)
\end{aligned}$$

One correction term:

$$\begin{aligned}
\Delta\bar{u}_{(Q=1)} &= h^6 \left(\left(\frac{1}{8}\theta^6 - \frac{1}{4}\theta^5 + \frac{1}{8}\theta^4 \right) V_1^2\eta_1 + \left(-\frac{7}{24}\theta^6 + \frac{1}{4}\theta^5 \right) V_1^2\eta_2 \right) \\
&\qquad + \mathcal{O}(h^7) \\
\Delta\bar{v}_{(Q=1)}/h &= h^6 \left(\left(\frac{1}{8}\theta^7 - \frac{1}{4}\theta^6 + \frac{1}{8}\theta^5 \right) V_1^2\eta_2 - \frac{1}{24}\theta^7 V_1^2\eta_3 \right) + \mathcal{O}(h^7) \\
\Delta\bar{u}'_{(Q=1)} &= h^6 \left(\left(\frac{1}{8}\theta^5 - \frac{1}{4}\theta^4 + \frac{1}{8}\theta^3 \right) V_1^2\eta_0 + \left(\frac{1}{12}\theta^5 - \frac{1}{4}\theta^4 + \frac{1}{8}\theta^3 \right) V_1^2\eta_1 \right. \\
&\qquad \left. -\frac{7}{24}\theta^5 V_1^2\eta_2 \right) + \mathcal{O}(h^7)
\end{aligned}$$

$$\Delta\bar{v}'_{(Q=1)}/h = h^6\left(\left(\frac{1}{8}\theta^6 - \frac{1}{4}\theta^5 + \frac{1}{8}\theta^4\right)V_1^2\eta_1 + \left(\frac{5}{24}\theta^6 - \frac{1}{4}\theta^5\right)V_1^2\eta_2\right) + \mathcal{O}(h^7)$$

Two correction terms:

$$\Delta\bar{u}_{(Q=2)} = h^9\left(\left(\frac{1}{48}\theta^9 - \frac{1}{16}\theta^8 + \frac{1}{16}\theta^7 - \frac{1}{48}\theta^6\right)V_1^3\eta_2 + \left(\frac{-1}{12}\theta^9 + \frac{7}{48}\theta^8 - \frac{1}{16}\theta^7\right)V_1^3\eta_3 + \frac{1}{48}\theta^9V_1^3\eta_4\right) + \mathcal{O}(h^{10})$$

$$\Delta\bar{v}_{(Q=2)}/h = h^9\left(\left(\frac{1}{48}\theta^{10} - \frac{1}{16}\theta^9 + \frac{1}{16}\theta^8 - \frac{1}{48}\theta^7\right)V_1^3\eta_3 + \left(-\frac{1}{48}\theta^{10} + \frac{1}{48}\theta^9\right)V_1^3\eta_4\right) + \mathcal{O}(h^{10})$$

$$\Delta\bar{u}'_{(Q=2)} = h^9\left(\left(\frac{1}{48}\theta^8 - \frac{1}{16}\theta^7 + \frac{1}{16}\theta^6 - \frac{1}{48}\theta^5\right)V_1^3\eta_1 + \left(-\frac{1}{24}\theta^7 + \frac{1}{16}\theta^6 - \frac{1}{48}\theta^5\right)V_1^3\eta_2 + \left(-\frac{7}{48}\theta^8 + \frac{7}{48}\theta^7\right)V_1^3\eta_3\right) + \mathcal{O}(h^{10})$$

$$\Delta\bar{v}'_{(Q=2)}/h = h^9\left(\left(\frac{1}{48}\theta^9 - \frac{1}{16}\theta^8 + \frac{1}{16}\theta^7 - \frac{1}{48}\theta^6\right)V_1^3\eta_2 + \left(\frac{1}{24}\theta^9 - \frac{5}{48}\theta^8 + \frac{1}{16}\theta^7\right)V_1^3\eta_3 - \frac{1}{48}\theta^9V_1^3\eta_4\right) + \mathcal{O}(h^{10})$$

To illustrate the second part of the theorem, we consider $\Delta\bar{p}_{(N)} = \bar{p}(\theta) - \bar{p}_N(\theta)$ whereby $\bar{p}_N(\theta)$ is obtained from $\bar{p}(\theta)$ in which $V_i = 0$ if $i > N$.

Truncation of ΔV to degree 0:

$$2\,\Delta\bar{u}_{(N=0)} = h^3\left(\left(\theta^3 - \theta^2\right)V_1\eta_0 - \theta^3V_1\eta_1\right) + \mathcal{O}(h^4)$$

$$\frac{2}{h}\Delta\bar{v}_{(N=0)} = h^3\left(\left(\theta^4 - \theta^3\right)V_1\eta_1\right) + \mathcal{O}(h^4)$$

$$
\begin{aligned}
2\,\Delta\bar{u}'_{(N=0)} &= h^3\left(\left(\theta^2-\theta\right)V_1\eta_{-1}+\left(\theta^2-\theta\right)V_1\eta_0\right)+\mathcal{O}(h^4)\\
\frac{2}{h}\Delta\bar{v}'_{(N=0)} &= h^3\left(\left(\theta^3-\theta^2\right)V_1\eta_0+\theta^3V_1\eta_1\right)+\mathcal{O}(h^4)
\end{aligned}
$$

Truncation of ΔV to degree 1:

$$
\begin{aligned}
2\,\Delta\bar{u}_{(N=1)} &= h^4\left(\left(2\theta^4-3\theta^3+\theta^2\right)V_2\eta_0+\left(-3\theta^4+3\theta^3\right)V_2\eta_1\right)+\mathcal{O}(h^5)\\
\frac{2}{h}\Delta\bar{v}_{(N=1)} &= h^4\left(\left(2\theta^5-3\theta^4+\theta^3\right)V_2\eta_1-\theta^5V_2\eta_2\right)+\mathcal{O}(h^5)\\
2\,\Delta\bar{u}'_{(N=1)} &= h^4\left(\left(2\theta^3-3\theta^2+\theta\right)V_2\eta_{-1}+\left(3\theta^3-3\theta^2+\theta\right)V_2\eta_0\right.\\
&\qquad \left.-3\theta^3V_2\eta_1\right)+\mathcal{O}(h^5)\\
\frac{2}{h}\Delta\bar{v}'_{(N=1)} &= h^4\left(\left(2\theta^4-3\theta^3+\theta^2\right)V_2\eta_0+\left(3\theta^4-3\theta^3\right)V_2\eta_1\right)+\mathcal{O}(h^5)
\end{aligned}
$$

Truncation of ΔV to degree 2:

$$
\begin{aligned}
2\,\Delta\bar{u}_{(N=2)} &= h^5\left(\left(5\theta^5-10\theta^4+6\theta^3-\theta^2\right)V_3\eta_0+\left(-10\theta^5+15\theta^4-6\theta^3\right)V_3\eta_1\right.\\
&\qquad \left.+5\theta^5V_3\eta_2\right)+\mathcal{O}(h^6)\\
\frac{2}{h}\Delta\bar{v}_{(N=2)} &= h^5\left(\left(5\theta^6-10\theta^5+6\theta^4-\theta^3\right)V_3\eta_1+\left(-5\theta^6+5\theta^5\right)V_3\eta_2\right)+\mathcal{O}(h^6)\\
2\,\Delta\bar{u}'_{(N=2)} &= h^5\left(\left(5\theta^4-10\theta^3+6\theta^2-\theta\right)V_3\eta_{-1}\right.\\
&\qquad +\left(10\theta^4-15\theta^3+6\theta^2-\theta\right)V_3\eta_0\\
&\qquad \left.+\left(-15\theta^4+15\theta^3\right)V_3\eta_1\right)+\mathcal{O}(h^6)\\
\frac{2}{h}\Delta\bar{v}'_{(N=2)} &= h^5\left(\left(5\theta^5-10\theta^4+6\theta^3-\theta^2\right)V_3\eta_0+\left(10\theta^5-15\theta^4+6\theta^3\right)V_3\eta_1\right.\\
&\qquad \left.-5\theta^5V_3\eta_2\right)+\mathcal{O}(h^6)
\end{aligned}
$$

Remark that in theorem 2.13, in contrast to theorem 2.11, no assumption is made on the size of $Z=\delta^2(V_0-E)$. In the proof of theorem 2.11 this distinction was necessary because error terms were expressed not only in terms of bounded functions $\eta_i(Z)$ but also in terms of the unbounded argument Z, whereas in the proof of theorem 2.13 all error terms were expressed in terms of $\eta_i(Z)$. This enables us to give $\mathcal{O}(h^{N+2})$ error estimates for all Z. However, for sufficiently large negative $Z(h)\ll 0$ the $\mathcal{O}(h^N/\sqrt{E})$ error estimate given in theorem 2.11 is sharper than the one given in theorem 2.13. This new result shows that the order P_{ass} for large and negative Z is the same as the order for small values for Z, thus $P_0=P_{ass}$. So from now on, we can denote a CP-method as CPM$\{r\}$, this method will be accurate with order $\mathcal{O}(h^r)$ if

$N = r - 2$ and $Q = \lfloor \frac{r}{3} \rfloor$. As a demonstration, we give the formulae for the fourth order method CPM{4}:

$$\begin{aligned}
2\bar{u}^{(4)} &= 2\eta_{-1} + h^3 \left(\left(\theta^3 - \theta^2\right) V_1 \eta_0 - \theta^3 V_1 \eta_1 \right) \\
&\quad + h^4 \left(\left(2\theta^4 - 3\theta^3 + \theta^2\right) V_2 \eta_0 + \left(-3\theta^4 + 3\theta^3\right) V_2 \eta_1 \right) \\
\frac{2}{h}\bar{v}^{(4)} &= 2\theta\eta_0 + h^3 \left(\left(\theta^4 - \theta^3\right) V_1 \eta_1 \right) + h^4 \left(\left(2\theta^5 - 3\theta^4 + \theta^3\right) V_2 \eta_1 - \theta^5 V_2 \eta_2 \right) \\
2\bar{u'}^{(4)} &= \frac{2Z}{h\theta}\eta_0 + h^2 \left(\left(\theta^2 - \theta\right) V_1 \eta_{-1} + \left(\theta^2 - \theta\right) V_1 \eta_0 \right) \\
&\quad + h^3 \left(\left(2\theta^3 - 3\theta^2 + \theta\right) V_2 \eta_{-1} + \left(3\theta^3 - 3\theta^2 + \theta\right) V_2 \eta_0 - 3\theta^3 V_2 \eta_1 \right) \\
\frac{2}{h}\bar{v'}^{(4)} &= \frac{2}{h}\eta_{-1} + h^2 \left(\left(\theta^3 - \theta^2\right) V_1 \eta_0 + \theta^3 V_1 \eta_1 \right) \\
&\quad + h^3 \left(\left(2\theta^4 - 3\theta^3 + \theta^2\right) V_2 \eta_0 + \left(3\theta^4 - 3\theta^3\right) V_2 \eta_1 \right).
\end{aligned}$$

Furthermore, theorem 2.13 also gives more insight in the number of correction terms Q needed to construct higher order methods. In `Matslise 2.0`, CPM$\{18, 16\}$ was constructed with $N = 16$ and $Q = 11$. Theorem 2.13 shows that all necessary terms can be generated with $Q = 6$ correction terms.

Evaluating eigenvalues

Once an eigenvalue is computed in `Matslise` or `Matslise 2.0`, the corresponding eigenfunction can be visualized. Since the mesh is coarse, such a visualization can only be achieved if extra function evaluations are available in a sufficient number of points. Neither `Matslise` nor `Matslise 2.0` were optimized do to so. To visualize the eigenfunction y_k that corresponds to a particular eigenvalue E_k, a new, sufficiently fine mesh is constructed, and all necessary computations are carried out once again.

To explain why this approach was chosen in `Matslise` and `Matslise 2.0`, let us consider the propagation matrix in (2.17). From theorem 2.10, we learn that each matrix element can be expressed as a polynomial in δ, whose coefficients depend via $Z = (V_0 - E)\delta^2$ on the functions η_i and δ-dependent polynomials C_j that also contain powers of the step length h and the coefficients V_n computed by (2.16). To compute the formulae in CPM{18} for instance, we will take a look at the expression for $u(\delta)$:

$$u(\delta) = \sum_{i=-1}^{6} C_i(\delta)\eta_i(\delta),$$

with C_i a polynomial in δ,

$$C_i = \sum_{j=0}^{16} d_{i,j}\delta^j,$$

where $d_{i,j}$ is a polynomial of degree $16-j$ in h with coefficients in $V_1,\dots,V_{16}$. As an example we give a small taste of what $d_{4,9}$ looks like:

$$\begin{aligned}d_{4,9} = h^7 \big(&0.03125\,V_2^4 + 0.40625\,V_1V_2^2V_3 + 0.21875\,V_1^2V_3^2 + 0.46875\,V_1^2V_2V_4\\ &+ 0.1875\,V_1^3V_5 - 67.8125\,V_3^2V_4 - 74.375\,V_2V_4^2 - 158.375\,V_2V_3V_5\\ &- 196.875\,V_1V_4V_5 - 96.5625\,V_2^2V_6 - 218.875\,V_1V_3V_6 - 275.0\,V_1V_2V_7\\ &- 199.125\,V_1^2V_8 + 159909.75\,V_1V_{11} + 101029.5\,V_{10}V_2 + 27184.5\,V_6^2\\ &+56164.5\,V_5V_7 + 62181.0\,V_4V_8 + 74868.75\,V_3V_9)\\ + h^6(&\dots) + \dots\end{aligned}$$

Although these coefficients can all be computed analytically, it was far from practical to implement these very long formulae in software. Since one was mainly interested in the computation of the eigenvalues, the matrix elements in the propagation matrix were in fact only implemented for $\delta = h$, in which case the formulae could be significantly simplified, which in turn had a positive impact on the computation speed.

For the applications we have in mind, this approach is too time-consuming. We decided to develop a new algorithm that is able to reuse all the data that was previously calculated. Therefore, we have chosen to implement in `Matslise 3.0` a δ-independent higher order method that is combined with a δ-dependent formula of lower order. The higher order method is the CPM{18}-formula (i.e. $N = 16$ and $Q = 6$) that was also used in `Matslise` and `Matslise 2.0`. For our error estimation, a δ-dependent version of CPM{16} is implemented ($N = 14$ and $Q = 5$), which also allows the fast and accurate evaluation of the eigenfunctions in arbitrary points. In section 2.5, we will present some numerical experiments in which eigenfunctions are computed as well.

2.4 Periodic 1D Schrödinger equation

In contrast to (separated) Robin boundary conditions, in this section we will study the case of generalized periodic boundary conditions. Assume we are solving a regular Sturm–Liouville equation

$$-\frac{\mathrm{d}p(x)y'(x)}{\mathrm{d}x} + q(x)y(x) = \lambda w(x)y(x) \tag{2.28}$$

on a domain $[a, b]$, with generalized periodic boundary conditions:

$$\begin{pmatrix} y(a) \\ p(a)y'(a) \end{pmatrix} = \mathbf{K} \begin{pmatrix} y(b) \\ p(b)y'(b) \end{pmatrix}. \tag{2.29}$$

In expression (2.29), $\mathbf{K}$ is assumed to be a 2×2 real matrix with determinant 1 [14]. If $\mathbf{K} = \mathbf{I}$, these boundary conditions are called *periodic*, if $\mathbf{K} = -\mathbf{I}$ then they are called *antiperiodic*.

In section 2.1.1, we have provided some theorems about regular Sturm–Liouville problems with homogeneous (separated) Robin boundary conditions. From all these theorems, only theorem 2.1 is also applicable here: all eigenvalues are real, and eigenfunctions can always be scaled to be real as well.

Following section 2.2 and 2.3, due to the Liouville-transformation (from section 2.1.2), it is sufficient to only consider Schrödinger problems with $p(x) = w(x) = 1$.

Since the equation has not changed, the idea and computation of a piecewise constant approximation, perturbation terms and a propagation matrix are still valid and valuable. There are just a few related problems. First, what do we propagate? No obvious initial conditions are available. And secondly, we are still only interested in continuous and continuously differentiable functions, so the idea of a matching error is relevant. But how can we compute the index of an eigenvalue? Without this, it would be difficult to ensure all requested eigenvalues are found.

For this we employ the results from Binding and Volkmer [15, 14] to expand the Prüfer transformation. In theorem 2.12, the Prüfer-angle function $\theta(x)$ was defined. This definition assumed initial conditions $y(a) = \beta_a$ and $p'(a)y(a) = -\alpha_a$. In the periodic case, no such initial conditions are available. Therefore, we define the Prüfer-angle as a function of the initial conditions $y(a) = \sin(\alpha)$ and $p(a)y'(a) = \cos(\alpha)$. For a fixed value of E, define $\theta_\alpha(x)$ with the ordinary differential equation (2.23) and initial conditions $\theta_\alpha(a) \in [0, \pi[$ and

$$\tan \theta_\alpha(a) = S(a) \tan(\alpha).$$

As in theorem 2.12, it is necessary to consider the difference with the rightmost boundary condition. For this we define the unique π-periodic function $\beta(\alpha)$ with $\beta(0) \in]-\pi, 0]$ and

$$\tan(\beta(\alpha)) = S(b) \frac{k_{11} \sin(\alpha) + k_{12} \cos(\alpha)}{k_{21} \sin(\alpha) + k_{22} \cos(\alpha)},$$

with

$$\mathbf{K} = \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}.$$

With this $\beta(\alpha)$, we define

$$\begin{aligned} \delta_\alpha(E) &= \theta_\alpha(b) - \beta(\alpha), \\ m(E) &= \min_{\alpha\in\mathbb{R}} \delta_\alpha(E) \\ \text{and } M(E) &= \max_{\alpha\in\mathbb{R}} \delta_\alpha(E). \end{aligned} \tag{2.30}$$

In these expressions we have made the dependence on E explicit, as E was previously assumed to be fixed.

The following lemma and theorems with proofs can be found in [15] and [13].

Lemma 2.14 (Binding and Volkmer 2012)**.**

- *The functions m and M are continuous and strictly increasing.*
- *For each E, we have the inequalities*

$$-\pi < m(E) < M(E) < m(E) + \pi.$$

For the following theorems we need to distinguish between $k_{12} < 0 \vee k_{12} = 0 < k_{11}$ and the opposite. Let us call the former the periodic case, and the latter the antiperiodic case.

Theorem 2.15 (Binding and Volkmer 2012)**.** *For each eigenvalue λ of the Sturm–Liouville equation* (2.28) *subject to generalized periodic boundary conditions, either $m(\lambda) = k\pi$ or $M(\lambda) = k\pi$ with k integer. In the periodic case, k will be even, in the antiperiodic case, k will be odd.*

The last theorem gives us a coupling between the function $m(E), M(E)$ and the eigenvalues of the Sturm–Liouville problem. To reliably determine the index of an eigenvalue we also need the converse theorem.

Theorem 2.16 (Binding and Volkmer 2012)**.** *Define λ_k^+ to be the unique value for which $m(\lambda_k^+) = k\pi$ and λ_k^- to be the unique value such $M(\lambda_k^-) = k\pi$. There is no value λ_0^-.*

- *If k is even (for the periodic case) or odd (for the antiperiodic case), then all values $\lambda_k^\pm$ are eigenvalues of the corresponding Sturm–Liouville problem.*

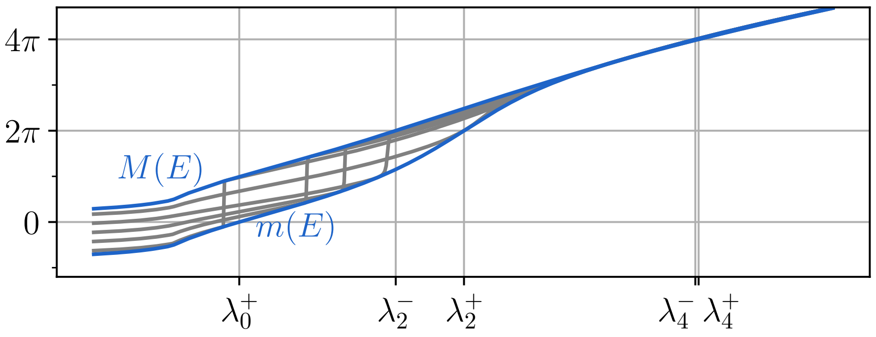


Figure 2.12: The scaled Prüfer transformation with different initial conditions (gray lines) for the Schrödinger problem with $q(x) = 3x^2(\pi - x)$ on $[0, \pi]$. The functions $m(E)$ and $M(E)$ from (2.30) are drawn as well.

- *There are no other eigenvalues.*
- $\lambda_k^+ < \lambda_{k+1}^-$ *for each* $k \geq 0$.
- *An eigenfunction belonging to* $\lambda_k^\pm$ *has exactly* k *zeros in the interval* $[a, b[$.

This theorem characterizes all eigenvalues of regular Sturm–Liouville problems with periodic boundary conditions. The eigenvalues are countable. They have a lower bound: λ_0^+ (for the periodic case) and λ_1^+ (for the antiperiodic case). Also, an eigenvalue has at most multiplicity two. Furthermore, it is known that $\lambda_k^\pm \to \infty$ if $k \to \infty$.

As an illustration, we provide figure 2.12. Here, the expression $\delta_\alpha(E)$ is visualized for varying initial conditions expressed with α. All these curves are captured in between $m(E)$ and $M(E)$. The first five eigenvalues of the considered problem are indicated as well. We see that for these values either $m(E)$ or $M(E)$ is an even multiple of π.

To determine the index of an eigenvalue, theorem 2.16 assures us that using the function $m(E)$ and $M(E)$ is sufficient. But efficiently computing these functions, is no easy task. Since they are defined as extreme values over all possible initial conditions, there is no straightforward way to evaluate them.

We propose to propagate two different initial conditions and use the inequality from lemma 2.14 to estimate $m(E)$ and $M(E)$. More concretely, we apply the propagation matrix from (2.18), not to a single vector, but to a pair of vectors,

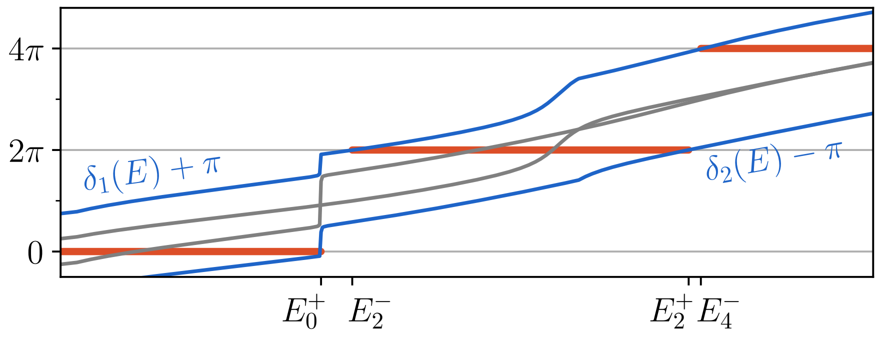


Figure 2.13: The functions $\delta_2(E)-\pi$ and $\delta_1(E)+\pi$ for the periodic Schrödinger problem with $q(x)=3x^2(\pi-x)$ on $[0,\pi]$. The search intervals from (2.31) are highlighted in red.

each corresponding to different initial conditions[10]. Both these vectors give rise to two estimates $\delta_1(E)$ and $\delta_2(E)$ of $\delta_\alpha(E)$. Without loss of generality we assume $m(E)\le\delta_1(E)\le\delta_2(E)\le M(E)$. With lemma 2.14, we can deduce

$$\delta_2(E)-\pi<m(E) \text{ and } M(E)<\delta_1(E)+\pi.$$

Suppose we want to find eigenvalues λ_k^- and λ_k^+. Now we search for the values E_k^- and E_k^+ such that

$$\delta_1(E_k^-)+\pi=k\pi \text{ and } \delta_2(E_k^+)-\pi=k\pi. \tag{2.31}$$

By construction, we know for certain that the interval $[E_k^-,E_k^+]$ now contains λ_k^- and λ_k^+, and that it contains no other eigenvalues. In figure 2.13, these intervals are visualized. The last issue left is that the matching error defined in (2.19) no longer exists. There, we started with a single fixed initial condition, propagated this to the matching point and verified that the resulting eigenfunction is continuous and continuously differentiable.

For the periodic problem, no initial values are known. So, if any initial condition exists such that the resulting eigenfunction is indeed smooth in the matching

[10]Any pair of linear independent initial conditions will do. In our implementation we have chosen to use $y_1(a)=1$ and $y_1'(a)=0$ for the first initial condition, $y_2(a)=0$ and $y_2'(a)=1$ for the second.

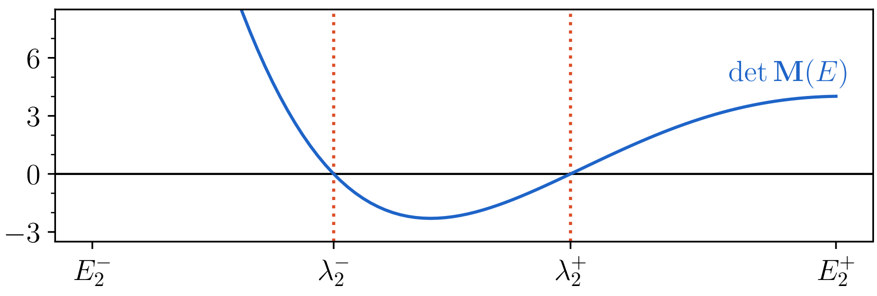


Figure 2.14: A close-up of the determinant of $\mathbf{M}(E)$ from (2.32) for the same problem as used in figure 2.12.

point, then the corresponding value for E is an eigenvalue of the periodic Schrödinger problem. We are not able to propagate all initial conditions. However, since we are working with a linear differential equation, the problem can be translated into finding any linear combination of $u(x)$ (with $u(a) = 1$ and $u'(a) = 0$) and $v(x)$ (with $v(a) = 0$ and $v'(a) = 1$) such that the function is smooth in the matching point x_m, while satisfying the boundary conditions. Call $u_L(x)$ the solution propagated from the left-hand side of the domain and $u_R(x)$ the solution propagated from the right-hand side of the domain, and analogous for $v_L(x)$ and $v_R(x)$. Now the question can be reformulated as: do there exist linear combinations $\mathbf{c_L}$ and $\mathbf{c_R}$ such that $\mathbf{c_L} = \mathbf{K}\mathbf{c_R}$ and

$$\begin{pmatrix} u_L(x_m) & v_L(x_m) \\ u'_L(x_m) & v'_L(x_m) \end{pmatrix} \mathbf{c_L} = \begin{pmatrix} u_R(x_m) & v_R(x_m) \\ u'_R(x_m) & v'_R(x_m) \end{pmatrix} \mathbf{c_R}?$$

Or, in other words: find values for E such that

$$\mathbf{M}(E) := \begin{pmatrix} u_L(x_m) & v_L(x_m) \\ u'_L(x_m) & v'_L(x_m) \end{pmatrix} \mathbf{K} - \begin{pmatrix} u_R(x_m) & v_R(x_m) \\ u'_R(x_m) & v'_R(x_m) \end{pmatrix} \tag{2.32}$$

becomes singular.

In summary, the function $\det \mathbf{M}(E)$ has exactly two (or one with double multiplicity) zeros in the interval $[E_1, E_2]$. As a demonstration we provide figure 2.14. Here $\det \mathbf{M}(E)$ is displayed for the same test problem as in figure 2.12. For $k = 2$ we found $E_1 \approx 8.064$ and $E_2 \approx 17.344$. The eigenvalues in this interval are $\lambda_2^- \approx 11.075$ and $\lambda_2^+ \approx 14.032$.

If an eigenvalue $\lambda_k^- = \lambda_k^+$ is not simple, that is to say, if it has a double multiplicity, then this will be visible in $\det \mathbf{M}(\lambda_k^-)$. For these values $\det \mathbf{M}(E)$ will have a double root[11].

In `Matslise 3.0` and `Pyslise` we have implemented this theory, for now only when $\mathbf{K} = \mathbf{I}$. In section 2.5.6 we present a numerical experiment where we demonstrate how our program can be used.

2.5 Numerical experiments

As a first numerical experiment, let us verify the order in h and make an estimate for the order in k. For this, we use the same problem as in figures 2.6 and 2.7.

2.5.1 A first example

Consider the Schrödinger problem with equation

$$-y'' + 100\cos^2(x)\, y = \lambda y, \tag{2.33}$$

on the domain $[a, b] = [0, \pi]$ and with homogeneous Dirichlet boundary conditions.

Before jumping into the graphs, we have to note that `Matslise 3.0` has automatic step size selection built in. This allows the algorithm to generate its own piecewise approximation of the potential. But, this also implies that a user is not directly able to control the number of subintervals used, which makes generating a figure like 2.6 or 2.7 more difficult.

To generate figures 2.15 and 2.16 we have solved 100 times the test problem with different tolerances between $2^{-10} \approx 10^{-3}$ and $2^{-30} \approx 10^{-9}$. We have grouped the errors of each of these solutions by the number of subintervals `Matslise 3.0` used, and averaged them for each eigenvalue.

In figure 2.15, we see that the number of subintervals varied between 4 and 10. Unsurprisingly, if `Matslise` used more subintervals, the results were more accurate. From the theory, we know that the propagation error from the constant perturbation method CPM{18} is of the order $\mathcal{O}(h^{18})$. We notice this same order in the error on the eigenvalue.

[11]Numerically, it could be possible that this function has no root at all. But then the minimum will be close to zero. When implementing this, care should be taken.

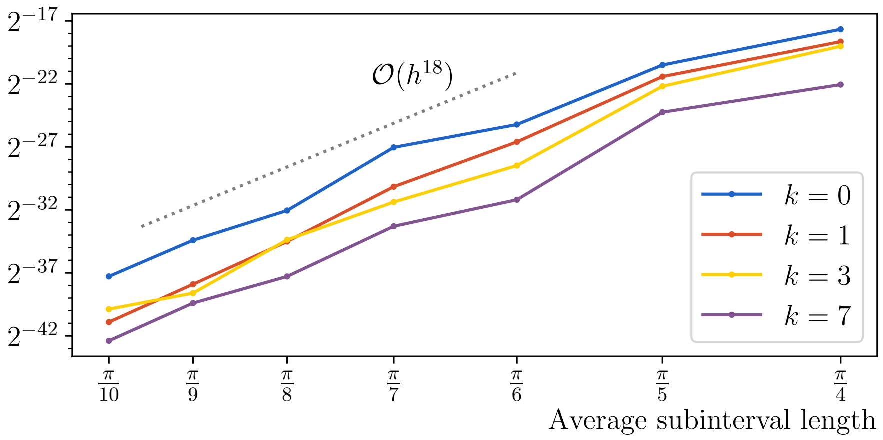


Figure 2.15: Relative error of the found eigenvalues of problem (2.33) by using `Matslise 3.0`.

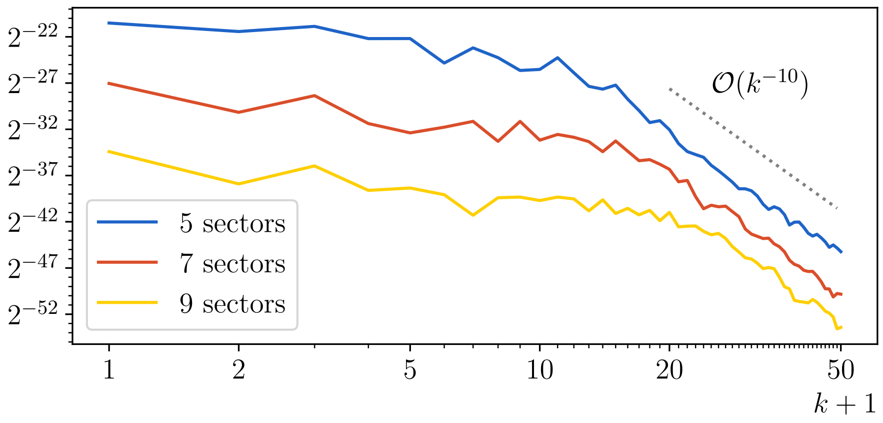


Figure 2.16: Relative error of the found eigenvalues of problem (2.33) by using `Matslise 3.0`. The graphs are in function of the index of the eigenvalue.

From the theory, we know little about the error estimate with respect to k. In theorem 2.6, we learned that for the Pruess method this order was $\mathcal{O}(k^{-4})$. With `Matslise` (both `2.0` and `3.0`) we observe a much more considerable value, visually we suspect this to be $\mathcal{O}(k^{-9})$ or $\mathcal{O}(k^{-10})$.

These numerical experiments may serve as a kind of tutorial to get to know `Pyslise`, this is the `python`-package built on top of `Matslise` `3.0`. So we will provide the code each time to solve the problem at hand. Here, to generate the first fifty eigenvalues the following code can be used.

```python
from pyslise import Pyslise
from math import pi, cos

V = lambda x: 100*cos(x)**2

problem = Pyslise(V, 0, pi, tolerance=1e-12)
print(problem.eigenvaluesByIndex(0, 50, (0, 1), (0, 1)))
```

The first few lines import the necessary functions. The next line defines the potential $V(x)$. Then, a `Pyslise` object is constructed. Here the user needs to provide the potential $V(x)$, and the left (0) and right (π) end points of the domain. Optionally a tolerance may be specified. Because we are working in `double` precision in `python`, anything less than approximately 10^{-16} is nonsensical.

Next, we find the eigenvalues. For this we use `.eigenvaluesByIndex`, a function defined on `Pyslise` objects. This function expects four arguments: i_{min}, i_{max}, $\mathbf{y_a}$ and $\mathbf{y_b}$. It will return all eigenvalues λ_i for which $i_{\text{min}} \leq i < i_{\text{max}}$ of the problem with boundary conditions

$$\begin{pmatrix} y(a) \\ y'(a) \end{pmatrix} = s_a \mathbf{y_a} \quad \text{and} \quad \begin{pmatrix} y(b) \\ y'(b) \end{pmatrix} = s_b \mathbf{y_b}.$$

In this expression, s_a and s_b are some scaling factors. Note that providing boundary conditions like this is different from the more mathematical notation

$$\alpha_a y(a) + \beta_a y'(a) = 0.$$

Our program expects a vector $\mathbf{y_a} = \begin{pmatrix}\beta_a & -\alpha_a\end{pmatrix}^\intercal$, and analogous for $\mathbf{y_b}$. We have chosen for these types of arguments to be more in line with how propagation is executed in the constant perturbation method.

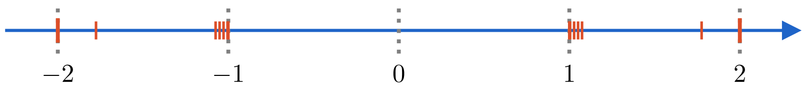


Figure 2.17: Visualization of the subintervals chosen by `Matslise 3.0` for the problem with potential (2.34), *without* specifying jumps.

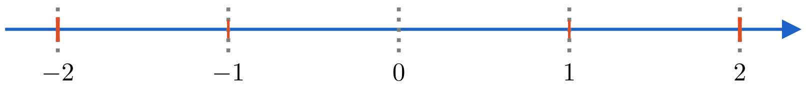


Figure 2.18: Visualization of the subintervals chosen by `Matslise 3.0` for the problem with potential (2.34), *with* specifying jumps.

2.5.2 A potential with jumps

In the next numerical experiment, we demonstrate another feature of constant perturbation methods. Since the potential function is piecewisely approximated, the potential may be discontinuous as long as the jumps are situated on the boundary between two consecutive subintervals.

Consider the Schrödinger problem with homogeneous Dirichlet boundary conditions on the interval $[-2, 2]$ with potential function

$$V(x) = \begin{cases} 0 & \text{if } |x| \leq 1, \\ 30 & \text{otherwise.} \end{cases} \tag{2.34}$$

If we use a similar program as before this may look like the following.

```python
from pyslise import Pyslise

def V(x):
    return 0 if abs(x) < 1 else 30

well = Pyslise(V, -2, 2, tolerance=1e-12)
print(well.eigenvaluesByIndex(0, 5, (0, 1), (0, 1)))
```

This program gives the correct results, but it used 15 subintervals. Those intervals are visualized in figure 2.17. If we change the line with the construction

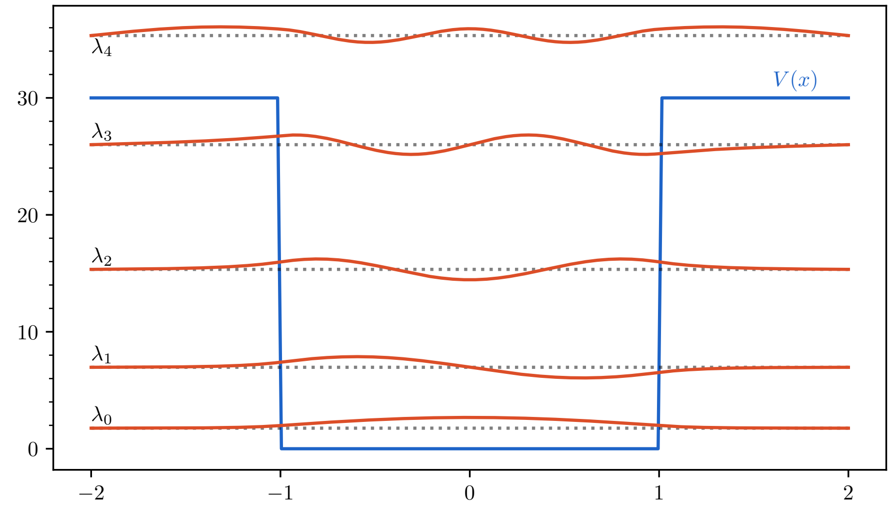


Figure 2.19: These are the first five eigenfunctions of the Schrödinger problem with potential (2.34) on $[-2, 2]$ with homogeneous Dirichlet boundary conditions.

of the `Pyslise` object to

```python
well = Pyslise(V, -2, 2, tolerance=1e-12, jumps=[-1, 1])
```

only 3 subintervals are used. For completeness, we have visualized these 3 subintervals in figure 2.18.

In any case, the first five eigenfunctions of this Schrödinger problem can be visualized with the following code.

```python
import numpy as np

x = np.linspace(-2, 2, 200)

plt.plot(x, np.vectorize(V)(x))
for i, E, f in well.eigenpairsByIndex(0, 5, (0, 1), (0, 1)):
    plt.plot(x, E + f(x)[0,:])
```

This yields a similar image to figure 2.19. In this code, `.eigenpairsByIndex` is a function which expects the same arguments as `.eigenvaluesByIndex`. But this function returns a list with tuples of three elements for each eigenvalue found: the index `i`, the eigenvalue `E` and the eigenfunction `f`. This `f` can be treated as any `python` function, and be evaluated in any value x inside the domain to get the value of the eigenfunction $f(x)$ and its derivative $f'(x)$.

2.5.3 Mathieu's equation

In the next examples we will compare the numerical accuracy of our new implementation `Matslise` 3.0 (and the `python` package `Pyslise`) with the original well-established `Matslise` 2.0.

As a first direct comparison, we will take a look at Mathieu's equation [80] on $[0, \pi]$ with homogeneous Dirichlet boundary conditions:

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + 2q\cos(2x)y(x) = \lambda y(x). \tag{2.35}$$

In this expression q is a parameter. In these experiments, we will only consider $q = 1$ and $q = 10$.

By using `Pyslise`, it is not difficult to find the first 10 eigenvalues of this equation:

```python
from pyslise import Pyslise
from math import pi, cos

q = 1
def V(x):
    return 2*q*cos(2*x)

mathieu = Pyslise(V, 0, pi, tolerance=1e-8)
print(mathieu.eigenvaluesByIndex(0, 10, (0,1), (0,1)))
```

The results of this computation can be compared with the results from `Matslise` 2.0. For different values of the parameter q ($q = 1$ and $q = 10$), the true eigenvalues (calculated with `Matslise` 2.0 and a tolerance of 10^{-14}) are used to calculate the error of the eigenvalues obtained from `Matslise` 2.0 and `Pyslise`, both with a tolerance of 10^{-8}. The execution times are reported as well. These were calculated using MATLAB's `timeit` function or Python's `timeit` function. Both execute the code multiple times to account for fluctuations caused by other system load. These comparisons were run on an Intel i7-8700K.

$q =$ 1	`Matslise 2.0`	`Pyslise`
	5.33ms	0.21ms
$-0.110\,248\,816\,992\,1$	$6.3 \cdot 10^{-10}$	$-2.6 \cdot 10^{-12}$
$3.917\,024\,772\,998\,5$	$-8.5 \cdot 10^{-10}$	$4.2 \cdot 10^{-12}$
$9.047\,739\,259\,809\,4$	$-1.1 \cdot 10^{-9}$	$-4.1 \cdot 10^{-12}$
$16.032\,970\,081\,405\,8$	$-6.1 \cdot 10^{-10}$	$-1.6 \cdot 10^{-11}$
$25.020\,840\,823\,289\,8$	$3.9 \cdot 10^{-10}$	$-2.6 \cdot 10^{-12}$
$36.014\,289\,910\,628\,2$	$5.3 \cdot 10^{-10}$	$-4.0 \cdot 10^{-12}$
$49.010\,418\,249\,423\,9$	$1.1 \cdot 10^{-9}$	$1.8 \cdot 10^{-11}$
$64.007\,937\,189\,249\,8$	$1.4 \cdot 10^{-9}$	$1.6 \cdot 10^{-11}$
$81.006\,250\,326\,632\,5$	$-4.8 \cdot 10^{-10}$	$4.5 \cdot 10^{-12}$
$100.005\,050\,675\,159\,4$	$-8.7 \cdot 10^{-10}$	$-9.3 \cdot 10^{-13}$

$q =$ 10	`Matslise 2.0`	`Pyslise`
	6.74ms	0.30ms
$-13.936\,552\,479\,250\,1$	$5.8 \cdot 10^{-9}$	$-5.4 \cdot 10^{-12}$
$-2.382\,158\,235\,956\,9$	$2.2 \cdot 10^{-8}$	$2.9 \cdot 10^{-12}$
$7.986\,069\,144\,681\,7$	$4.7 \cdot 10^{-10}$	$1.4 \cdot 10^{-11}$
$17.381\,380\,678\,623\,0$	$-3.4 \cdot 10^{-8}$	$6.4 \cdot 10^{-12}$
$26.766\,426\,360\,480\,1$	$-3.1 \cdot 10^{-8}$	$-8.2 \cdot 10^{-12}$
$37.419\,858\,776\,724\,2$	$-4.7 \cdot 10^{-8}$	$-5.1 \cdot 10^{-12}$
$50.054\,157\,213\,557\,2$	$-2.9 \cdot 10^{-8}$	$1.1 \cdot 10^{-12}$
$64.800\,440\,293\,021\,5$	$2.7 \cdot 10^{-8}$	$4.1 \cdot 10^{-12}$
$81.628\,313\,184\,383\,1$	$2.7 \cdot 10^{-8}$	$-3.1 \cdot 10^{-11}$
$100.506\,769\,462\,878\,4$	$4.0 \cdot 10^{-8}$	$-7.1 \cdot 10^{-12}$

Table 2.1: The first 10 eigenvalues for the Mathieu problem (2.35) for $q = 1$ and $q = 10$, the execution times and the absolute errors obtained with `Matslise 2.0` and `Pyslise` with a tolerance of 10^{-8}.

From table 2.1 there are two things to note. First, `Pyslise` is more than 20 times faster than `Matslise` 2.0. This means that our efforts to improve efficiency are definitely worth it.

Second, the accuracy of `Pyslise` is a lot higher. This can be explained by the fact that `Pyslise`'s error estimates are very conservative. The error estimation is used to choose the mesh to use. Because `Pyslise`'s errors are less sharp, it tends to use more subintervals than `Matslise` 2.0. These extra intervals lead to more accurate results. These more accurate results may seem like a benefit, however in reality, it is not. We requested an accuracy for both programs of 10^{-8}, and `Matslise` 2.0 respects this beautifully. In other words `Matslise` 2.0 is able to use the lowest number of subintervals possible, while respecting the requested accuracy. `Pyslise` on the other hand, goes above and beyond to ensure the requested accuracy and overshoots it with quite a lot. This means that it uses more subintervals than necessary, and is therefore a little less efficient than it could be.

In practice, and for our use-cases in the following chapters, we have found `Matslise` 3.0 and `Pyslise` to be sufficiently fast. Knowing this, we decided to keep our error estimate as it is, even though it is more conservative than it needs to be.

Eigenfunctions
As explained in section 2.3, one of the reasons we developed more complicated formulae and built a new implementation was the evaluation of the eigenfunctions. To evaluate the eigenfunction in `Matslise` 2.0, the partition of the domain has to be recomputed to include the requested evaluation points. If there are only a limited set of points, and these are known beforehand, then `Matslise` 2.0's methods is sufficient. In many applications however, this is insufficient.

Assume[12] we want to compute an orthogonal projection of a function $f(x)$ to the truncated basis of the eigenfunctions y_i: $f(x) = \sum_{i=0} c_i y_i(x)$. To compute these numerically we need to approximate the following integrals for all necessary values for i:

$$\int_a^b f(x) y_i(x) \, \mathrm{d}x.$$

In `Matslise` 2.0, one would choose a fixed grid beforehand and evaluate all eigenfunctions in exactly these grid points. To approximate the integral, any

[12]Or take a look at section 3.2.3, where we need to do exactly this.

$q = 1$	$n = 100$	$n = 1000$	$n = 10000$
`Matslise 2.0`	65.3ms	647ms	10180ms
`Pyslise`	0.387ms	3.05ms	29.3ms

Table 2.2: Execution times needed to compute the first 10 eigenfunctions of the Mathieu problem (2.35) in n equidistant points.

	y_0	y_1	y_2	y_3
`Matslise 2.0`	$1.0 \cdot 10^{-10}$	$6.2 \cdot 10^{-9}$	$1.7 \cdot 10^{-10}$	$1.4 \cdot 10^{-9}$
`Pyslise`	$4.7 \cdot 10^{-9}$	$5.9 \cdot 10^{-9}$	$2.8 \cdot 10^{-9}$	$1.9 \cdot 10^{-9}$
	y_4	y_5	y_6	y_7
`Matslise 2.0`	$4.0 \cdot 10^{-11}$	$5.7 \cdot 10^{-10}$	$8.2 \cdot 10^{-11}$	$8.6 \cdot 10^{-10}$
`Pyslise`	$3.9 \cdot 10^{-9}$	$2.6 \cdot 10^{-9}$	$1.4 \cdot 10^{-9}$	$1.3 \cdot 10^{-9}$

Table 2.3: Maximum absolute error in $n = 100$ equidistant points of the eigenfunction y_n corresponding to λ_n for the Mathieu problem (2.35) with $q = 1$.

quadrature rule can be used, as long as it only uses the values in exactly these grid points. If a large basis is used, or if $f(x)$ is unpredictable with some steep regions for example, this approach will only be accurate if a prohibitively dense grid is used. Ideally some adaptive quadrature rule should be employed. These numerical integration techniques evaluate the integrand in as many values as needed to ensure accuracy. When the integrand becomes 'difficult', more evaluations points are dynamically chosen. In the regions where the integrand is 'easy', much fewer evaluations are used. But, these methods, of course, need a way to evaluate the integrand dynamically, and do not work with a fixed grid. With this application in mind, our new implementation already has the benefit of being able to do this, with relatively little computational cost. But to see the true power of `Matslise 3.0` (using the `python`-package `Pyslise`), we compare the computation time for the evaluation of eigenfunctions with `Matslise 2.0`. These results are reported in table 2.2.

In this scenario the difference in efficiency becomes apparent. `Pyslise` is two orders of magnitude faster than `Matslise 2.0`. On top of that, this extreme speedup is achieved without dropping below the requested accuracy. In table 2.3 the maximum error of the eigenfunction for each eigenvalue is tabulated. One notices that `Matslise 2.0` is more accurate, but remember that the requested accuracy in this experiment was for both programs 10^{-8}. As long as the results are more accurate than 10^{-8}, they are successful.

It is remarkable that for the eigenvalues `Pyslise` was more accurate than required. For the eigenfunctions, the situation is reversed. This reversion can be explained by the fact that `Pyslise` is able to reuse the same grid (which was accurate to the required tolerance), but `Matslise 2.0` computes a new partition with the grid points used for the computation of the eigenvalue together with all requested evaluation points of the eigenfunction. This grid is much denser, and thus unnecessarily more accurate.

2.5.4 Coffey–Evans problem

Another, more challenging problem is the Coffey–Evans equation [81].

$$-y''(x) + (\beta^2 \sin^2(2x) - 2\beta\cos(2x))y(x) = \lambda y(x), \tag{2.36}$$

with $x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and homogeneous Dirichlet boundary condition.

The Coffey–Evans equation has two interesting features: first, it is a symmetric problem, and second it has triplets of close eigenvalues and the closeness increases dramatically as the parameter β increases. Accurately discriminating these close but different eigenvalues is a challenge.

To exploit this symmetry, `PysliseHalf` can be used. Besides this, the few lines of Python needed to find the first eigenvalues are very similar to the other examples.

```python
from pyslise import PysliseHalf
from math import pi, sin, cos

beta = 15
def V(x):
    return beta**2 * sin(2*x)**2 - 2*beta * cos(2*x)

coffey_evans = PysliseHalf(V, pi/2, tolerance=1e-8)
print(coffey_evans.eigenvaluesByIndex(0, 10, (0,1)))
```

In the last line we notice that supplying only one boundary condition is sufficient; symmetric boundary conditions are expected when using a half-range reduction. In table 2.4, we present for $\beta = 5$, $\beta = 15$ and $\beta = 25$ the results of our experiment. Here the reference values of the first nine eigenvalues are

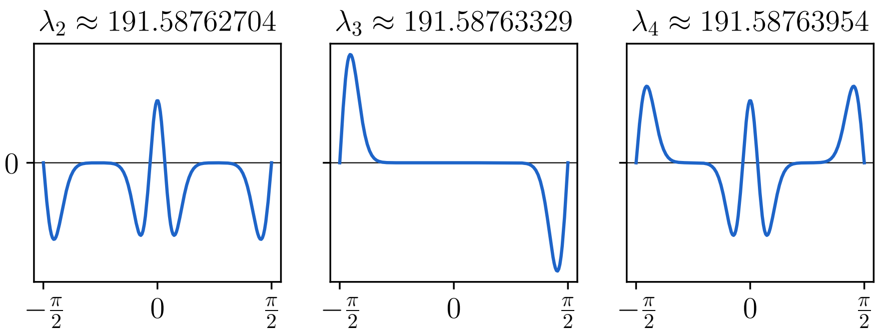


Figure 2.20: A plot of the eigenfunctions y_2, y_3 and y_4 of the Coffey–Evans problem (2.36) with $\beta = 25$.

computed with `Matslise 2.0` with a tolerance of 10^{-14}. In the next column the errors obtained with `Matslise 2.0` and a tolerance of 10^{-8} are given. In the last column, the errors of `Pyslise` with the same tolerance of 10^{-8} are presented.

Besides the numerical results, also the computational running time is reported in table 2.4. Similarly to the results from the Mathieu problem, `Pyslise` was significantly faster and more accurate. But in contrast to the previous example, now one could argue that `Matslise 2.0` no longer respects the requested accuracy of 10^{-8} for $\beta = 25$. However, the relative error is sufficiently small.

As seen previously, eigenfunctions can be visualized. If we want to recreate graphs similar to figure 2.20, the following code can be used.

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-pi/2, pi/2, 200)

for i, E, f in coffey_evans.eigenpairsByIndex(2, 5, (0, 1)):
    plt.plot(x, f(x)[0,:])
    plt.show()
```

Notice here that by providing the arguments `2` and `5` to `.eigenpairsByIndex`, we only request the eigenvalues and eigenfunctions with index 2, 3 and 4. Visually these graphs look convincing. But, as a more rigorous verification we

	$\beta = 5$	`Matslise 2.0`	`Pyslise`
		8.94ms	0.33ms
0	0.000 546 303 058 8	$4.5 \cdot 10^{-10}$	$2.4 \cdot 10^{-10}$
1	17.420 130 075 140 1	$3.1 \cdot 10^{-9}$	$1.4 \cdot 10^{-11}$
2	26.831 329 487 520 7	$8.2 \cdot 10^{-9}$	$5.2 \cdot 10^{-12}$
3	30.651 459 704 462 5	$1.1 \cdot 10^{-9}$	$-3.2 \cdot 10^{-12}$
4	37.970 984 812 717 8	$-1.6 \cdot 10^{-8}$	$9.2 \cdot 10^{-12}$
5	49.410 037 596 678 3	$-6.0 \cdot 10^{-9}$	$-1.8 \cdot 10^{-11}$
6	62.211 503 944 335 8	$-6.7 \cdot 10^{-9}$	$-3.7 \cdot 10^{-12}$
7	76.999 768 785 165 0	$-3.1 \cdot 10^{-8}$	$-4.1 \cdot 10^{-11}$
8	93.892 340 837 247 8	$-1.5 \cdot 10^{-8}$	$-4.5 \cdot 10^{-11}$
	$\beta = 15$	`Matslise 2.0`	`Pyslise`
		9.71ms	0.54ms
0	0.000 000 000 000 3 5	$-1.7 \cdot 10^{-7}$	$-1.1 \cdot 10^{-11}$
1	57.883 306 848 610 8	$3.2 \cdot 10^{-7}$	$4.8 \cdot 10^{-11}$
2	111.202 333 351 229 3	$1.9 \cdot 10^{-7}$	$1.1 \cdot 10^{-10}$
3	111.227 069 441 859 2	$3.1 \cdot 10^{-7}$	$4.9 \cdot 10^{-11}$
4	111.251 880 818 312 7	$1.9 \cdot 10^{-7}$	$1.1 \cdot 10^{-10}$
5	159.182 672 432 314 4	$5.3 \cdot 10^{-9}$	$1.7 \cdot 10^{-10}$
6	197.332 858 363 147 6	$6.3 \cdot 10^{-8}$	$-7.5 \cdot 10^{-11}$
7	199.869 005 533 835 6	$2.6 \cdot 10^{-9}$	$6.3 \cdot 10^{-13}$
8	203.229 529 502 001 5	$2.7 \cdot 10^{-8}$	$-2.1 \cdot 10^{-10}$
	$\beta = 25$	`Matslise 2.0`	`Pyslise`
		24.36ms	0.61ms
0	−0.000 000 000 000 000 0	$-4.4 \cdot 10^{-7}$	$-2.4 \cdot 10^{-11}$
1	97.934 561 686 363 7	$7.9 \cdot 10^{-7}$	$-3.2 \cdot 10^{-11}$
2	191.587 627 039 665 6	$8.3 \cdot 10^{-7}$	$4.8 \cdot 10^{-11}$
3	191.587 633 291 399 9	$1.2 \cdot 10^{-6}$	$-1.3 \cdot 10^{-11}$
4	191.587 639 543 137 1	$7.2 \cdot 10^{-7}$	$4.6 \cdot 10^{-11}$
5	280.614 245 270 678 6	$1.7 \cdot 10^{-7}$	$1.7 \cdot 10^{-10}$
6	364.551 423 917 178 6	$6.2 \cdot 10^{-8}$	$-4.4 \cdot 10^{-12}$
7	364.555 644 201 143 3	$-8.5 \cdot 10^{-8}$	$2.2 \cdot 10^{-10}$
8	364.559 865 747 062 5	$6.1 \cdot 10^{-8}$	$2.4 \cdot 10^{-10}$

Table 2.4: Eigenvalues for the Coffey–Evans problem (2.36), with absolute errors obtained by `Matslise 2.0` and `Pyslise` with a tolerance of 10^{-8}.

$\beta = 5$	y_0	y_1	y_2	y_3
`Matslise 2.0`	$2.0 \cdot 10^{-11}$	$3.2 \cdot 10^{-10}$	$1.5 \cdot 10^{-9}$	$2.1 \cdot 10^{-10}$
`Pyslise`	$4.2 \cdot 10^{-9}$	$8.5 \cdot 10^{-10}$	$1.4 \cdot 10^{-9}$	$1.4 \cdot 10^{-9}$
$\beta = 5$	y_4	y_5	y_6	y_7
`Matslise 2.0`	$1.5 \cdot 10^{-9}$	$5.4 \cdot 10^{-10}$	$5.0 \cdot 10^{-10}$	$2.2 \cdot 10^{-9}$
`Pyslise`	$1.9 \cdot 10^{-9}$	$1.4 \cdot 10^{-9}$	$1.8 \cdot 10^{-9}$	$1.8 \cdot 10^{-9}$

$\beta = 15$	y_0	y_1	y_2	y_3
`Matslise 2.0`	$2.8 \cdot 10^{-9}$	$1.2 \cdot 10^{-8}$	$4.0 \cdot 10^{-6}$	$1.7 \cdot 10^{-5}$
`Pyslise`	$2.0 \cdot 10^{-10}$	$7.3 \cdot 10^{-10}$	$4.6 \cdot 10^{-9}$	$5.7 \cdot 10^{-9}$
$\beta = 15$	y_4	y_5	y_6	y_7
`Matslise 2.0`	$4.0 \cdot 10^{-6}$	$1.8 \cdot 10^{-10}$	$1.2 \cdot 10^{-8}$	$1.1 \cdot 10^{-9}$
`Pyslise`	$3.5 \cdot 10^{-9}$	$2.3 \cdot 10^{-9}$	$2.2 \cdot 10^{-9}$	$3.3 \cdot 10^{-9}$

$\beta = 25$	y_0	y_1	y_2	y_3
`Matslise 2.0`	$4.6 \cdot 10^{-9}$	$1.9 \cdot 10^{-8}$	$8.2 \cdot 10^{-2}$	$3.1 \cdot 10^{-1}$
`Pyslise`	$2.0 \cdot 10^{-10}$	$6.9 \cdot 10^{-10}$	$7.0 \cdot 10^{-6}$	$6.5 \cdot 10^{-6}$
$\beta = 25$	y_4	y_5	y_6	y_7
`Matslise 2.0`	$6.5 \cdot 10^{-2}$	$3.7 \cdot 10^{-9}$	$7.8 \cdot 10^{-6}$	$2.9 \cdot 10^{-5}$
`Pyslise`	$7.0 \cdot 10^{-6}$	$2.2 \cdot 10^{-9}$	$2.8 \cdot 10^{-8}$	$2.7 \cdot 10^{-9}$

Table 2.5: Maximum error in 100 equidistant points of the eigenfunction corresponding to each of first eight eigenvalues for the Coffey–Evans problem (2.36) with $\beta = 5$, $\beta = 15$ and $\beta = 25$.

$\beta = 15$	$n = 100$	$n = 1000$	$n = 10000$
`Matslise 2.0`	59.9ms	525ms	7508ms
`Pyslise`	0.344ms	2.58ms	24.5ms

Table 2.6: Computational time used to evaluate the first ten eigenfunctions in n equidistant points for the Coffey–Evans problem (2.36) with $\beta = 15$.

have provided table 2.5. Here the maximum absolute errors in the eigenfunctions are tabulated. Again, we have used `Matslise 2.0` with a tolerance of 10^{-14} to obtain reference results to compare with. For $\beta = 5$, both programs reach the requested accuracy. When $\beta = 15$, `Matslise 2.0` starts to lose accuracy, especially for y_2, y_3 and y_4. Only for $\beta = 25$, starts `Pyslise` to lose accuracy as well, albeit much less dramatic than `Matslise 2.0`.

From the theory we expect that the time needed to evaluate the eigenfunction does not depend on the difficulties to find eigenvalues for that particular problem. Once an eigenvalue is found, the corresponding eigenfunction should be easily calculated. The computation time to evaluate an eigenfunction for an equidistant grid of n points is reported in table 2.6. The remarkable speed-up we were able to achieve with our new implementation can be observed.

2.5.5 Truncated hydrogen potential

As a next example we will take a look at the truncated hydrogen potential [80]:

$$V(x) = -\frac{1}{x} + \frac{2}{x^2}, \tag{2.37}$$

on the domain $[0, 1000]$ with homogeneous Dirichlet boundary conditions. The lower eigenvalues are a good approximation for the eigenvalues of the non-truncated problem (on the domain $x \in [0, \infty[$). By now, the code to solve this problem may seem very familiar.

```python
from pyslise import Pyslise

def V(x):
    return -1/x + 2/x**2

hydrogen = Pyslise(V, 0, 1000, tolerance=1e-8)
print(hydrogen.eigenvaluesByIndex(0, 10, (0, 1), (0, 1)))
```

	`Matslise 2.0`	`Pyslise`
	30.96ms	7.09ms
$-0.062\,500\,000\,000\,0$	$2.1 \cdot 10^{-12}$	$8.9 \cdot 10^{-15}$
$-0.027\,777\,777\,777\,8$	$4.9 \cdot 10^{-12}$	$-2.2 \cdot 10^{-15}$
$-0.015\,625\,000\,000\,0$	$3.4 \cdot 10^{-12}$	$-9.0 \cdot 10^{-16}$
$-0.010\,000\,000\,000\,0$	$9.4 \cdot 10^{-13}$	$-2.7 \cdot 10^{-15}$
$-0.006\,944\,444\,444\,4$	$-1.1 \cdot 10^{-12}$	$1.2 \cdot 10^{-13}$
$-0.005\,102\,040\,816\,3$	$-2.3 \cdot 10^{-12}$	$-6.3 \cdot 10^{-13}$
$-0.003\,906\,250\,000\,0$	$-2.9 \cdot 10^{-12}$	$-2.3 \cdot 10^{-13}$
$-0.003\,086\,419\,753\,1$	$-3.0 \cdot 10^{-12}$	$1.1 \cdot 10^{-13}$
$-0.002\,500\,000\,000\,0$	$-2.7 \cdot 10^{-12}$	$-9.6 \cdot 10^{-14}$
$-0.002\,066\,115\,702\,5$	$-1.9 \cdot 10^{-12}$	$1.6 \cdot 10^{-14}$

Table 2.7: The first ten eigenvalues for the truncated hydrogen problem (2.37), the execution times and the errors obtained with `Matslise 2.0` and `Pyslise` for a tolerance of 10^{-8}.

The potential is unbounded for the left endpoint of the integration interval, thus this is a non-regular problem. `Matslise 2.0` is well-suited to solve such singular problems. It contains many checks to identify and routines to work around singularities of the problem. For efficiency reasons, `Pyslise` does not have these extra features.

Despite these missing features, `Pyslise` can still solve the problem. It is even faster and more accurate. We do note that speedup is less significant than for non-singular problems. In table 2.7 we see that `Pyslise` is at least as accurate as `Matslise` and approximately four times faster (in contrast to 20 times for the Mathieu en Coffey–Evans problems).

For the evaluation of the eigenfunctions, this singularity does not matter. The eigenfunctions are evaluated with a maximal error of less than 10^{-9}, with timings similar to the previous test problems.

2.5.6 Periodic problem with an asymmetric potential

As described in section 2.4, `Matslise 3.0` is able to solve Schrödinger problems with periodic boundary conditions. To demonstrate this we will solve a problem also found in [3]. Consider the Schrödinger equation

$$-y''(x) + x^2(\pi - x)y(x) = \lambda y(x)$$

on the domain $[0, \pi]$ with periodic boundary conditions: $y(0) = y(\pi)$ and $y'(0) = y'(\pi)$.

To solve this with `Pyslise` we need to import a different solver, but most of the code is similar.

```python
from pyslise import PyslisePeriodic
from math import pi

def V(x):
    return x*x*(pi-x)

problem = PyslisePeriodic(V, 0, pi, tolerance=1e-8)

print(problem.eigenvaluesByIndex(0, 20))
```

In this code there are two notable differences to the previous examples. First, `PyslisePeriodic` is used as solver. And second, no boundary conditions are provided to the `.eigenvaluesByIndex` function. These are unnecessary because periodic boundary conditions are implied by `PyslisePeriodic`.

Another difference can be found in the output:

```
[
  (0, 2.0294161514952878, 1), (1, 6.500490696092834, 1),
  ...,
  (18, 326.5881759903209, 1), (19, 402.5834632545595, 1)
]
```

The function `.eigenvaluesByIndex` returns a list of all eigenvalues found. Now for each eigenvalue the tuple has three elements: the index, the eigenvalue and the multiplicity. From the theory, we know that for periodic problems, eigenvalues no longer are guaranteed to be unique. An eigenvalue can have a double multiplicity, and if `Pyslise` detects this, multiplicity 2 is returned. For example, if we request much higher eigenvalues with the line

```python
print(problem.eigenvaluesByIndex(1000, 1005))
```

then the following output is generated.

```
[
  (999, 1000002.5873939216, 2), (1001, 1004006.5873655227, 2),
  (1003, 1008018.5849614257, 2), (1005, 1012038.5812170412, 2)
]
```

	Andrew[3]	Vanden Berghe[100]	`Pyslise` (10^{-12})
0			2.029 416 151 495
1	6.500 5	6.500 49	6.500 490 696 093
2	7.015 1	7.015 06	7.015 056 863 171
3	18.584 8	18.584 77	18.584 772 142 361
4	18.665 5	18.665 48	18.665 481 445 221
5	38.581 6	38.581 62	38.581 627 865 277
6	38.621 5	38.621 54	38.621 542 547 363
7	66.582 1	66.582 04	66.582 047 792 290
8	66.605 4	66.605 37	66.605 365 007 694
9	102.582 5	102.582 52	102.582 525 988 841
10	102.597 7	102.597 72	102.597 720 549 763
11	146.582 9	146.582 86	146.582 865 091 809
12	146.593 5	146.593 52	146.593 522 877 933
13	198.583 1	198.583 10	198.583 098 056 935
14	198.591 0	198.599 98	198.590 975 696 561
15	258.583 3	258.583 26	258.583 260 746 092
16	258.589 3	258.589 31	258.589 315 759 431
17	326.583 4	326.583 38	326.583 377 751 489
18	326.588 2	326.588 17	326.588 175 990 321
19	402.583 5	402.583 47	402.583 463 254 559

Table 2.8: The first twenty eigenvalues for the periodic problem from section 2.5.6.

These higher eigenvalues are degenerate, so `Pyslise` returns multiplicity 2. This can also be seen in the index of the returned eigenvalues.

In [3], the first twenty eigenvalues are computed to four digits behind the decimal point. In [100], an extra fifth digit is computed. `Matslise 3.0` produces much more accurate results. In table 2.8, we have tabulated the results from [3], from [100] and the results obtained with `Pyslise` and a tolerance of 10^{-12}. Our computed values agree with the literature up to all five digits, except for the last eigenvalue where we have found a different rounding.

As an illustration we present the graph of the first three eigenfunctions in figure 2.21.

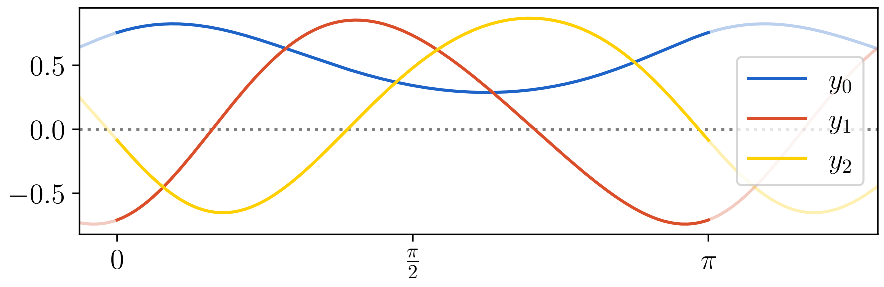


Figure 2.21: A plot of the first three eigenfunctions y_0, y_1 and y_2 with eigenvalues $\lambda_0 \approx 2.029$, $\lambda_1 \approx 6.500$ and $\lambda_0 \approx 7.015$ of the periodic problem from section 2.5.6.

2.5.7 Sturm–Liouville problems

As a last example we present the following Sturm–Liouville problem from [91]:

$$-\left((1+x)^2 y'(x)\right)' + \left(x^2 - 2\right) y(x) = \lambda e^x y(x)$$

on the domain $[0, 1]$ with a homogeneous Dirichlet boundary condition on the left and a homogeneous Neumann boundary condition on the right: $y(0) = 0$ and $y'(1) = 0$.

Solving this in `Pyslise` can be achieved with the following program.

```python
from pyslise import SturmLiouville
from math import exp

def p(x):
    return (1 + x)**2

def q(x):
    return x * x - 2

def w(x):
    return exp(x)

slp = SturmLiouville(p, q, w, 0, 1, tolerance=1e-8)
print(slp.eigenvaluesByIndex(0, 10, (0, 1), (1, 0)))
```

Siedlecki [91]	`Matslise 2.0`	`Pyslise` error
1.170 492 62	1.170 492 759 902	$2.2 \cdot 10^{-12}$
26.863 361 7	26.863 367 646 410	$8.3 \cdot 10^{-12}$
78.549 002 6	78.549 045 264 140	$2.4 \cdot 10^{-12}$
156.079 996	156.080 156 224 823	$7.2 \cdot 10^{-12}$
259.455 041	259.455 476 210 930	$4.7 \cdot 10^{-12}$
388.673 823	388.674 790 600 773	$5.7 \cdot 10^{-12}$
543.736 155	543.738 037 129 327	$5.2 \cdot 10^{-12}$
724.641 861	724.645 192 282 213	$4.5 \cdot 10^{-13}$

Table 2.9: Comparing numerical results for the problem from section 2.5.7. In the right most column the absolute error of our implementation with a specified tolerance of 10^{-8} are reported.

Here we are using the `SturmLiouville` solver. We now have to specify the three defining functions $p(x)$, $q(y)$ and $w(x)$. Notice that in the last line we have specified $y(0) = 0$ and $y'(0) = 1$ to mark the left boundary condition, and $y(1) = 1$ and $y'(1) = 0$ to mark the right boundary conditions. Remember that solutions will only satisfy these boundary conditions after some rescaling.

Under the hood, the `SturmLiouville` solver will compute Liouville's transformation as in section 2.1.2. This yields a Schrödinger problem for which `Matslise 3.0` will use the constant perturbation method, just as for the `Pyslise` solver.

In table 2.9, we have compared the accurate values from `Matslise 2.0` to the reported values in [91] and our new implementation. Note that for our implementation the absolute errors are reported, which means that the relative error is close to machine precision.

2.6 Matslise 3.0 – A new implementation

Later on in chapters 3 and 4, we develop methods to solve time-independent two-dimensional Schrödinger problems. These methods will depend upon our ability to solve the one-dimensional problem accurately and efficiently. Not only eigenvalues will be required, but evaluating eigenfunctions will be essential.

This need for speed drove us away from `Matslise` or `Matslise 2.0`. These are efficient and accurate packages which mainly focus on the computation of the eigenvalues with automatic regularity detection. In our case, we need a package that allows the fast and accurate solution of the eigenvalues as well as of the eigenfunctions of regular one-dimensional Schrödinger-problems. We investigate possible bottlenecks of performance of the `Matslise 2.0` package and its routines.

Let us start with the computation of the eigenvalues. The original `Matslise` package as well as its successor `Matslise 2.0` are highly optimized packages, in the sense that they are designed to compute the eigenvalues of various types of regular and irregular Schrödinger and Sturm–Liouville problems as efficiently as possible. For this, only one coarse mesh, typically consisting of only a few mesh points, is determined. The mesh is then used for all eigenvalue computations. This means that all evaluations of the coefficient functions $C_i^{(q)}$ from theorem 2.10 are performed during the construction of the mesh and are saved for later reuse. This makes the actual computation of the eigenvalues, at least in an interpreted language such as MATLAB, as fast as possible. For the actual computation of the eigenvalues, the main bottleneck of performance of `Matslise 2.0` is MATLAB. This environment creates an interpreted language which is inherently slower than a compiled one like `Fortran` or `C`. However, using MATLAB has the great benefit of being user-friendly. Even less seasoned programmers are able to easily implement their Sturm–Liouville problem. In `Matslise 2.0`, a graphical user interface is provided to aid in specifying the user's problem, without any programming knowledge needed.

Since we wanted to reimplement `Matslise` into a faster compiled language, we had to consider what we would gain but also what we would lose. Speed and efficiency only go so far if the program is implemented in a prohibitively difficult package to use. As stated, one of the great strengths of `Matslise 2.0` is its ease of use. We were hesitant to use compiled languages for this reason.

In the current day and age, one of the more popular languages, with a larger community than `Fortran`'s or MATLAB's is `python`. Like MATLAB, this language is interpreted, which makes it equally slow [20, 97]. However, in `python` it is not hard to add native packages. These packages are not written in `python` but in some other, more low-level, language. This other language is compiled to native machine code, and nicely packaged to be called from within `python`.

These native packages are the answer to the efficiency versus user-friendliness problem we were faced with. The high performing part can be efficiently implemented in a compiled language, but the code that the user will develop can be written in the more approachable `python`. The last question before we could build our new implementation of `Matslise` was: "Which compiled language?". This language should be able to support the building of `python`-packages, it should have support for linear algebra, preferably with optimized BLAS libraries, and it should support code architecture for numerical libraries. All of these requests makes `C++` a prime candidate. In combination with `pybind` [53] for the `python` compatibility and `Eigen` v3 [33] as linear algebra library we have the perfect tools to satisfy all our needs.

Try it out!
The new implementation, which we called `Matslise 3.0`, can be accessed and executed in three different ways.

- As a `C++` program: the source code of the package is hosted at `https://github.com/twist-numerical/matslise`. There, instructions to compile it can be found.
- As a `python`-package: in order to avoid that users need to compile the packages themselves, we also provide a 64-bit prebuilt python package `Pyslise`, for `Linux`, `Windows` and `macOS`. Installing `Pyslise` is as easy as running:

```
pip install pyslise
```

- Online: lastly we also have created the possibility to run this code inside the web browser: `https://matslise.ugent.be/ti1d`.

2.6.1 Implementation challenges

Building a new implementation from an existing package allows for incorporating some learned lessons from the previous implementations, and simplifying some functionality. In practice this is harder than it sounds. For example, when simplifying some functionality one has to carefully consider why the original code did not include this simplification. Did the programmer forget to consider it? Was it a remnant of a previous version, or of removed code? Or something else entirely? In the hard way, I have learned that, in numerical algorithms especially, the original programmer probably had very good reasons not to include something. More times than I would like to admit, I simplified some code of `Matslise 2.0` to include in the new implementation, only to discover that my 'simplification' did not cover all edge-cases or all possible scenarios. In the best cases I made these discoveries quickly upon testing that code. Other times these discoveries only occurred a few months or even years later when getting stuck on a particular difficult bug.

In this section, we want to highlight some challenges we encountered while building the new implementation. Some of these are a consequence of the choice of language we have made, some are some inherent challenges of the constant perturbation method and others are self-inflicted by the ambitious goals we set out for ourselves.

2.6.1.1 Computation of the perturbation term formulae

In theorem 2.10, a nice procedure is outlined to determine each of the propagators $u(\delta)$ and $v(\delta)$. In short both propagators are written in the following form:

$$u(\delta) = \sum_{q=-1}^{Q} c_q(\delta)\eta_q(\delta)$$

with c_i a polynomial in δ,

$$c_q(\delta) = \sum_{i=0}^{N} d_{q,i}\delta^i, \tag{2.38}$$

and $d_{q,i}$ is a polynomial in h with coefficients in $V_1, \dots, V_N$.

Once a mesh is constructed and $V_1, \dots, V_N$ are determined, the values for $d_{q,i}$ are fixed on each subinterval. This means that these values can be computed once and be reused for each guess for λ. In practice, the formulae for the values of $d_{i,j}$ are quite complicated. For small values of N and Q, it would be feasible to implement these values by hand, for the large N and Q we are using here, this is intractable.

We could compute these formulae each time we construct a mesh by implementing some symbolic computation in `C++`. This could work, but it is less than ideal. Not only would this include a large computational burden, it would also be quite labor-intensive to implement the symbolic tools needed to pull this off. A better strategy is to use a symbolic toolbox like `maple`, `mathematica`, or `sage` and let it generate the necessary `C++` code only once, before compiling the program. Computing the formulae once and generating code for them, also has the added benefit that the `C++` compiler can optimize them and generate the best possible machine-code.

In the appendices of [67], some `maple` programs are present which are able to determine the necessary values if $h = \delta$. For our case, because we want to have these formulae for general δ, a new program needs to be written. In the next chapter, the method we develop also uses constant perturbation formulae for

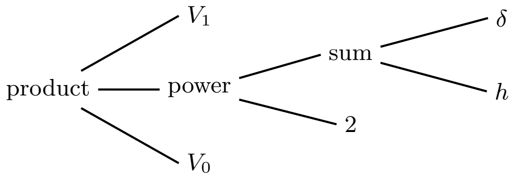


Figure 2.22: A possible tree of operations for $V_1(\delta + h)^2 V_0$.

coupled systems of Schrödinger equations, therefore it would also be valuable if this new program is also able to compute these. And lastly, it would be a nice-to-have if these formulae could be generated in a matter of minutes instead of weeks, for example.

We have implemented this in `sage` [89]. Here it is possible to precisely specify what type each variable is. Mathematically, all expressions are polynomials in h, δ and $V_0, \dots, V_N$. As such, all computations will be executed in the following polynomial ring:

$$\mathbb{Q}[V_0, \dots, V_n][h][\delta].$$

This means that all expressions in `sage` will have the following structure with coefficients $r \in \mathbb{Q}$:

$$\sum_i \left(\sum_j \left(\sum_{k_0, \dots, k_N} r_{i,j,k_0,\dots,k_N} V_0^{k_0} \cdots V_N^{k_N} \right) h^j \right) \delta^i.$$

This structure allows `sage` to optimize each expression, in contrast to working with general symbolic expressions, which would also allow other operations such as division or sin for example.

As an example, the `sage` program starts with defining the variables V_0, V_1, V_2, h and δ.

```
pr_V = QQ['V0', 'V1', 'V2']
pr_h = pr_V['h']
pr_delta = pr_h['delta']

V0, V1, V2 = pr_V.gens()
h = pr_h.gen()
detla = pr_delta.gen()
```

Now we can write an expression such as $V_1(\delta+h)^2V_0$, which `sage` would immediately translate to the expanded expression:

$$V_0V_1\delta^2+2V_0V_1h\delta+V_0V_1h^2.$$

This expression is much more efficient to work with. For the first expression, a computer algebra system has to save this as a tree of operations, see figure 2.22. This tree is cumbersome and relatively slow to work with. By specifying all variables to be polynomials, `sage` can save these much more efficiently as a few simple arrays. For small expressions the difference may be negligible. But for the giant expressions we are working with, the difference is very significant.

Besides the computational benefit, we also have the ability to easily specify that $V_0,\dots,V_N$ are matrices when generating formulae for the constant perturbation method on coupled systems of equations. We do not specify $V_0,\dots,V_N$ to be in a polynomial ring, but in a general free algebra without commutation rules.

```
FreeAlgebra(QQ, ['V0', 'V1', 'V2'])
```

An expression such as $(V_1+V_0)^2$ will now internally be stored as:

$$V_0^2+V_0V_1+V_1V_0+V_1^2.$$

Using these polynomial rings, it is straightforward to implement the formulae from theorem 2.10. The last thing to do is generating the `C++`-code to compute the values $d_{q,i}$ from (2.38). For this we generate for each element of the propagation matrix $u(\delta)$, $v(\delta)$, $u'(\delta)$ and $v'(\delta)$ the $(Q+2)\times(N+1)$ matrix $\mathbf{T_u},\dots,\mathbf{T_{v'}}$ of formulae for $d_{q,i}$. For example:

$$(\mathbf{T_v})_{1,3}=h\cdot(-0.5V_1+h\cdot(0.5V_2+h\cdot(-0.5V_3+h\cdot(\dots)))).$$

As one may expect, this generates a huge file of code. In figure 2.23 we have provided the cleaned-up[13] generated code for $(T_u)_{4,10}$. This expression is one example of the over 300 complicated terms needed to generate the propagation matrix.

To aid the compiler, before generating the code, we implement a crude common subexpression elimination in `sage`. Because we are working in a multivariate polynomial ring (or a free algebra in the case of coupled systems), expressions

[13]In practice, we have to ensure all scalars have the right type, see section 2.6.1.2. So a simple fraction like $\frac{37}{80}$ has to be implemented as `Scalar(37L)/Scalar(80L)`, here is `Scalar` a template type variable.

```
Tu[4, 10] = (
  37/80*v1*v1*v2 - 1065/56*v3*v3 - 165/4*v2*v4 - 261/4*v1*v5
  + 19305/2*v8
  + h*(-279/80*v1*v2*v2 - 69/16*v1*v1*v3
  + 945/2*v3*v4 + 567*v2*v5 + 891*v1*v6 - 328185/2*v9
  + h*(-7/192*v1*v1*v1*v1 + 39/8*v2*v2*v2 + 691/20*v1*v2*v3
  + 185/8*v1*v1*v4 - 12405/8*v4*v4 - 13251/4*v3*v5
  - 16731/4*v2*v6 - 26235/4*v1*v7 + 2953665/2*v10
  + h*(31/96*v1*v1*v1*v2 - 513/10*v2*v2*v3 - 953/16*v1*v3*v3
  - 1061/8*v1*v2*v4 - 735/8*v1*v1*v5 + 15120*v4*v5
  + 16956*v3*v6 + 21978*v2*v7 + 137709/4*v1*v8 - 18706545/2*v11
  + h*(-43/48*v1*v1*v2*v2 - 2/3*v1*v1*v1*v3 + 11427/80*v2*v3*v3
  + 12517/80*v2*v2*v4 + 1443/4*v1*v3*v4 + 1697/4*v1*v2*v5
  + 4781/16*v1*v1*v6 - 57765/2*v5*v5 - 60447*v4*v6
  - 70110*v3*v7 - 92169*v2*v8 - 144144*v1*v9 + 93532725/2*v12
  + h*(-1/3840*v1*v1*v1*v1*v1 + 31/32*v1*v2*v2*v2
  + 313/96*v1*v1*v2*v3 + 61/48*v1*v1*v1*v4 - 1809/16*v3*v3*v3
  - 2943/4*v2*v3*v4 - 7319/16*v1*v4*v4 - 34047/80*v2*v2*v5
  - 7811/8*v1*v3*v5 - 11823/10*v1*v2*v6 - 67137/80*v1*v1*v7
  + 192726*v5*v6 + 208521*v4*v7 + 247401*v3*v8
  + 1311453/4*v2*v9 + 2045043/4*v1*v10 - 392837445/2*v13
  + h*(1/768*v1*v1*v1*v1*v2
  - 23/64*v2*v2*v2*v2 - 77/16*v1*v2*v2*v3 - 517/192*v1*v1*v3*v3
  - 181/32*v1*v1*v2*v4 - 217/96*v1*v1*v1*v5 + 6161/8*v3*v3*v4
  + 13289/16*v2*v4*v4 + 14079/8*v2*v3*v5 + 8655/4*v1*v4*v5
  + 21051/20*v2*v2*v6 + 9605/4*v1*v3*v6 + 118279/40*v1*v2*v7
  + 33663/16*v1*v1*v8 - 558747/2*v6*v6 - 577323*v5*v7
  - 640332*v4*v8 - 771309*v3*v9 - 1027026*v2*v10
  - 1594593*v1*v11
)))))))
```

Figure 2.23: The δ-dependent formula for term $(T_u)_{4,10}$, with $Q = 7$ and $N = 16$ as used by `Matslise 3.0`. In reality, this is only one of the over 300 terms needed to compute the propagation matrix, with each term just as complicated as this one.

which are the product of some V_i's such as $V_1^2 V_2$ will be common. And, many of them will be present in more than one formula. We can start the code generation by extracting some of these products in their own variables. For example:

```
...
v1_v2 = v1 * v2
v1_v1_v2 = v1 * v1_v2
...
```

In the scalar case, this works wonderfully. All tested compilers are able to process this generated file without many difficulties. For coupled systems, this generated file is much harder on the compiler because $V_0, \ldots, V_N$ are no longer simple scalars. They have become matrices from the `Eigen` library. `Eigen` relies upon inlining and other compiler optimizations to generate the best possible machine-code. For some compilers (most notably for the Microsoft Visual Studio compiler) this becomes too much, and they run out of available system memory. To combat this issue partially, we introduce more temporaries and split the giant formulae into more manageable simpler expressions.

2.6.1.2 Generalizing the scalar-type

One of the very cool features of the linear algebra library `Eigen` is that it does not care what kind of scalars you are working with. All their algorithms are implemented agnostic of the scalar type. This is achieved by using `C++`-templates. For readers unfamiliar with this concept we will provide a very brief summary of this concept. For a thorough description, see [93, chapter 23].

First, let us take a look at the `C`-functions to compute the square root of a floating point number.

```
float sqrtf(float arg);
double sqrt(double arg);
long double sqrtl(long double arg);
```

The `C`-standard has to provide three different functions for three different floating point types: `float` is 32 bits wide, `double` is 64 bits wide and `long` `double` is commonly[14] 80 bits wide.

It is cumbersome to have to differentiate between which function to call, depending on which argument types are used. In `C++` this is solved more elegantly, by allowing function overloads depending on the supplied arguments. But `C++` goes even further, a programmer can build a function for different types with the same code. For example, the following code implements the simple operations $x^2 + y^2$, for all possible types all at once.

```cpp
template<typename Number>
Number squared_norm(Number x, Number y) {
    Number a = x * x;
    Number b = y * y;
    return a + b;
}
```

This function can now be called with different numeric types.

```cpp
squared_norm<int>(3, 4);
squared_norm<float>(3.0f, 4.0f);
squared_norm<double>(3.0, 4.0);
```

The library `Eigen` is implemented by using these templates extensively. This library provides the `Eigen::Matrix<Scalar, rows, cols>` type which can be used to build for example the following matrices.

```cpp
// A 3 by 3 integer matrix
Eigen::Matrix<int, 3, 3>
```

```cpp
// A double-type column vector with a dynamic number of rows
Eigen::Matrix<double, Eigen::Dynamic, 1>
```

```cpp
// A complex 10 by 5 matrix, the real and imaginary part of
// each entry is saved in a long double format.
Eigen::Matrix<std::complex<long double>, 10, 5>
```

[14]The `C` and `C++`-standards give surprisingly vague definitions of numeric types. For example: the `int`-type is guaranteed to be *at least* 16 bits wide. The two floating point types `float` and `double` are atypical in the sense that they are guaranteed to be the IEEE-754 binary32 and binary64 formats respectively. For the `long double` type, the specification is less strict: *"extended precision floating-point type. Matches IEEE-754 binary128 format if supported, otherwise matches IEEE-754 binary64-extended format if supported, otherwise matches some non-IEEE-754 extended floating-point format as long as its precision is better than binary64 and range is at least as good as binary64, otherwise matches IEEE-754 binary64 format."* On x86 systems, this is a 80 bit wide floating point type with a 15 bits exponent and 64 bits significand.

```cpp
#include <iostream>
#include <boost/format.hpp>
#include <boost/math/constants/constants.hpp>
#include <boost/multiprecision/float128.hpp>
#include <matslise/matslise.h>

using boost::math::constants::pi;
using boost::multiprecision::float128;

float128 mathieuPotential(float128 x) {
    return 2 * cos(2 * x);
}

int main() {
    matslise::Matslise<float128> problem(
            &mathieuPotential, 0, pi<float128>(), 1e-25q);

    auto boundary = matslise::Y<float128>::Dirichlet();
    auto eigs = problem.eigenvaluesByIndex(0, 7, boundary);
    for (auto [i, E]: eigs) {
        auto error = problem.eigenvalueError(E, boundary, i);
        std::cout << boost::format(
                "Eigenvalue %1$d:%2$30.25f  (error: %3$.1e)")
                        % i % E % error << std::endl;
    }
    return 0;
}
```

```
Eigenvalue 0:  -0.1102488169920951699065478  (error: 1.7e-25)
Eigenvalue 1:   3.9170247729984711867034169  (error: 1.6e-25)
Eigenvalue 2:   9.0477392598093749823749465  (error: 1.9e-25)
Eigenvalue 3:  16.0329700814057944092457252  (error: 4.3e-25)
Eigenvalue 4:  25.0208408232897663652258825  (error: 4.8e-25)
Eigenvalue 5:  36.0142899106282223466625724  (error: 7.7e-25)
Eigenvalue 6:  49.0104182494238719005911294  (error: 9.8e-25)
```

Figure 2.24: A sample program to compute the first eight eigenvalues of the Mathieu problem from section 2.5.3, in 128 bits wide floating point precision, and the output generated.

In our implementation we have also adopted this philosophy by using templates to allow for any scalar type. This means that `Matslise 3.0` is not only able to solve Sturm–Liouville equations in `double`-precision (like `Matslise 2.0`) but also in the more accurate `long double`-precision. We even have enabled support for a 128 bits floating point numeric type by using Boost's [17] `boost::multiprecision::float128` type.

Unfortunately, `python` only has support for the double type. So, if we want to leverage this higher precision, we will have to write a `C++` program. As an example, in figure 2.24 we provide the code to find the first few eigenvalues of the Mathieu problem from section 2.5.3 in quadruple precision (128 bit).

2.6.1.3 Memory management

In computer science, a garbage collector is a method to do automatic memory management. The idea here is to automatically detect if allocated memory (for an array or object for example) is no longer used. If such memory is found, this can be returned to the operating system, for later use by our own or another program. Detecting unused memory is essential to ensure a reliable working of the system. If one program hoards memory, without returning it to the operating system, then sooner or later the operating system runs out of memory and programs need to be unexpectedly aborted.

To avoid this, many interpreted languages (for example `python`, `javascript`, MATLAB...) have such a garbage collector built in. Even some compiled languages (such as `java` or `C#`) do memory management with a garbage collector. But this is not the only way to manage memory. The language `rust` for example, keeps track of used memory at compile time, with what they call *lifetimes.* `C` and `C++` on the other hand, have no memory management built in, these languages rely on the programmer to clean up after themselves.

In theory, cleaning up after yourself is easy. When you allocate data, just make sure you deallocate it once you no longer need it. In practice, this view is too simple. In `C++` some memory management is handled automatically by constructors and destructors. But other, more complicated scenarios, still have to be handled by the programmer. The situation that caused problems was the interoperability with `python`. We had to specify very carefully whose responsibility it was to clean up objects. Was it an object that was managed on the `C++` side by us, the programmer, or could we let the garbage collector of `python` do the work? Luckily, `pybind` supports `std::shared_ptr<...>`. Using this solves the responsibility issue in most cases.

The only problematic case left, is when the lifetime of one object depends upon another object. Consider for example the following `python` code, which computes and stores the sixth eigenfunction of three different Schrödinger problems.

```python
from pyslise import Pyslise

some_functions = []
for q in [1, 10, 100]:
    problem = Pyslise(lambda x: q*x, 0, 1)
    i, E, f = problem.eigenpairsByIndex(5, 6, (0, 1))[0]
    some_functions.append(f)
```

Here, in the body of the loop, the eigenfunction object `f` assumes that the original `Pyslise` object `problem` exists. If `problem` were destroyed, evaluating the eigenfunction `f` would trigger undefined behavior. In the best case, this would crash the program, in the worst case wrong results would be returned. To ensure `problem` not to be destroyed too early we have to explicitly mark a dependence of each of the returned eigenfunctions on the original problem.

In hindsight, these issues and fixes may seem obvious. However, they highlight some difficulties in getting `python` and `C++` to play nicely together.

2.6.1.4 Automatic testing

Another challenge we want to focus on is an often underappreciated issue in mathematical software development. When programming, there is an almost universal truth: all software contains bugs. An art in writing software is maximizing the chance you catch bugs early. One of the best tools available for this is automatic tests. These tests should be extremely easy to run and require no input from the programmer whatsoever. They should just result in a pass/fail condition. Using a well established test framework makes writing tests a breeze, and encourages implementing more tests when new functionality is added or when new edge-cases are discovered.

We have chosen to use `catch` [39] as a test framework, and we implemented 43 test scenarios with over 2 000 000 conditions to check. These tests range from simple unit-tests to ensure the Legendre-polynomials are computed correctly or that our program is able to evaluate the η_i functions in all floating point types, to full integration tests where a Sturm–Liouville problem is solved, and the accuracy is verified. As a summary we provide a small description of some of these automatic tests.

- Some simple unit-tests of some auxiliary procedures.
  - Least squares approximations of trigonometric functions with shifted Legendre polynomials.
  - The evaluation of the η_i-funtions in many points for all scalar types.
  - The domain and the Schrödinger potential after Liouville's transformation applied to $p(x) = \cos(x)$, $q(x) = 0$ and $w(x) = \tan(x)\sin(x)$.
- Some Sturm–Liouville problems with exact known eigenvalues.
  - The Schrödinger problem with $V(x) = 0$ on $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ with homogeneous Dirichlet boundary conditions. The first 50 eigenfunctions are compared in many points to the analytical solution.
  - The Schrödinger problem with $V(x) = x^2$ on $[-15, 15]$ with homogeneous Dirichlet boundary conditions. The first 50 eigenvalues and the orthonormality of the eigenfunctions are verified, in `double`, `long double` and `float128`.
  - Klotter's problem [58] with $p(x) = 1$, $q(x) = \frac{3}{4x^2}$ and $w(x) = \frac{64\pi^2}{9x^6}$ on $\left[\frac{8}{7}, 8\right]$.
- Some Schrödinger problems with eigenvalues found in the literature and with `Matslise 2.0`.
  - The Mathieu problem $V(x) = 2\cos(2x)$ on $[0, \pi]$. The first 200 eigenvalues and the orthonormality of the eigenfunctions are checked, in `double`, `long double` and `float128`. The first and fourth eigenfunctions are compared in different values computed with `Matslise 2.0`.
  - Marletta's problem: $V(x) = 3\frac{x-31}{4(x+1)(x+4)^2}$ on $[0, 12]$ with $y(0) = 0$ and $5y(12) + 8y'(12) = 0$.
  - The Coffey–Evans problem: $V(x) = -2\beta\cos(2x) + \beta^2\sin^2(2x)$ on $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ with homogeneous Dirichlet boundary conditions and half-range reduction for $\beta \in \{20, 25\}$ with all scalar types.
- Sturm–Liouville problems from [91] are checked to their reported values. For the second and third problem, the reported results in [91] are not as accurate as we desire, so `Matslise 2.0` was used to provide more accurate values.
  - $p(x) = 1$, $q(x) = 0$ and $w(x) = (1+x)^{-2}$ with homogeneous Dirichlet boundary conditions on $[0, 1]$.

  - $p = (1 + x)^2$, $q(x) = x^2 - 2$ and $w(x) = \exp(x)$ on $[0, 1]$ with homogeneous Dirichlet boundary conditions left and homogeneous Neumann conditions on the right.
  - $p(x) = 2 + \sin(2\pi x)$, $q(x) = -10$ and $w(x) = 1 + \sqrt{x}$ on $[0, 1]$ with boundary conditions $y(0) = 0$ and $10y(1) + p(1)y'(1) = 0$. The first three eigenfunctions are also compared to values computed with `Matslise 2.0`.

- The following Schrödinger problems with periodic boundary conditions from [3].
  - The first 20 eigenvalues of the periodic problem with potential $V(x) = x^2(\pi - x)$.
  - For $V(x) = x^2(\pi - x)$ a selection of eigenvalues with index up to 40.

All these tests can be easily run during development. They are automatically run on each code change with the help of GitHub Actions. Here the program is compiled, and the tests are executed on a clean test system on GitHub's servers for `Linux`, `macOS` and `Windows`. This also builds and tests the Python-package. Working with GitHub Actions for automated testing allows us to quickly detect code regressions and failing tests on other operating system.

2.7 Future work

In this chapter, we have introduced and improved upon the constant perturbation method for Sturm–Liouville problems, and more specifically time-independent one-dimensional Schrödinger equations. In section 2.3 we started from the complicated propagation formulae, and complicated them even further by explicitly keeping the dependence on δ. These even more intimidating formulae enabled us to evaluate eigenfunctions extremely efficiently, as seen in the numerical experiments in section 2.5.

Besides this new theoretical advancement, and the efficient implementation accompanying it (as detailed in section 2.6), we also applied the constant perturbation method to periodic problems in section 2.4. As far as we have found in the literature, problems with non-separated boundary conditions were not yet tackled with CP-methods. Furthermore, implementations that are able to solve problems with periodic boundary conditions are rare. So, we plan to publish section 2.4 in a separate article. Regarding 'future work', this is a concrete example.

In the next chapters we will use the method and the implementation developed here to solve time-independent two-dimensional Schrödinger problems. Our advancements in this chapter were invaluable in the developments from the next chapters.

Since the constant perturbation methods are well-established and thoroughly researched, it is hard to fundamentally improve them. Many researchers already advanced this topic. However, we still see some opportunities to apply constant perturbation in more general Sturm–Liouville problems. One of the first assumptions we made when introducing the CP-methods was that the domain $\Omega = [a, b]$ is a bounded interval. If one is interested in solving Sturm–Liouville problems on infinite domains ($[0, +\infty[$ or $]-\infty, +\infty[$ for example), the domain has to be truncated. For Schrödinger problems where the potential quickly diverges to $+\infty$, this truncation does not pose a big issue. However, for problems with potentials such as the hydrogen potential (from section 2.5.5), eigenvalues close to, but less than, zero are very sensitive to this truncation. Mitigating this is difficult. If the domain is chosen too small, boundary effects change the computed eigenvalues. If the domain is too large, selecting the optimal step size is difficult.

As a possible avenue for future research, we believe that CP-methods can be used on truly infinite domains, without truncation. Up to now, solutions are always propagated from the end points of the interval to a matching point somewhere in the interior. But in principle, we could also propagate solutions from the middle of the domain to the end points. On an infinite domain there will not be two endpoints. This does not need to be a problem, as we can propagate solutions until we find convergence for both the eigenfunction and its derivative to zero. I suspect that implementing this will be far from trivial. The program should be able to dynamically add more steps in both directions if larger eigenvalues are requested.

Chapter 3

A shooting method for 2D time-independent Schrödinger equations

There are many general purpose methods available for solving partial differential equations. Each method has its own benefits and disadvantages. As a rule of thumb, one can say that a method that is very general and widely applicable, will be less efficient or less accurate or both, than a method that is specifically tuned for the problem at hand. With that in mind, there is a real advantage to gain when investing time and research into a highly-tuned optimized method for a specific problem.

In this and the next chapter we will study two-dimensional time-independent Schrödinger equations

$$-\nabla^2\psi(x,y) + V(x,y)\psi(x,y) = E\psi(x,y) \tag{3.1}$$

on the domain $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. We will only consider homogeneous Dirichlet boundary conditions, this means $\forall(x,y) \in \partial\Omega : \psi(x,y) = 0$. The function $V : \mathbb{R}^2 \to \mathbb{R}$ is called the potential function. This potential, together with the domain and the boundary conditions, define the Schrödinger problem. When *solving* the time-independent Schrödinger equation, one is searching for values for E for which a function $\psi(x,y)$ exists such that they together satisfy the Schrödinger problem (3.1). Such a value E is called an *eigenvalue* corresponding to the *eigenfunction* $\psi(x,y)$.

From a functional analysis perspective, the Schrödinger problem can also be interpreted as finding the eigenvalues and eigenfunctions of the *Hamiltonian* $\mathcal{H}$, this is the linear functional operator:

$$\mathcal{H} := -\nabla^2 + V(x,y),$$

defined on the domain Ω with given boundary conditions.

Theoretically, working within the functional analysis framework is extremely useful. Many powerful results are available for all kinds of operators on all kinds of domains. In our case, we are interested in self-adjoint elliptic operators, for if $V(x,y)$ is bounded and continuous then the Hamiltonian is self-adjoint and elliptic. Proving this here is non-trivial. Not because it is a particular difficult proof, in some textbooks this is merely an example of proven theorems, rather since all proofs require a thoroughly developed functional analysis framework. As an example of this framework: in the series of books starting with [85], the authors start with defining functional spaces and provide some basic topological ideas to finally, after almost 200 pages, define a functional operator. It takes another 50 pages before they consider the spectrum of such an operator. So, in these 250 pages they were able to define very rigorously the needed concepts. Only in the fourth volume [84], all parts of the puzzle are available to prove the Hamiltonian operator is self-adjoint. This self-adjointness in turn implies many theorems proven in volume two [83].

Even though numerical methods are able to take a few theoretical shortcuts[1], having a thorough understanding of the theory can be quite instructive. For now, we will not provide the functional analysis background, or rigorous proofs for the properties of Hamiltonian operators. However, we will state some useful theorems specifically for the Schrödinger problem at hand. The theorems we provide here are all special cases of more general results from functional analysis, proofs for these can be found in many textbooks, for example [85] and other books in the series.

But first, let us define a *regular* Schrödinger problem[2] to be equation (3.1)

[1]For example, if we numerically approximate the integral $\int_a^b f(x)\,\mathrm{d}x$ we use our favorite quadrature rule and input the function f. We are not concerned if f is sufficiently integrable. We are not concerned that f could be not *almost everywhere* continuous. The only thing we need is to be able to evaluate f in the points required by the quadrature rule.

[2]The definition we provide here is only for two-dimensional problems. Changing the number of dimensions will not alter the provided theorems.

defined on a bounded Lipschitz[3] domain $\Omega \subseteq \mathbb{R}$ with linear, real and continuous boundary conditions and let $V(x, y) : \Omega \to \mathbb{R}$ be continuous (and therefore bounded) on Ω. With this in hand, we can state analogous theorems as in the one-dimensional case.

Theorem 3.1. *All eigenvalues of a regular Schrödinger problem are real. Corresponding eigenfunctions can always be scaled such that they are real.*

This first theorem states that to find solutions, no complex numbers are necessary. This significantly simplifies the implementation of a method.

Theorem 3.2. *The number of eigenvalues of a regular Schrödinger problem are countable. All eigenvalues have a lower bound and no upper bound. This implies that these can be written as:*

$$E_0 \leq E_1 \leq E_2 \leq \cdots \to \infty.$$

Note that in contrast to theorem 2.2, eigenvalues are no longer guaranteed to be simple. With this we mean that for the same eigenvalue, multiple linear independent eigenfunctions can be found. Such eigenvalues are said to be degenerate. If the corresponding eigenfunction space has dimension 3, for example, the eigenvalue is said to have multiplicity 3.

The last theorem we provide for now will give some information about the eigenfunctions.

Theorem 3.3. *For two different eigenvalues E_m and E_n, the corresponding eigenfunctions $\psi_m(x, y)$ and $\psi_n(x, y)$ will be orthogonal on Ω.*

$$\int_\Omega \psi_m(\mathbf{x})\psi_n(\mathbf{x})\,\mathrm{d}\mathbf{x} = 0 \qquad \textit{if } E_m \neq E_n.$$

These theorems give a very clear picture about what a solution will look like. But also what kind of questions we want to be able to handle, for example: "What is the smallest eigenvalue?", or "Draw a graph of the first few eigenfunctions.". To put these theorems in context we consider the following example.

3.0.1 A first example

As an example we consider the Schrödinger problem on the domain $[0, \pi] \times [0, \pi]$ with homogeneous Dirichlet boundary conditions and zero potential. Its

[3]Formally, a Lipschitz domain has a Lipschitz continuous boundary. Intuitively, a Lipschitz domain has a 'sufficiently regular' boundary [23].

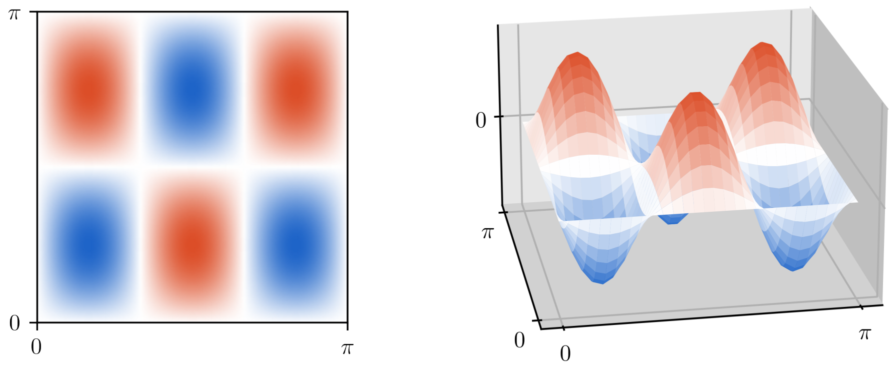


Figure 3.1: A 2D and 3D plot of the eigenfunction corresponding to $i = 3$ and $j = 2$ from equation (3.5), the eigenvalue is $E_{3,2} = 13$.

equation is given as

$$-\frac{\partial^2\psi}{\partial x^2} - \frac{\partial^2\psi}{\partial y^2} = E\psi(x, y). \tag{3.2}$$

Now we write an eigenfunction $\psi(x, y)$ as a y-dependent linear combination of the x-dependent functions $\sin(x)$, $\sin(2x)$, $\sin(3x)$, … which satisfy the boundary conditions:

$$\psi(x, y) = \sum_{i=1}^{\infty} c_i(y)\sin(ix). \tag{3.3}$$

Since $\{\sin(x), \sin(2x), \dots\}$ constructs a basis for $L_0^2(\Omega)$, this is the space of compactly supported square integrable functions on Ω, all possible eigenfunctions are expressed with equation (3.3). If we plug this into the Schrödinger equation then we get

$$\sum_{i=1}^{\infty} i^2 c_i(y)\sin(ix) - \sum_{i=1}^{\infty} \frac{\mathrm{d}^2 c_i}{\mathrm{d}y^2}\sin(ix) = \sum_{i=1}^{\infty} Ec_i(y)\sin(ix).$$

Linear combinations of basis functions can only be equal if their coefficients are equal, thus:

$$(i^2 - E)c_i(y) = \frac{\mathrm{d}^2 c_i}{\mathrm{d}y^2} \quad \text{with } c_i(0) = c_i(\pi) = 0 \text{ for each } i \in \{1, 2, \dots\}. \tag{3.4}$$

Define $j := \sqrt{E - i^2}$. The ordinary differential equation (3.4) will only have solutions that satisfy the given boundary conditions if j is a strictly positive integer, specifically: $c_i(y) = \sin\left(\sqrt{E - i^2} y\right)$. This implies that all eigenvalues of (3.2) are

$$E_{i,j} = i^2 + j^2 \quad \text{for all } i, j \in \{1, 2, \dots\}$$

with corresponding eigenfunction

$$\psi_{i,j} = \sin(ix)\sin(jy). \tag{3.5}$$

The eigenvalues are summarized in the following table.

0	1,2	3	4,5	...	30, 31, 32	...
$E_{1,1}$	$E_{1,2} = E_{2,1}$	$E_{2,2}$	$E_{1,3} = E_{3,1}$	...	$E_{1,7} = E_{5,5} = E_{7,1}$	...
2	5	8	10	...	50	...

Here we see that there are many degenerate eigenvalues, the 30th eigenvalue 50 even has multiplicity three. The 234th eigenvalue with value 325 even has multiplicity six. With some number theory, one can prove that the multiplicity of an eigenvalue can be unbounded.

A visualization of the eigenfunctions is found in figure 3.1. Here we have visualized the surface in a three-dimensional plot on the right. For clarity, we will use most of the time the two-dimensional representation on the left. Note that no color-legend is provided in the two-dimensional representation, as eigenfunctions may always be scaled.

The eigenvalue corresponding to the eigenfunction from figure 3.1 has multiplicity two. Another linear independent eigenfunction can be found by swapping x and y.

For almost all potential functions V however, the corresponding Schrödinger equation cannot be solved symbolically. So when one is interested in solutions, one has to resort to numerical methods. When only the ground state, that is the lowest eigenvalue, or maybe only few of the lowest eigenvalues are required, general numerical methods may suffice. Some examples of such techniques are finite difference based methods, or a finite element analysis. When higher eigenvalues are required, the eigenfunctions become more and more oscillatory. For the one-dimensional problem we have seen that general methods have difficulties with highly oscillatory functions. These same difficulties are expected for two-dimensional problems. In chapter 4, we will study such a more general method, and develop our own method.

This chapter is dedicated to the study and improvement of the method proposed in [47] by Ixaru. In section 3.1, we will follow [47] and study the method itself. Later on, in section 3.2 we will highlight some challenges with using this method, and propose some improvements. Section 3.3 contains the new theory we have developed to determine the index of an eigenfunction. And lastly, in section 3.4 some numerical experiments are presented.

3.1 Ixaru's method

The main idea of Ixaru's method is built on the well-established technique (by, among others, Titchmarsh [95]) of writing a solution as a linear combination of well-chosen one-dimensional basis functions $b_i(x)$: $\psi(x,y) = \sum_{i=1}^{\infty} b_i(x)c_i(y)$. A disadvantage of this known technique is that many basis functions are necessary to represent an eigenfunction accurately along the whole of the domain. Ixaru mitigates this by proposing multiple sets of basis functions, depending on the position in the domain. More concretely, he suggests splitting the domain into K different sectors along the y-axis[4]:

$$y_{\min} = y_0 < y_1 < y_2 < \cdots < y_k < \cdots < y_{K-1} < y_K = y_{\max}.$$

This split in sectors is illustrated in figure 3.2.

On each sector k (with domain $[x_{\min}, x_{\max}] \times [y_{k-1}, y_k]$), a solution $\psi(x,y)$ will be approximated as a linear combination of N basis functions:

$$\psi(x,y) \approx \sum_{i=1}^{N} b_i^{(k)}(x)c_i^{(k)}(y) = \mathbf{b}^{(k)\intercal}(x)\,\mathbf{c}^{(k)}(y). \tag{3.6}$$

Ideally, the used basis on sector k should be related to the resulting eigenfunction on this sector. Of course, the eigenfunction is unknown, so using it is not an option. In principle, there are many bases to choose from, a Fourier-basis is possible, or some basis based upon orthogonal polynomials. But, these well-known choices do not take advantage of the shape of the potential in the sector. In [47], the author proposes to use the eigenfunctions from the following one-dimensional Schrödinger problem:

$$-\frac{\partial^2 b_i^{(k)}}{\partial x^2} + \bar{V}^{(k)}(x)b_i^{(k)}(x) = \lambda_i^{(k)} b_i^{(k)}(x) \tag{3.7}$$

[4]In the original article [47], the domain is split along the x-axis. But for notational purposes, it is more convenient to split along the y-axis. Analogous for the 3d version of the method, the split would happen along the z-axis.

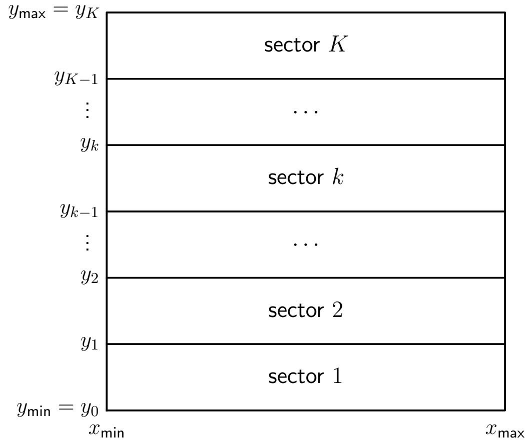


Figure 3.2: An illustration of the split in sectors along the y-axis for the domain $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$.

with boundary conditions $b_i^{(k)}(x_{\min}) = b_i^{(k)}(x_{\max}) = 0$. The function $\bar{V}^{(k)}(x)$ is a constant (in the y-direction) approximation of the potential V on the k^{th} sector.

Using this basis is extremely promising. The basis functions are oscillatory for regions where $\bar{V}$ is small, and have an exponential behavior when $\bar{V}$ is large. This same behavior is expected for the two-dimensional eigenfunctions as well. For a potential function which becomes large, the two-dimensional eigenfunction is expected to be most present in the regions where $V(x, y)$ is small. The chosen one-dimensional basis functions $b_i^{(k)}(x)$ express this same behavior.

In [47], $\bar{V}^{(k)}(x) := V\left(x, \frac{y_{k-1}+y_k}{2}\right)$ is used as the potential of the one-dimensional problem on sector k. We will use this as well, for now. Later, we will remark that in some cases other choices may be beneficial.

Just like the constant perturbation methods for one-dimensional problems, this method employs shooting to locate the eigenvalues. To do so, for a fixed value of E, formulae are needed to propagate solutions from the bottom of the domain (along $y = y_{\min}$) upwards, and from the top of the domain ($y = y_{\max}$) downwards. So, given a solution at the beginning of sector k expressed in the basis $b_i^{(k)}$: $\psi(x, y_{k-1}) = \sum_{i=1}^{N} b_i^{(k)}(x)\, c_i^{(k)}(y_{k-1})$, an expression is constructed to compute $c_i^{(k)}(y_k)$ at the end of the sector. Substituting (3.6) into (3.1) using (3.7) gives rise to

$$-\frac{\partial^2 \mathbf{c}^{(k)}}{\partial y^2} + \mathbf{V}^{(k)}(y)\mathbf{c}^{(k)}(y) = E\mathbf{c}^{(k)}(y). \tag{3.8}$$

In this expression $\mathbf{V}^{(k)}$ is an $N \times N$ matrix depending on y:

$$\mathbf{V}_{ij}^{(k)}(y) = \int_{x_{\min}}^{x_{\max}} b_i^{(k)}(x) b_j^{(k)}(x) \left(V(x,y) - \bar{V}^{(k)}(x) \right) \mathrm{d}x + \delta_{ij}\lambda_i^{(k)}. \tag{3.9}$$

The system of ordinary differential equations given in (3.8) is a coupled system of Schrödinger equations. For coupled systems, there are implementations of constant perturbation methods available, `LILIX` [46] or `MatSCS` [63] for example.

For the accurate computation of the integral in (3.9) we have developed specialized formulae. These will be presented later on in section 3.2.3.

To be able to propagate a solution along the whole domain, it is vital to have an expression to transfer solutions between consecutive sectors. Eigenfunctions should be continuous and continuously differentiable. To ensure this when transitioning between sectors, we impose for all values $x \in [x_{\min}, x_{\max}]$:

$$\begin{aligned} \mathbf{b}^{(k)\mathsf{T}}(x)\,\mathbf{c}^{(k)}(y_k) &= \mathbf{b}^{(k+1)\mathsf{T}}(x)\,\mathbf{c}^{(k+1)}(y_k) \\ \text{and} \quad \mathbf{b}^{(k)\mathsf{T}}(x)\,\frac{\partial \mathbf{c}^{(k)}}{\partial y}(y_k) &= \mathbf{b}^{(k+1)\mathsf{T}}(x)\,\frac{\partial \mathbf{c}^{(k+1)}}{\partial y}(y_k)\,. \end{aligned}$$

Multiplying both sides with $\mathbf{b}^{(k+1)}(x)$ and integrating along the x-axis yields

$$\begin{aligned} \mathbf{c}^{(k+1)}(y_k) &= \mathbf{M}^{(k)}\mathbf{c}^{(k)}(y_k) \\ \frac{\partial \mathbf{c}^{(k+1)}}{\partial y}(y_k) &= \mathbf{M}^{(k)}\frac{\partial \mathbf{c}^{(k)}}{\partial y}(y_k) \end{aligned}$$

with

$$\mathbf{M}_{ij}^{(k)} = \left\langle b_i^{(k+1)} \middle| b_j^{(k)} \right\rangle = \int_{x_{\min}}^{x_{\max}} b_i^{(k+1)}(x) b_j^{(k)}(x)\mathrm{d}x.$$

Here we assumed each $b_i^{(k+1)}(x)$ to be normalized $\langle b_i^{(k+1)} | b_i^{(k+1)} \rangle = 1$, or in matrix notation:

$$\int_{x_{\min}}^{x_{\max}} \mathbf{b}^{(k+1)}\mathbf{b}^{(k+1)\mathsf{T}}\,\mathrm{d}x = \mathbf{I}. \tag{3.10}$$

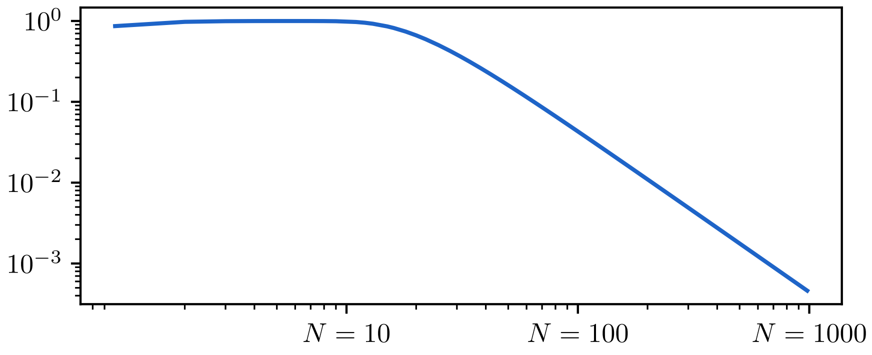


Figure 3.3: Let $\mathbf{N}$ be the upper left $N \times N$ block of $\mathbf{M}$, with $\mathbf{M}$ the transition between the bases defined by the Schrödinger problems with potentials $V(x) = (x-1)^2$ and $V(x) = (x+1)^2$. This graph displays $\|\mathbf{N}\mathbf{N}^\intercal - I\|_2$ as a function of N.

Since in the infinite case

$$c_i^{(k+1)} = \sum_{i=0}^{\infty} \left\langle b_i^{(k+1)} \middle| b_j^{(k)} \right\rangle c_j^{(k)} \tag{3.11}$$

exactly, this matrix $\mathbf{M}^{(k)}$ is orthogonal. However, if the sum is truncated, then the equality in (3.11) no longer holds and $\mathbf{M}^{(k)\intercal}$ no longer reverses this transformation. Therefore, $\mathbf{M}^{(k)}$ is in general no longer orthogonal. As an extreme example consider the transition matrix $\mathbf{M}$ from the basis defined by $-b_i'' + (x-1)^2 b_i = \lambda_i b_i$ to the basis defined by $-b_i'' + (x+1)^2 b_i = \lambda_i b_i$. The infinite matrix $\mathbf{M}$ is orthogonal, any finite upper left block of $\mathbf{M}$ is not orthogonal. This is demonstrated in figure 3.3. This loss of orthogonality is a minor inconvenience, inherent to this method.

With these tools available, all that is left is to formalize how the shooting can be executed. Instead of propagating with a single starting condition (i.e. a column vector $\mathbf{c}^{(1)}(y_{\min})$), all possible initial values (for homogeneous Dirichlet boundary conditions) are propagated at once. Therefore, we propose to start with:

$$\mathbf{C}^{(1)}(y_{\min}) = \mathbf{0}_{N\times N} \qquad \frac{\partial}{\partial y}\mathbf{C}^{(1)}(y_{\min}) = \mathbf{I}_{N\times N}$$

and

$$\mathbf{C}^{(K)}(y_{\max}) = \mathbf{0}_{N\times N} \qquad \frac{\partial}{\partial y}\mathbf{C}^{(K)}(y_{\max}) = \mathbf{I}_{N\times N}.$$

As we are using a multiple shooting procedure in the y-direction, the propagated values come together in a matching line $y = y_m$. Since eigenfunctions have to be continuous and continuously differentiable, a 'matching' condition can be formulated. Let us define $\mathbf{C}_{\text{bottom}}$ and $\mathbf{C}'_{\text{bottom}}$ to be the values of $\mathbf{C}^{(m)}(y_m)$ and $\frac{\partial}{\partial y}\mathbf{C}^{(m)}(y_m)$ respectively when propagated from the bottom of the domain upwards. Analogous $\mathbf{C}_{\text{top}}$ and $\mathbf{C}'_{\text{top}}$ can be defined. Now the value E is an eigenvalue of the original problem if and only if there exist vectors $\mathbf{u}_{\text{bottom}}$ and $\mathbf{u}_{\text{top}}$ such that

$$\begin{aligned} \mathbf{C}_{\text{bottom}} \cdot \mathbf{u}_{\text{bottom}} &= \mathbf{C}_{\text{top}} \cdot \mathbf{u}_{\text{top}} \\ \text{and} \quad \mathbf{C}'_{\text{bottom}} \cdot \mathbf{u}_{\text{bottom}} &= \mathbf{C}'_{\text{top}} \cdot \mathbf{u}_{\text{top}}. \end{aligned} \tag{3.12}$$

In [47], $\mathbf{C}_{\text{bottom}}$ and $\mathbf{C}_{\text{top}}$ are implicitly assumed to be non-singular. With this, equations (3.12) can be rearranged. This makes it equivalent with saying: E is an eigenvalue of (3.1) if and only if the mismatch matrix

$$\mathbf{\Phi}(E) := \mathbf{C}'_{\text{bottom}}\mathbf{C}^{-1}_{\text{bottom}} - \mathbf{C}'_{\text{top}}\mathbf{C}^{-1}_{\text{top}} \tag{3.13}$$

has a zero eigenvalue. Each linear independent eigenvector of $\mathbf{\Phi}(E)$ corresponding to eigenvalue 0 implies a linear independent eigenfunction $\psi(x, y)$ of the Schrödinger equation. Conversely, each eigenfunction of the Schrödinger equation corresponding to E will emit a singular vector for (3.13). Therefore, the geometric multiplicity of the zero eigenvalue of this matrix is the same as the multiplicity of E as an eigenvalue of (3.1). In the original article no details are provided about how one should find the values of E for which the matrix $\mathbf{\Phi}(E)$ becomes singular. Later, in section 3.2.4, we will present our method for finding those values. In section 3.3, we will go even further and demonstrate a technique which allows to determine the index of the eigenvalue in question.

This concludes our overview of the method described by Ixaru in [47]. There are a few differences between our overview and the method as described in [47]. Most notably, as stated in the beginning, we have swapped the roles of x and y. To highlight the recursive nature of this method, we have chosen to apply the split along the y-axis. To use this method for three-dimensional problems, we split the domain along the z-axis, and for the basis functions on each sector we solve the two-dimensional Schrödinger problem in the $x\,y$-plane. For each of these two-dimensional problems, we split the domain along the y-axis and solve some one-dimensional Schrödinger problems along the x-axis.

3.2 Our improvements

The numerical experiment in [47] makes this method seem promising. However, we have identified some possibilities where Ixaru's method can be expanded or improved. This section follows primarily our work from [9].

We have made four large improvements and present them here in no particular order. First in section 3.2.1, we make the program able to automatically choose the optimal sector size. This automatic sector selection allows for a user to only need to specify the required accuracy of the results. In section 3.2.2, we construct a formula to compute the inner product of two eigenfunctions. This formula allows us to normalize eigenfunctions and to construct an orthogonal basis of the eigenspace for degenerate eigenvalues. Section 3.2.3 is dedicated to the computation of the integral in equation (3.9). And the last improvement we present here is a robust way to locate eigenvalues in section 3.2.4. For this last improvement we have to develop some new theoretical results, which we will cover in section 3.3.

3.2.1 Automatic optimal sector size

When developing numerical methods (for any problem, not only for differential equations), it is most user-friendly to ask a user to only specify to which accuracy results are required. Everything else should be automatic[5]. In `Matslise 3.0`, we have followed `Matslise 2.0`, which has automatic sector size built in. The usability of this two-dimensional method improves if we include something similar.

The same ideas from `Matslise` can be used here as well. The only minor difference is the way we compute the error for a given sector. In the one-dimensional case, the difference between the 16^{th} and 18^{th} order propagation matrix is used to estimate the error. In Ixaru's method we are propagating a coupled system of Schrödinger equations on each sector. This propagation is implemented with an adaptation of `MatSCS` [63]. In this adaptation, the difference between propagating with a 8^{th} order and a 10^{th} order method is used as an error estimate.

In theory and in practice this works beautifully, but some optimizations are

[5]For example, an implementation of an adaptive quadrature rule should let the user specify only the function to integrate and the accuracy required. No extra parameters, such as the initial number of grid points or the order of the method for example, should be required. But ideally, a user who is experimenting or wants more control should be able to specify any of these extra parameters.

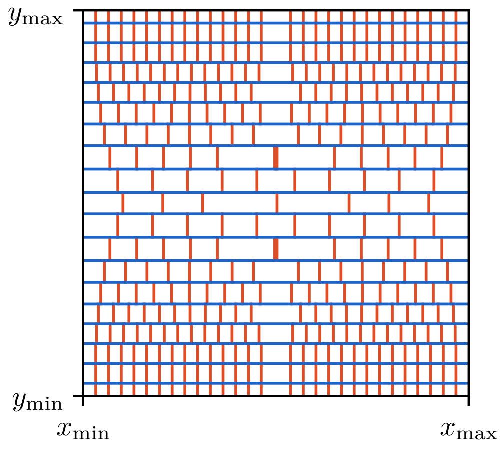


Figure 3.4: A possible sector selection for problem 3.4.3 with potential $V(x,y) = (1+x^2)(1+y^2)$. The blue lines indicate sector boundaries. The red lines visualize the piecewise approximation used by `Matslise 3.0` for each of the x-directional problems.

still possible. The selection algorithm works by guessing an initial sector size s_0. When the error is too large, a smaller sector size s_1 is chosen and the process starts over. One big disadvantage of this method is that when a sector was computed with an error that is too large, all computations are thrown away and everything is recomputed. This is quite wasteful when the sector size is only slightly adjusted. One of the things we can reuse are the basis functions. Initially these are computed as the solution of

$$-\frac{\partial^2 b_i}{\partial x^2} + V(x, \bar{y}) = \lambda_i b_i,$$

with $\bar{y} = y_k + \frac{s_0}{2}$. But if $\frac{s_1}{2} \approx \frac{s_0}{2}$ then it is maybe not necessary to recompute these basis functions. In our testing, we have seen that the algorithm is not very sensitive to when $\frac{s_1}{2}$ is no longer considered close to $\frac{s_0}{2}$. We have chosen the heuristic that $y_k + \frac{s_1}{3} \leq \bar{y} \leq y_k + \frac{2s_1}{3}$. Intuitively this means that our program reuses the basis functions b_i as long as the corresponding $\bar{y}$ is situated within the middle third of a sector. Another big advantage of not needing to recompute the basis functions is that the expensive recursive expressions for the integrals $\int_0^h b_i(\delta)b_j(\delta)\delta^n \mathrm{d}\delta$ can be reused as well. These expressions will be calculated in section 3.2.3.

A possible automatic sector selection is visualized in figure 3.4. Here we have run our implementation on the problem from section (3.4.3). We notice that in the middle of the domain, where the graph of the potential $V(x,y) = (1+x^2)(1+y^2)$

lies lower and is less steep, the sectors are larger.

3.2.2 Orthonormalization of eigenfunctions

For the one-dimensional case, we know from theory that eigenfunctions corresponding to different eigenvalues are necessarily orthogonal. In theorem 3.3 we have seen that eigenfunctions for different eigenvalues will always be orthogonal. But up to now, nothing has been stated about eigenfunctions corresponding to the same eigenvalue. For degenerate eigenvalues, the eigenspace will be multidimensional.

Ideally, the eigenfunctions returned by our program should form an orthonormal basis. For this we need to normalize eigenfunctions, and ensure that the returned basis of a multidimensional eigenspace is orthogonal. To execute normalization, the inner product $\langle u|u\rangle := \int_\Omega u^2$ of an eigenfunction u with itself should be computed. To orthogonalize two linear independent eigenfunctions in the same eigenspace, a Gram–Schmidt process can be used. For this process, there has to be a way to compute the inner product $\langle u|v\rangle$ of two functions u and v in this eigenspace.

To normalize an eigenfunction or to orthogonalize eigenfunctions, in both cases the inner product of two functions should be computable. One way would be to numerically estimate this value with some quadrature rules. In principle this is a valid approach, but in practice, this is quite computationally expensive. Therefore in theorem 3.4, we construct specialized formulae to compute these inner products.

Calculating the inner product on the whole domain at once is difficult due to the changes in the used basis $\{b_i^{(k)}\}$ for different sectors. To combat this, theorem 3.4 is formulated with the assumptions that these $b_i^{(k)}(x)$ are constant in the y-direction. To compute the inner product of two eigenfunctions on the whole domain, the theorem can be repeatedly applied on each sector. The proof of this theorem follows the same idea used for the normalization of solutions found with a constant perturbation method for one-dimensional Sturm–Liouville problems.

Theorem 3.4. *Let $\psi_a(x,y)$ and $\psi_b(x,y)$ be two, not necessarily distinct, real eigenfunctions of the Schrödinger operator corresponding to the same eigenvalue E. Denote both eigenfunctions as linear combinations of one-dimensional orthonormal basis functions with $b_i(x_{min}) = b_i(x_{max}) = 0$ (as described in* *section 3.1, on a single sector).*

$$\psi_a(x,y) = \mathbf{b}(x)^\intercal \mathbf{c_a}(y) = \sum_i b_i(x) c_{a,i}(y),$$
$$\psi_b(x,y) = \mathbf{b}(x)^\intercal \mathbf{c_b}(y) = \sum_i b_i(x) c_{b,i}(y).$$

On a rectangular domain $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ *the inner product of these two functions can be expressed as*

$$\langle \psi_a | \psi_b \rangle = \int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} \psi_a \psi_b \, \mathrm{d}y \, \mathrm{d}x = \left(\frac{\partial \mathbf{c_a}^\intercal}{\partial E} \mathbf{c_b}' - \mathbf{c_b}^\intercal \frac{\partial \mathbf{c_a}'}{\partial E} \right) \Bigg|_{y=y_{min}}^{y=y_{max}},$$

with $\mathbf{c_a}' = \frac{\mathrm{d}\mathbf{c_a}}{\mathrm{d}y}$ *and* $\mathbf{c_b}' = \frac{\mathrm{d}\mathbf{c_b}}{\mathrm{d}y}$*.*

Proof. By assumption ψ_a and ψ_b are solutions of the Schrödinger equation:

$$-\nabla^2 \psi_a + V\psi_a = E\psi_a$$
$$-\nabla^2 \psi_b + V\psi_b = E\psi_b.$$

Differentiating the first equation with respect to E and multiplying with ψ_b gives

$$-\psi_b \nabla^2 \frac{\partial \psi_a}{\partial E} + V(x,y) \frac{\partial \psi_a}{\partial E} \psi_b = E \frac{\partial \psi_a}{\partial E} \psi_b + \psi_a \psi_b.$$

The second equation can be multiplied with $\frac{\partial \psi_a}{\partial E}$ to result in

$$-\frac{\partial \psi_a}{\partial E} \nabla^2 \psi_b + V(x,y) \psi_b \frac{\partial \psi_a}{\partial E} = E \psi_b \frac{\partial \psi_a}{\partial E}.$$

The difference of these last two expressions can be integrated over the domain $[x_{\text{min}}, x_{\text{max}}] \times [y_{\text{min}}, y_{\text{max}}]$ to obtain an expression for the inner product $\langle \psi_a | \psi_b \rangle$. For brevity of notation, we omit the integration domain, this is assumed to be $[x_{\text{min}}, x_{\text{max}}] \times [y_{\text{min}}, y_{\text{max}}]$.

$$\iint \psi_a \, \psi_b = \iint \frac{\partial \psi_a}{\partial E} \nabla^2 \psi_b - \iint \psi_b \nabla^2 \frac{\partial \psi_a}{\partial E}$$

The right-hand side can be calculated by applying Green's second identity[6]

[6]Let $f : \Omega \to \mathbb{R}$ and $g : \Omega \to \mathbb{R}$ be twice differentiable functions on $\Omega \subseteq \mathbb{R}^d$. Green's second identity tells us $\int_\Omega f \nabla^2 g - g \nabla^2 f \mathrm{d}\Omega = \oint_{\partial\Omega} (f \nabla g - g \nabla f) \cdot \mathrm{d}\mathbf{n}$, with $\mathbf{n}$ the normal of $\partial\Omega$.

$$= \oint \left(\frac{\partial \psi_a}{\partial E} \nabla \psi_b \cdot \mathbf{n} - \psi_b \nabla \frac{\partial \psi_a}{\partial E} \cdot \mathbf{n} \right).$$

This can be written explicitly on the integration domain as

$$= \int_{x_{\min}}^{x_{\max}} \left(\frac{\partial \psi_a}{\partial E} \frac{\partial \psi_b}{\partial y} - \psi_b \frac{\partial^2 \psi_a}{\partial E \partial y} \right) \mathrm{d}x \Bigg|_{y=y_{\min}}^{y=y_{\max}} + \int_{y_{\min}}^{y_{\max}} \left(\frac{\partial \psi_a}{\partial E} \frac{\partial \psi_b}{\partial x} - \psi_b \frac{\partial^2 \psi_a}{\partial E \partial x} \right) \mathrm{d}y \Bigg|_{x=x_{\min}}^{x=x_{\max}}.$$

Taking into account that the eigenfunctions ψ_a and ψ_b are expressed in terms of an orthonormal set of basis functions $\mathbf{b}(x)$, as in equation (3.10), we can simplify the expression to

$$= \left(\frac{\partial \mathbf{c_a}^\intercal}{\partial E} \mathbf{c_b}' - \mathbf{c_b}^\intercal \frac{\partial \mathbf{c_a}'}{\partial E} \right) \Bigg|_{y=y_{\min}}^{y=y_{\max}} + \int_{y_{\min}}^{y_{\max}} \left(\mathbf{b}^\intercal \frac{\partial \mathbf{c_a}}{\partial E} \mathbf{b}'^\intercal \mathbf{c_b} - \mathbf{b}^\intercal \mathbf{c_b} \mathbf{b}'^\intercal \frac{\partial \mathbf{c_a}}{\partial E} \right) \mathrm{d}y \Bigg|_{x=x_{\min}}^{x=x_{\max}}.$$

Since the basis functions satisfy the Dirichlet boundary conditions[7] $\mathbf{b}(x_{\min}) = \mathbf{b}(x_{\max}) = \mathbf{0}$ the last term has to be zero, which proves the theorem. □

In this theorem, it is assumed that the basis in which the eigenfunctions are expanded is constant throughout the whole domain. In our case, this basis changes between sectors. Therefore, when computing the inner product of two eigenfunctions on the domain, theorem 3.4 will have to be repeatedly employed on each sector separately, with $y_{\min} = y_{k-1}$ and $y_{\max} = y_k$. Summing these values across all sectors gives the inner product over the whole domain. This value, in turn, can be used to normalize an eigenfunction or orthogonalize different eigenfunctions for the same eigenvalue.

3.2.3 Calculation of $\mathbf{V}^{(k)}(y)$

Another improvement we have made can be found in the calculation of the matrix $\mathbf{V}^{(k)}(y)$ from equation (3.9). When one wants to implement the right-hand side of (3.9) accurately, some considerations have to be made. As the formula consist of an integral with a relatively complicated integrand, we have

[7]Note that this proof also works if the basis $\mathbf{b}$ is assumed to satisfy homogeneous Robin boundary conditions.

to consider in which points this integrand is known. Furthermore, evaluating more points is non-trivial.

In [47], the author points out that not only the values of the basis functions are known but also the values of their derivatives. To use this, ideas from [57] are combined with the treatment of exponential fitting with two frequencies in [48]. This yielded a procedure close to `CENC1` from [52].

In our case, the story is quite different. Because of our improvements to `Matslise`, we can improve accuracy even further. In particular, the computation of $b_i^{(k)}(x)$ is now efficiently possible in arbitrary points. This allows us to use off-the-shelf adaptive quadrature rules, to ensure accuracy. As this was only a first test to approximate $\mathbf{V}^{(k)}(x)$, we did not use specialized exponentially fitted methods. When profiling our implementation we found that the time to execute these quadrature formulae was almost negligible.

So in practice, there was little need to improve even further, with the more complicated quadrature formulae from [48, 57, 21]. But, as numerical analysts, we were still curious if some more fundamental improvements were possible. After all, the eigenfunctions $b_i^{(k)}(x)$ are approximated piecewise by a truncated series in the step size δ and the special functions η_{-1}, η_0, ... But also, the function $\bar{V}^{(k)}$ is already approximated by a sixteenth degree polynomial. The only 'wildcard', so to speak, is the unknown general function $V(x, y)$, for a fixed value of y. In the middle of the k^{th} sector, this function is approximated piecewisely. As such, it is reasonable to assume that a piecewise sixteenth degree polynomial can also be accurately fitted on this function $V(x, y)$, if the same grid as for $\bar{V}^{(k)}$ is used.

More formal, let us split the interval $[x_{\min}, x_{\max}]$ into the same partition that `Matslise 3.0` used for the piecewise approximation $x_{\min} = x_0, x_1, \ldots, x_K = x_{\max}$. For a fixed value of y on the section $[x_{l-1}, x_l]$, the potential $V(x, y)$ can be approximated by a sixteenth order polynomial, just like $\bar{V}^{(k)}(x)$. We thus approximate $V(x, y) - \bar{V}^{(k)}(x)$ on $[x_{l-1}, x_l]$ by a polynomial $q_l(x)$. This means that to compute (3.9), it is sufficient to be able to evaluate

$$\int_0^h b_i^{(k)}(x_{l-1} + \delta) b_j^{(k)}(x_{l-1} + \delta) \delta^n \mathrm{d}\delta \tag{3.14}$$

with $h := x_l - x_{l-1}$.

In chapter 2, we have provided expressions for $b_i^{(k)}(x_{l-1} + \delta)$ as a function of δ

and $Z_i := \delta^2\left(\bar{V}^{(k)}_{l,0} - \lambda^{(k)}_i\right)$:

$$b^{(k)}_i(x_{l-1}+\delta) = \sum_{m=-1} c_m(\delta)\eta_m(Z_i).$$

In this expression, $c_m(\delta)$ are known polynomials in δ.

Thus, to reconstruct the value of (3.14), it is sufficient to compute the value of

$$I^n_{k,l} := \int_0^h \eta_k(Z_i)\eta_l(Z_j)\delta^n \mathrm{d}\delta \tag{3.15}$$

for all appropriate k, l and n. In the next section, we will develop recursive formulae for this expression.

3.2.3.1 Weighted integral of product of two η-functions

Here, the goal is to develop an expression to compute (3.15). Due to the recursive property of η-functions

$$\eta_k(Z) = \frac{1}{Z}\left(\eta_{k-2}(Z) - (2k-1)\eta_{k-1}(Z)\right),$$

if the first values of (3.15) are known for all n, the others follow. So we calculate four integrals symbolically (denote $\theta_i := \bar{V}^{(m)}_0 - \lambda_i$ and $\hat{Z}_i = h^2\theta_i$):

$$\begin{aligned}
F^0 := I^0_{-1,-1} &= \frac{h}{\theta_i - \theta_j}\left(\theta_i\eta_0(\hat{Z}_i)\eta_{-1}(\hat{Z}_j) - \theta_j\eta_{-1}(\hat{Z}_i)\eta_0(\hat{Z}_j)\right),\\
G^0 := I^1_{-1,0} &= \frac{-1}{\theta_i - \theta_j}\left(\eta_{-1}(\hat{Z}_i)\eta_{-1}(\hat{Z}_j) - \hat{Z}_i\eta_0(\hat{Z}_i)\eta_0(\hat{Z}_j) - 1\right),\\
H^0 := I^1_{0,-1} &= \frac{1}{\theta_i - \theta_j}\left(\eta_{-1}(\hat{Z}_i)\eta_{-1}(\hat{Z}_j) - \hat{Z}_j\eta_0(\hat{Z}_i)\eta_0(\hat{Z}_j) - 1\right),\\
J^0 := I^2_{0,0} &= \frac{h}{\theta_i - \theta_j}\left(\eta_{-1}(\hat{Z}_i)\eta_0(\hat{Z}_j) - \eta_0(\hat{Z}_i)\eta_{-1}(\hat{Z}_j)\right).
\end{aligned}$$

In the case where $i = j$, these values are given as

$$\begin{aligned}
F^0 := I^0_{-1,-1} &= \frac{h}{2}\left(\eta_{-1}(\hat{Z}_i)\eta_0(\hat{Z}_i) + 1\right),\\
G^0 := I^1_{-1,0} &= \frac{h^2}{2\hat{Z}_i}\left(\eta_{-1}(\hat{Z}_i)^2 - 1\right),\\
H^0 := I^1_{0,-1} &= G^0,\\
J^0 := I^2_{0,0} &= \frac{h^3}{2\hat{Z}_i}\left(\eta_{-1}(\hat{Z}_i)\eta_0(\hat{Z}_i) - 1\right).
\end{aligned}$$

We next show that it is possible, by partial integration, to compute formulae for $F^n := I^n_{-1,-1}$ and $J^n := I^{n+2}_{0,0}$ using values for $G^{n-1} := I^n_{-1,0}$ and $H^{n-1} := I^n_{0,-1}$. Analogous, formulae for G^n and H^n use the values of F^{n-1} and J^{n-1}. Let us consider the case of F^n. Differentiating both sides of

$$\int_0^h \eta_{-1}(Z_i)\eta_{-1}(Z_j)\mathrm{d}\delta = \frac{h}{\theta_i - \theta_j}\left(\theta_i\eta_0(\hat{Z}_i)\eta_{-1}(\hat{Z}_j) - \theta_j\eta_{-1}(\hat{Z}_i)\eta_0(\hat{Z}_j)\right),$$

with respect to h, and replacing h by δ, gives:

$$\eta_{-1}(Z_i)\eta_{-1}(Z_j) = \frac{\partial}{\partial\delta}\left(\frac{\delta}{\theta_i - \theta_j}\left(\theta_i\eta_0(Z_i)\eta_{-1}(Z_j) - \theta_j\eta_{-1}(Z_i)\eta_0(Z_j)\right)\right).$$

Then, the partial integration formula

$$\int_0^h f'(\delta)g(\delta)\mathrm{d}\delta = f(h)g(h) - f(0)g(0) - \int_0^h f(\delta)g'(\delta)\mathrm{d}\delta$$

is applied to $g(\delta) = \delta^n$ and $f'(\delta) = \eta_{-1}(Z_i)\eta_{-1}(Z_j)$. Since

$$f(\delta) = \frac{\delta}{\theta_i - \theta_j}\left(\theta_i\eta_0(Z_i)\eta_{-1}(Z_j) - \theta_j\eta_{-1}(Z_i)\eta_0(Z_j)\right),$$

we obtain

$$\begin{aligned} F^n &= \int_0^h \eta_{-1}(Z_i)\eta_{-1}(Z_j)\delta^n\mathrm{d}\delta \\ &= h^n F^0 - \frac{n}{\theta_i - \theta_j}\int_0^h \delta^n\left(\theta_i\eta_0(Z_i)\eta_{-1}(Z_j) - \theta_j\eta_{-1}(Z_i)\eta_0(Z_j)\right)\mathrm{d}\delta \\ &= h^n F^0 - \frac{n}{\theta_i - \theta_j}\left(\theta_i H^{n-1} - \theta_j G^{n-1}\right). \end{aligned}$$

With similar computations we obtain the following formulae

$$\begin{aligned}
G^n &= \int_0^h \eta_{-1}(Z_i)\eta_0(Z_j)\delta^{n+1}\mathrm{d}\delta \\
&= h^n G^0 + \frac{n}{\theta_i - \theta_j}\int_0^h \delta^{n-1}\left(\eta_{-1}(Z_i)\eta_{-1}(Z_j) - \theta_i\delta^2\eta_0(Z_i)\eta_0(Z_j) - 1\right)\mathrm{d}\delta \\
&= h^n G^0 + \frac{n}{\theta_i - \theta_j}\left(F^{n-1} - \theta_i J^{n-1}\right) - \frac{h^n}{\theta_i - \theta_j} \\
H^n &= h^n H^0 - \frac{n}{\theta_i - \theta_j}\left(F^{n-1} - \theta_j J^{n-1}\right) + \frac{h^n}{\theta_i - \theta_j} \\
J^n &= \int_0^h \eta_0(Z_i)\eta_0(Z_j)\delta^{n+2}\mathrm{d}\delta \\
&= h^n J^0 - \frac{n}{\theta_i - \theta_j}\int_0^h \delta^n\left(\eta_{-1}(Z_i)\eta_0(Z_j) - \eta_0(Z_i)\eta_{-1}(Z_j)\right)\mathrm{d}\delta \\
&= h^n J^0 - \frac{n}{\theta_i - \theta_j}\left(G^{n-1} - H^{n-1}\right).
\end{aligned}$$

Calculating these values for increasing n can be a numerically unstable process. A more stable algorithm is made by reversing the recursion with decreasing values of n. This gives rise to the recursion:

$$\begin{aligned}
F^{n-1} &= \frac{1}{n}\left(x_1\theta_i + x_2\theta_j + h^n\right) \\
J^{n-1} &= \frac{1}{n}\left(x_1 + x_2\right) \\
G^{n-1} &= \frac{1}{n}\left(y_1 + \theta_i y_2\right) \\
H_{n-1} &= \frac{1}{n}\left(y_1 + \theta_j y_2\right)
\end{aligned}$$

with:

$$\begin{aligned}
x_1 &= H^0 h^n - H^n \\
x_2 &= G^0 h^n - G^n \\
y_1 &= F^0 h^n - F^n \\
y_2 &= J^0 h^n - J^n.
\end{aligned}$$

In practice, we have found reliable convergence when starting this recursion with n sufficiently large and F^n, G^n, H^n and J^n replaced by 0.

Lastly, the other values for $I^n_{k,l}$, with $k, l > 0$, can be computed via the recursion relation between the η-functions:

$$I^n_{k,l} = \frac{1}{\theta_i}\left(I^{n-2}_{k-2,l} - (2k-1)I^{n-2}_{k-1,l}\right)$$
$$I^n_{k,l} = \frac{1}{\theta_j}\left(I^{n-2}_{k,l-2} - (2l-1)I^{n-2}_{k,l-1}\right).$$

In summary, the formulae developed here can now be used to determine values for (3.15), for all n, k and l.

3.2.3.2 Calculation of $\mathbf{M}^{(k)}$

In the previous section, we have obtained formulae to compute (3.14). In principle, these formulae could also be extended to compute overlap integrals

$$\mathbf{M}^{(k)}_{ij} = \int_{x_{\min}}^{x_{\max}} b^{(k+1)}_i(x) b^{(k)}_j(x) \mathrm{d}x,$$

but there are a few hurdles. First, we would like to use the automatic sector selection algorithm in `Matslise`, as this has the big advantage of ensuring the requested accuracy. Using this automatic sector selection has the consequence of choosing a different partition along the x-axis for different sectors in the y-direction. However, since $\mathbf{M}^{(k)}$ requires basis functions on two different sectors, it would be very cumbersome to construct appropriate formulae.

Second, we noted that these recursive expressions for $\mathbf{V}^{(k)}(y)$ are quite expensive to compute. For $\mathbf{V}^{(k)}(y)$, this cost can be justified by also reducing function evaluations of the potential $V(x, y)$.

In the case of $\mathbf{M}^{(k)}$, a similar justification is hard to find because in `Matslise 3.0` it is now possible to evaluate $b^{(k+1)}_i(x)$ and $b^{(k)}_j(x)$ very cheaply in arbitrary points.

For these reasons we have opted to use classical quadrature rules to compute $\int_{x_{\min}}^{x_{\max}} b^{(k+1)}_i(x) b^{(k)}_j(x)\mathrm{d}x$. To ensure sufficiently accurate integration results, the 31-points adaptive Gauss–Kronrod formulae are used. These have a proven track record [77].

3.2.4 Locating eigenvalues

In section 3.1, we have studied a method to determine, for a given value E, whether it is an eigenvalue. This E will only be an eigenvalue if the matrix from equation (3.13) becomes singular. In [47], it is suggested to keep track of the smallest (in modulus) eigenvalue of $\Psi(E)$. No information is provided about how one can find such values for E.

In the literature, many methods to find roots of functions are available. In introductory textbooks to scientific computing (e.g. [37, Chapter 5]), the most well-known methods are presented. One of the easiest to implement is the method of interval bisection. In this method the root of a scalar function $f : \mathbb{R} \to \mathbb{R}$ is determined by repeatedly halving a search interval. The position of the root can be tracked by ensuring the function f has different signs on the end points of the search interval. This requires that an initial guess for the interval already contains one root. But also, notice that the method will not work when the initial search interval contains two roots, or a root with higher multiplicity for example.

Another well-known technique is Newton's method. Here, the root of a scalar function f can be approximated by iterating the following scheme:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

For single roots, this method has a faster rate of convergence. Choosing initial values is also easier, because a value x_0 should be chosen only to be 'sufficiently close' to a true root. One of the biggest drawbacks of this method is that the derivative of $f(x)$ should be available. For complicated functions, this can be difficult. Extensions of this method exist for which the derivative is not necessary, the secant method for example.

In all these well-known methods, the problem is that they only can be used to find a single root, not all roots. Furthermore, they only work on scalar function $\mathbb{R} \to \mathbb{R}$ or vector-functions $\mathbb{R}^n \to \mathbb{R}^n$ for which roots are uniquely defined. In our case, we are interested in the situation when any of the eigenvalues of (3.13) becomes zero. This does not map directly on any of these classical root-finding algorithms.

To combat these issues with the classical algorithms, we propose a small modification to Newton's method. Suppose an inaccurate approximation E_0 of an eigenvalue of the Schrödinger equation (3.1) is given. Starting from this approximation, we want a fast converging algorithm to find the true eigenvalue E. As stated earlier, E is an eigenvalue if and only if it is a value such that the mismatch matrix $\mathbf{\Phi}(E)$, as defined by (3.13), is singular.

$$\mathbf{\Phi}(E) := \mathbf{C}'_{\text{bottom}}\mathbf{C}_{\text{bottom}}^{-1} - \mathbf{C}'_{\text{top}}\mathbf{C}_{\text{top}}^{-1}.$$

In other words, the matrix $\mathbf{\Phi}(E)$ has to have a zero eigenvalue.

In Newton's method, the derivative of the function in question should be available. Fortunately, in [46] and in [65], procedures are available to compute the derivative of solutions with respect to E. In our case, this means that the matrices

$$\frac{\partial \mathbf{C}_{\text{bottom}}}{\partial E}, \frac{\partial \mathbf{C}'_{\text{bottom}}}{\partial E}, \frac{\partial \mathbf{C}_{\text{top}}}{\partial E} \text{ and } \frac{\partial \mathbf{C}'_{\text{top}}}{\partial E}$$

are available. This allows us to also compute the derivative of $\mathbf{\Phi}$ itself. First note, for any invertible matrix $\mathbf{A}$ that $\left(\mathbf{A}^{-1}\right)' = -\mathbf{A}^{-1}\mathbf{A}'\mathbf{A}^{-1}$, since $\mathbf{0} = \left(\mathbf{A}\mathbf{A}^{-1}\right)' = \mathbf{A}'\mathbf{A}^{-1} + \mathbf{A}\left(\mathbf{A}^{-1}\right)'$. It then follows that

$$\begin{aligned}\frac{\partial \mathbf{\Phi}}{\partial E} = &\left(\frac{\partial \mathbf{C}'_{\text{bottom}}}{\partial E} - \mathbf{C}'_{\text{bottom}}\mathbf{C}^{-1}_{\text{bottom}}\frac{\partial \mathbf{C}_{\text{bottom}}}{\partial E}\right)\mathbf{C}^{-1}_{\text{bottom}} \\ &- \left(\frac{\partial \mathbf{C}'_{\text{top}}}{\partial E} - \mathbf{C}'_{\text{top}}\mathbf{C}^{-1}_{\text{top}}\frac{\partial \mathbf{C}_{\text{top}}}{\partial E}\right)\mathbf{C}^{-1}_{\text{top}}.\end{aligned}$$

Since we want to find roots within the eigenvalues of $\mathbf{\Phi}$, it is also valuable to be able to compute the derivative of an eigenvalue with respect to E. The derivatives of the eigenvalues of a matrix-function are already known for a long time [60].

Theorem 3.5 (Lancaster 1964)**.** *Let $\mathbf{A}(x)$ be a matrix function $\mathbb{C} \to \mathbb{C}^{n\times n}$ such that each coefficient is continuously differentiable with respect to x in the point x_0. Furthermore, assume $\mathbf{A}(x_0)$ to be diagonalizable*[8].

Let λ be a simple eigenvalue of $\mathbf{A}(x_0)$ with left and right eigenvectors $\mathbf{v}$, $\mathbf{u}$ respectively. It now holds that

$$\left(\frac{\mathrm{d}\lambda}{\mathrm{d}x}\right)_{x=x_0} = \frac{1}{\mathbf{v}^\intercal\mathbf{u}}\left(\mathbf{v}^\intercal\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}x}\mathbf{u}\right)_{x=x_0},$$

where $\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}x}$ is the matrix with, as coefficients, the derivatives of the corresponding coefficients of $\mathbf{A}(x)$ with respect to x.

[8]In [60], it is noted that different, less stringent, assumptions may be made on the structure of $\mathbf{A}(x)$: "This is equivalent to saying that every eigenvalue of $[\mathbf{A}(x_0)]$ has only linear elementary divisors; or that $[\mathbf{A}(x_0)]$ is diagonable, non-derogatory, non-defective, or similar to a diagonal matrix. This assumption could be replaced by the less restrictive condition that only the eigenvalue $[\lambda]$ should have linear elementary divisors."

Proof. As $\mathbf{u}$ and $\mathbf{v}$ are right, respectively left, eigenvectors corresponding to the eigenvalue λ, it follows that:

$$\mathbf{v}^\intercal \mathbf{A}(x_0)\mathbf{u} = \lambda \mathbf{v}^\intercal \mathbf{u}. \tag{3.16}$$

Because all involved variables are dependent on x, and we will only take the value (or the value of the derivative) in x_0, we will, to ease notation, omit this x dependency and implied evaluation in the point x_0. Furthermore, we will denote the derivative with respect to x with a prime "$'$". This allows us to write down the derivative of both sides of (3.16). These derivatives can further be simplified by using the product rule when applied to matrix multiplications:

$$\begin{aligned} & (\mathbf{v}^\intercal \mathbf{A}(x_0)\mathbf{u})' = (\lambda \mathbf{v}^\intercal \mathbf{u})' \\ \implies \quad & \mathbf{v}^{\intercal\prime}\mathbf{A}\mathbf{u} + \mathbf{v}^\intercal\mathbf{A}'\mathbf{u} + \mathbf{v}^\intercal\mathbf{A}\mathbf{u}' = \lambda'\mathbf{v}^\intercal\mathbf{u} + \lambda\mathbf{v}^{\intercal\prime}\mathbf{u} + \lambda\mathbf{v}^\intercal\mathbf{u}'. \end{aligned}$$

Using that $\mathbf{u}$ and $\mathbf{v}$ are eigenvectors ($\mathbf{A}\mathbf{u} = \lambda\mathbf{u}$ and $\mathbf{v}^\intercal\mathbf{A} = \lambda\mathbf{v}^\intercal$), gives us the required expression:

$$\begin{aligned} & \mathbf{v}^\intercal\mathbf{A}'\mathbf{u} = \lambda'\mathbf{v}^\intercal\mathbf{u} \\ \implies \quad & \lambda' = \frac{\mathbf{v}^\intercal\mathbf{A}'\mathbf{u}}{\mathbf{v}^\intercal\mathbf{u}}. \end{aligned}$$

Here we assumed $\mathbf{v}^\intercal\mathbf{u} \neq 0$. This concern is addressed in the appendix of [60]: by considering the subspaces of left and right eigenvectors corresponding to λ, it can be proven that $\mathbf{v}^\intercal\mathbf{u}$ never vanishes. □

Some care should be taken when two eigenvalues coincide. In [60], this is handled by considering the left and right eigenvector space corresponding to the multiple eigenvalue. But since we are dealing with only numerical results this (almost) never happens. Theorem 3.5 applied to $\mathbf{\Phi}(E)$ gives

$$\frac{\partial\lambda}{\partial E} = \frac{1}{\mathbf{v}^\intercal\mathbf{u}}\mathbf{v}^\intercal\frac{\partial\mathbf{\Phi}}{\partial E}\mathbf{u},$$

with left and right eigenvectors $\mathbf{v}$ and $\mathbf{u}$ respectively corresponding to the eigenvalue λ.

With the values for all these derivatives, all tools are available to formulate our proposed modified Newton's scheme. To simplify the notation, we introduce the matrix-operator σ which returns the spectrum of a matrix. This operator is defined on the $n \times n$-matrix $\mathbf{A}$ as

$$\sigma(\mathbf{A}) := \{\lambda \in \mathbb{C} \mid \exists\mathbf{u} \in \mathbb{C}^n \setminus \{\mathbf{0}\} : \mathbf{A}\mathbf{u} = \lambda\mathbf{u}\}.$$

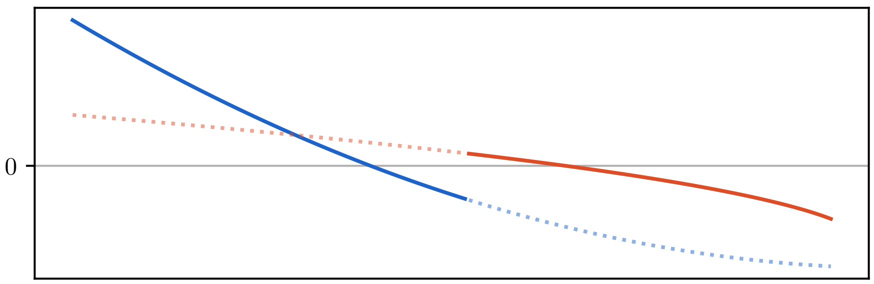


Figure 3.5: Two eigenvalues of $\mathbf{\Phi}(E)$ are plotted. Whichever is preferred by $f(E)$ from (3.17) is highlighted.

Now, we define the scalar function $f(E) : \mathbb{R} \to \mathbb{R}$ as

$$f(E) = \operatorname*{arg\,min}_{\lambda \in \sigma(\mathbf{\Phi}(E)) \cap \mathbb{R}} \left| \frac{\lambda}{\partial \lambda / \partial E} \right|. \tag{3.17}$$

On this function $f(E)$, the classical Newton's method can be used.

In figure 3.5, the idea of $f(E)$ from (3.17) is demonstrated. Assume the blue line and red line each represents an eigenvalue of $\mathbf{\Phi}(E)$. The regions where this eigenvalue is chosen by $f(E)$ are indicated. The idea is that f selects the function with the closest possible root in a linear approximation.

3.2.4.1 A numerical example

To provide some intuition about the eigenvalues λ of the mismatch matrix $\mathbf{\Phi}(E)$, we will analyze a numerical example. Consider the two-dimensional time-independent Schrödinger equation with potential function

$$V(x, y) = (1 + x^2)(1 + y^2) \tag{3.18}$$

on the domain $[-5.5; 5.5] \times [-5.5; 5.5]$ with homogeneous Dirichlet boundary conditions. This problem is also the numerical example from Ixaru's work [47].

Throughout this research, one of the first graphs we studied can be found in figure 3.6. Here all eigenvalues of the mismatch matrix $\mathbf{\Phi}(E)$ are plotted. For each value of E in this problem, $\mathbf{\Phi}(E)$ has N real eigenvalues, with N number of functions $b_i^{(k)}(x)$ on each sector, as described in section 3.1. Upon

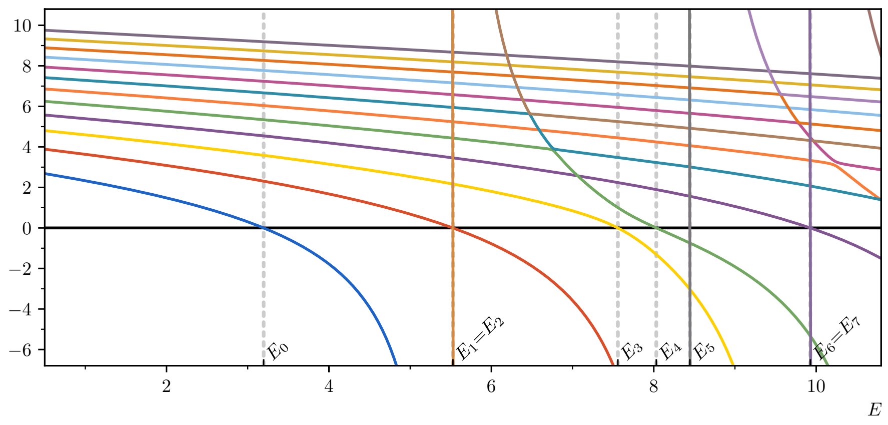


Figure 3.6: Each of the eigenvalues λ of the mismatch matrix $\mathbf{\Phi}(E)$ of the Schrödinger problem (3.18) with $N = 12$ as a function of E. The true eigenvalues of this problem are indicated with E_0, E_1, ... The lines are colored and continued to aid in the clarity of this illustration. But, do notice that this is only a best guess approximation.

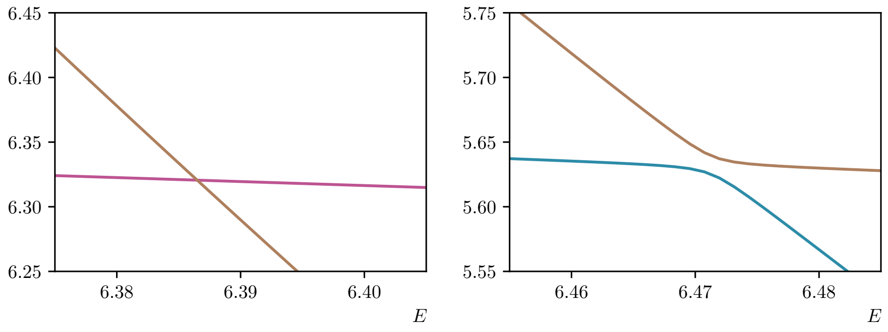


Figure 3.7: These are zoomed in views of the graph from 3.6. Some surprising behavior of the eigenvalues of $\mathbf{\Phi}(E)$ are visible.

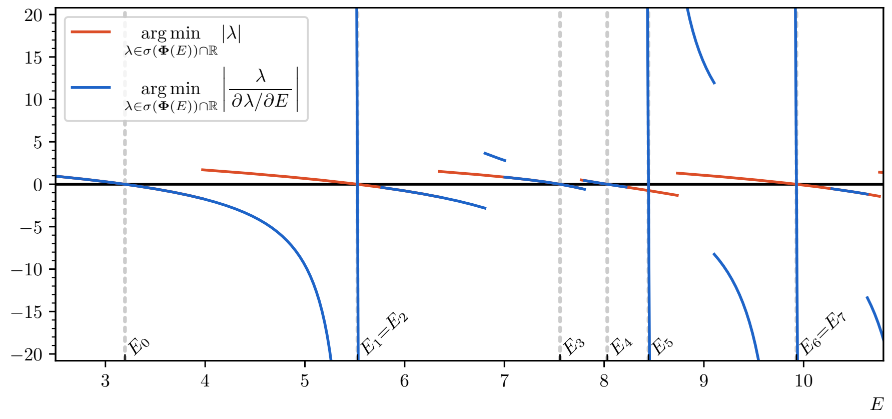


Figure 3.8: The function (3.17) for problem (3.18) is plotted in blue. In red, we have indicated the smallest real eigenvalue (in absolute value) of $\mathbf{\Phi}(E)$. In regions where the red curve seems missing, it coincides with the blue curve. At first glance, the striking vertical lines may be assumed to be asymptotes. This is not the case: in these regions, $f(E)$ changes rapidly.

seeing this graph, it is easy to say that one can 'follow' the trajectory of a single eigenvalue as E changes. In reality, this is not easy at all. For example, sometimes two eigenvalue curves intersect, other times they barely avoid each other, and seemingly switch which curve they follow. In figure 3.7, this strange behavior is visible.

These difficulties and nuances are hidden away behind the colors and seemingly continuous lines present in figure 3.6. Practically, this graph is generated from the eigenvalues (and their derivatives) of $\mathbf{\Phi}(E)$ for 10 000 values of E. Two eigenvalues $\lambda_j^{(i)}$, $\lambda_k^{(i+1)}$ for two adjacent values E_i and E_{i+1}, are determined to correspond to the same curve (and thus get the same color) if

$$\frac{\lambda_k^{(i+1)} - \lambda_j^{(i)}}{E_{i+1} - E_i} \approx \frac{\mathrm{d}\lambda_k^{(i+1)}}{\mathrm{d}E}$$

holds within a given accuracy.

In figure 3.8, the graph of the function $f(E)$ from equation (3.17) is plotted. Comparing this with figure 3.6, allows us to get an intuitive understanding of our application of Newton's method. But as can be seen in figure 3.8, this method is not perfect. For example, the region in which eigenvalue E_5 can be detected is relatively small. If the more straightforward $f(x) = \arg\min_{\lambda \in \sigma(\Phi(E)) \cap \mathbb{R}} |\lambda|$ was used, this region would be extremely tiny, as can be seen on figure 3.8. But still, even with (3.17), numerically it would be very unlikely to stumble upon this region, to detect that an eigenvalue was missing. For this reason we have developed a robust way to determine the number of eigenvalues less than the value E. This allows us to detect these missing values, and locate them. In the next section (3.3), we will develop some new theory to locate all eigenvalues reliably.

3.3 Determining the index of eigenvalues

This section is based upon as yet unpublished work, in collaboration with professor Hans Vernaeve.

In mathematical physics there are many problems dependent on, or consisting of, determining eigenvalues of a linear elliptic operator on a given domain with homogeneous Dirichlet boundary conditions. Examples include the Schrödinger equation, the wave equation, and the linear theory of elasticity to name a few. There are many methods, analytical as well as numerical, to solve for or to approximate these eigenvalues, e.g. shooting methods [51, 47], finite difference methods [102], methods for finding the lowest eigenvalues [18], or even the methods discussed in chapters 2, 3 and 4. There even exist (numerical) methods that are able to find eigenvalues in the neighborhood of a given value E without the need to compute all lower eigenvalues. For this last group of methods we have developed a theorem to count eigenvalues.

Up until now, there was no method, that the authors know of, to reliably determine the exact number of eigenvalues lower than a given value E for multidimensional problems. The best known result is already almost a century old: Courant's nodal domain theorem [22, vol I, chapter VI, paragraph 2, theorem 2][9]. This theorem states that, for an eigenfunction u with corresponding eigenvalue λ, there are at least as many eigenvalues (counted with multiplicity) less than λ, as the number of nodal domains of u minus one. A nodal domain is defined as a largest connected set of the domain on which the eigenfunction u does not become zero. The nodal domain theorem gives, when an eigenvalue λ with corresponding eigenfunction is known, a lower bound on the number of eigenvalues less than λ. However, only for restricted classes

[9]A more modern formulation of this theorem can be found in for example [12, theorem 1.1].

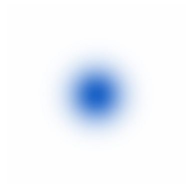
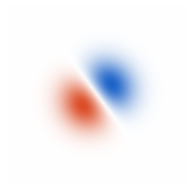
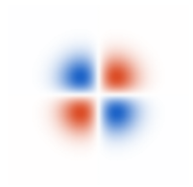
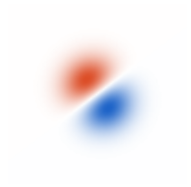
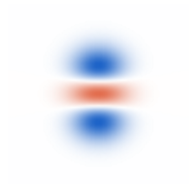
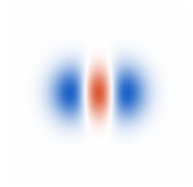
$\lambda_0 \approx 2.00$	$\lambda_{1,2} \approx 4.00$	$\lambda_{3,4,5} \approx 6.00$
 $\mathbf{u_{1,1}}$ **1**	 $\mathbf{u_{1,2}}$ **2**	 $\mathbf{u_{2,2}}$ **4**
	 $\mathbf{u_{2,1}}$ **2**	 $u_{1,3}$ 3
		 $u_{3,1}$ 3

Table 3.1: Density plots (see also figure 3.9) of the first six eigenfunctions of the quantum harmonic oscillator, $-\nabla^2 u + (x^2 + y^2)u = \lambda u$, on the square $[-5; 5] \times [-5; 5]$, with homogeneous Dirichlet boundary conditions. For each eigenfunction, the number of nodal domains is indicated. The eigenfunctions for which Courant's nodal domain theorem is strict, are highlighted in bold. These four eigenfunctions presented here are the only ones for which this strictness holds, for the harmonic oscillator.

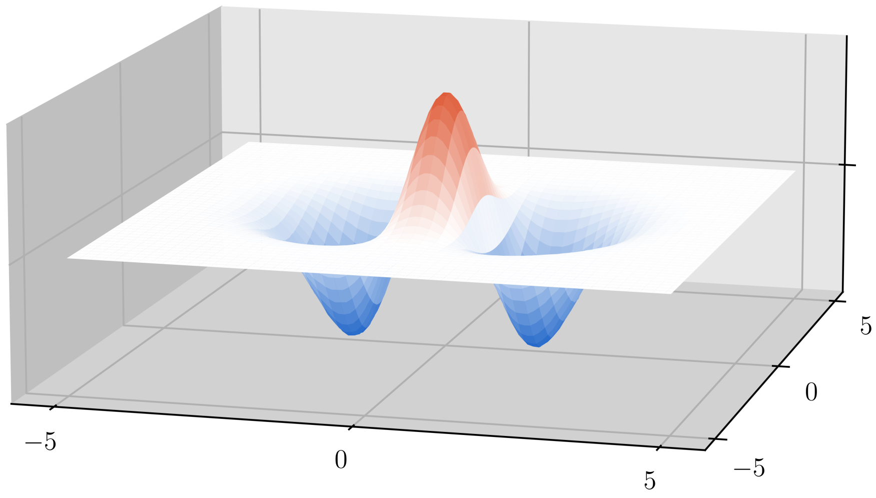


Figure 3.9: A three-dimensional graph of the eigenfunction $u_{3,1}$ of the quantum harmonic oscillator $-\nabla^2 u + (x^2 + y^2)u = \lambda u$, on the square $[-5; 5] \times [-5; 5]$, with homogeneous Dirichlet boundary conditions. This graph illustrates how the density plots from table 3.1 and figure 3.19 should be interpreted.

of one-dimensional problems, such as regular Sturm–Liouville problems with linear separated boundary conditions on bounded domains, this value is exact. In table 3.1, this fact is demonstrated by showing that the number of nodal domains of an eigenfunction does not lead to an exact estimate for the index of the corresponding eigenvalue.

A practical problem in applying Courant's nodal domain theorem is that numerically, it is quite difficult to reliably determine the number of nodal domains of a function. This motivated us to formulate theorem 3.9.

To develop the theoretical framework for this theorem, some definitions are needed. These will simplify the notation and proof of our results.

Definition 3.6. *The one-dimensional cross-section $l_{\Omega,i}(\mathbf{x})$ of a Lipschitz domain Ω along the i^{th} dimension through $\mathbf{x}$ is defined as:*

$$l_{\Omega,i}(\mathbf{x}) := \{\mathbf{y} \in \Omega \mid \forall j \in \{1, \ldots, n\} \setminus \{i\} : \mathbf{x}_j = \mathbf{y}_j\}\,.$$

Note that, when Ω is bounded, $l_{\Omega,i}(\mathbf{x})$ is the union of collinear line segments.

As such, the following definition represents the maximal sum of the lengths of each of these segments.

Definition 3.7. *The* directional diameter, $\mathrm{dirdiam}_i(\Omega)$, *of a bounded Lipschitz domain* $\Omega \subseteq \mathbb{R}^n$ *is defined as:*

$$\mathrm{dirdiam}_i(\Omega) := \sup_{\mathbf{x}\in\Omega} \int_{-\infty}^{\infty} I_\Omega(\mathbf{x}_1, \dots, \mathbf{x}_{i-1}, t, \mathbf{x}_{i+1}, \dots, \mathbf{x}_n)\, \mathrm{d}t.$$

Here $I_\Omega(\mathbf{x})$ *is defined as* 1 *if* $\mathbf{x} \in \Omega$ *and zero otherwise.*

In theorem 3.10, we will be able to identify each eigenvalue by using a continuous sequence of domains Ω_ϵ. For this sequence of domains, it should hold that if $\epsilon \to 0$, the domains should become unboundedly small. The notion of the *directional diameter* allows us to quantify this.

In figures 3.10 and 3.11, two examples of series of domains are given. Both of these sets of domains are applicable for theorem 3.10. We would like to note that the domain from figure 3.11 is particularly valuable for [47, 8].

The next section will be dedicated to proving this theorem. In the last section, we will provide some examples of and motivation for our work.

3.3.1 The main theorem

Throughout this section, the L^2-norm $\|\cdot\|_{L^2(\Omega)}$ and Sobolev norm $\|\cdot\|_{H^k(\Omega)}$ are being used. For clarity, we will provide the necessary definitions. For a more thorough discussion of these norms and their properties, many texts are available, for example [1].

Let $L^2(\Omega)$ be the space of functions $u : \Omega \to \mathbb{R}$ such that the norm

$$\|u\|_{L^2(\Omega)} := \left(\int_\Omega |u|^2 \right)^{\frac{1}{2}}$$

exists and is finite.

The Sobolev space $H^k(\Omega) = W^{k,2}(\Omega)$ is the space of all functions $u \in L^2(\Omega)$ such that, for the multi-index α with order $|\alpha|$ at most k, each weak partial derivative $D^\alpha u$ of u is a function in $L^2(\Omega)$. The Sobolev norm of u is defined as:

$$\|u\|_{H^k(\Omega)} := \left(\sum_{|\alpha|\le k} \int_\Omega |D^\alpha u|^2 \right)^{\frac{1}{2}}.$$

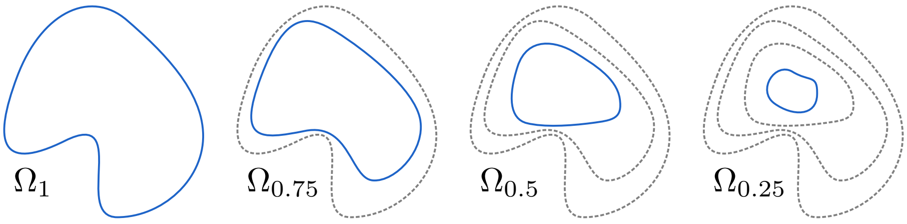


Figure 3.10: An example of a sequence of domains Ω_ϵ that satisfy the necessary conditions for theorem 3.10: the domains are nested, all domains are Lipschitz, and the horizontal as well as the vertical directional diameter converge to zero for $\epsilon \to 0^+$.

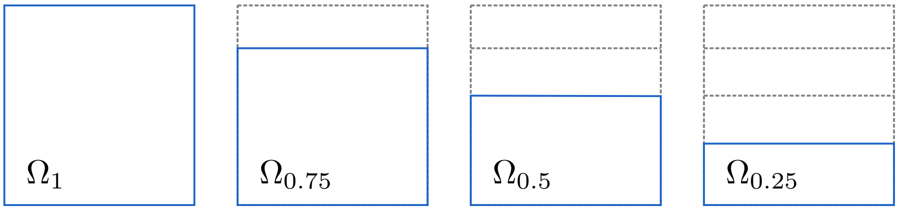


Figure 3.11: Another example of a sequence of domains Ω_ϵ that also satisfy the necessary conditions for theorem 3.10. Note that these conditions only require that any directional diameter converges to zero. In this example, the horizontal directional diameter does not converge to zero: $\lim_{\epsilon\to 0^+} \operatorname{dirdiam}_1(\Omega_\epsilon) \neq 0$. The vertical directional diameter does vanish: $\lim_{\epsilon\to 0^+} \operatorname{dirdiam}_2(\Omega_\epsilon) = 0$.

The function space $C^\infty(\Omega)$ consists of all infinitely differentiable functions defined on Ω, and $C_0^\infty(\Omega)$ consists of all infinitely differentiable functions f defined on Ω such that $f(\mathbf{x}) = 0$ for all $\mathbf{x} \in \partial\Omega$.

The subspace $H_0^k(\Omega)$ of $H^k(\Omega)$ is the closure of $C_0^\infty(\Omega)$ in $H^k(\Omega)$.

Before proving the main theorem of this section, we provide a kind of Friedrichs's inequality to estimate a lower bound for $\|u\|_{H^k(\Omega)}$ in terms of $\|u\|_{L^2(\Omega)}$, and the diameter[10] and any directional diameter of Ω.

Lemma 3.8. *Let $\Omega \subset \mathbb{R}^n$ be a bounded domain with Lipschitz boundary, and*

[10]The diameter of a domain Ω is defined as $\operatorname{diam}\Omega := \sup_{\mathbf{x},\mathbf{y}\in\Omega} \|\mathbf{x}-\mathbf{y}\|_2$.

$k \in \mathbb{N}^+$. *For each* $u \in H_0^k(\Omega)$, *and any* $j \le n$

$$\|u\|_{L^2(\Omega)} \le \sqrt{\operatorname{diam}(\Omega)\, \operatorname{dirdiam}_j(\Omega)}\|u\|_{H^k(\Omega)}. \tag{3.19}$$

Proof. For ease of notation, we will assume $j = 1$. Note that this proof is equally valid for any other j. Assume also $u \in C_0^\infty(\Omega) \cap H_0^k(\Omega)$. Because of the homogeneous Dirichlet boundary conditions, the fundamental theorem of calculus[11] can be used to estimate $|u(\mathbf{x})|^2$ for each $\mathbf{x} = (x_1, \dots, x_n) \in \Omega$

$$|u(\mathbf{x})|^2 = \left| \int_{-\infty}^{x_1} \frac{\partial u}{\partial x_1}(t, x_2, \dots, x_n)\, I_\Omega(t, x_2, \dots, x_n) \mathrm{d}t \right|^2.$$

In this formula, and throughout the proof, we formally pose that $u(\mathbf{x}) \equiv 0$ for $\mathbf{x} \notin \Omega$. Note that the fundamental theorem of calculus is applicable because u is compactly supported and in $C^1(\Omega)$.

Applying the Cauchy–Schwarz inequality provides some simplifications. Extending the integration domain removes the dependency on x_1.

$$\begin{aligned}
|u(\mathbf{x})|^2 &\le \int_{-\infty}^{x_1} I_\Omega(t, x_2, \dots, x_n)\mathrm{d}t \int_{-\infty}^{x_1} \left| \frac{\partial u}{\partial x_1}(t, x_2, \dots, x_n) \right|^2 \mathrm{d}t \\
&\le \operatorname{dirdiam}_1(\Omega) \int_a^b \left| \frac{\partial u}{\partial x_1}(t, x_2, \dots, x_n) \right|^2 \mathrm{d}t
\end{aligned}$$

with $a = \inf_{\mathbf{x}\in\Omega}\{\mathbf{u}_1 \,|\, \mathbf{u} \in l_{\Omega,1}(\mathbf{x})\}$ and $b = \sup_{\mathbf{x}\in\Omega}\{\mathbf{u}_1 \,|\, \mathbf{u} \in l_{\Omega,1}(\mathbf{x})\}$. Integrating both sides over Ω, applying Fubini's theorem[12], and noting that $b - a \le \operatorname{diam}(\Omega)$, yields the required expression for $u \in C_0^\infty(\Omega)$.

$$\begin{aligned}
\|u\|_{L^2(\Omega)}^2 &\le \operatorname{dirdiam}_1(\Omega) \int_\Omega \int_a^b \left| \frac{\partial u}{\partial x_1}(t, x_2, \dots, x_n) \right|^2 \mathrm{d}t\, \mathrm{d}\mathbf{x} \\
&= \operatorname{dirdiam}_1(\Omega)\,(b - a) \int_\Omega \left| \frac{\partial u}{\partial x_1}(\mathbf{x}) \right|^2 \mathrm{d}\mathbf{x} \\
&\le \operatorname{dirdiam}_1(\Omega)\, \operatorname{diam}(\Omega)\, \|u\|_{H^k(\Omega)}^2.
\end{aligned}$$

[11]The fundamental theorem of calculus states that if $f : \mathbb{R} \to \mathbb{R}$ is a continuous and differentiable function with $f'(x)$ integrable then

$$\int_a^b f'(x)\,\mathrm{d}x = f(b) - f(a).$$

[12]Fubini's theorem allows (under some easily checked conditions) to reorder the integration order in multidimensional integrals.

To prove this for each $u \in H_0^k(\Omega)$, we only have to note that $C_0^\infty(\Omega)$ is $H^k(\Omega)$-dense in $H_0^k(\Omega)$. Now $u \in H_0^k(\Omega)$ can be written as $u = \lim_{i \to +\infty} u_i$ with $u_i \in C_0^\infty(\Omega)$. This implies that

$$\|u\|^2_{L^2(\Omega)} \leq \operatorname{dirdiam}_1(\Omega) \operatorname{diam}(\Omega) \|u\|^2_{H^k(\Omega)},$$

for each $u \in H_0^k(\Omega)$. □

Now we are able to prove lemma 3.9. This prepares the main result of this section: theorem 3.10.

Lemma 3.9. *Let $\{\Omega_\epsilon \subseteq \mathbb{R}^n \mid \forall \epsilon \in]0,1]\}$ be a continuous family of diffeomorphic bounded Lipschitz domains, such that $a < b \implies \Omega_a \subsetneq \Omega_b$, and*

$$\exists j \in \{1, \ldots, n\} : \lim_{\epsilon \to 0^+} \operatorname{dirdiam}_j(\Omega_\epsilon) = 0.$$

Let $\mathcal{L}$ be a linear self-adjoint uniformly elliptic operator, defined on Ω_1. Define $\lambda_{\epsilon,i}$ as the i^{th} eigenvalue of $\mathcal{L}$ limited to Ω_ϵ with homogeneous Dirichlet boundary conditions. Then for each $i \in \mathbb{N}$, the values $\lambda_{\epsilon,i}$ as functions of ϵ

- *are continuous,*
- *are strictly decreasing,*
- *and are unbounded for $\epsilon \to 0^+$.*

Proof. Already in 1924, Courant and Hilbert [22, vol I. chapter V. paragraph 13] noted the continuity of the spectrum under perturbations of the elliptic operator. Hale described (e.g. [36]) how to translate regular perturbations in the domain to perturbations in the operator. Our set of diffeomorphic domains satisfies the necessary assumptions formulated by Hale, so continuity is guaranteed.

That $\lambda_{\epsilon,i}$ is a decreasing function of ϵ can be shown by application of [22, vol I, chapter VI, paragraph 2, theorem 3] *"Under the boundary condition $u = 0$ the n^{th} eigenvalue for a domain G never exceeds the n^{th} eigenvalue for a subdomain of G".* The strictness is addressed in a footnote by this theorem: *"In fact, it is always smaller when we are dealing with a proper subdomain."*.

To prove that $\lambda_{\epsilon,i}$ grows unboundedly, we start with Gårding's inequality [86, section 9.2.3]. It says that there exist C_1 and C_2 such that for all $u \in H_0^k(\Omega_\epsilon)$:

$$\frac{\langle \mathcal{L}u | u \rangle_{L^2(\Omega_\epsilon)}}{\|u\|^2_{L^2(\Omega_\epsilon)}} + C_1 \geq C_2 \frac{\|u\|^2_{H^k(\Omega_\epsilon)}}{\|u\|^2_{L^2(\Omega_\epsilon)}}. \tag{3.20}$$

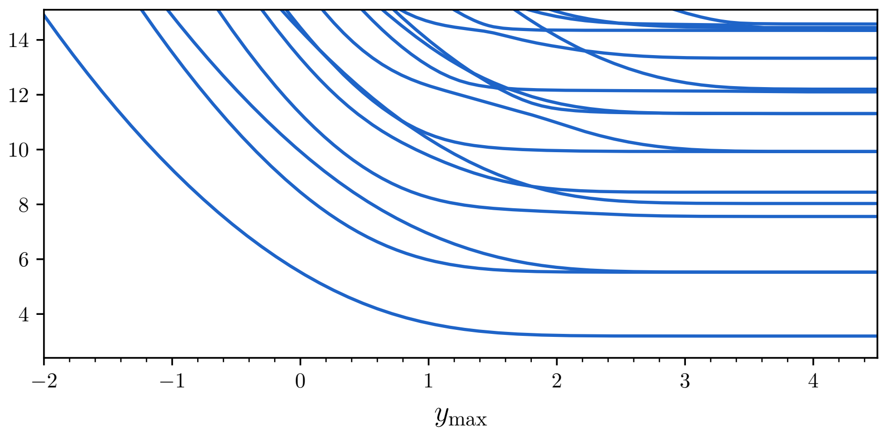


Figure 3.12: This graph displays all eigenvalues less than 15 for the two-dimensional Schrödinger problems $-\nabla^2\psi + (1+x^2)(1+y^2)\psi = E\psi$ on the rectangle $[-5.5; 5.5] \times [-5.5; y_{\max}]$ with homogeneous Dirichlet boundary conditions.

Note that the same C_1 and C_2 as for Ω_1 will also work for Ω_ϵ. Lemma 3.8 provides an estimate for the right-hand side of (3.20), which, in turn, provides an estimate for the operator $\mathcal{L}$:

$$\frac{\langle \mathcal{L}u|u\rangle_{L^2(\Omega_\epsilon)}}{\|u\|^2_{L^2(\Omega_\epsilon)}} \geq \frac{C_3}{\sqrt{\operatorname{dirdiam}_j(\Omega_\epsilon)}} - C_1,$$

for a constant C_3. By assumption the right-hand side is unbounded for $\epsilon \to 0$. Substituting the eigenfunction $u_{\epsilon,i}$, corresponding to the eigenvalue $\lambda_{\epsilon,i}$, into the left-hand side yields that $\lambda_{\epsilon,i}$ has to be unbounded as well. $\square$

As an example, the spectrum of a Schrödinger operator is plotted as a function of the nested sequence of domains $[-5.5; 5.5] \times [-5.5; y_{\max}]$ in figure 3.12. Each curve corresponds to an eigenvalue. The figure suggests that all of these lines are continuous, decreasing, and unbounded for $y_{\max} \to -5.5^+$, in accordance with lemma 3.9. A horizontal line for a fixed value E will intersect each curve corresponding to a lower eigenvalue on the whole domain. This number of intersections thus gives the number of eigenvalues lower than this given value E. If E itself is an eigenvalue, then we have also found its index.

The graph from figure 3.12 and the notion of a horizontal line with fixed value of E, give rise to following theorem.

Theorem 3.10. *Let $\mathcal{L}$ be a linear self-adjoint uniformly elliptic operator on a bounded Lipschitz domain $\Omega_1 \subseteq \mathbb{R}^n$ with homogeneous Dirichlet boundary conditions. Following [36], define a topology on all domains Ω which are C^k-diffeomorphic to Ω_1. Assume a continuous family of such C^k-diffeomorphic domains Ω_ϵ for $\epsilon \in]0,1]$ is given such that $\Omega_a \subseteq \Omega_b$ if $a < b$ and $\exists j \in \{1,\ldots,n\} : \lim_{\epsilon \to 0^+} \operatorname{dirdiam}_j(\Omega_\epsilon) = 0$.*

For a given value E, the number of eigenvalues of $\mathcal{L}$ on Ω_1 less than or equal to E is exactly the same as the number of domains Ω_ϵ on which E is an eigenvalue of $\mathcal{L}$ limited to Ω_ϵ with homogeneous Dirichlet boundary conditions. Domains on which E is an eigenvalue with multiplicity d are counted d times.

As a necessary condition we imposed that, for the sequence of domains Ω_ϵ, $\operatorname{dirdiam}_j(\Omega_\epsilon)$ should converge to zero. This may seem a quite restrictive condition, however in practice, it is not. Most (natural) sequences of domains do satisfy this condition in at least one axis. In the next section, we provide some examples of different kinds of such domains.

For direct methods, such as (semi-)discretization methods, theorem 3.10 cannot directly be applied. However, for shooting methods ([47, 8]), this theorem is extremely useful. In such methods, for a fixed value of E, an error function is computed by propagating solutions over the domain. The error function measures the mismatch between the propagated solution and the boundary conditions.

While propagating a solution, one can keep track of how many times the boundary conditions are met inside the domain. Each of these occurrences corresponds to a subdomain on which E is an eigenvalue, which in turn, by theorem 3.10, corresponds to a unique eigenvalue less than E.

3.3.2 Examples

We provide a few examples. The first example illustrates that our result is a more general formulation of a well known theorem for the one-dimensional problem. In the second example we will do some symbolic calculations for a disc-shaped domain. The third example will demonstrate our implemention of theorem 3.10 in the method developed in section 3.1. Finally, we will take a look at a much more exotic family of domains to illustrate the applicability of our theorem to use-cases far beyond what our numerical method is capable of.

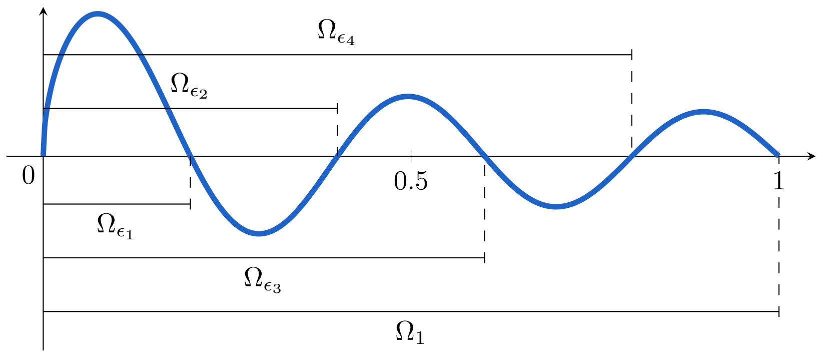


Figure 3.13: Eigenfunction y_4, with $\lambda_4 \approx 246.74$, of the 1/2-order Bessel equation [95] $-\frac{\mathrm{d}x}{\mathrm{d}}\left(x\frac{\mathrm{d}y}{\mathrm{d}x}\right)+\frac{1}{4x}y=\lambda xy$ on $[0,1]$, with $y(0)=y(1)=0$. The four subintervals, $\Omega_{\epsilon_1}=[0,\epsilon_1],\ldots,\Omega_{\epsilon_4}=[0,\epsilon_4]$, on which λ_4 is also an eigenvalue of this equation, are indicated.

3.3.2.1 One-dimensional Sturm–Liouville equation

Let λ_i be the $(i+1)^{\text{th}}$ eigenvalue[13], with eigenfunction $y_i(x)$, of the regular Sturm–Liouville equation

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}y_i}{\mathrm{d}x}\right)+q(x)y_i=\lambda_i w(x)y_i$$

on the interval $[a,b]$ with homogeneous Dirichlet boundary conditions $y_i(a)=y_i(b)=0$. Defining the series of domains $\Omega_\epsilon=[a,a+\epsilon\,(b-a)]$ allows us to apply theorem 3.10. Thus, there should be i intervals $[a,c]$ with $a<c<b$, on which λ_i is an eigenvalue. As eigenfunctions are unique, up to a scaling factor, y_i should have exactly i zeros inside $[a,b]$. This is a well-known property in Sturm–Liouville theory, see theorem 2.4. This fact is also illustrated in figure 3.13 for the Bessel equation.

Multiple shooting procedures [9, 68, 45] have been developed that implement this well-known property for Sturm–Liouville problems using the Prüfer transformation [79].

[13]The first eigenvalue is denoted as λ_0, the second as $\lambda_1\ldots$

3.3.2.2 Standing waves on a disc

Consider the eigenvalues and eigenfunctions of the two-dimensional wave equation

$$-\nabla^2\psi(x,y) = \lambda\psi(x,y) \tag{3.21}$$

on the unit disc with homogeneous Dirichlet boundary condition.

In many introductory textbooks to (partial) differential equations, the study of the wave equation on a drum is considered. In most texts, if not all, the idea boils down to: with a transformation into polar coordinates and separation of variables, this two-dimensional problem can be transformed into two one-dimensional problems. One of these can be solved directly, the other gives rise to the so-called Bessel functions.

Here we will give a brief overview of the symbolic calculations to solve (3.21) on the unit disc $B(\mathbf{0},1)$, with homogeneous Dirichlet boundary conditions. For more details one could consult for example [7, chapter 4]. As stated, we first transform (3.21) to polar coordinates. For this we pose $x = r\cos(\theta)$ and $y = r\sin(\theta)$, with $r \in [0,1]$ and $\theta \in [0,2\pi]$. The solutions $u(r,\theta) := \psi(r\cos(\theta), r\sin(\theta))$ are now considered in the transformed domain. Applying the chain rule to $\nabla^2\psi$ yields the new equation

$$-\frac{\partial^2 u}{\partial r^2} - \frac{1}{r}\frac{\partial u}{\partial r} - \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} = -\lambda u. \tag{3.22}$$

The boundary conditions are likewise transformed: $u(1,\theta) = 0$ for all $\theta \in [0,2\pi]$ and $u(r,0) = u(r,2\pi)$ and $\frac{\partial u}{\partial \theta}(r,0) = \frac{\partial u}{\partial \theta}(r,2\pi)$ for all $r \in [0,1]$.

Now we use the method of separation of variables to transform (3.22) into two one-dimensional problems. For this, we separate $u(r,\theta) = R(r)\Theta(\theta)$ into a product of an r-dependent function and a θ-dependent function. This allows us to write

$$-\frac{r^2R''}{R} - \frac{rR'}{R} - \frac{\Theta''}{\Theta} = \lambda r^2.$$

Rearranging terms to an r-dependent side and a θ-dependent side yields two separate equations

$$\lambda r^2 + \frac{r^2R''}{R} + \frac{rR'}{R} = k \qquad \text{and} \qquad -\frac{\Theta''}{\Theta} = k,$$

with k a constant and boundary conditions $R(1) = 0$, $\Theta(0) = \Theta(2\pi)$ and $\Theta'(0) = \Theta'(2\pi)$.

Solutions for $\Theta(\theta)$ can directly be found as a solution of a linear second order differential equation with constant coefficients. Thus, Θ only has periodic solutions

$$\Theta(\theta) = A_m \cos(m\theta) + B_m \sin(m\theta),$$

if $k = m^2$ with $m \in \{0, 1, 2, \dots\}$. Note that if $m > 0$, there are two linear independent solutions for Θ.

The differential equation for $R(r)$ becomes

$$r^2 R'' + rR + (\lambda r^2 - m^2)R = 0.$$

We find that solutions to our equation can be written using the mth Bessel function of the first kind $R(r) = J_m(r\sqrt{\lambda})$, for more details about these functions see for example [7, section 4.7]. These solutions only satisfy the boundary condition $R(1) = 0$ if and only if $\sqrt{\lambda}$ is a positive zero of J_m. If we denote $j_{m,n}$ as the nth positive root of the mth Bessel function J_m, we find that all eigenvalues of (3.22), and also of (3.21), are given as the squares of roots of the Bessel functions. So $\lambda = j_{m,n}^2$ is an eigenvalue of (3.21). It is a single eigenvalue if $m = 0$, otherwise it is a double eigenvalue. The following table contains the first ten eigenvalues, counted with multiplicity.

Eigenvalue	λ_0	$\lambda_{1,2}$	$\lambda_{3,4}$	λ_5	$\lambda_{6,7}$	$\lambda_{8,9}$
Analytical value	$j_{0,1}^2$	$j_{1,1}^2$	$j_{2,1}^2$	$j_{0,2}^2$	$j_{3,1}^2$	$j_{1,2}^2$
Numerical value	5.783	14.68	26.37	30.47	40.71	49.22

To apply theorem 3.10 we need to define a family of domains. A most obvious choice would be all concentric discs with radius at most one. With a variable substitution we know the eigenvalues of the wave equation on $\Omega_\epsilon := B(\mathbf{0}, \epsilon)$ to be $\frac{\lambda_i}{\epsilon^2}$ for all eigenvalues λ_i on the unit disc. It is obvious that the eigenvalues $\frac{\lambda_i}{\epsilon^2}$ as a function of ϵ, are continuous, decreasing, and unbounded as $\epsilon \to 0^+$. This confirms lemma 3.9.

Concentric discs may not be the most thrilling examples. But in the example in section 3.3.2.4 we will consider a much more interesting family of domains for the wave equation on a disc.

3.3.2.3 A two-dimensional Schrödinger equation

In this example, we will take a look at a two-dimensional time-independent Schrödinger equation with homogeneous Dirichlet boundary conditions on a rectangle:

$$-\nabla^2\psi + V(x,y)\psi = E\psi.$$

In section 3.4.3, we will study the Schrödinger problem with the potential

$$V(x,y) = (1+x^2)(1+y^2)$$

on the square $[-5.5; 5.5] \times [-5.5; 5.5]$ with homogeneous Dirichlet boundary conditions. The first eigenvalues are reported in [47] to be as follows.

E_0	3.1959181	$E_6 = E_7$	9.9280611
$E_1 = E_2$	5.5267439	$E_8 = E_9$	11.3118171
E_3	7.5578033	E_{10}	12.1032536
E_4	8.0312723	E_{11}	12.2011790
E_5	8.4445814	E_{12}	13.3323313

For this problem we will use the method as described in section 3.1. In summary, this is a shooting method, which means that it guesses a value for E and calculates a matching error. This matching error is then used to improve the estimation of E. This process is repeated until E is found up to the desired accuracy.

By propagating all possible eigenfunctions $u(x,y)$ from the bottom and the top of the domain to the matching line, this matching error is calculated. In the context of theorem 3.10, this propagation can be thought of as going through all possible subdomains from figure 3.11. If, while propagating, any of the possible eigenfunctions u becomes identically zero on a line, then a subdomain is found on which E is an eigenvalue.

For each E, while propagating, one can keep track of how many u did become zero on a line, or in other words, on how many subdomains E is an eigenvalue. Following theorem 3.10, this number tells us how many eigenvalues there are less than E on the whole domain. In figure 3.14, keeping track of when an eigenfunction becomes zero on a line is demonstrated. For $E = 11$, all domains are visualized where any $u(x,y) = 0$ for all $x \in [x_{\min}, x_{\min}]$. Notice that there may be domains on which multiple eigenfunctions are found for that particular value of E. These domains should be counted multiple times. In figure 3.14, we see that $y_{\max} \approx 0.80845$ is counted twice. From the tabulated true eigenvalues in [47], we know that there are indeed exactly 8 eigenvalues less than $E = 11$. This example thus verifies theorem 3.10 as well.

For completeness, we will also verify lemma 3.9 by taking a closer look at the graph from figure 3.12. In figure 3.15 we see a zoomed version. These graphs

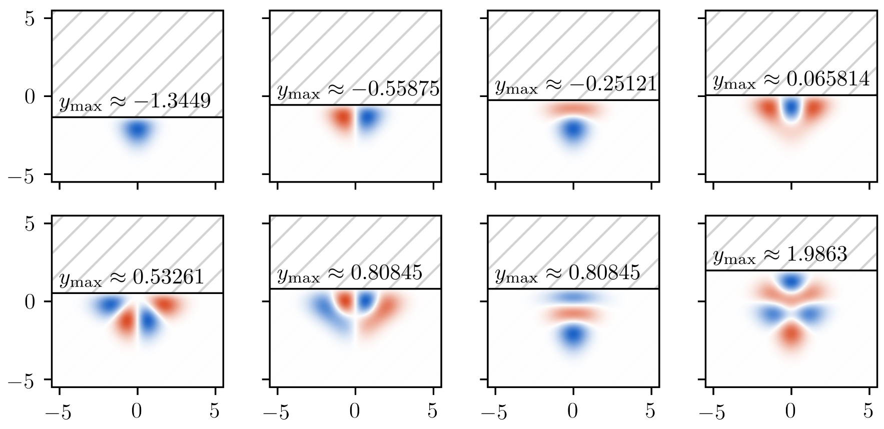


Figure 3.14: There are 8 (smaller) rectangles on which $E = 11$ is an eigenvalue of the Schrödinger problem from example 3.3.2.3.

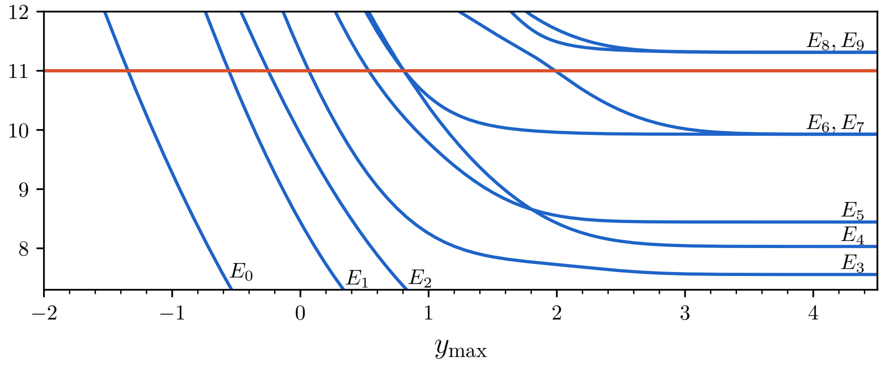


Figure 3.15: Here, a zoomed in version of figure 3.12 is displayed. Each blue graph represents an eigenvalue of the Schrödinger equation from example 3.3.2.3. The fixed line $E = 11$ is represented in red. Each crossing of the red line and a blue graph represents a smaller subdomain on which $E = 11$ is an eigenvalue. These subdomains, with corresponding eigenfunction, are illustrated in figure 3.14.

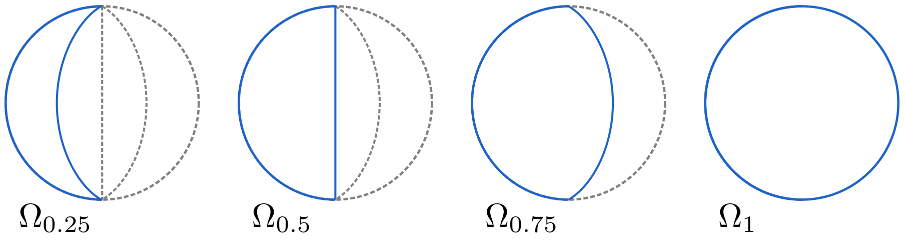


Figure 3.16: An illustration of the family of domains we consider in section 3.3.2.4.

describe how the eigenvalues evolve through the growing domain. If we draw a horizontal line at $E = 11$, as in figure 3.15, eight curves intersect this line. Each curve corresponds to a lower lying eigenvalue, just as lemma 3.9 predicts.

3.3.2.4 A more exotic family of domains

As a numerical demonstration for theorem 3.10 with less trivial domains, we consider the wave equation on a series of moon-shaped domains with homogeneous Dirichlet boundary conditions. Visually, these domains can be found in figure 3.16. Mathematically, we define the transformation $T : [-\frac{\pi}{2}, \frac{\pi}{2}] \times [-1, 1] \to \mathbb{R}^2$ as:

$$T(\alpha, t) = \begin{pmatrix} \cot(\alpha)\sin(t\alpha) \\ \frac{1}{\sin(\alpha)} - \cot(\alpha)\cos(t\alpha) \end{pmatrix}. \tag{3.23}$$

Strictly speaking, for $\alpha = 0$, $T(\alpha, t)$ is undefined. In these points, the limit $\lim_{\alpha\to 0} T(\alpha, t)$ should be considered. In figure 3.17, this transformation is visualized. Note that when t is limited to $[-1, 2\epsilon - 1]$ the transformation results in a moon-shape.

To illustrate the applicability of theorem 3.9, we will consider the family of continuous subdomains

$$\Omega_\epsilon = \left\{ T(\alpha, t) \,\middle|\, \forall \alpha \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \forall t \in [-1, 2\epsilon - 1] \right\}.$$

On each of these subdomains Ω_ϵ, we will approximate the eigenvalues of the Schrödinger equation with zero potential:

$$-\nabla^2 \psi(x, y) = \lambda \psi(x, y). \tag{3.24}$$

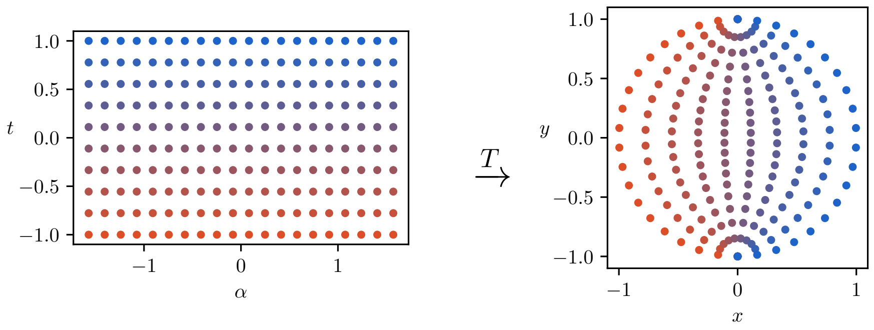


Figure 3.17: The transformation $T(\alpha, t)$ from (3.23) is applied to a grid of points.

As a boundary condition we impose $\psi(x, y) = 0$ on the boundary $\partial\Omega_\epsilon$.

Later on, the approximate eigenvalues found can be compared to the eigenvalues on the unit disc Ω_1, as described in section 3.3.2.2. Theorem 3.10 promises that, for an eigenvalue λ_i on the unit disc, exactly i moon-shaped subdomains can be found on which this λ_i is also an eigenvalue.

Transforming the problem onto a rectangular domain

Approximating solutions for the eigenvalues of (3.24) on such an exotic domain Ω_ϵ is not a trivial task. In principle, this problem could be seen as solving the Schrödinger equation on $[-1, 1]^2$ with the potential

$$V(x, y) = \begin{cases} 0 & \text{if } (x, y) \in \Omega_\epsilon \\ +\infty & \text{otherwise.} \end{cases}$$

Numerically, this does not make the problem easier. On the one hand, the rectangular domain would allow us to employ the earlier developed method. On the other hand however, the potential becomes infinite, which is very difficult to implement with sufficiently high accuracies. Also, Ixaru's two-dimensional method has difficulties with non-continuous potentials. Note that the inability to tackle non-continuous problems is a phenomenon present in many high-order numerical methods. In section 2.5.2, we have seen that with one-dimensional problems, `Matslise` can avoid these issues by smartly choosing its piecewise approximation. For higher-dimensional problems, these kinds of tricks are no longer possible.

We propose another technique. We can transform equation (3.24) from these moon-shaped domains to a more manageable rectangular domain. The cost of this transformation is that (3.24) will no longer be a Schrödinger equation.

As stated earlier, the transformation we will apply is the function $(x, y) = T(\alpha, t)$ from (3.23). To formalize this, we introduce $\phi(\alpha, t)$ as

$$\phi(\alpha, t) = \psi(T(\alpha, t)).$$

With this transformation, the Schrödinger equation (3.24) transforms into

$$-\mathcal{D}\phi(\alpha, t) = \lambda\phi(\alpha, t), \tag{3.25}$$

with $\phi(\alpha, t)$ defined on the domain $\Xi_\epsilon = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \times [-1, 2\epsilon - 1]$. Note that $T(\Xi_\epsilon) = \Omega_\epsilon$. As boundary conditions we impose $\phi(\alpha, t) = 0$ if $(\alpha, t) \in \partial\Xi_\epsilon$. In this expression, $\mathcal{D}$ is the differential operator corresponding to $\nabla^2\psi(x, y)$. Calculating this operator by hand is tedious, therefore we will use `sage` to do the symbolic heavy lifting for us. To be able to work with the operator $\mathcal{D}$, we first need a procedure to compute it. To ease notation, we will denote $T(\alpha, t) = (T^x(\alpha, t), T^y(\alpha, t))$, and omit the arguments to the functions ϕ, ψ and T. Derivatives will be denoted as $\phi_\alpha = \frac{\partial\phi}{\partial\alpha}$ and $\phi_{\alpha\alpha} = \frac{\partial^2\phi}{\partial\alpha^2}$. We compute all first and second derivatives of $\phi(\alpha, t)$.

$$\begin{cases} \phi_\alpha = \psi_x T^x_\alpha + \psi_y T^y_\alpha \\ \phi_t = \psi_x T^x_t + \psi_y T^y_t \\ \phi_{\alpha\alpha} = \psi_x T^x_{\alpha\alpha} + \psi_y T^y_{\alpha\alpha} + \psi_{xx}(T^x_\alpha)^2 + 2\psi_{xy}T^x_\alpha T^y_\alpha + \psi_{yy}(T^y_\alpha)^2 \\ \phi_{\alpha t} = \psi_x T^x_{\alpha t} + \psi_y T^y_{\alpha t} + \psi_{xx}(T^x_\alpha)(T^x_t) + \psi_{xy}\left(T^x_\alpha T^y_t + T^x_t T^y_\alpha\right) \\ \qquad\quad + \psi_{yy}(T^y_\alpha)(T^y_t) \\ \phi_{tt} = \psi_x T^x_{tt} + \psi_y T^y_{tt} + \psi_{xx}(T^x_t)^2 + 2\psi_{xy}T^x_t T^y_t + \psi_{yy}(T^y_t)^2 \end{cases} \tag{3.26}$$

The question now is: can we isolate $\psi_{xx} + \psi_{yy}$ from the right-hand side of the system from (3.26)? As this is a linear system in the derivatives of ψ, there is a unique linear combination $\mathbf{c}(\alpha, t) = (c^\alpha, c^t, c^{\alpha\alpha}, c^{\alpha t}, c^{tt})$ such that:

$$c^\alpha\phi_\alpha + c^t\phi_t + c^{\alpha\alpha}\phi_{\alpha\alpha} + c^{\alpha t}\phi_{\alpha t} + c^{tt}\phi_{tt} = \psi_{xx} + \psi_{yy}.$$

This linear combination allows us to finally determine the operator $\mathcal{D}$, and by extension, the partial differential equation (3.24) transforms into. Note that, since $\mathbf{c}$ is dependent on α and t, the operator $\mathcal{D}$ has this dependency as well,

$$\mathcal{D} = c^\alpha\frac{\partial}{\partial\alpha} + c^t\frac{\partial}{\partial t} + c^{\alpha\alpha}\frac{\partial}{\partial\alpha\alpha} + c^{\alpha t}\frac{\partial}{\partial\alpha t} + c^{tt}\frac{\partial}{\partial tt}.$$

A specialized numerical method

Before we can analyze the eigenvalues of the transformed operator $-\mathcal{D}$ in relation to the domain Ω_ϵ, we still need a method to approximate these eigenvalues. For general partial differential equations with appropriate boundary conditions, some standard techniques are available. [37, Chapter 11] contains an overview of some of these methods. Our choices include, but are not limited to, a finite difference method, a finite element method, or a semidiscrete method with shooting. For our purposes, a method that is easy to implement that gives reliable results will be the best choice. A simple finite difference scheme will be sufficient.

We will place an $n_\alpha \times n_t$ grid on the domain Ξ_ϵ. This gives rise to the equidistant points $-\frac{\pi}{2} = \alpha_0, \alpha_1, \dots, \alpha_{n_\alpha} = \frac{\pi}{2}$, and the points $-1 = t_0, t_1, \dots t_{n_t} = -1 + 2\epsilon$. The distances between two points are given by $h_\alpha = \frac{\pi}{n_\alpha}$ and $h_t = \frac{2\epsilon}{n_t}$. Because of the homogeneous Dirichlet boundary conditions $\phi_{0,j} = \phi_{n_\alpha,j} = \phi_{i,0} = \phi_{i,n_t} = 0$ for all i and j. These grid points allow us to write down approximations of the first and second derivatives of $\phi(\alpha, t)$ in each of the grid points $\phi_{i,j} = \phi(\alpha_i, t_j)$.

$$\begin{aligned}
\frac{\partial \phi}{\partial \alpha}(\alpha_i, t_j) &\approx \frac{1}{2h_\alpha}\left(\phi_{i+1,j} - \phi_{i-1,j}\right) \\
\frac{\partial \phi}{\partial t}(\alpha_i, t_j) &\approx \frac{1}{2h_t}\left(\phi_{i,j+1} - \phi_{i,j-1}\right) \\
\frac{\partial^2 \phi}{\partial \alpha^2}(\alpha_i, t_j) &\approx \frac{1}{h_\alpha^2}\left(\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}\right) \\
\frac{\partial^2 \phi}{\partial \alpha \partial t}(\alpha_i, t_j) &\approx \frac{1}{4h_\alpha h_t}\left(\phi_{i+1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j+1} + \phi_{i-1,j-1}\right) \\
\frac{\partial^2 \phi}{\partial t^2}(\alpha_i, t_j) &\approx \frac{1}{h_t^2}\left(\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}\right).
\end{aligned}$$

When working with finite difference schemes, especially for a more advanced operator such as $\mathcal{D}$, writing out all formulae becomes quite tedious. Therefore, it is much more comprehensive if we introduce some matrix notation. Let us denote $\mathbf{I}_n$ as the $n \times n$ identity matrix. Furthermore, a diagonal matrix with $a_1, \dots, a_n$ on the diagonal will be denoted as $\operatorname{diag}_n(a_1, \dots, a_n)$, and an $n \times n$ tridiagonal Toeplitz matrix, with c_0 on the main diagonal, c_{-1} below it and c_1 above it, will be denoted as $\operatorname{tridiag}_n(c_{-1}, c_0, c_1)$. To aid the notation of the finite difference approximations we introduce $\mathbf{D}_n^{(1)} = \frac{1}{2}\operatorname{tridiag}_n(-1, 0, 1)$ and $\mathbf{D}_n^{(2)} = \operatorname{tridiag}_n(1, -2, 1)$.

The Kronecker product of a $k \times l$ matrix $\mathbf{A}$ and an $m \times n$ matrix $\mathbf{B}$ is defined as the $km \times ln$ block matrix:

$$\mathbf{A} \otimes \mathbf{B} := \begin{pmatrix} A_{1,1}\mathbf{B} & A_{1,2}\mathbf{B} & \dots & A_{1,l}\mathbf{B} \\ A_{2,1}\mathbf{B} & A_{2,2}\mathbf{B} & \dots & A_{2,l}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ A_{k,1}\mathbf{B} & A_{k,2}\mathbf{B} & \dots & A_{k,l}\mathbf{B} \end{pmatrix}.$$

To approximate (3.25) as a matrix problem, we need to aggregate the grid points $\phi_{i,j}$ as a vector. For this we define

$$\boldsymbol{\phi} := \begin{pmatrix} \phi_{1,1} & \phi_{2,1} & \dots & \phi_{n_\alpha-1,1} & \phi_{1,2} & \dots & \phi_{n_\alpha-1,n_t-1} \end{pmatrix}^\intercal.$$

Also, to be able to write down $\mathcal{D}$ as a matrix operation, we need to define

$$\mathbf{C}^t := \operatorname{diag}\left(c^t(\alpha_1, t_1), \dots, c^t(\alpha_{n_\alpha-1}, t_1), c^t(\alpha_1, t_2), \dots, c^t(\alpha_{n_\alpha-1}, t_{n_t-1})\right),$$

and analogously for $\mathbf{C}^\alpha$, $\mathbf{C}^{\alpha\alpha}$, $\mathbf{C}^{\alpha t}$ and $\mathbf{C}^{tt}$.

All these notations allow us to approximate (3.25) as a matrix eigenvalue problem:

$$-\mathbf{M}\boldsymbol{\phi} = \lambda\boldsymbol{\phi}$$

with

$$\begin{aligned} \mathbf{M} = {} & \frac{1}{h_\alpha}\mathbf{C}^\alpha\left(\mathbf{I}_{n_t-1} \otimes \mathbf{D}^{(1)}_{n_\alpha-1}\right) + \frac{1}{h_t}\mathbf{C}^t\left(\mathbf{D}^{(1)}_{n_t-1} \otimes \mathbf{I}_{n_\alpha-1}\right) \\ & + \frac{1}{h_\alpha^2}\mathbf{C}^{\alpha\alpha}\left(\mathbf{I}_{n_t-1} \otimes \mathbf{D}^{(2)}_{n_\alpha-1}\right) + \frac{1}{h_t^2}\mathbf{C}^{tt}\left(\mathbf{D}^{(2)}_{n_t-1} \otimes \mathbf{I}_{n_\alpha-1}\right) \\ & + \frac{1}{h_\alpha h_t}\mathbf{C}^{\alpha t}\left(\mathbf{I}_{n_t-1} \otimes \mathbf{D}^{(1)}_{n_\alpha-1}\right)\left(\mathbf{D}^{(1)}_{n_t-1} \otimes \mathbf{I}_{n_\alpha-1}\right). \end{aligned}$$

Analysis of eigenvalues in relation to the domain
With a numerical method to compute eigenvalues of the wave equation (3.24) on moon-shaped domains Ω_ϵ, and a comprehensive analysis of the true eigenvalues of this equation on the disc Ω_1, we are able to illustrate theorem 3.9.

By employing our numerical method on different domains Ω_ϵ, we can construct figure 3.18. This figure was computed with $n_\alpha = n_t = 61$ (which makes $\mathbf{M}$ a 3600×3600 sparse matrix) for 200 equidistantly-spaced values for ϵ. Here, each line corresponds to a different eigenvalue. The lowest blue line visualizes the changes in the lowest eigenvalue according to the domain. For small domains

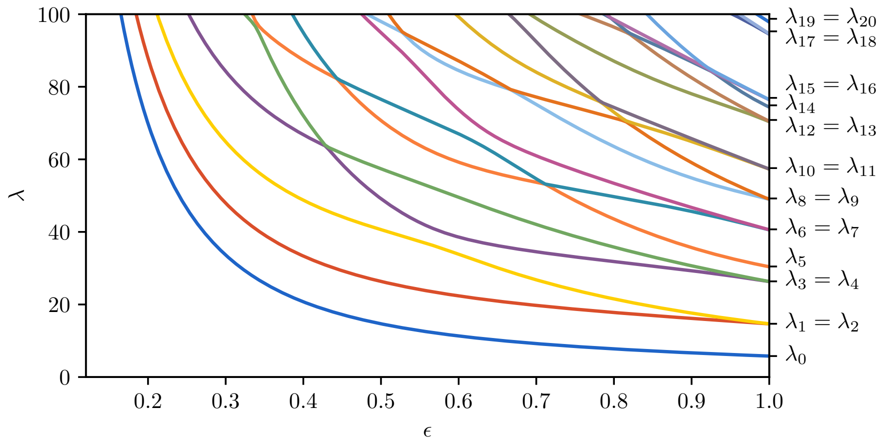


Figure 3.18: Eigenvalues of the Schrödinger equation with zero potential on the family of domains of section 3.3.2.4 in relation to ϵ.

$\epsilon \to 0$, this value becomes larger. The red line right above the blue line corresponds to the eigenvalue with index 1 of the wave equation on Ω_ϵ. For Ω_1, this eigenvalue has a double multiplicity $\lambda_1 = \lambda_2$. But for all smaller domains, this degeneracy disappears, and λ_1 is a single eigenvalue. Which eigenvalue each line represents is marked on the right side. In accordance with theorem 3.9, all lines in figure 3.18 are continuous, decreasing and unbounded when $\epsilon \to 0$.

Consider now the fixed value of $E = 45$. From the analysis in 3.3.2.2 we know that there are eight eigenvalues, counted with multiplicity, less than E. This means that, following theorem 3.10, there should be exactly eight moon-shaped domains on which E is also an eigenvalue. These eight domains, with the corresponding eigenfunction of $E = 45$, are plotted in figure 3.19.

3.4 Numerical experiments

In the previous section, we have made some theoretical advancements. To demonstrate the relevance and effectiveness of the provided theory, we proudly present some numerical results. In summary, the obtained results are at least as accurate as the original method in [47], but they are found significantly quicker, even when corrected for the more powerful modern CPUs.

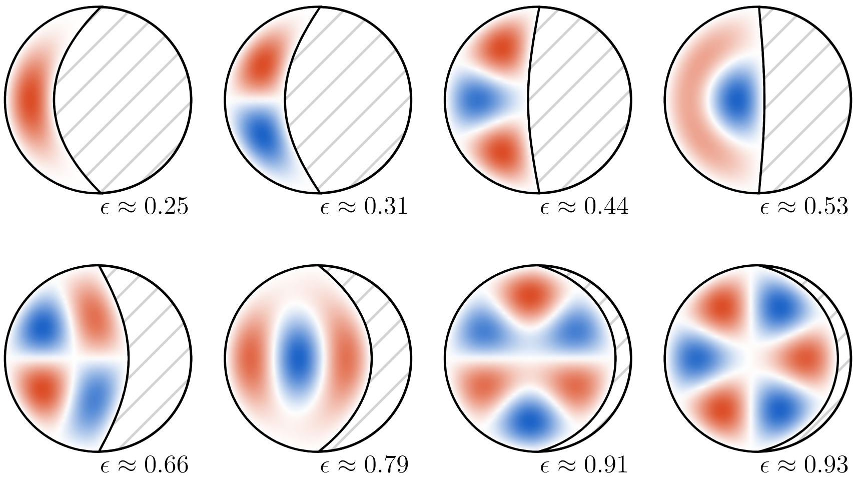


Figure 3.19: There are eight smaller moon-shaped domains on which $\lambda = 45$ is an eigenvalue of the wave equation $-\nabla^2\psi = \lambda\psi$ with homogeneous Dirichlet boundary conditions.

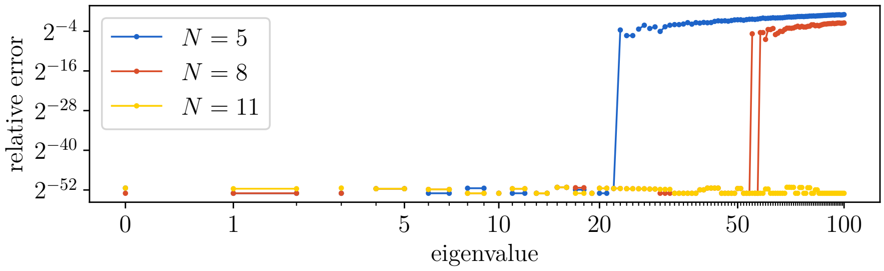


Figure 3.20: The relative error for the first hundred eigenvalues of the zero potential Schrödinger problem on the square $[0, \pi] \times [0, \pi]$ with homogeneous Dirichlet boundary conditions. Degenerate eigenvalues are connected. If a color seems missing for a value, it coincides with another drawn point.

Another large improvement can be found in the 'autonomy' of the algorithm. In [47], results for different steps sizes are tabulated. Since we implemented automatic step size selection, this distinction is no longer relevant. To demonstrate that our improvements facilitate using this method, we will also provide some `python`-code. The example code makes use of the package `Pyslise2D`. This package contains our implementation (in `C++`, see the discussion from section 2.6.1) of the earlier described algorithm, together with our improvements.

Via `pip`, the package `Pyslise2D` can be installed.

```
pip install pyslise2d
```

3.4.1 The zero potential

As a first example we will find eigenvalues of the Schrödinger equation with a zero potential

$$-\nabla\psi(x, y) = \lambda\psi(x, y)$$

on the domain $[0, \pi] \times [0, \pi]$ with homogeneous Dirichlet boundary conditions.

In section 3.0.1, we have calculated the eigenvalues to be of the form $i^2 + j^2$ for any $i, j \in \mathbb{N}^+$, and the corresponding eigenfunction is given by $\psi(x, y) = \sin(ix)\sin(jy)$. To reiterate, the first few eigenvalues are

$$2, 5, 5, 8, 10, 10, 13, 13, 17, 17, 18, 20, 20, 25, 25, 26, 26, 29, 29, 32, \ldots$$

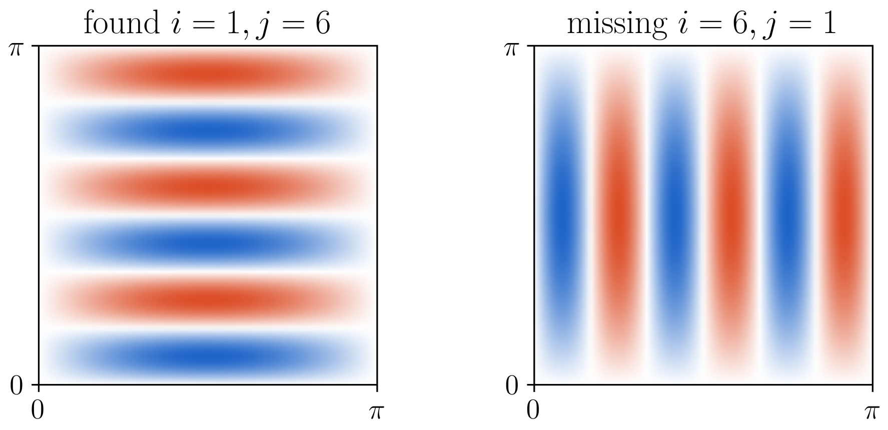


Figure 3.21: When using $N = 5$. On the left: the found 22nd eigenfunction. On the right: the missing 23th eigenfunction.

Despite this being the easiest Schrödinger problem to consider, it is non-trivial in the sense that the multiplicity of an eigenvalue is rather unpredictable.

In figure 3.20, the relative errors of the first hundred eigenvalues found with `Pyslise2D` are presented. Even for an extremely low number of basis functions used on each sector ($N = 5$), we are still able to reach the most accurate results possible for the first two dozen eigenvalues. However, once the index of the eigenvalue is sufficiently high, the relative error jumps to a value close to 1.

The reason for these sudden wrong results can be found in the fact that the Schrödinger problem with zero potential on a rectangular domain is definitely an easy problem for this method. Upon close inspection, we have found that `Pyslise2D` only uses a total of three sectors on the whole domain, independent of the requested accuracy. On each sector, exactly the same one-dimensional problem is solved, yielding exactly the same basis functions $b_i(x) = \sin(ix)$. And coincidentally, the two-dimensional eigenfunction corresponding to $E = i^2 + j^2$ can be written as $\psi(x, y) = c(y)\sin(ix)$ for a y-dependent function $c(y)$. In summary for this method, by construction, the Schrödinger problem with zero potential gets solved exactly. Yet, for $N = 5$ we see that $E_{22} = E_{23}$ is not found correctly. Notice that $E_{22} = E_{23} = 37 = 1^2 + 6^2$. When only five basis functions are considered on a sector, the necessary function $\sin(6x)$ will not be present, and therefore this eigenvalue cannot be correctly found. As an illustration, figure 3.21 visualizes this missing eigenfunction.

Let us also address a possible concern: why did our method not detect this missing eigenvalue, using theorem 3.10? Our method checks how many times there was a linear combination of the propagated functions that vanishes. If an insufficient number of basis functions are used, no such vanishing combination will be found, and the index will not be correctly determined. Here, the importance of a sufficiently large basis is apparent.

To illustrate the use of `Pyslise2D` we present some sample code to solve this Schrödinger problem.

```python
from pyslise2d import Pyslise2D
from math import pi

problem = Pyslise2D(lambda x, y: 0, 0, pi, 0, pi,
                          tolerance=1e-8, N=10)
print(problem.eigenvaluesByIndex(0, 10))
```

This code will output a list of all eigenvalues as tuples. For each eigenvalue we get its index, the eigenvalue itself and its multiplicity. This multiplicity is numerically detected, thus this may be inaccurate. An eigenvalue can be present multiple times in the list to compensate.

3.4.2 The harmonic oscillator potential

As the zero potential is quite easy for this method, next we will consider the quantum harmonic oscillator with potential $V(x, y) = x^2 + y^2$. On the infinite domain $\mathbb{R}^2$, the true eigenvalues are known to be

$$2, 4, 4, 6, 6, 6, 8, 8, 8, 8, 10, \dots, 10, 12, \dots$$

The method does not support infinite domains, thus we will assume homogeneous Dirichlet boundary conditions on $\Omega = [-9.5, 9.5] \times [-9.5, 9.5]$. This allows a comparison to section 4.2.3.1.

Our implementation is relatively autonomous, there are only two parameters to study. First we can specify a tolerance to influence the automatic sector size selection. And second, we can influence the number of basis functions N to use on each sector.

In figure 3.22, we provide a comparison of the relative error of the first few eigenvalues when varying these two parameters. The first thing we must remark

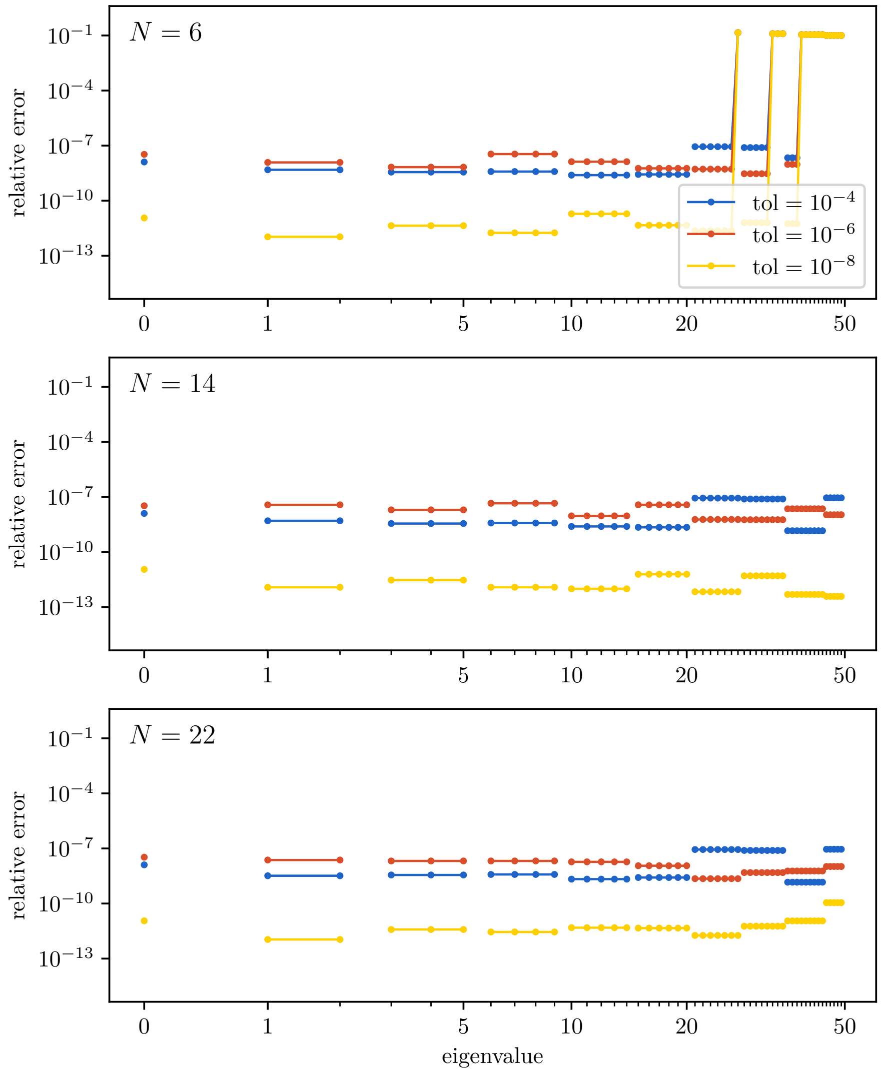


Figure 3.22: A comparison of relative error of the first fifty eigenvalues of the harmonic oscillator problem for different basis sizes N and requested tolerances tol. For $N = 22$, the algorithm with different tolerances used respectively 19, 29 and 40 sectors.

is that our implementation has degenerate eigenvalue detection. This can be seen in the equal error for each of these eigenvalues.

For a small basis size ($N = 6$), we observe the same behavior as in figure 3.20. From a certain point onward, eigenvalues will be plain wrong. For the first fifty eigenvalues, this behavior disappears once N is sufficiently large.

For $N = 14$ and $N = 22$, we are able to verify that the method respects the requested accuracy of 10^{-4}, 10^{-6} and 10^{-8} respectively. Even stronger, this accuracy is respected with a very comfortable margin. This can be attributed to a conservative error estimate in the automatic sector size detection. Similar as in `Matslise 3.0`, we have opted to err on the side of caution in this estimation. This way, we are much more confident that the requested accuracy is actually reached.

As a reference, the following code can be used to solve this problem.

```python
from pyslise2d import Pyslise2D

def V(x, y):
    return x * x + y * y

harmonic = Pyslise2D(V, -9.5, 9.5, -9.5, 9.5
                     tolerance=1e-8, N=14)
print(harmonic.eigenvaluesByIndex(0, 30))
```

3.4.3 Ixaru's potential

For the next example we follow [47] and study the problem from section 3.3.2.3. Consider the Schrödinger problem with potential

$$V(x, y) = (1 + x^2)(1 + y^2)$$

on the square domain $[-5.5, 5.5] \times [-5.5, 5.5]$ with homogeneous Dirichlet boundary conditions.

For this problem no analytical results are available. However, in [47] numerical results are reported.

By using `pyslise2d`, the code to solve this problem is quite straightforward.

```python
from pyslise2d import Pyslise2D

def V(x, y):
```

	Ixaru [47]	Our results
E_0	3.1959181	3.1959180852
$E_1 = E_2$	5.5267439	5.5267438744
E_3	7.5578033	7.5578033268
E_4	8.0312723	8.0312723403
E_5	8.4445814	8.4445813616
$E_6 = E_7$	9.9280611	9.9280610570
$E_8 = E_9$	11.3118171	11.3118170506
E_{10}	12.1032536	12.1032535787
E_{11}	12.2011790	12.2011789679
E_{12}	13.3323313	13.3323312712

Table 3.2: The first few eigenvalues of the problem with potential $V(x, y) = (1 + x^2)(1 + y^2)$ on the domain $[-5.5; 5.5] \times [-5.5; 5.5]$.

```
    return (1+x*x)*(1+y*y)

problem = Pyslise2D(V, -5.5,5.5, -5.5,5.5,
                    tolerance=1e-8)
problem.eigenvaluesByIndex(0, 12)
```

In table 3.2, our results are compared to the reference values calculated by Ixaru in [47]. There, the computation took 45 seconds. Our values were computed within 1.4 seconds on an Intel i7-8700K. Blindly comparing these run-times is ill-informed as the hardware is significantly different. Regardless, we are quite satisfied with the speed our program is able to achieve.

Upon close inspection of table 3.2, we remark that our results agree with these from [47] in all digits.

Alternative to a table of values, we plot the relative errors of the found eigenvalues for different basis sizes N and requested tolerances in figure 3.23. Here the reference values were not computed with `Pyslise2D`, rather we employed our method from the next chapter.

Upon analysis of figure 3.23, we can observe in all panels a clear trend in the blue curve with $N = 6$: if the index of the eigenvalue becomes larger, the accuracy decreases. For the red curve with $N = 12$, a similar trend can be suspected in the eigenvalues with index close to 30.

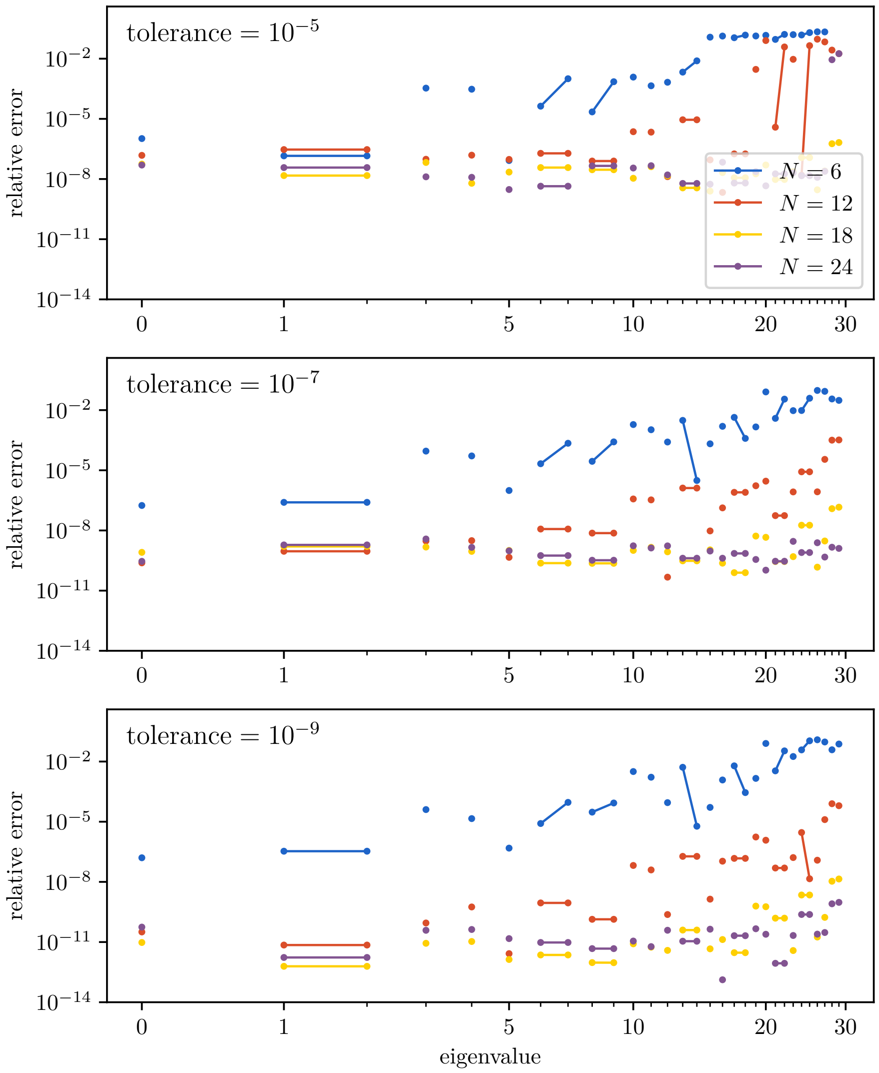


Figure 3.23: A comparison of relative error of the first thirty eigenvalues of the Schrödinger problem with Ixaru's potential from section 3.4.3 for different basis sizes N and requested tolerances.

	Braun et al. [18]	Our results
E_0	0.998 595	0.998 594 772 6
$E_1 = E_2$	1.990 077	1.990 076 760 1
E_3	2.956 243	2.956 242 989 6
$E_4 = E_5$	2.985 326	2.985 326 428 1
$E_6 = E_7$	3.925 964	3.925 963 750 1
E_8	3.982 417	3.982 417 294 5
E_9	3.985 761	3.985 760 926 5
E_{10}	4.870 144	4.870 144 338 0
$E_{11} = E_{12}$	4.898 644	4.898 644 675 5
$E_{13} = E_{14}$	4.986 251	4.986 251 029 7
E_{15}	5.817 019	5.817 019 662 6
E_{16}	5.817 027	5.817 027 592 2

Table 3.3: The first few eigenvalues of the problem with potential $V(x,y) = x^2 + y^2 + \frac{1}{6\sqrt{5}}y\left(3x^2 - y^2\right)$ on the domain $[-6;6] \times [-6;6]$. The results are reported divided by 2 to provide compatibility with [18].

3.4.4 The Hénon–Heiles potential

Another interesting example is the Hénon–Heiles potential. The corresponding Schrödinger problem is given by:

$$-\nabla^2\psi + \left(x^2 + y^2 + \frac{\sqrt{5}}{30}y\left(3x^2 - y^2\right)\right)\psi = E\psi.$$

Following [18], we impose homogeneous Dirichlet boundary conditions on the square $[-6,6] \times [-6,6]$. Our results are reported in table 3.3. The computation only took 1.3 seconds, and the results are identical, within the available precision in [18].

By now the code for solving this problem will look familiar.

```python
from pyslise2d import Pyslise2D
from math import sqrt

def V(x, y):
    return x*x + y*y + sqrt(5)/30 * y * (3*x*x - y*y)

```

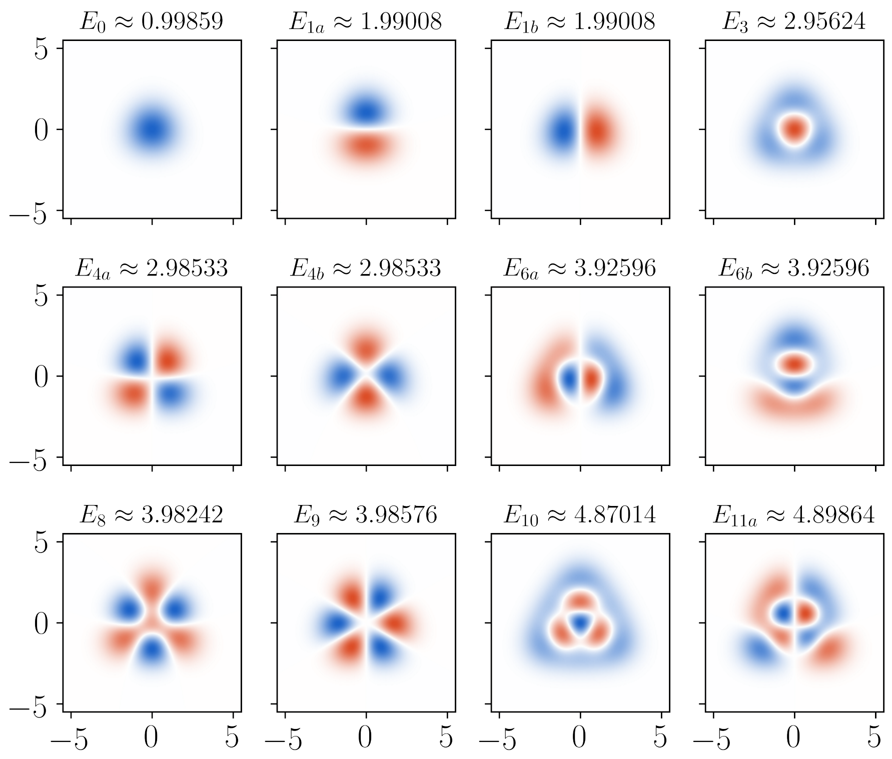


Figure 3.24: The first twelve eigenfunctions of the Schrödinger problem with the Hénon–Heiles potential on $[-6, 6] \times [-6, 6]$.

```
problem = Pyslise2D(V, -6,6, -6,6, tolerance=1e-8)
problem.eigenvaluesByIndex(0, 16)
```

With our program, also the eigenfunctions can be computed. This is demonstrated in figure 3.24.

Here we have opted to not include similar figures to visualize the relative errors as in the previous experiments. These graphs do not let us gain any more new insights.

3.5 Conclusions

In this chapter, we have studied the method introduced by Ixaru in [47] for two-dimensional time-independent Schrödinger equations. The original work presented an intriguing innovative method accompanied by a promising numerical example. Section 3.1 presents an overview of this method as first described. In section 3.2, we proudly presented our additions and improvements to this method. In particular, we have devised a way to reliably determine the index of an eigenvalue. For this, we developed some new theory in section 3.3.

Section 3.4 was dedicated to some numerical experiments. In this thesis, in contrast to an article, we have the luxury to take the time for a much deeper dive into some numerical experiments. These experiments have been promising and were able to reach the requested accuracy. This verifies that the method and our improvements are definitely valuable in solving time-independent two-dimensional Schrödinger equations.

However, this is not the *perfect* method. The accuracy of higher eigenvalues is limited by the considered basis size N. Since we are solving many coupled systems of Schrödinger equations, due to numerical instability choosing N extremely large becomes infeasible. Nonetheless, this method has proven itself to be capable, and worth investigating. During the process of our investigation, we were able to undoubtedly improve this method and discovered some new, more widely applicable theoretical results.

Implementing this method is (and has been) difficult. There are many interconnected parts, with each their own numerical nuances. In the next chapter we will start with the analysis of a relatively simple technique based upon a finite difference scheme. We will see that this technique is much easier to implement, is able to reach extremely accurate results, and all this without significantly more computation time.

Chapter 4

Strands: a new method for 2D time-independent Schrödinger equations

In chapter 2, we studied a constant perturbation method for one-dimensional Sturm–Liouville problems. We have seen a brief history about these CP-methods, as well as a thorough overview about how these methods can be implemented. The numerical examples illustrated the benefits and accuracy of the studied techniques.

Chapter 3 was dedicated to the treatment of a recent method to solve time-independent two-dimensional Schrödinger equations. This method aims to use the strengths of the constant perturbation methods for higher dimensional problems. This new method is promising, and we developed many improvements upon the original idea.

One of the unique powers of the CP-methods is their ability to not only compute low eigenvalues accurately, but even to increase accuracy for higher eigenvalues. For Sturm–Liouville problems there are methods that can accurately compute low eigenvalues and there are other methods that are accurate for higher eigenvalues (e.g. asymptotic methods [59, 104]). A CP-method, however, is one of the few techniques that is accurate for low as well as high eigenvalues.

For two dimensions, this property did not translate cleanly. The method described in chapter 3 tries to capture this by considering solutions of a one-dimensional Schrödinger problem in the x-direction, while a method for a coupled system of Schrödinger equations has been used in the y-direction. In theory, this method is capable of computing any eigenvalue. In practice, this is not the case. A two-dimensional eigenfunction is represented as a linear combination $\sum_i c_i(y)\beta_i(x)$ of one-dimensional basis functions $\beta_i(x)$. This basis is well-chosen, such that even a few functions can already describe a two-dimensional eigenfunction sufficiently well. When implementing this, the basis has to be finite. As eigenfunctions corresponding to higher eigenvalues will become more and more oscillatory, the chosen finite basis will no longer be able to express all necessary details.

That the basis is finite thus limits the accuracy for higher eigenvalues, which negates one of the strongest benefits of the employed CP-methods. As such, we believe that the perfect method for the higher-dimensional time-independent Schrödinger equation is one which does not decrease in accuracy as higher eigenvalues are requested, just like the CP-methods are for the one-dimensional case. Developing such a method will, most likely, require new and very complicated formulae.

For clarity, we did not develop such a perfect method. But in the last few years I have played with the idea...

More realistically, during our research into the method of Ixaru, we have found other work, also focussing upon the time-independent Schrödinger problem. These ideas and methods have inspired us to develop our own technique. The new methods we propose try to fix or mitigate some issues present in the other methods.

4.1 Inspiration

After careful implementation and thorough testing of Ixaru's method from chapter 3, I browsed through the literature with renewed appreciation for the Schrödinger problem. When investigating that method, many obstacles and challenges arose. It was an interesting task to balance computation time, symbolic formulae and numerical accuracy. So with this in mind, I came across [102] by Wang and Shao.

In this article, the authors proposed a new kind of discretization scheme for solving two-dimensional time-independent Schrödinger equations. Before studying the details of a new method, I like to review its numerical results.

In [102], the algorithm was tested on two potential functions. First, the harmonic oscillator was considered. Schrödinger problems with this potential function are not extremely difficult to solve, but they capture some of the challenges that numerical algorithms may face, while still having symbolic solutions for the eigenvalues. Second, they provided results for the Hénon–Heiles potential. Here, the exact solutions are not known, but reliable approximations exist. For both potentials, the results were quite impressive. They reached a high accuracy for a significant number of eigenvalues without using excessive computation time.

But the thing that strikes me most about [102] is the simplicity of the method. They constructed formulae for a discrete approximation on a grid. Finding these kinds of results within the literature is disheartening and inspirational at the same time. At first glance, they seem to be able to reach higher accuracy with a simpler method than Ixaru's work [47] and our improvements [8] to it. Before throwing away all our work, and declaring [102] to be superior, let us analyze it ourselves.

4.1.1 A finite difference scheme

As in chapter 3, we are considering the Schrödinger equation

$$-\psi''(x,y)+V(x,y)\psi(x,y)=E\psi(x,y) \tag{4.1}$$

on a rectangular domain $\Omega=[x_{\min},x_{\max}]\times[y_{\min},y_{\max}]$ with homogeneous Dirichlet boundary conditions, so $\psi(x,y)=0,\forall(x,y)\in\partial\Omega$. The potential $V(x,y)$ is known, and eigenvalues E with corresponding eigenfunctions $\psi(x,y)$ which satisfy (4.1) are sought.

In [102], the authors discretize the domain Ω with an equidistant $n_x\times n_y$ grid. With this, they approximate the second partial derivative of a function $\psi(x,y)$ with, what they call, a new scheme:

$$\frac{\partial^2}{\partial x^2}\psi(x,y)=-\frac{1}{h^2}\sum_{i=-N_x}^{N_x}c_i\psi(x+ih,y).$$

In this expression, h is the x-step of the equidistant grid. An analogous formula for the second partial derivative of $\psi(x,y)$ with respect to y is proposed, with y-step η:

$$\frac{\partial^2}{\partial y^2}\psi(x,y)=-\frac{1}{\eta^2}\sum_{i=-N_y}^{N_y}c_i\psi(x,y+i\eta).$$

To determine the value of c_i, the authors propose to assume $N = N_x = N_y$ and use the Taylor series expansion of the left-hand side of

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\psi(x,y) = -\frac{1}{h^2}\sum_{i=-N}^{N} c_i \left(\psi(x+ih,y) + \left(\frac{h}{\eta}\right)^2 \psi(x,y+i\eta)\right).$$

In [102], the coefficients c_i for $N_q = 3, 4, 5, 6$ are provided. For points close to the boundary of the domain, the authors propose to assume that the unknown eigenfunction $\psi(x,y)$ will always be identically zero outside the domain Ω. With this assumption, the same formulae can be used for every point in the discretization, even those close to the boundary of the domain.

Using these formulae, the problem now reduces to a simple, albeit very large, square matrix eigenvalue problem:

$$\left(-\mathbf{T}_{n_x} \otimes \mathbf{I}_{n_y} - \mathbf{I}_{n_x} \otimes \mathbf{T}_{n_y} + \operatorname{diag}\begin{pmatrix} V(x_1,y_1) \\ V(x_1,y_2) \\ \vdots \end{pmatrix}\right)\mathbf{u} = \lambda\mathbf{u}. \tag{4.2}$$

Here, $\mathbf{T}_{n_x}$ is the symmetric Toeplitz[1] $n_x \times n_x$ matrix with the first row $\begin{pmatrix} c_0 & c_1 & \cdots & c_{N_x} & 0 & \cdots \end{pmatrix}$, analogous for $\mathbf{T}_{n_y}$. The $n \times n$ identity matrix is denoted as $\mathbf{I}_n$. The operation $\cdot \otimes \cdot$ is the Kronecker product. And finally, the vector of unknowns $\mathbf{u}$ is the approximation of ψ in each of the grid points: $\mathbf{u} \approx \begin{pmatrix} \psi(x_1,y_1) & \psi(x_1,y_2) & \cdots \end{pmatrix}^\intercal$. The matrix on the left-hand side of (4.2) can be directly computed. It is valuable to note that most of the entries of this matrix will be zero, so considering it as a sparse matrix is preferred. Many classical, well-tested solvers exist for sparse eigenvalue problems. The authors of [102] have used algorithms provided by `mathematica` [41].

Before reviewing, and recalculating their numerical results, I want to take the time to thoroughly analyze their formulae. Using an equidistant grid and coefficients which optimize for the Taylor series of the function in question, is oddly reminiscent of finite difference approximations. To confirm this similarity

[1]A matrix is called Toeplitz if each diagonal only contains a constant value. In other words, a matrix $\mathbf{A}$ is Toeplitz if it can be written as follows, and it is symmetric if $a_{-i} = a_i$.

$$\begin{pmatrix} a_0 & a_1 & a_2 & \cdots \\ a_{-1} & a_0 & a_1 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix}$$

we have calculated the central finite difference approximations of the second order derivative.

The following table contains the first few central symmetric finite differences.

Order	c_0	$c_{-1} = c_1$	$c_{-2} = c_2$	$c_{-3} = c_3$	$c_{-4} = c_4$	$c_{-5} = c_5$
2	-2	1				
4	$-\frac{5}{2}$	$\frac{4}{3}$	$-\frac{1}{12}$			
6	$-\frac{49}{18}$	$\frac{3}{2}$	$-\frac{3}{20}$	$\frac{1}{90}$		
8	$-\frac{205}{72}$	$\frac{8}{5}$	$-\frac{1}{5}$	$\frac{8}{315}$	$-\frac{1}{560}$	
10	$-\frac{5269}{1800}$	$\frac{5}{3}$	$-\frac{5}{21}$	$\frac{5}{126}$	$-\frac{5}{1008}$	$\frac{1}{3150}$

These give exactly the same coefficients as in [102]. It is quite disingenuous to call this a *new* scheme. The study of finite differences can be traced back to, among others, Newton. One of the first English books on this topic was written by Boole [16] in 1860, with many more textbooks to follow [94, 54]. Even the symbolic construction of high-order approximations has already been studied and tabulated as early as 1967 [11, 56, 31]. In the context of one-dimensional Schrödinger equations, our research group already studied techniques based upon finite difference approximations some 40 years ago [29, 30].

Besides this remark, [102] is very valuable as an application of very high order versions of these well-known formulae to the two-dimensional time-independent Schrödinger equation. Their numerical results are still valid and nonetheless impressive.

4.1.1.1 Numerical experiments

The harmonic oscillator
As a first example, we will replicate the results of [102] for the harmonic oscillator:

$$-\nabla^2\psi(x,y) + \left(x^2+y^2\right)\psi(x,y) = E\psi(x,y) \tag{4.3}$$

on the domain $[-9.5, 9.5] \times [-9.5, 9.5]$ with homogeneous Dirichlet boundary conditions. Notice that in [102], the authors have chosen to scale the potential and eigenvalue by an extra factor of two. We did not do this in our experiment, as such our computed eigenvalues will be twice as large. However, the relative errors remain the same.

In figure 4.1, the relative errors for the first 100 eigenvalues are plotted. This is done for $h = \eta \in \left\{\frac{19}{40}, \frac{19}{60}, \frac{19}{100}\right\}$ and $N_x = N_y \in \{5, 10, 20, 25\}$. In these

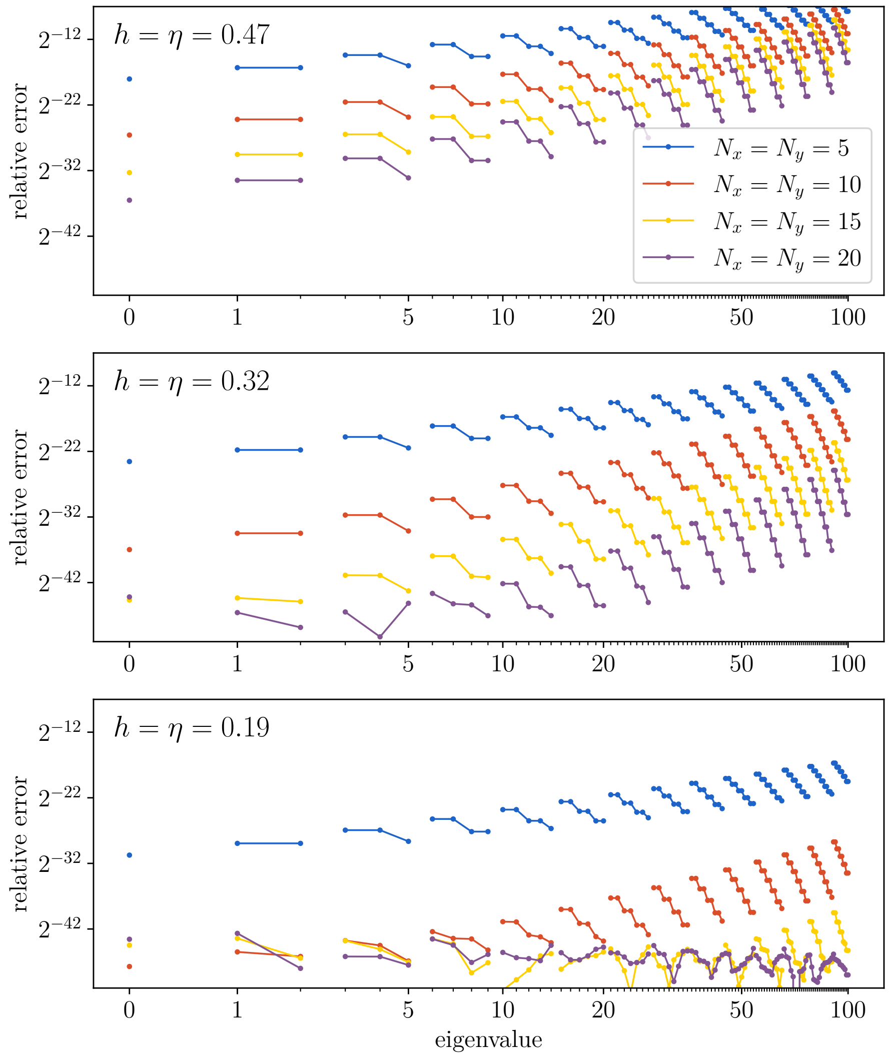


Figure 4.1: These graphs display the relative error of each of the first 100 eigenvalues of the harmonic oscillator (4.3), computed for different values of $h = \eta$ and $N_x = N_y$. Repeated eigenvalues are connected.

graphs, a few things can be noted. Unsurprisingly, when $N_x = N_y$ is increased, the results become more accurate. When $h = \eta$ is decreased, that is to say, more grid points are used, the results also become more accurate. When larger eigenvalues are computed, the accuracy decreases, as is the case with most algorithms for approximating Schrödinger equations. The corresponding eigenfunctions become more and more oscillatory, and are thus more difficult to approximate accurately.

But most impressively, all these graphs were computed within a minute on a simple laptop. More detailed runtime analysis can be found in section 4.2.4. Furthermore, the obtained results by themselves are also impressive. In the most accurate case $h = \eta = \frac{19}{100}$ and $N_x = N_y = 20$, the first 100 eigenvalues are found with 45 bits of precision, which is close to the ideal machine precision of 53 bits.

To get a deeper understanding of this finite difference method, it may be beneficial to study the asymptotic behavior of the error, with respect to the number of discretization points and the index of the eigenvalue. The finite difference approximation error can be used as a jumping point. Assume $N = N_x = N_y$, then the finite difference approximation of the second derivative of a function $f : \mathbb{R} \to \mathbb{R}$ is of the form:

$$f''(x) = \frac{1}{h^2} \sum_{i=-N}^{N} c_i f(x + ih) + \mathcal{O}\big(h^{2N}\big).$$

Going further with a theoretical analysis becomes unnecessarily difficult. We will estimate the order by using the numerical results for the first 100 eigenvalues of the harmonic oscillator, calculated for the values of $h = \eta = \frac{1}{n}$ for $n \in \{20, 30, 40, 50, 60, 70, 80, 90, 100\}$. These nine different sets of parameters yield for each of the 100 eigenvalues (with index $k \in \{1, 2, \dots, 100\}$) a data point. Next, through all these points, we find the best fitting curve which follows the expression $\alpha_1 h^{\alpha_2} k^{\alpha_3}$. In this case, we define the best fitting curve to be such that for the error $\epsilon_k^{(h)}$ (for each eigenvalue k calculated with a step size of $h = \eta$), the following is minimized:

$$\sum_{k,h} \left| \log |\epsilon_k^{(h)}| - \log(c_1 h^{c_2} k^{c_3}) \right|^2 .$$

These best fits, with $N_x = N_y = N$ are summarized in the following table.

N	5	10	15	20
Error	$\mathcal{O}\big(h^{7.59}k^{2.10}\big)$	$\mathcal{O}\big(h^{14.00}k^{3.38}\big)$	$\mathcal{O}\big(h^{19.50}k^{3.91}\big)$	$\mathcal{O}\big(h^{21.96}k^{4.23}\big)$

It is important to note that these values are estimates, and are quite sensitive to the used values of n and k. But some trends are visible. We first note that increasing N indeed increases the order in h as well, maybe not as much as theoretical assumed ($\mathcal{O}\big(h^{2N}\big)$), but nonetheless significantly. One of the reasons this theoretical value of h^{2N} is not reached is because of the limited precision available in the `double` floating point type. For low eigenvalues, dense grids, and higher values of N, the error is so small that the numerical precision of the datatype becomes the main source of error. Another visible behavior is that a more accurate (higher order in h) method also has a larger exponent with respect to k. This means that higher eigenvalues are asymptotically less accurate. This property can also be seen in the graphs of figure 4.1. A virtual line through the points corresponding to $N = 5$ is less steep than a line through the points of $N = 10$.

For one-dimensional problems the behavior of the error with respect to k has already been well studied and improved. In [76], expressions are constructed which provide a correction to the numerical approximation with a finite difference scheme for Sturm–Liouville problems. These correction formulae bring the order in k down from $\mathcal{O}(k^4h^2)$ to $\mathcal{O}(kh^2)$. For multidimensional Schrödinger equations, no such corrections are available, let alone for the extremely high order schemes considered here.

Zero potential

- Besides the harmonic oscillator, it is also instructive to consider the problem with a zero potential function on the domain $\Omega = [0,\pi]\times[0,\pi]$ and homogeneous Dirichlet boundary conditions. So, find all eigenvalues E for which there exists an eigenfunction $\psi(x,y)$ on the domain with $\psi(x,y) = 0$ on $\partial\Omega$ such that the following holds:

$$-\nabla^2\psi(x,y) = E\psi(x,y). \tag{4.4}$$

By separation of variables, the exact solutions of this equation can be found: $\psi_{i,j}(x,y) = \sin(ix)\sin(jy)$ with eigenvalue $\lambda_{i,j} = i^2 + j^2$ for all $i,j \in \mathbb{N}^+$. The

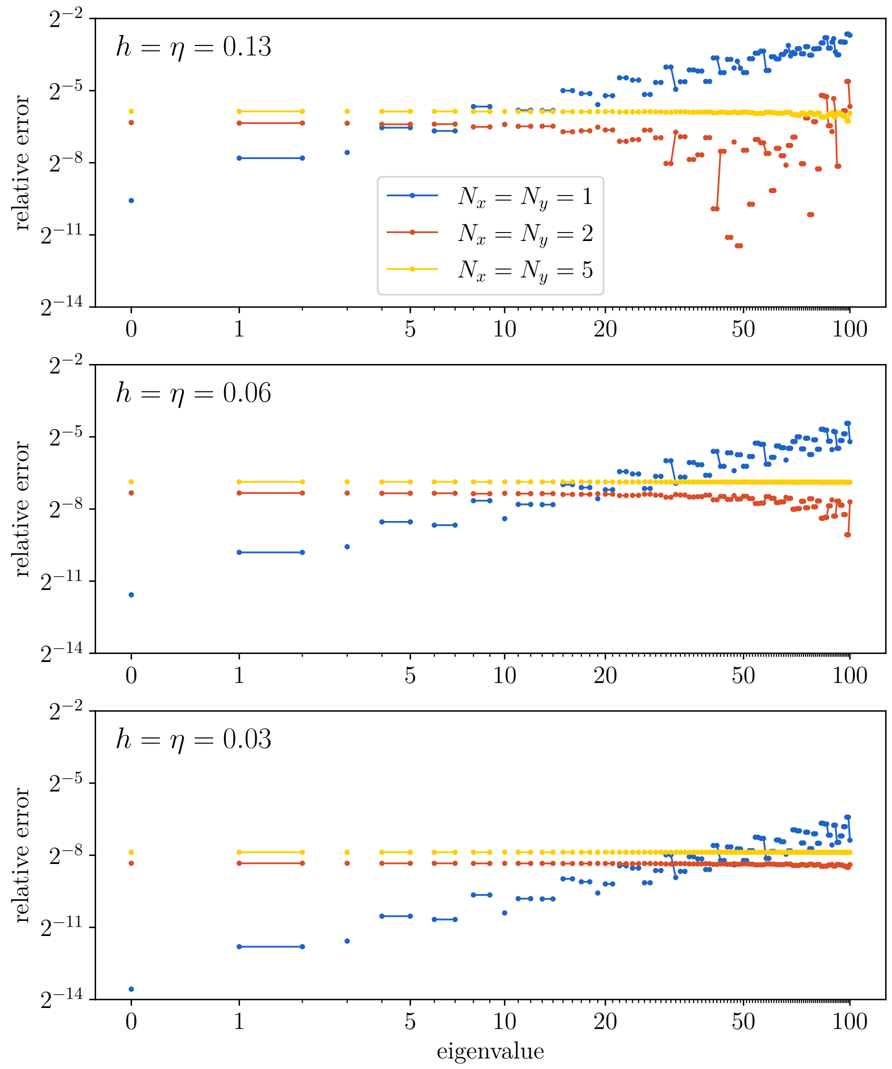


Figure 4.2: These graphs display the relative error of each of the first 100 eigenvalues of a Schrödinger problem with a zero potential on the rectangular domain $[0, \pi] \times [0, \pi]$ with homogeneous Dirichlet boundary conditions (4.4). This error is computed for different values of $h = \eta$ and $N_x = N_y$. Repeated eigenvalues are connected.

first few of these eigenvalues are listed.

$$\begin{aligned}
\lambda_{1,1} &= 2 & \lambda_{3,3} &= 18 & \lambda_{3,5} = \lambda_{5,3} &= 34\\
\lambda_{1,2} = \lambda_{2,1} &= 5 & \lambda_{2,4} = \lambda_{4,2} &= 20 & \lambda_{1,6} = \lambda_{6,1} &= 37\\
\lambda_{2,2} &= 8 & \lambda_{3,4} = \lambda_{4,3} &= 25 & \lambda_{2,6} = \lambda_{6,2} &= 40\\
\lambda_{1,3} = \lambda_{3,1} &= 10 & \lambda_{1,5} = \lambda_{5,1} &= 26 & \lambda_{4,5} = \lambda_{5,4} &= 41\\
\lambda_{2,3} = \lambda_{3,2} &= 13 & \lambda_{2,5} = \lambda_{5,2} &= 29 & \lambda_{3,6} = \lambda_{6,3} &= 45\\
\lambda_{1,4} = \lambda_{4,1} &= 17 & \lambda_{4,4} &= 32 & \lambda_{1,7} = \lambda_{5,5} = \lambda_{7,1} &= 50
\end{aligned}$$

In a certain way, these eigenvalues are more interesting than those from the harmonic oscillator, as their multiplicities are less predictable.

In figure 4.2, the relative error in each of the first 100 eigenvalues is displayed for different values of $h = \eta$ and $N_x = N_y$. When comparing this figure to the results of the harmonic oscillator 4.1, some striking differences can be seen. Maybe most surprisingly, increasing the order does not give more accurate results! For low eigenvalues, the method with $N_x = N_y = 1$ is most accurate, this is simply the second order central finite difference formula. When using higher order methods the error does not get any better than $2^{-8} \approx 0.00391$, which is not at all as accurate as the results for the harmonic oscillator, where the error for $N_x = N_y = 5$ lies between $2^{-32} \approx 2.3 \cdot 10^{-10}$ and at most $2^{-16} \approx 1.5 \cdot 10^{-5}$. This is quite disconcerting behavior for a numerical method to have.

These results bring to light one of the biggest drawbacks of the method from [102]. This method only works if it is assumed that the eigenfunctions always are zero outside the domain. For the harmonic oscillator, this is the case, as $V(x, y) \to +\infty$ if $\|x, y\| \to +\infty$. For the zero potential function, this is definitely not the case as the eigenfunctions are periodic. One could argue that, in theory, an eigenfunction may be anything outside the domain Ω. So in particular, we could define them to be zero there. But, most numerical methods, and this method in particular, assume that solutions are sufficiently smooth. To cleanly define the problem, at least $C_0^2(\Omega)$ is needed. When using N^{th} order central finite difference schemes, $C_0^N(\Omega)$ is implicitly assumed. And this assumption fails if we define the function to be zero outside Ω.

To combat this issue, on the boundary of the domain asymmetric schemes could be used, the authors of [102] did not do this. As an example, we have tabulated all relevant seven-point formulae.

c_{-3}	c_{-2}	c_{-1}	c_0	c_1	c_2	c_3	c_4	c_5	error
$\frac{1}{90}$	$-\frac{3}{20}$	$\frac{3}{2}$	$-\frac{49}{18}$	$\frac{3}{2}$	$-\frac{3}{20}$	$\frac{1}{90}$			$\mathcal{O}(h^6)$
	$-\frac{13}{180}$	$\frac{19}{15}$	$-\frac{7}{3}$	$\frac{10}{9}$	$\frac{1}{12}$	$-\frac{1}{15}$	$\frac{1}{90}$		$\mathcal{O}(h^5)$
		$\frac{137}{180}$	$-\frac{49}{60}$	$-\frac{17}{12}$	$\frac{47}{18}$	$-\frac{19}{12}$	$\frac{31}{60}$	$-\frac{13}{180}$	$\mathcal{O}(h^5)$

There are two difficulties when using these formulae. First, to reach the required order, one extra point should be used. And second, when using asymmetric formulae, the matrix from (4.2) is no longer symmetric. In principle this should not matter. In practice however, many sparse matrix eigenvalue solvers are much more efficient when working with a real symmetric matrix.

4.1.1.2 Computational cost of the method

Finite differences are employed in many real-world applications, as they are easy to implement, yet are able to reach high accuracies. In practice, it is rare to encounter these very high orders. However, in this setting these high orders seem to be very well applicable.

Yet, these finite difference methods are not without issues. One of the most prominent drawbacks is the expensive computations involved. To get accurate results, high orders combined with fine grids have to be used. This results in huge sparse matrices. Only due to this sparsity are algorithms able to compute the eigenvalues. Now, for higher order schemes, the matrices become less sparse, and as such, more expensive to work with.

Alternatively, a method that is able to reduce the size of the matrices involved without losing accuracy may be preferable.

4.2 Strands: A line-based collocation method

In contrast to the previous methods, we propose a technique which is not necessarily restricted to cases where the domain is rectangular. So consider a finite domain $\Omega \subseteq \mathbb{R}^2$ on which we are searching for eigenvalues $E \in \mathbb{R}$ and eigenfunctions $\psi : \Omega \to \mathbb{R}$ such that for a given potential $V : \Omega \to \mathbb{R}$ the following holds

$$-\nabla^2\psi + V(x, y)\psi = E\psi. \tag{4.5}$$

Still, we impose homogeneous Dirichlet boundary conditions: $\psi(x, y) = 0$ for $(x, y) \in \partial\Omega$.

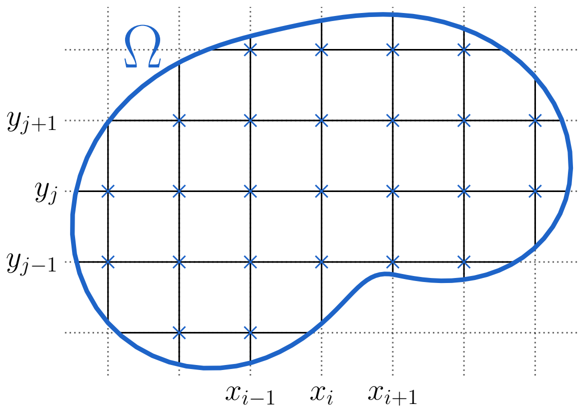


Figure 4.3: The grid used by Strands in section 4.2.

The main idea underlying this method is that we want to represent the eigenfunctions ψ efficiently. A fully discretized method may represent eigenfunctions as its values on certain grid points. Our first attempt to develop a more continuous approximation of the eigenfunction used ideas from chapter 3 and led to the ideas for a new method, where we approximated the eigenfunction as a linear combination of well-chosen basis functions on parallel lines throughout the domain. This led to a continuous approximation along one direction and a discrete approximation along the other direction of the domain.

In this section, we are going to construct a new method in which we will bring this continuous approximation to both directions of the domain. For this, we place a grid over the domain Ω, as can be seen in figure 4.3. This grid does not need to be equidistant. When developing this new method, we strived to allow for maximal flexibility, by avoiding as many restrictions on Ω as possible. This has as consequence that, since Ω does not need to be a rectangle, the number of intersections per grid line is not necessarily constant. Even worse, because we do not even require Ω to be convex, the intersection of a grid line with Ω may not be connected. Upon formulating and implementing this new method, this varying number of intersections and these unconnected grid lines are some examples of difficulties that arise by explicitly allowing more flexibility.

After placing the grid on Ω, the next step is to approximate the unknown eigenfunction ψ as a linear combination of basis functions on each of the grid lines. On the vertical line $x = x_i$, we denote these basis functions as $\beta_k^{(x_i)}(y)$, for each value of k. This yields the expression

$$\psi(x_i, y) = \sum_{k=0}^{\infty} c_k^{(x_i)} \beta_k^{(x_i)}(y). \tag{4.6}$$

For horizontal lines $y = y_j$, the basis functions $\beta_k^{(y_j)}(x)$ yield a similar expression

$$\psi(x, y_j) = \sum_{k=0}^{\infty} c_k^{(y_j)} \beta_k^{(y_j)}(x). \tag{4.7}$$

Notice that the domain of each of these β-functions is the intersection of its corresponding grid line with Ω. As stated earlier, this domain may not be connected. In theory this is not a problem, as long as the definition of β allows this. Indeed, it will allow this. In practice, when one wants to implement this, care has to be taken. It may be beneficial to construct multiple separate sets of basis functions for each of the connected parts of the grid line. Because the theory has no problem with non-connectedness, we will not burden the notation and explanation with this separation into connected parts.

To ensure $\psi(x, y)$ is uniquely defined in each point in Ω, we have to require that in each intersection point (x_i, y_j), $\psi(x_i, y_j)$ has only one solution:

$$\psi(x_i, y_j) = \sum_{k=0}^{\infty} c_k^{(x_i)} \beta_k^{(x_i)}(y_j) = \sum_{k=0}^{\infty} c_k^{(y_j)} \beta_k^{(y_j)}(x_i). \tag{4.8}$$

Before deciding on which basis functions to use, let us consider the Schrödinger equation (4.5) on each intersection point (x_i, y_j) with this new representation of ψ:

$$-\sum_{k=0}^{\infty} c_k^{(x_i)} \beta_k^{\prime\prime(x_i)}(y_j) - \sum_{k=0}^{\infty} c_k^{(y_j)} \beta_k^{\prime\prime(y_j)}(x_i) + (V(x_i, y_j) - E)\psi(x_i, y_j) = 0.$$

This last formula suggests choosing $\beta_k^{(x_i)}$ and $\beta_k^{(y_j)}$ such that their second derivative contains, in a certain sense, $V(x_i, y_j)$. Together with the idea from chapter 3 to use a one-dimensional Schrödinger equation, this leads us to propose $\beta_k^{(x_i)}$ to be the ordered eigenfunctions which satisfy the one-dimensional Schrödinger equation

$$-\beta_k^{\prime\prime(x_i)}(y) + \frac{V(x_i, y)}{2} \beta_k^{(x_i)}(y) = \lambda_k^{(x_i)} \beta_k^{(x_i)}(y)$$

with homogeneous Dirichlet boundary conditions. The domain for this one-dimensional problem is the intersection of the vertical line $x = x_i$ and the two-dimensional domain Ω. Similarly, for the horizontal line $y = y_j$, we propose $\beta_k^{(y_j)}(x)$ to be the eigenfunctions of

$$-\beta_k''^{(y_j)}(x) + \frac{V(x, y_j)}{2}\beta_k^{(y_j)}(x) = \lambda_k^{(y_j)}\beta_k^{(y_j)}(x)$$

with homogeneous Dirichlet boundary conditions, and as domain the intersection of $y = y_j$ and Ω.

By choosing half the original potential in each of the approximations, in each intersection (x_i, y_j), equation (4.5) simplifies, and $V(x, y)$ disappears:

$$\sum_{k=0}^{\infty} \lambda_k^{(x_i)} c_k^{(x_i)} \beta_k^{(x_i)}(y_j) + \sum_{k=0}^{\infty} \lambda_k^{(y_j)} c_k^{(y_j)} \beta_k^{(y_j)}(x_i) = E\psi(x_i, y_j). \tag{4.9}$$

Notice that in this expression, only E (the eigenvalue), $c_k^{(x_i)}$ and $c_k^{(y_j)}$ are unknown, as $\psi(x_i, y_j)$ depends linearly on $c_k^{(x_i)}$ and $c_k^{(y_j)}$. So, expression (4.9) is, in fact, a linear problem.

Before writing this as a matrix-problem, we first have to limit the range of the sum. It is, of course, impossible to implement these formulae while the function bases are still infinite. Therefore, we limit the basis on the line $x = x_i$ to the first K_{x_i} functions and to the first K_{y_j} functions on the line $y = y_j$. Notice that the size of a basis on each line should not be greater than the number of intersections on that line. If the basis size is too large, the system (4.8) will be underdetermined. In this case, solutions no longer are uniquely defined, which is a problem. In contrast, if the basis size is smaller, the system (4.8) is overdetermined. This does not lead to issues, because we can reformulate it as finding solutions in a least squares sense.

With these finite sums, equations (4.8) and (4.9) become, on the intersection (x_i, y_j):

$$\begin{aligned}\sum_{k=0}^{K_{x_i}-1} \lambda_k^{(x_i)} c_k^{(x_i)} \beta_k^{(x_i)}(y_j) &+ \sum_{k=0}^{K_{y_j}-1} \lambda_k^{(y_j)} c_k^{(y_j)} \beta_k^{(y_j)}(x_i) \\ &= E \sum_{k=0}^{K_{x_i}-1} c_k^{(x_i)} \beta_k^{(x_i)}(y_j) = E \sum_{k=0}^{K_{y_j}-1} c_k^{(y_j)} \beta_k^{(y_j)}(x_i).\end{aligned} \tag{4.10}$$

The introduction of appropriate vectors and matrices will allow us to translate (4.10) into a matrix problem. The unknowns will be summarized into the two vectors $\mathbf{c_x}$ and $\mathbf{c_y}$, with sizes $n_x := \sum_i K_{x_i}$ and $n_y := \sum_j K_{y_j}$ respectively:

$$\begin{aligned} \mathbf{c_x} &= \begin{pmatrix} c_0^{(x_0)} & c_1^{(x_0)} & \cdots & c_{K_{x_0}-1}^{(x_0)} & c_0^{(x_1)} & c_1^{(x_1)} & \cdots \end{pmatrix}^\mathsf{T} \\ \text{and } \mathbf{c_y} &= \begin{pmatrix} c_0^{(y_0)} & c_1^{(y_0)} & \cdots & c_{K_{y_0}-1}^{(y_0)} & c_0^{(y_1)} & c_1^{(y_1)} & \cdots \end{pmatrix}^\mathsf{T}. \end{aligned}$$

Furthermore, we introduce the $n_x \times n_x$ diagonal matrix $\mathbf{\Lambda_x}$ and the $n_y \times n_y$ diagonal matrix $\mathbf{\Lambda_y}$ which contain the eigenvalues of the one-dimensional Schrödinger problems used to define the basis functions:

$$\begin{aligned} \mathbf{\Lambda_x} &= \operatorname{diag}\left(\lambda_0^{(x_0)}, \lambda_1^{(x_0)}, \cdots, \lambda_{K_{x_0}-1}^{(x_0)}, \lambda_0^{(x_1)}, \lambda_1^{(x_1)}, \cdots\right) \\ \text{and } \mathbf{\Lambda_y} &= \operatorname{diag}\left(\lambda_0^{(y_0)}, \lambda_1^{(y_0)}, \cdots, \lambda_{K_{y_0}-1}^{(y_0)}, \lambda_0^{(y_1)}, \lambda_1^{(y_1)}, \cdots\right). \end{aligned}$$

Last, we define the matrices that contain the values of the basis functions in each of the grid points. For this, let us define m as the total number of intersection points. Now, we define the $m \times n_x$ matrix $\mathbf{B_x}$ and the $m \times n_y$ matrix $\mathbf{B_y}$. Each row $r_{i,j}$ of these matrices corresponds to a grid point (x_i, y_j). The non-zero entries on each of the rows $r_{i,j}$ of the matrices $\mathbf{B_x}$ and $\mathbf{B_y}$ can be calculated as

$$\begin{aligned} (\mathbf{B_x})_{r_{i,j}, o_i^{(x)}+k} &= \beta_k^{(x_i)}(y_j) \text{ for } k \in \{0, 1, \ldots, K_{x_i} - 1\} \\ \text{and } (\mathbf{B_y})_{r_{i,j}, o_j^{(y)}+k} &= \beta_k^{(y_j)}(x_i) \text{ for } k \in \{0, 1, \ldots, K_{y_i} - 1\}. \end{aligned} \tag{4.11}$$

In this expression, the values $o_i^{(x)}$ and $o_j^{(y)}$ are offsets depending on the row, defined as: $o_i^{(x)} := \sum_{i'<i} K_{x_{i'}}$ and $o_j^{(y)} := \sum_{j'<j} K_{y_{j'}}$. To aid an intuitive understanding of the $\mathbf{B_x}$ and $\mathbf{B_y}$ matrices, figure 4.4 provides a schematic view of them, calculated from a small numerical example.

As promised, these definitions allow us to rewrite the system (4.10) with m equations more compactly as:

$$\mathbf{B_x \Lambda_x c_x} + \mathbf{B_y \Lambda_y c_y} = E\mathbf{B_x c_x} = E\mathbf{B_y c_y}. \tag{4.12}$$

This formulation is not yet reminiscent of any classical linear algebra problem. One of the unfamiliar parts of this expression is the fact that there are two

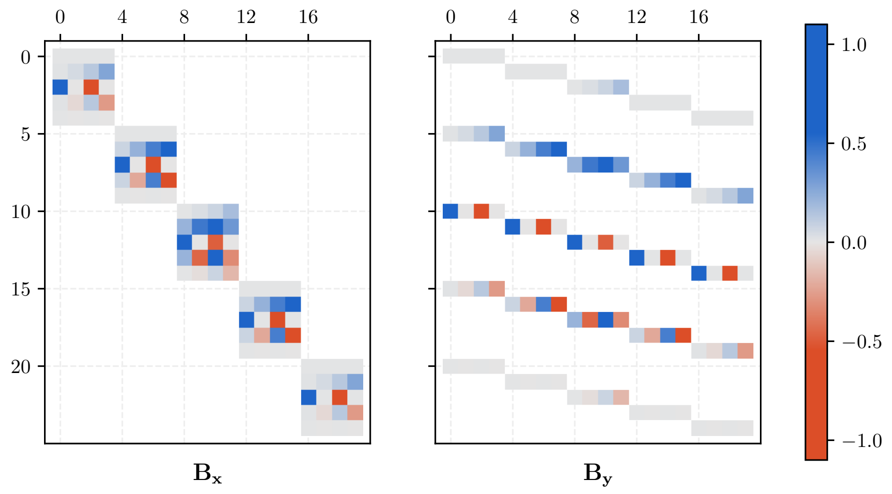


Figure 4.4: The non-zero entries of the matrices $\mathbf{B_x}$ and $\mathbf{B_y}$ from equation (4.11) are visualized. These were calculated on the problem from section 4.2.3.4 with a 5×5 internal grid and $K_{x_i} = K_{y_j} = 4$ basis functions per line. In practice, these matrices are much larger.

vectors of unknowns. Another unfamiliar part is that (4.12) defines, in fact, two different equations:

$$\begin{cases} \mathbf{B_x \Lambda_x c_x} + \mathbf{B_y \Lambda_y c_y} = E\mathbf{B_x c_x} \\ \mathbf{B_x c_x} = \mathbf{B_y c_y}. \end{cases} \tag{4.13}$$

There are a few strategies to further translate this problem into a form for which efficiently implemented and well-studied algorithms exist. Ideally, the solution would take the sparsity of the involved matrices into account.

4.2.1 Solving the matrix approximation

As a first strategy, we investigate how this problem can be directly transformed into a standard mathematical problem.

4.2.1.1 Direct transformation into an eigenvalue problem

We tackle equation (4.13) by rewriting it as

$$\begin{cases} \mathbf{B_x \Lambda_x c_x} + \mathbf{B_y \Lambda_y c_y} = E\mathbf{B_x c_x} \\ \mathbf{B_x \Lambda_x c_x} + \mathbf{B_y \Lambda_y c_y} = E\mathbf{B_y c_y}. \end{cases}$$

This system can be seen as the following generalized rectangular eigenvalue problem:

$$\begin{pmatrix} \mathbf{B_x \Lambda_x} & \mathbf{B_y \Lambda_y} \\ \mathbf{B_x \Lambda_x} & \mathbf{B_y \Lambda_y} \end{pmatrix} \begin{pmatrix} \mathbf{c_x} \\ \mathbf{c_y} \end{pmatrix} = E \begin{pmatrix} \mathbf{B_x} & \mathbf{0} \\ \mathbf{0} & \mathbf{B_y} \end{pmatrix} \begin{pmatrix} \mathbf{c_x} \\ \mathbf{c_y} \end{pmatrix}. \tag{4.14}$$

At first glance, this may seem to enable us to solve the problem elegantly. But, there are a few issues apparent with this translation. One of the most visible problems is that we have translated this into a matrix problem which is twice as large as the original matrices $\mathbf{B_x}$ and $\mathbf{B_y}$ in both rows and columns. On the other hand, one can argue that the matrices are clearly sparse. The matrices $\mathbf{B_x}$ and $\mathbf{B_y}$ are sparse indeed, but the problem is that there are few algorithms available that are even able to solve a generalized rectangular eigenvalue problem, let alone a *sparse* generalized rectangular eigenvalue problem.

Yet, a lack of sparse implementations does not deter us from trying out some numerical experiments. In this first experiment, we will have to fall back to algorithms working on dense matrices. Of course, the runtime will suffer, but from a numerical point of view the results will still be valuable.

Consider the Schrödinger equation with the harmonic oscillator potential:

$$-\nabla^2\psi(x,y) + \left(x^2+y^2\right)\psi(x,y) = E\psi(x,y)$$

on the domain $[-9.5, 9.5] \times [-9.5, 9.5]$ with homogeneous Dirichlet boundary conditions[2]. We apply the described method on a grid with 30 lines in each direction and 16 basis functions per line. This yields two 900×480 matrices $\mathbf{B_x}$ and $\mathbf{B_y}$. The rectangular problem from (4.14) has 1 800 equations and 960 variables. As this eigenvalue problem is heavily overdetermined, we have to consider that solutions will only be accurate in a least squares sense. Much research is already dedicated to solving this kind of problem. As such we will, for now, use the first method from [40] to find solutions.

If we write equation (4.14) symbolically as $\mathbf{Ac} = E\mathbf{Dc}$, and the thin singular value decomposition[3] of $\mathbf{D}$ as $\mathbf{D} = \mathbf{U_D\Sigma_D V_D^H}$, then any solution of (4.14) will also be a solution of the generalized square eigenvalue problem

$$\mathbf{U_D^H A V_D v} = E\mathbf{\Sigma_D v},$$

with $\mathbf{c} = \mathbf{V_D v}$. Applying this method to the problem at hand yields 960 eigenvalues. First, it is important to remark that the resulting values are elements of $\mathbb{C}$. But, in this case, the imaginary part of all values lies between $-1.68 \cdot 10^{-14}$ and $1.68 \cdot 10^{-14}$, so all values may be considered real. The lowest few values are given:

$$\underbrace{-2.85 \cdot 10^{-14} \quad \dots \quad 5.39 \cdot 10^{-14}}_{\text{256 values close to zero}} \quad 2.00 \quad 3.93 \quad 3.93 \quad 4.00 \quad 4.00 \quad \dots$$

This can be more compactly summarized when the number of times an eigenvalue is (up to a precision of 10^{-6}) repeated, is indicated by a subscript:

$$0.00_{256} \quad 2.00 \quad 3.93_2 \quad 4.00_2 \quad 5.36_2 \quad 6.00_3 \quad 6.78_2 \quad 6.87_2 \quad 8.00_4 \quad \dots$$

One immediately notices the many returned zero values. Some of these can be explained by the structure of the matrix on the left-hand size of (4.14). This is a $2 \cdot 900 \times (480 + 480)$ matrix with repeated rows. The rank is thus at most 900, which explains $960 - 900 = 60$ zero eigenvalues. The others are more difficult to explain, but suggestions can be found in the fact that $\mathbf{B_x\Lambda_x c_x}$ and $\mathbf{B_y\Lambda_y c_y}$ describe (up to a least squares approximation) the same

[2]We also considered this problem in section 4.1.1.1.

[3]A thin singular value decomposition of an $m \times n$-matrix results in Σ being a square $k \times k$ diagonal matrix, with $k = \min(m, n)$.

points. So, $\mathbf{c_x}$ and $\mathbf{c_y}$ are not independent, which implies that the rank of the matrix $\begin{pmatrix}\mathbf{B_x\Lambda_x} & \mathbf{B_y\Lambda_y}\end{pmatrix}$ will probably not be maximal. But, these zero eigenvalues indicate another problem. Namely, that for these values it is not at all guaranteed that $\mathbf{B_x c_x} = \mathbf{B_y c_y}$. So, these eigenvalues will be nonsensical in the context of the original Schrödinger problem.

Remember that the true eigenvalues of the harmonic oscillator are 2, 4, 4, 6, 6, 6, 8, 8, 8, 8, ... And, we notice that these true values can indeed be found in our solutions, but also many other values are present. Since the method from [40] for solving generalized rectangular eigenvalue problems may return more solutions than the original problem has, we still have to filter some out. One way to only end up with true eigenvalues is by substituting them back into the original problem (4.13) and verifying the residuals of both equations:

$$r_1 = \left\| \mathbf{B_x\Lambda_x c_x} + \mathbf{B_y\Lambda_y c_y} - \frac{E}{2}\left(\mathbf{B_x c_x} + \mathbf{B_y c_y}\right)\right\| \text{ and } r_2 = \left\|\mathbf{B_x c_x} - \mathbf{B_y c_y}\right\|.$$

The found possible eigenvalue with the lowest residuals r_1 and r_2 are also true eigenvalues of the Schrödinger problem. When we sort all possibilities by their first residual r_1, we obtain the following table.

E	2.000 000	4.000 000	4.000 000	6.000 000	7.999 998
r_1	$1.5 \cdot 10^{-5}$	$4.7 \cdot 10^{-4}$	$4.7 \cdot 10^{-4}$	$4.9 \cdot 10^{-4}$	$5.0 \cdot 10^{-3}$
r_2	$1.2 \cdot 10^{-5}$	$2.3 \cdot 10^{-4}$	$2.3 \cdot 10^{-4}$	$1.6 \cdot 10^{-4}$	$1.2 \cdot 10^{-3}$

E	7.999 998	9.999 997	9.999 997	10.000 006	8.000 007
r_1	$5.0 \cdot 10^{-3}$	$6.6 \cdot 10^{-3}$	$6.7 \cdot 10^{-3}$	$1.0 \cdot 10^{-2}$	$1.4 \cdot 10^{-2}$
r_2	$1.2 \cdot 10^{-3}$	$1.3 \cdot 10^{-3}$	$1.3 \cdot 10^{-3}$	$2.1 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$

E	8.000 007	13.999 988	5.999 974	5.999 974	10.000 151
r_1	$1.4 \cdot 10^{-2}$	$1.9 \cdot 10^{-2}$	$3.8 \cdot 10^{-2}$	$3.8 \cdot 10^{-2}$	$4.8 \cdot 10^{-2}$
r_2	$3.5 \cdot 10^{-3}$	$2.7 \cdot 10^{-3}$	$1.3 \cdot 10^{-2}$	$1.3 \cdot 10^{-2}$	$9.6 \cdot 10^{-3}$

Here, we see that when the first residual r_1 is low, the other is as well. Also, in the first few eigenvalues, only true solutions are present.

Upon studying this direct method to solve the system of equations (4.13), we have noticed some drawbacks. First, the proposed system (4.14) is, in a certain sense, twice as large as the discrete problem we started from. Eigenvalue algorithms are at least cubic in complexity. So, this doubling in size implies an eightfold runtime penalty. Second, and numerically interesting, we obtain many more 'solutions' than the ones we are looking for. Each generalized eigenvalue, with its eigenvector, has to be computed and checked against the residuals. This will throw away many eigenvalues.

4.2.1.2 Restriction to a null space

One improvement we can make to decrease the runtime is by not solving a system that is twice as large, but by solving two smaller systems. Furthermore, instead of ending up with too many eigenvalues and trying to filter out the wrong ones, we have devised a way to, a priori, limit the number of solutions. The idea here is that, before solving an eigenvalue problem, we solve the second equation of (4.13) and only take those solutions into account.

The first equation resembles a generalized rectangular eigenvalue problem, the second is a classical linear system. Let us only consider $n_x + n_y$ dimensional vectors $\mathbf{c} = \begin{pmatrix}\mathbf{c_x}^\mathsf{T} & \mathbf{c_y}^\mathsf{T}\end{pmatrix}^\mathsf{T}$ which solve

$$\begin{pmatrix}\mathbf{B_x} & -\mathbf{B_y}\end{pmatrix}\mathbf{c} = \mathbf{0}.$$

This allows us to unify $\mathbf{c_x}$ and $\mathbf{c_y}$, while ensuring $\mathbf{B_x c_x} = \mathbf{B_y c_y}$ is satisfied. To expand on this idea, we will write $\mathbf{c}$ to be an element of the right kernel of $\begin{pmatrix}\mathbf{B_x} & -\mathbf{B_y}\end{pmatrix}$. For this, we define the columns of the $(n_x + n_y) \times z$ matrix $\mathbf{Z}$ to be a basis of this right kernel:

$$\begin{pmatrix}\mathbf{B_x} & -\mathbf{B_y}\end{pmatrix}\mathbf{Z} = \mathbf{0}. \tag{4.15}$$

Computing this kernel numerically is definitely not trivial. For rectangular matrices, there are a few well-known methods to compute the kernel. In the next paragraphs we will explore these methods and consider how well they are suited for our problem. For ease of notation we will be solving the rectangular system $\mathbf{AZ} = \mathbf{0}$ with $\mathbf{A}$ a (sparse) $m \times n$ matrix and $\mathbf{Z}$ a yet unknown $n \times z$ matrix, whose columns are a basis for the right null space of $\mathbf{A}$.

Finding the null space of a matrix
One of the first methods for finding the null space of a matrix one finds when browsing the literature is by computing its singular value decomposition. For extended background about using a singular value decomposition for null space construction, we refer to [32, section 2.4]. In this book the following theorem can be found.

Theorem 4.1. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ be a real $m \times n$ matrix with singular value* *decomposition*[4]

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal,$$

and ordered singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p$ *with* $p = \min(m, n)$*. If* $\mathbf{A}$ *has* r *positive singular values, then*[5] $\mathrm{rank}(A) = r$ *and*

$$\begin{aligned}\mathrm{null}(\mathbf{A}) &= \mathrm{span}\{\mathbf{v}_{r+1}, \dots, \mathbf{v}_n\},\\ \mathrm{ran}(\mathbf{A}) &= \mathrm{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\},\end{aligned}$$

with $\mathbf{u}_i$ *the* i^{th} *column of* $\mathbf{U}$*, analogous* $\mathbf{v}_i$ *the* i^{th} *column of* $\mathbf{V}$*.*

The singular value decomposition gives us a constructive way to find a basis of the null space of a matrix $\mathbf{A}$. For dense matrices this works very well, but is computationally quite expensive. For sparse matrices the story is more complicated. There are many available routines for calculating singular value decompositions. Some of them are constructed specifically for computing the singular value decomposition, others are adapted from symmetric eigenvalue solvers on the matrix $\mathbf{A}^\intercal\mathbf{A}$ or on $\mathbf{A}\mathbf{A}^\intercal$, whichever is more efficient, for example: `SLEPc` [38], `spectra` [82] or `SciPy` [101] with `ARPACK` [70]. We have experimented with many solvers. One of the first difficulties is that all of them require specifying beforehand how many singular values are required. In our case, the dimension of the null space is unknown, so it is impossible to exactly specify this. The solver which seemed most promising to combat this issue was `SLEPc`. There, one can instruct the solver to continue to find eigenvalues as long as a custom condition is not met. In our case, we could provide that as long as the singular values are close to zero that then the search should continue. But, on the other hand, the algorithms used there are using a small-dimensional subspace in which convergence to the eigenvectors happens. This space should at least have as many dimensions as the number of eigenvalues required, and has to be specified beforehand. As noted previously, in our case, this is difficult. But, even ignoring this complication, and just specifying a sufficiently high number of required singular values, did not solve it either. All algorithms had trouble to reliably converge to the required number of singular values. In the simplest test problem, the best results we were able to obtain this way were a few dozen of the smallest singular values, when we required a few hundred.

[4]In this expression the matrix $\mathbf{U}$ ($\mathbf{V}$ respectively) are orthogonal $m \times m$ ($n \times n$ respectively) matrices. $\mathbf{\Sigma}$ is a diagonal $m \times n$ matrix with the ordered singular values on the diagonal: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)}$.

[5]The null space $\mathrm{null}(\mathbf{A})$ of $\mathbf{A}$ is the vector space of all vectors $\mathbf{x}$ for which $\mathbf{Ax} = \mathbf{0}$. The range $\mathrm{ran}(\mathbf{A})$ is the vector space of all vectors $\mathbf{y}$ such that there exists a vector $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{y}$

Another popular method [96] to find the null space of a matrix is by using a QR-decomposition with pivoting. For this we decompose the $m \times n$ matrix $\mathbf{A}$ as $\mathbf{A}^\intercal\mathbf{E} = \mathbf{QR}$, with $\mathbf{E}$ an $m \times m$ permutation matrix, $\mathbf{Q}$ a square $n \times n$ orthogonal matrix and $\mathbf{R}$ an $n \times m$ rectangular upper triangular matrix, with the elements on its diagonal sorted (descending in absolute value). To find the kernel, we now consider for which vectors $\mathbf{z}$ the expression $\mathbf{Az} = \mathbf{ER}^\intercal\mathbf{Q}^\intercal\mathbf{z}$ becomes zero. Notice that this only happens when $\mathbf{z}$ is in the linear span of the columns of $\mathbf{Q}$ corresponding to a diagonal item in $\mathbf{R}$ that is (close to) zero and, if $n > m$ the last $n - m$ columns of $\mathbf{Q}$. So an orthogonal basis of the null space of $\mathbf{A}$ can be found in a selection of columns of $\mathbf{Q}$ in the QR-decomposition of $\mathbf{A}^\intercal$. For dense matrices this procedure also works very well. We can quite easily compute the full kernel with built-in QR routines. Upon selecting which columns to include, the diagonal elements of $\mathbf{R}$ can even be used to take into account a prespecified tolerance. It is also valuable to notice that computing a QR-decomposition is significantly more efficient than finding all singular values. So, for dense matrices, this second method is preferred. But for sparse matrices, the story is, again, quite different. There are many well-tested routines to compute a sparse QR-decomposition, for example SuiteSparseQR [25] and `Eigen`'s QR [33]. Yet, we have tested this algorithm with both solvers, without success. `Eigen` does not contain a rank revealing sparse QR-decomposition, and as such is unable to compute the full kernel. SuiteSparseQR, on the other hand, was sometimes able to compute the kernel, but this computation was each time very sensitive to the tolerance of when to pivot and which columns of $\mathbf{Q}$ to include. Even worse, the perfect 'tolerance' was different for each test problem, or even for the same problem with different parameters.

Solving the matrix approximation

In conclusion, coming back to equation (4.15), for now we will ignore the sparsity present and use the algorithm for the dense QR-decomposition from `Eigen` [33].

The vector $\mathbf{c}$ can now be written as a linear combination of columns of $\mathbf{Z}$:

$$\mathbf{c} = \begin{pmatrix} \mathbf{c_x} \\ \mathbf{c_y} \end{pmatrix} = \begin{pmatrix} \mathbf{Z_x} \\ \mathbf{Z_y} \end{pmatrix} \mathbf{u} = \mathbf{Zu}.$$

Another benefit of considering $\mathbf{Zu}$, besides only considering solutions of $\mathbf{B_x c_x} = \mathbf{B_y c_y}$, is that we unified the two vectors of unknowns $\mathbf{c_x}$ and $\mathbf{c_y}$ into one (much) smaller vector $\mathbf{u}$. This simplifies the two problems of equation (4.13) into

$$\begin{pmatrix} \mathbf{B_x \Lambda_x} & \mathbf{B_y \Lambda_x} \end{pmatrix} \mathbf{Zu} = E\mathbf{B_x Z_x u} = E\mathbf{B_y Z_y u}.$$

For the right-hand side, by construction $\mathbf{B_x Z_x} = \mathbf{B_y Z_y}$. Now, this problem has become a generalized rectangular eigenvalue problem.

As stated in section 4.2.1.1, only few implementations to solve these matrix problems are available. In our first tests we have found the first method from [40] to be sufficient.

If $\mathbf{B_x}$ and $\mathbf{B_y}$ are viewed as dense matrices, this technique for solving (4.13) works well. The correct eigenvalues are found within reasonable computation time.

If relatively high accuracies are required, the used grid size should be increased. This also has a significant impact on the size of the involved matrices. As these become larger, using their sparsity becomes a necessity. This is where both of the previous techniques to solve (4.13) fall short. There, existing algorithms for sparse matrices were not able to perform the required computations efficiently and accurately. To combat this, we have developed an alternative method, explicitly tuned to be able to use the sparsity of the matrices.

4.2.1.3 Least squares approximation

We start with the observation that classical matrix problems normally do not contain two vectors of unknowns. So, we need a way to unify them. With the previous technique we considered vectors in the right kernel of a given matrix. We have discovered that this is surprisingly difficult to compute, while respecting the sparsity of the matrices involved. As an alternative to unify $\mathbf{c_x}$ and $\mathbf{c_y}$, one could fix $\mathbf{c_x}$ and compute $\mathbf{c_y}$ as the least-squares solution of $\mathbf{B_x c_x} = \mathbf{B_y c_y}$. Or, the other way around, fix $\mathbf{c_y}$ and compute $\mathbf{c_x}$. Both seem like artificial asymmetric choices. Therefore, we propose to find an m-dimensional vector $\mathbf{z}$, such that $\mathbf{c_x}$ and $\mathbf{c_y}$ can be calculated as least square solutions of $\mathbf{B_x c_x} = \mathbf{B_y c_y} = \mathbf{z}$. With normal equations, we can define this more formally as

$$\begin{aligned} \mathbf{c_x} &= \left(\mathbf{B_x^\intercal B_x}\right)^{-1} \mathbf{B_x^\intercal z} \\ \text{and } \mathbf{c_y} &= \left(\mathbf{B_y^\intercal B_y}\right)^{-1} \mathbf{B_y^\intercal z}. \end{aligned}$$

Notice that the computation of $\left(\mathbf{B_x^\intercal B_x}\right)^{-1}$ is not at all as difficult as this may seem: $\mathbf{B_x}$ is a block diagonal matrix. So, these computations, calculating an inverse or multiplying with another block diagonal matrix, can be done for each of the much smaller dense subblocks on the diagonal. From figure 4.4, we know that the matrix $\mathbf{B_y}$ is not block diagonal. But, if we apply an appropriate

$m \times m$ permutation matrix $\mathbf{P}$ we can obtain that $\widetilde{\mathbf{B}}_\mathbf{y} := \mathbf{P}\mathbf{B}_\mathbf{y}$ is, in fact, also a block diagonal matrix. So, the computation of $\mathbf{c}_\mathbf{y}$ can be written as:

$$\mathbf{c}_\mathbf{y} = \left(\widetilde{\mathbf{B}}_\mathbf{y}^\intercal \widetilde{\mathbf{B}}_\mathbf{y}\right)^{-1} \widetilde{\mathbf{B}}_\mathbf{y}^\intercal \mathbf{P}\mathbf{z}.$$

And again, as the involved matrices (except $\mathbf{P}$) are block diagonal, all computation can be delegated to the much smaller dense subblocks of $\widetilde{\mathbf{B}}_\mathbf{y}$.

Numerically, directly computing inverse matrices should be avoided. Therefore, we compute the matrix $\mathbf{Z}_\mathbf{x} := \left(\mathbf{B}_\mathbf{x}^\intercal \mathbf{B}_\mathbf{x}\right)^{-1} \mathbf{B}_\mathbf{x}^\intercal$ as the least squares solution of $\mathbf{B}_\mathbf{x}\mathbf{Z}_\mathbf{x} = \mathbf{I}_{m\times m}$, and analogous for $\mathbf{Z}_\mathbf{y} = (\widetilde{\mathbf{B}}_\mathbf{y}^\intercal \widetilde{\mathbf{B}}_\mathbf{y})^{-1}\widetilde{\mathbf{B}}_\mathbf{y}^\intercal$. And again, all these computations can be executed on each of much smaller dense diagonal blocks.

The expressions for $\mathbf{c}_\mathbf{x}$ and $\mathbf{c}_\mathbf{y}$ now allow us to rewrite the first equation of (4.13) into

$$\left(\mathbf{B}_\mathbf{x}\mathbf{\Lambda}_\mathbf{x}\mathbf{Z}_\mathbf{x} + \mathbf{P}^\intercal \widetilde{\mathbf{B}}_\mathbf{y}\mathbf{\Lambda}_\mathbf{y}\mathbf{Z}_\mathbf{y}\mathbf{P}\right)\mathbf{z} = E\mathbf{z}. \tag{4.16}$$

This problem is a classical square non-symmetric sparse eigenvalue problem. The main reason we developed this technique is that many well-tested extremely-efficient sparse eigenvalue solvers exist. Before declaring this technique as superior, there is still one issue left to solve. Up until now, we have ignored that $\mathbf{B}_\mathbf{x}\mathbf{c}_\mathbf{x} = \mathbf{B}_\mathbf{y}\mathbf{c}_\mathbf{y}$ should hold. So one way to ensure this, is by checking that each eigenvector $\mathbf{z}$ of (4.16) satisfies (up to the required tolerance)

$$\mathbf{Z}_\mathbf{x}\mathbf{z} - \mathbf{Z}_\mathbf{y}\mathbf{P}\mathbf{z} = 0. \tag{4.17}$$

Surprisingly, some sparse eigenproblem solvers allow us to do something more efficient. `SLEPc` [38] for example, allows one to provide a *deflation space*. When this is provided, the eigensolver restricts solutions to the orthogonal complement of the deflation space. This functionality exists to allow users to continue an eigenvalue search, by only seeking solutions orthogonal to earlier seen eigenvectors. Or, a user may provide the null space of the matrix to exclude all zero eigenvalues.

Constructing a deflation space

In our case, this feature can be used with the span of the rows of $\mathbf{Z}_\mathbf{x} - \mathbf{Z}_\mathbf{y}\mathbf{P}$ as deflation space. Then, only solutions will be considered which satisfy (4.17). While implementing and testing, we have found that the algorithms within `SLEPc` have trouble converging to the low end of the spectrum of the matrix from (4.16). One of the issues `SLEPc` encounters is that the matrix is highly singular. The employed algorithms for finding the low end of the spectrum are advanced adaptations of inverse power iterations. Therefore, during the execution many linear systems will be solved with our singular matrix. To solve these systems, they are using an LU-decomposition of the matrix in question. And, this decomposition is not rank-revealing and has issues with the singularity of the matrix. The developers of `SLEPc` propose that a user should use a different, external, linear solver. Or, one could compute the null space independently and provide the results as the deflation space.

Using an external solver is definitely not an attractive solution. Besides the difficulties of finding such a solver in the first place, or getting it to work together with `SLEPc` and `Eigen` (which we are already using), numerically we can do better. As most of the null space of the matrix in (4.16) is irrelevant for the original Schrödinger problem we are trying to solve, it would be advantageous to be able to exclude this null space before searching for eigenvalues. Trying to numerically compute this null space, is reminiscent of the analysis we have made in section 4.2.1.2. There we came to the conclusion that an implementation which takes the sparsity into account is truly difficult to get working. Therefore, we would prefer a theoretical analysis which provides a sufficiently large null space, such that `SLEPc` converges, even when searching for the low end of the spectrum.

The first step is to investigate where this singularity comes from. We can use the degrees of freedom as a guide. Initially, we were looking for many linear combinations of basis functions, contained within the n_x-dimensional vector $\mathbf{c_x}$ and the n_y-dimensional vector $\mathbf{c_y}$. So the number of degrees of freedom, ignoring the constraints $\mathbf{B_x c_x} = \mathbf{B_y c_y}$, starts of at $n_x + n_y$. To unify these two vectors, we have introduced $\mathbf{z}$ in (4.16). This $\mathbf{z}$ has m rows. In principle, m has nothing to do with our initial degrees of freedom n_x and n_y. So, there will be choices for $\mathbf{z}$ which imply that the corresponding $\mathbf{c_x} = \mathbf{Z_x z}$ or $\mathbf{c_y} = \mathbf{Z_y P z}$ are zero. So the null space of the right-hand side of (4.16) can be in large part found in the null spaces of $\mathbf{Z_x}$ and $\mathbf{Z_y P}$. Luckily for us, the null spaces for these last matrices can be easily computed. These matrices are block diagonal. This means that we can find their null spaces by computing the null space of each of the dense subblocks, for which `Eigen` provides many well-tested algorithms. In our case, a dense QR-decomposition with column pivoting suffices.

With these null spaces in hand, it is still not as easy as adding them to the deflation space with `SLEPc`. For this library, it is not necessary that the vectors in the deflation space are orthogonal, as they orthogonalize the space themselves by a Gram–Schmidt-like procedure. The user manual of `SLEPc` does not mention this, but this procedure assumes that no redundant vectors are given in the deflation space. If we add all basis vectors from the null spaces of $\mathbf{Z_x}$ and $\mathbf{Z_y P}$ to the deflation space, `SLEPc` throws an error indicating that the whole space is spanned by the deflation space, which should not be the case. The two null spaces we are considering do intersect. By adding all their basis vectors, `SLEPc` will force all these vectors to be orthogonal. Because of numerical approximation errors, some of these resulting vectors will no longer lay in any of the null spaces, but in a random direction. So instead of adding all basis vectors directly, we will first construct an orthogonal basis of the intended deflation space. This new basis can be found as the columns of the Q-matrix in the QR-decomposition of the matrix with as columns the basis vectors from $\mathbf{Z_x}$ and $\mathbf{Z_y P}$.

We are finally able to request the low end of the spectrum of the right-hand side of (4.16). And the numerical results are promising! However, we noticed that this sparse technique is significantly slower than the dense implementation from 4.2.1.2.

We suspect the reason for this slowness to be due to the computation of the smaller basis with the QR-decomposition. Even with using a sparse QR-decomposition, this still takes a significant amount of runtime. Besides this, there is also a more implementation specific time sink: one can provide the deflation space in `SLEPc` as a list of vectors. In this library all vectors are dense. With the large deflation space we are using, this results in many dense vector operations on this huge list of dense vectors. This defeats our goal of maximally using the sparsity of the system. We did not find another library that supports deflation spaces, let alone sparse deflation spaces. But even if we would have, this still would not have worked. After the QR-decomposition, the vectors from our new basis for the deflation space are dense. This inherently slows everything down.

Using deflation gave promising numerical results, but due to implementation drawbacks it was still too slow. But, fortunately, we are able to add deflation to any eigenvalue problem, even if the solver does not support it. In the literature there are a few techniques available [87, section 4.2][73]. The ideas that are used for sparse problems mostly involve ensuring that all vectors from the deflation space have eigenvalue zero. This makes sure that iterative schemes, like power iteration or (restarted) Arnoldi iteration [6][96, Chapter 6], never find eigenvectors contained within the deflation space.

Solving the square eigenvalue problem
To solve (4.16), a matrix-free eigenvalue solver[6] can be used (e.g. [70, 38, 82]). In our implementation we have chosen for `spectra` [82], which is an implicitly restarted Arnoldi method. Since we are only interested in the lowest lying real eigenvalues, we will use a lower bound of the ground state as a shift to make the interesting eigenvalues strictly positive. A simple choice for this shift is $\sigma = \min_{\mathbf{x}\in\Omega} V(\mathbf{x})$. Now the values we are interested in are the smallest positive real eigenvalues of $\mathbf{A} - \sigma\mathbf{I}$. For finding these, there are two possible strategies. First, most eigenvalue solvers allow users to specify which values to select, the largest negative real values for example. All solvers are best at finding the outer boundaries of the spectrum. However, some also allow selecting values deep inside the spectrum, like the small positive reals, with the caveat that convergence may suffer. On the other hand, we could also search for the largest positive real eigenvalues of the matrix $(\mathbf{A} - \sigma\mathbf{I})^{-1}$. In this case, the convergence will be better, but the runtime may suffer due to the needed linear solves, for example with an LU-decomposition.

In any case, we want to exclude the null spaces of $\mathbf{Z_x}$ and $\mathbf{Z_y P}$ to be considered. Let the columns of $\mathbf{K_x}$ be a basis of the null space of $\mathbf{Z_x}$ and the columns of $\mathbf{K_y}$ a basis of the null space of $\mathbf{Z_y P}$. Following [87, 73], let us define $\mathbf{D} := (\mathbf{I} - \mathbf{K_y}\mathbf{K_y}^\intercal)(\mathbf{I} - \mathbf{K_x}\mathbf{K_x}^\intercal)$ as what we call the deflation matrix. Applying this operator $\mathbf{D}$ to any vector orthogonally projects it out of the null spaces to their orthogonal complement. We need to ensure that the operator of which we are searching eigenvalues only returns vectors which are orthogonal to the unwanted space. On the other hand, we also want to ensure that when solving linear systems, the incoming vectors are orthogonal to the null space. With this in mind, we propose to find the smallest real strictly positive eigenvalues of

$$\mathbf{D}\left(\mathbf{A} - \sigma\mathbf{I}\right)\mathbf{D}$$

or the largest real positive eigenvalues of

$$\mathbf{D}\left(\mathbf{A} - \sigma\mathbf{I}\right)^{-1}\mathbf{D}. \tag{4.18}$$

In practice, it is important to note that this product most likely will be a dense matrix. When using a matrix-free sparse eigenvalue solver this can

[6]A classical (dense) eigenvalue solver assumes the problem is given as $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ with $\mathbf{A}$ a matrix. For sparse problems, constructing the matrix $\mathbf{A}$ explicitly may be inefficient or may introduce unwanted dense matrices. For example, if $\mathbf{A}$ is the product of two sparse matrices, in general this will itself not be as sparse. To combat this, many solvers implement what is called a matrix-free operation. Here, the user provides the linear operator $\mathcal{L}$ to find the eigenvalues of as a function: $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^n : \mathbf{x} \to \mathcal{L}(\mathbf{x})$. So there is no need to construct any matrix at all.

be implemented as a few sparse matrix-vector products, which is much more efficient. As an example, the pseudocode for finding eigenvalues with the operation from equation (4.18) can be found in algorithm 1.

Algorithm 1 The pseudocode of the algorithm to apply projection deflation with a matrix-free eigenvalue solver in shift-invert mode, see equation (4.18).

```
Compute the sparse LU decomposition of $\mathbf{A} - \sigma\mathbf{I}$

function ApplyMatrix(vector $\mathbf{v}$, result $\mathbf{r}$)
    $\mathbf{u} = (\mathbf{I} - \mathbf{K_y}\mathbf{K_y}^\intercal)(\mathbf{I} - \mathbf{K_x}\mathbf{K_x}^\intercal)\mathbf{v}$
    Use LU-decomposition to solve $(\mathbf{A} - \sigma\mathbf{I})\mathbf{x} = \mathbf{u}$ for $\mathbf{x}$.
    $\mathbf{r} = (\mathbf{I} - \mathbf{K_y}\mathbf{K_y}^\intercal)(\mathbf{I} - \mathbf{K_x}\mathbf{K_x}^\intercal)\mathbf{x}$
end function

eigenvalueSolver(ApplyMatrix)        ▷ Call matrix-free eigensolver library
```

In section 4.2.4, we will take a closer look at how both solving strategies perform.

In summary, we have studied three techniques to solve (4.13). With all three techniques, we were able to approximate the sought eigenvalues. However, only with the last technique, we were able to build a sparse implementation. This method is significantly faster than the other two, without loosing any accuracy.

Before performing some numerical experiments in section 4.2.3, we will develop in the next section a method to evaluate the found eigenfunctions in arbitrary points.

4.2.2 Computing eigenfunctions

This new method is able to compute eigenvalues of Schrödinger equations efficiently and accurately. One of the disadvantages of many simpler numerical schemes is their inability to evaluate eigenfunctions. The method from [102], for example, provides an approximation to the value of the eigenfunctions in a finite number of fixed evaluation points. The eigenfunction is only known in the grid points used to construct the matrix problem. For us, this is unsatisfactory.

In recent years, we have focused upon accurately computing eigenfunctions in arbitrary points. If we look back at the one-dimensional case for a moment, we see that in `Matslise 2.0` [68] it was possible to compute eigenfunctions.

However, each of the required evaluation points needed to be specified beforehand. Also, when requesting the value of the eigenfunction in many points, the runtime increased dramatically. Only in our work from chapter 2 and [9], we were able to build an algorithm which could compute the value of the eigenfunction in many points efficiently. These points did not need to be specified beforehand. In a more technical sense, when eigenfunctions are requested, `Matslise 3.0` returns a function, not just a grid of values. This has many benefits: first, `Matslise 3.0` may decide its own step size, based upon the required accuracy and jumps in the domain, without taking a ridiculous amount of steps, even if we want to evaluate the eigenfunction in many points. Second, besides these clear efficiency gains, doing computations with these eigenfunctions becomes a lot more user-friendly. Many numerical procedures are adaptive, for example ODE-solvers with adaptive step size, based upon an error estimate. Similarly, the computation of a numerical quadrature is ideally done with adaptive formulae to get a grip on the error of the result. When using these adaptive procedures it is not known beforehand which values of the involved functions will be required. Therefore, we strived to build the dynamic computation of eigenfunctions into the core of our algorithms.

In our own work, see [8] or chapter 3 or even this chapter, we already benefited immensely from the ability to compute eigenfunctions efficiently in arbitrary points. Of course, we want to build this feature also into the new method developed in this section.

This new method provides not only the eigenvalues but also the corresponding vector $\mathbf{c_x}$ and $\mathbf{c_y}$, containing each of the values $c_k^{(x_i)}$ and $c_k^{(y_j)}$, for each i, j and k. These can be used to reconstruct the two-dimensional eigenfunction $\psi(x, y)$ on each of the grid lines (see also equations (4.6) and (4.7)):

$$\psi(x_i, y) = \sum_{k=0}^{\infty} c_k^{(x_i)} \beta_k^{(x_i)}(y)$$

$$\text{and } \psi(x, y_j) = \sum_{k=0}^{\infty} c_k^{(y_j)} \beta_k^{(y_j)}(x).$$

Because $\beta_k^{(x_i)}$ and $\beta_k^{(y_j)}$ are computed with `Matslise 3.0`, these functions can be cheaply evaluated in any point. This means, visually, that the value of the two-dimensional eigenfunction can be calculated on each of the horizontal and vertical lines of figure 4.3.

Almost all points in the domain Ω are not contained on any of these lines. It is not trivial how one would reconstruct the eigenfunction $\psi(x, y)$ in such a general

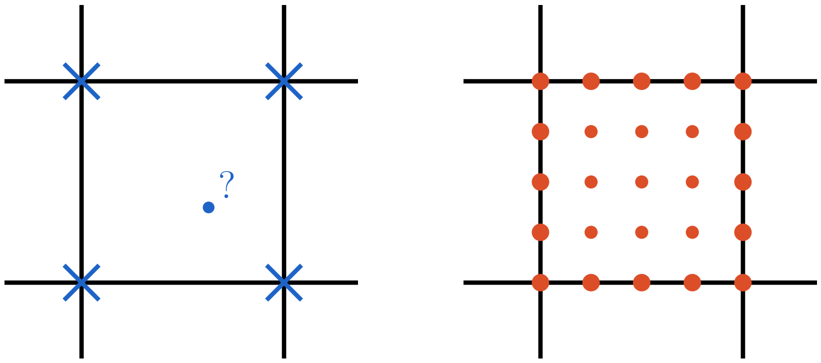


Figure 4.5: On the left: a closer look at an internal rectangle from figure 4.3. On the right: the points calculated with a small finite difference approximation.

point, from the results $\mathbf{c_x}$ and $\mathbf{c_y}$ from the matrix problem. Let us simplify the problem by only considering a rectangle between two consecutive grid lines, in both the x- and y-direction. The left side of figure 4.5 contains such a single rectangle. The issue now is: how do we reconstruct the eigenfunction on this rectangle, given its value on the whole boundary? Since we have to approximate an unknown function, given the values on a fixed number of points, interpolation comes to mind. As a first idea, we have tried building a polynomial basis to interpolate the eigenfunction with nodes on the boundary of such a rectangle.

The most natural k-degree polynomial basis to consider are the monomials $x^i y^j$ for all i, j with $i + j \leq k$. For $k = 4$, for example, this yields:

$$1,\ y,\ y^2,\ y^3,\ y^4,\ x,\ xy,\ xy^2,\ xy^3,\ x^2,\ x^2y,\ x^2y^2,\ x^3,\ x^3y \text{ and } x^4.$$

This basis already highlights one of the fundamental issues with this approach. In figure 4.6, two different linear combinations of basis functions are plotted: $x^2 + y^2 - 1$ and x^2y^2. Clearly, these functions are different. Yet, their values on the boundary of the unit square are identical. This indicates that it will not be possible to reconstruct the eigenfunction by using polynomial basis functions based upon values from the boundary alone.

To be able to calculate an interpolating polynomial uniquely, we also require values in the interior of the domain. The vectors $\mathbf{c_x}$ or $\mathbf{c_y}$ provide the value of the eigenfunction on the boundary of the small internal rectangle. They tell us nothing about the value of the eigenfunction in the interior. But, on this

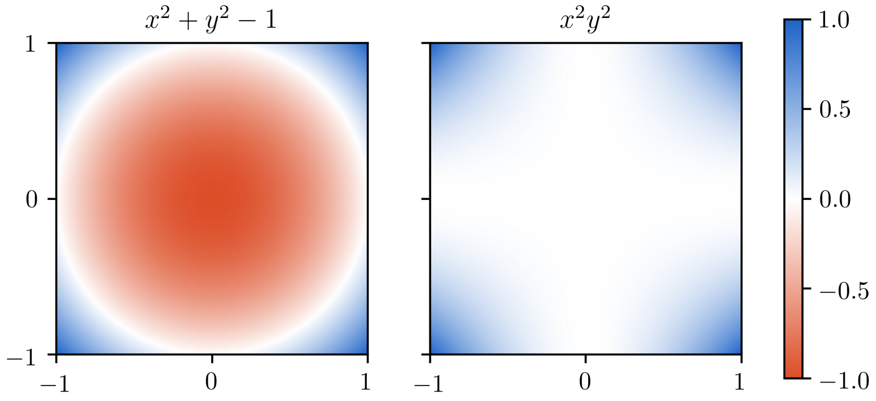


Figure 4.6: Two different fourth degree polynomial functions with the same value on the boundary of the region $[-1,1]\times[-1,1]$.

internal rectangle $\Omega_{i,j} = [x_i, x_{i+1}] \times [y_j, y_{j+1}]$ we know that the eigenfunction $\psi(x,y)$ satisfies the equation

$$-\nabla^2\psi(x,y) + V(x,y)\psi(x,y) = E\psi(x,y) \tag{4.19}$$

with E fixed and boundary conditions:

$$\begin{aligned}
\psi(x_i,y) &= \sum_{k=0}^{\infty} c_k^{(x_i)}\beta_k^{(x_i)}(y),\\
\psi(x_{i+1},y) &= \sum_{k=0}^{\infty} c_k^{(x_{i+1})}\beta_k^{(x_{i+1})}(y),\\
\psi(x,y_j) &= \sum_{k=0}^{\infty} c_k^{(y_j)}\beta_k^{(y_j)}(x)\\
\text{and } \psi(x,y_{j+1}) &= \sum_{k=0}^{\infty} c_k^{(y_{j+1})}\beta_k^{(y_{j+1})}(x).
\end{aligned}$$

Using this, we place a $K\times K$ grid on this small rectangular domain $\Omega_{i,j}$, as in the right-hand side of figure 4.5. The grid points can now be described as $x_i = x_0^{(i)}, x_1^{(i)}, \dots, x_m^{(i)}, \dots, x_K^{(i)} = x_{i+1}$, and analogous for y: $y_j = y_0^{(j)}$,

$y_1^{(j)}, \ldots, y_n^{(j)}, \ldots, y_K^{(j)} = y_{j+1}$. This allows the eigenfunction $\psi(x,y)$ to be approximated in each of these points by:

$$\psi_{m,n}^{(i,j)} \approx \psi(x_m^{(i)}, y_n^{(j)}) \quad \text{for each } m,n \in \{0,1,\ldots,K\}.$$

With this notation in hand, equation 4.19 can be approximated in each grid point by a finite difference approximation. Because $\Omega_{i,j}$ is only a small part of the total domain Ω, the grid size K may be chosen relatively small, without a penalty on accuracy. In our case, we found that fixing $K = 4$ gives satisfactory results, without too much computational overhead. So for the following discussion about a finite difference approximation on $\Omega_{i,j}$, we assume $K = 4$.

Because only a relatively small grid is used, the finite difference scheme should be as accurate as possible, without making any assumptions on points outside $\Omega_{i,j}$. Denote by f_0'' the value we want to approximate on an equidistant grid with step size h, on which the function in question takes the values $\ldots, f_{-2}, f_{-1}, f_0, f_1, f_2, \ldots$ around the central point. The finite difference approximation can be written as

$$f_0'' \approx \sum_{i=a}^{b} \frac{\alpha_i}{h^2} f_i,$$

with appropriate choices for a and b such that each f_i lies within the grid of approximations. The relevant five-point formulae we will be using are provided in the following table.

α_{-3}	α_{-2}	α_{-1}	α_0	α_1	α_2	α_3
		$\frac{11}{12}$	$\frac{-5}{3}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{-1}{12}$
	$\frac{-1}{12}$	$\frac{4}{3}$	$\frac{-5}{2}$	$\frac{4}{3}$	$\frac{-1}{12}$	
$\frac{-1}{12}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{-5}{3}$	$\frac{11}{12}$		

Let us, for ease of notation, summarize these values into the following matrix:

$$\mathbf{D} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ \frac{11}{12} & \frac{-5}{3} & \frac{1}{2} & \frac{1}{3} & \frac{-1}{12} \\ \frac{-1}{12} & \frac{4}{3} & \frac{-5}{2} & \frac{4}{3} & \frac{-1}{12} \\ \frac{-1}{12} & \frac{1}{3} & \frac{1}{2} & \frac{-5}{3} & \frac{11}{12} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

With this finite difference approximation (4.19) can now be approximated as a nine-dimensional linear system with for each $m, n \in \{1, 2, 3\}$:

$$\begin{cases} \vdots \\ \displaystyle\sum_{l=0}^{4} \frac{D_{m,l}}{\Delta x_i^2} \psi_{l,n}^{(i,j)} + \sum_{l=0}^{4} \frac{D_{n,l}}{\Delta y_j^2} \psi_{m,l}^{(i,j)} = \left(V(x_m^{(i)}, y_n^{(j)}) - E \right) \psi_{m,n}^{(i,j)} \\ \vdots \end{cases}$$

The nine unknown variables are $\psi_{m,n}^{(i,j)}$ for $m, n \in \{1, 2, 3\}$. The step sizes Δx_i and Δx_j are defined as $(x_{i+1} - x_i)/K$ and $(y_{i+1} - y_i)/K$ respectively. Notice that for all $m, n \in \{0, 1, \ldots, K\}$, if any of $m, n \in \{0, 4\}$, the values $\psi_{m,n}^{(i,j)}$ are known, as they can be computed as the boundary values.

This relatively small linear system can now be numerically solved to obtain values of the eigenfunction within the small rectangle $\Omega_{i,j}$. On the right-hand side of figure 4.5, all red dots are now computed values and can be used to construct a polynomial interpolant. As a basis, we have chosen the 25 monomials $x^i y^j$ for all $i, j \in \{0, 1, 2, 3, 4\}$.

4.2.3 Numerical experiments

When developing a new method, it needs to be tested thoroughly. This inescapably means that the program should be run on many problems with a wide variety of settings. Doing this methodically and consistently after code changes can be a cumbersome task. Being aware of this, we have from the very first versions onwards included many automatic tests. On every code change, these tests would run on all supported platforms (`Linux`, `Windows` and `macOS`) to verify that our changes still produce sufficiently accurate results for the considered problems. For the development of this new method we continuously test the time-independent two-dimensional Schrödinger problems with the following potential: the harmonic oscillator potential (section 4.2.3.1), the Hénon–Heiles potential (section 4.2.3.3) and the quartic oscillator potential. All these problems are tested on simple rectangular domains. The Schrödinger problem with zero potential (4.2.3.2) is tested on a rectangular as well as a circular domain. The Schrödinger problem with Ixaru's potential is tested on a rectangle, a disc and a 45° rotated rectangle. All tests are conducted in `double`-precision and most problems are also tested in de extended `long double` datatype.

These automated tests are extremely valuable during development to quickly notice regressions, as also mentioned in section 2.6.1.4. But for a thorough mathematical analysis, more work is required. In this section we will provide many graphs and figures to evaluate the accuracy of our method. The evaluation of the runtime performance is another can of worms which will be treated in section 4.2.4.

Our implementation is built in `C++` using `Eigen` [33] for linear algebra with `spectra` [82] as a matrix-free eigenvalue solver, and `Matslise 3.0` (see chapter 2 or [9]) for the computation of one-dimensional basis functions. Installation is possible through `python`'s package system `pip`.

```
pip install strands
```

As a user guide, we provide for all numerical experiments a Python program with which a reader can replicate the experiments on their own computer.

4.2.3.1 Harmonic oscillator potential

Following section 4.1.1.1, the first example we will consider is the harmonic oscillator with equation

$$-\nabla^2\psi(x,y) + \left(x^2+y^2\right)\psi(x,y) = E\psi(x,y) \tag{4.20}$$

on the domain $[-9.5, 9.5] \times [-9.5, 9.5]$ with homogeneous Dirichlet boundary conditions.

Using `Strands`, the first ten eigenvalues of this problem can be found within a second with the following code.

```python
from strands import Schrodinger2D, Rectangle

schrodinger = Schrodinger2D(
    lambda x, y: x*x + y*y, Rectangle(-9.5, 9.5, -9.5, 9.5),
    gridSize=(40, 40), maxBasisSize=30)
print(schrodinger.eigenvalues(10))
```

In our algorithm, there are two main variables to experiment with. The first, most obvious one, is the density of the used grid. Our grid density is similar to the step size that is used by the finite difference approximation of section 4.1.1. In our method, the size of the involved matrices, and thus the runtime, is directly proportional to the total number of grid points. Being able to use a less dense grid would therefore bring a significant computational benefit to the table.

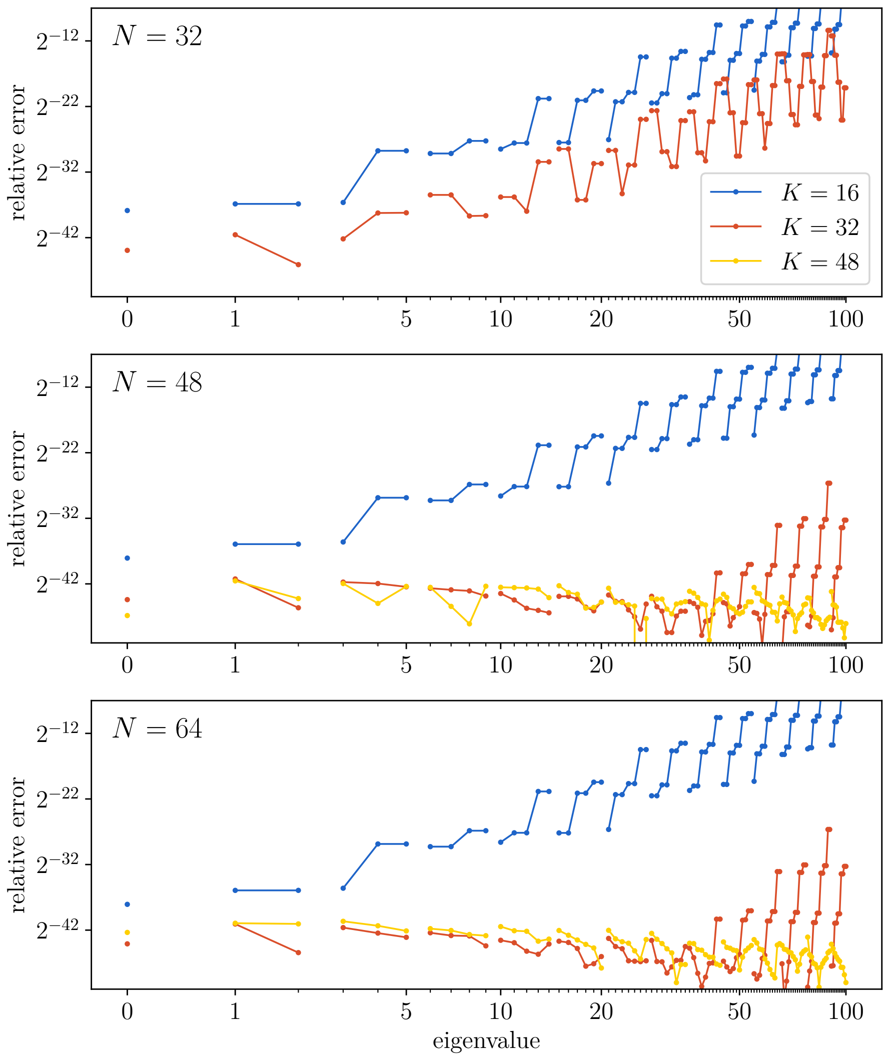


Figure 4.7: These graphs display the relative error of each of the first 100 eigenvalues of the harmonic oscillator (4.20) on a rectangular domain $[-9.5, 9.5] \times [-9.5, 9.5]$, computed for different grid sizes N and basis sizes K. Repeated eigenvalues are connected.

The second parameter we can tweak is the number of basis functions on each of the grid lines. On each grid line, in both directions, `Matslise` needs to compute these basis functions. This computation requires, of course, a part of the runtime.

In a very crude analysis, we can take a look at the computational complexity of the most relevant operations. Assume there are N grid lines in both directions. This yields N^2 grid points, and $2N$ grid lines. On each of these lines we compute $K \leq N$ basis functions, thus the computational overhead of the many one-dimensional problems is $\mathcal{O}(KN)$. For the computation of the eigenvalues of the two-dimensional problem, the analysis is more nuanced. Finding accurate and reliable complexity estimates is more difficult than one would expect. The restarted Arnoldi procedure (without shift-invert), has a complexity of $\mathcal{O}(k^2n)$ when k eigenvalues are requested for a sufficiently sparse $n \times n$ matrix [69]. If the linear operator has a dense operation, this dominates the complexity $\mathcal{O}(kn^2)$. In our case, we support a shift-invert scheme. In this scheme, solving the sparse system is definitely something to take into account. `Eigen`'s built-in methods are based upon techniques from `SuperLU` [26, 71]. In the performance analysis of this software, the authors provide a detailed overview of which properties have an impact on performance. An important property, which contributes to the runtime, is the structure of the matrix. If the non-zero elements of the involved matrix are situated so that the algorithm can exploit it, it helps to reduce the computational cost. It is not known whether our matrix has an exploitable structure or not. It would be surprising if our matrix had the perfect structure, independent of the size and shape of the domain.

In summary, a first complexity analysis of our method indicates that the one-dimensional problems have a complexity of $\mathcal{O}(KN)$ and solving the matrix eigenvalue problem has a complexity of at least $\mathcal{O}(kN^2)$. One has to be wary to only rely on a theoretical analysis to conclude that solving the one-dimensional problems has a negligible computational cost. In section 4.2.4, we will conduct a thorough analysis of the runtime of our method. For now, we will focus on numerical accuracy.

Figure 4.7 contains three graphs illustrating how the two main variables (grid size and basis size) work together to obtain accurate results. For this figure three different basis sizes were considered $K = 16$, $K = 32$ and $K = 48$. As grid sizes we have chosen $N = 32$, $N = 48$ and $N = 64$. Notice that $N = 32$ and $K = 48$ was omitted from the figure as these give exactly the same results as $N = 32$ and $K = 32$, due to the limitation that the number of basis functions can never exceed the number of grid points on the relevant grid line.

When comparing figure 4.7 with figure 4.1, we notice that our new method is able to reach the same accuracy with only a grid size of 48 (with 48 basis functions). This is considerably less than the required grid size of 100 with spacing $h = 0.19$ when using the finite difference scheme. As a direct comparison, our method was solving a $2\,304 \times 2\,304$ sparse eigenvalue problem (from equation 4.16), while the method with finite difference approximation needed to solve a sparse $9\,801 \times 9\,801$ eigenvalue problem (from equation 4.2).

As a second, highly related experiment we consider the harmonic oscillator problem (equation (4.20)) on a circular domain around $(0, 0)$ with radius 9.5. In principle, this is the same problem as before. But in practice, solving Schrödinger problems on non-rectangular domains was not at all possible with other grid-based methods. The following code solves this problem on a circular domain.

```python
from strands import Schrodinger2D, Circle

schrodinger = Schrodinger2D(
    lambda x, y: x * x + y * y, Circle((0, 0), 9.5),
    gridSize=(40, 40), maxBasisSize=30)
print(schrodinger.eigenvalues(10))
```

Within a second, the first ten eigenvalues of this problem are printed, accurate to about 10^{-13}.

Since we have constructed this method with non-rectangular domains in mind, our program is able to tackle these more exotic problems. For this problem, figure 4.8 answers the most important question: are the results accurate? The graphs displayed are calculated in the same way as figure 4.7. Three different grid sizes and three different maximal basis sizes are considered. Notice that we used the term *maximal* basis size. For non-rectangular domains the number of grid points per grid line varies. Grid lines close to the boundary are short chords containing only a few grid points. As the number of points limits the size of the basis, this size has to vary as well. For our algorithm, it is still valuable to place an upper bound on the basis size, especially for those lines that run through the middle of the domain, containing many grid points.

The results of figure 4.8 are eerily similar to those of figure 4.7. For the harmonic oscillator the domain seems to have no effect on the accuracy of the method. This can be explained by the fact that for all eigenfunctions $f(\mathbf{x})$: $\lim_{\|\mathbf{x}\|\to+\infty} f(\mathbf{x}) = 0$. As a consequence, this means that once the domain is sufficiently large the boundary, and corresponding boundary conditions,

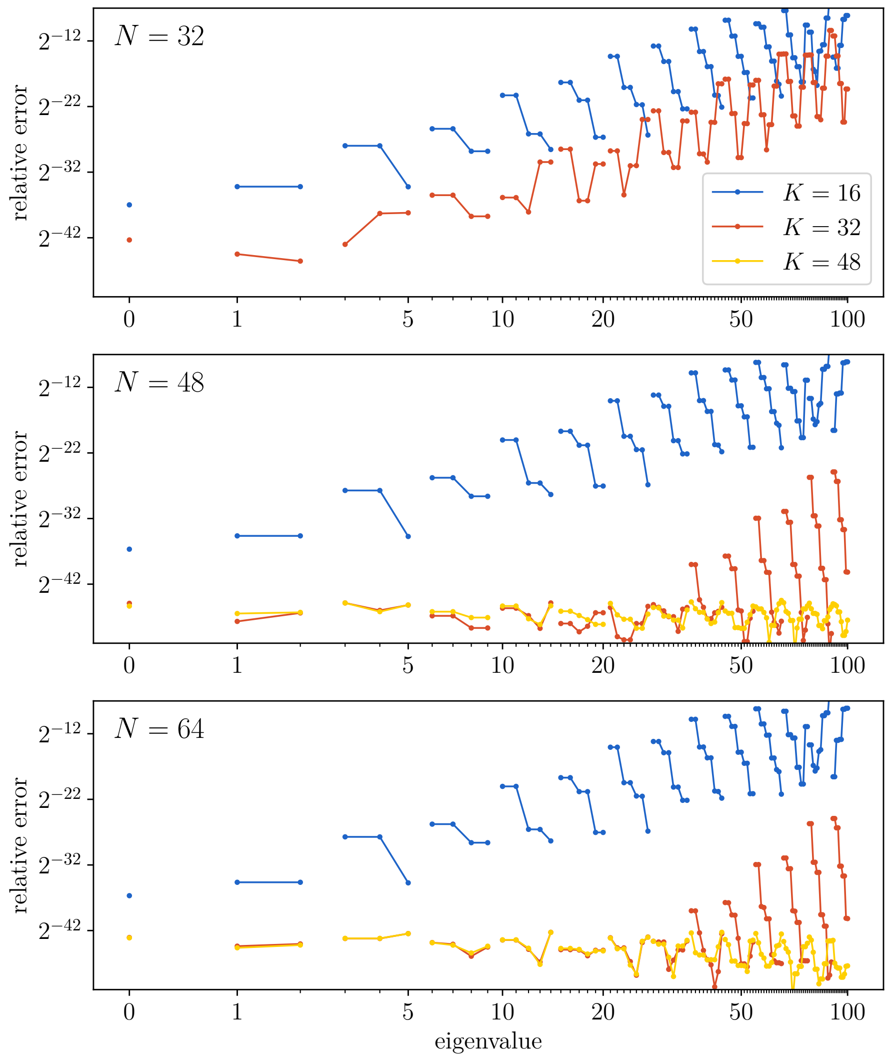


Figure 4.8: These graphs display the relative error of each of the first 100 eigenvalues of the harmonic oscillator (4.20) on the disc around **0** with radius 9.5, computed for different grid sizes N and maximal basis sizes $K \leq N$. Repeated eigenvalues are connected.

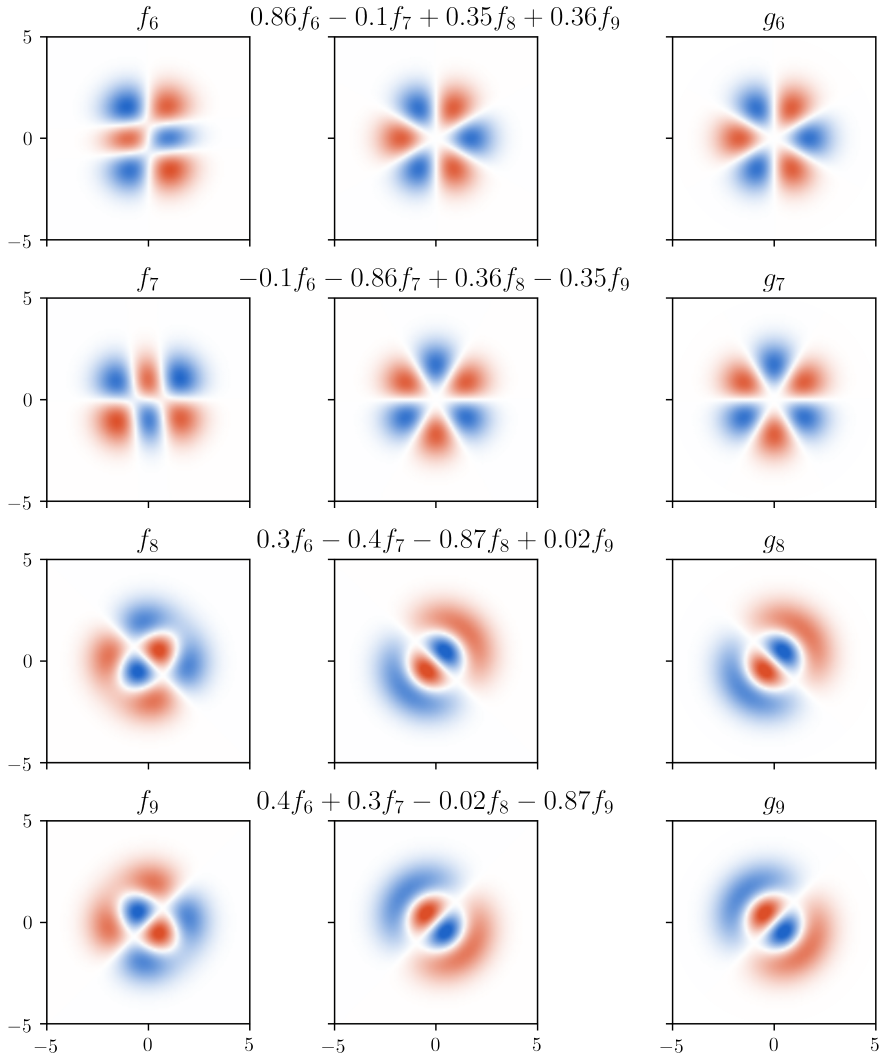


Figure 4.9: Left: four eigenfunctions with the same eigenvalue of the harmonic oscillator on the rectangular domain $[-5, 5]$. Right: the corresponding eigenfunctions on the circular domain with radius 5. Middle: linear combinations of the left eigenfunctions are chosen such that they equal the right eigenfunctions.

become less and less important. One of the great benefits of considering this circular domain is its size. For example, if $N = 64$ the rectangular domain has 4096 grid points. The circular domain on the other hand, only has 3300 grid points. This reduction in grid points makes the involved matrices smaller, and therefore reduces the computational cost.

As a last visual experiment with the harmonic oscillator, we will take a look at the eigenfunctions. As a reminder: the true eigenvalues of the two-dimensional harmonic oscillator on $\mathbb{R}^2$ are:

$$2, 4, 4, 6, 6, 6, 8, 8, 8, 8, 10, 10, 10, 10, 10, 12, \dots$$

`Strands` can be used to find and evaluate the eigenfunctions. The following code yields similar graphs as in figure 4.9.

```python
from strands import Schrodinger2D, Rectangle
import numpy as np
import matplotlib.pyplot as plt

xs = np.linspace(-9.5, 9.5, 200)
ys = np.linspace(-9.5, 9.5, 200)
X, Y = np.meshgrid(xs, ys)

harmonic = Schrodinger2D(
    lambda x, y: x*x + y*y, Rectangle(-9.5, 9.5, -9.5, 9.5),
    gridSize=(40, 40), maxBasisSize=30)

for E, f in harmonic.eigenfunctions(10):
    plt.pcolormesh(X, Y, f(X, Y))
    plt.show()
```

Figure 4.9 plots a basis for all eigenfunctions corresponding to eigenvalue $\lambda_6 = \lambda_7 = \lambda_8 = \lambda_9 = 8$. In the left-most column, the basis for the eigenfunctions on the square domain $[-5, 5] \times [-5, 5]$ can be seen, and in the right-most column the basis found on a circular domain around zero with radius 5. We have opted here for a smaller domain, not because of computational cost (on the larger domain these eigenfunctions can be drawn just as quickly), but for visual usefulness. On a larger domain, the interesting center part of the image would be a lot smaller and thus less visible.

Upon a first viewing of these eigenfunctions (left and right columns), one may be surprised that these are different. But, as the eigenspace corresponding to eigenvalue 8 is multidimensional, the basis is no longer unique. To be able to compare these results we have introduced the middle column. Here, a linear combination of basis functions from the rectangular domain is calculated such that we obtain the same basis functions as found on the circular domain. This illustrates some of the challenges when trying to compare or numerically verify eigenfunctions.

4.2.3.2 Zero potential

Analogous to section 4.1.1.1, the next example we will study is the Schrödinger equation with a zero potential. Here the equation simplifies to:

$$-\nabla^2\phi(x,y) = \lambda\phi(x,y)$$

on the domain Ω, with homogeneous Dirichlet boundary conditions.

If the domain Ω is the rectangle $[0,\pi]\times[0,\pi]$, exact solutions can be obtained with separation of variables. This gives $\forall i,j \in \mathbb{N}^+$ that i^2+j^2 is an eigenvalue of this Schrödinger problem with corresponding eigenfunction $\phi(x,y)=\sin(ix)\sin(jy)$.

Not much has to be changed from the previous code samples to solve this problem.

```python
from strands import Schrodinger2D, Rectangle
from math import pi

schrodinger = Schrodinger2D(
    lambda x, y: 0, Rectangle(0, pi, 0, pi),
    gridSize=(32, 32), maxBasisSize=32)
print(schrodinger.eigenvalues(10))
```

In figure 4.10, the results for the Schrödinger problem with zero potential are visualized. Unsurprisingly, our method works extremely well for this. The main reason can be found in the fact that the basis functions `Matslise` calculates on each line are exactly $\sin(x)$, $\sin(2x)$, $\sin(3x)$... and analogous for y. These functions are able to exactly represent the true eigenfunctions $\sin(ix)\sin(jy)$ on each line.

For non-rectangular domains with the zero potential, the story is different. Let us consider the Schrödinger equation with the zero potential and the circular domain around $(0,0)$ with radius 1 and homogeneous Dirichlet boundary

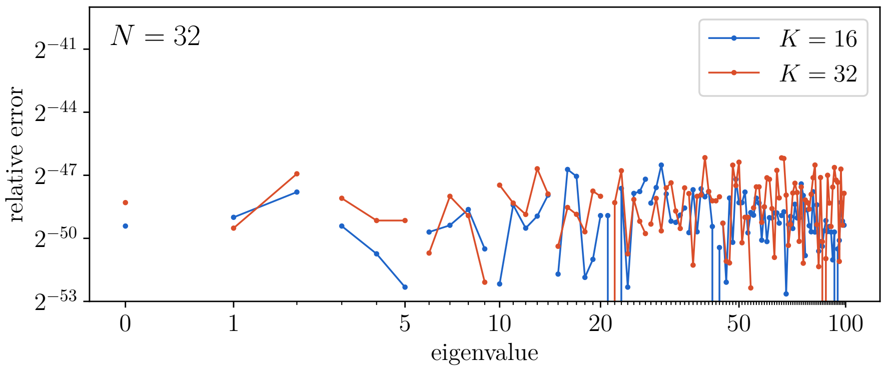


Figure 4.10: This graph displays the results of our new method for the zero potential on the square $[0, \pi] \times [0, \pi]$, for a fixed grid size of $N = 32$ and a varying basis size. Notice the zoomed-in y-axis, close to the machine precision of 2^{-53}.

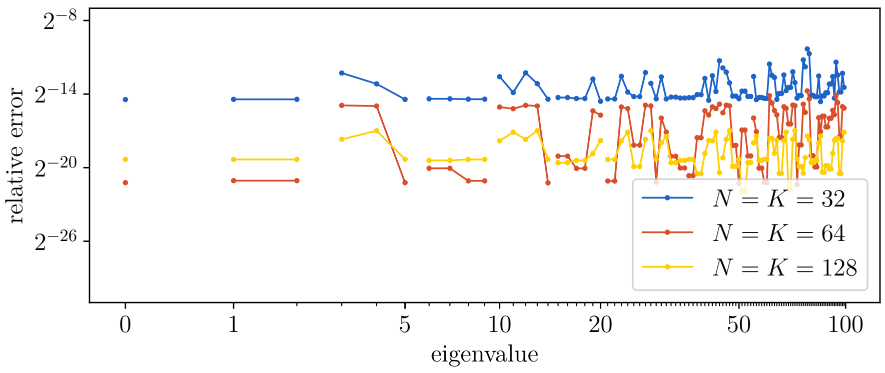


Figure 4.11: This graph displays the results of our new method for the zero potential on the circular domain around $(0, 0)$ with radius 5, for varying grid and basis sizes. Notice the shifted y-axis, to larger errors.

conditions. The exact eigenvalues are known as the squares of the roots of the Bessel functions. See section 3.3.2.2 for the full symbolic calculation.

The relative errors of the first hundred eigenvalues found with our new method are shown in figure 4.11. We have tested this with the grid size and maximal basis size both equal to 32, 64 or 128. The results are less impressive than for rectangular domains as the maximal accuracy reached is somewhere around 10^{-6}. But nevertheless, being able to compute these values at all is quite unique.

The behavior we observe in figure 4.11 is concerning. The method seems to be unable to further improve accuracy, even if parameters which should result in more accurate approximations are used. The eigenvalues found with $N = K = 128$ are not consistently more accurate than the values found with $N = K = 64$.

Determining the source of these inaccuracies is difficult. The graphs are reminiscent of those in figure 4.2 where we used a high order finite difference scheme. There, the high order scheme (wrongly) assumed the eigenfunctions to be identically zero outside the domain. In our case, no such assumption is explicitly made. Implicitly however, for grid-lines close to the boundary of the disc, only a few basis functions are available. On these lines, it is thus impossible to accurately represent the sought eigenfunctions. We suspect this to introduce unwanted boundary effects. It would explain the similarities between figure 4.11 and figure 4.2.

4.2.3.3 Hénon–Heiles potential

Another often used test problem is the Hénon–Heiles system. In the context of time-independent two-dimensional Schrödinger equations the potential in question is given as:

$$V(x, y) = x^2 + y^2 + \frac{\sqrt{5}}{30}(3yx^2 - y^3). \tag{4.21}$$

We refer back to section 3.4.4, where we also considered this potential.

In [102], the authors remark to take care when choosing the size of the domain. This should not be too small to allow the eigenfunctions to converge to zero on the boundary. But it should neither be too large, because this can generate some physically nonsensical eigenvalues. In [102], the authors simply use the domain $[-10, 10] \times [-10, 10]$ and leave it at that. Looking at the available literature about this problem, we find for example [24]. Here, the proposed

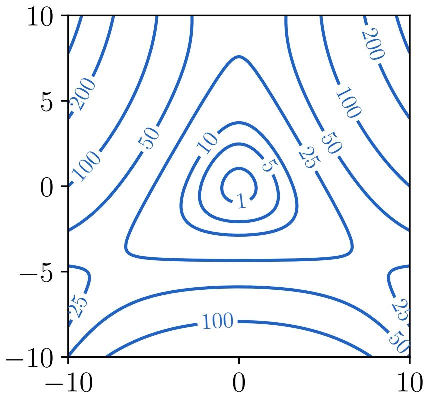


Figure 4.12: A contour plot of the potential of the Hénon–Heiles problem as in equation (4.21).

method does not use a clear defined boundary, but uses basis functions with infinite support. However, the interesting regions of these basis functions are only present in a radius much smaller than 10 around zero. In [18] the grid is limited to a $[-6, 6] \times [-6, 6]$ domain, and in [8], the authors follow [18]. For this experiment we will also use a $[-10, 10] \times [-10, 10]$ domain. A contour plot of this potential can be seen in figure 4.12.

The code for solving this problem should be familiar by now.

```
from strands import Schrodinger2D, Rectangle
from math import sqrt

def V(x, y):
    return x*x + y*y + sqrt(5)/30 * y * (3*x*x - y*y)

henon = Schrodinger2D(V, Rectangle(-10, 10, -10, 10),
                      gridSize=(48, 48), maxBasisSize=32)
print(henon.eigenvalues(10))
```

In figure 4.13, the relative errors obtained with our method our displayed for this experiment. We used the results from [102] as reference. Before analyzing these results, we need to mention that for $N = K = 32$ and $N = K = 48$, our method found many nonsensical small eigenvalues. We know that they have no physical meaning because the corresponding eigenfunctions are only non-zero inside the valleys at the boundary. These unwanted eigenvalues would disappear if the domain were a little smaller.

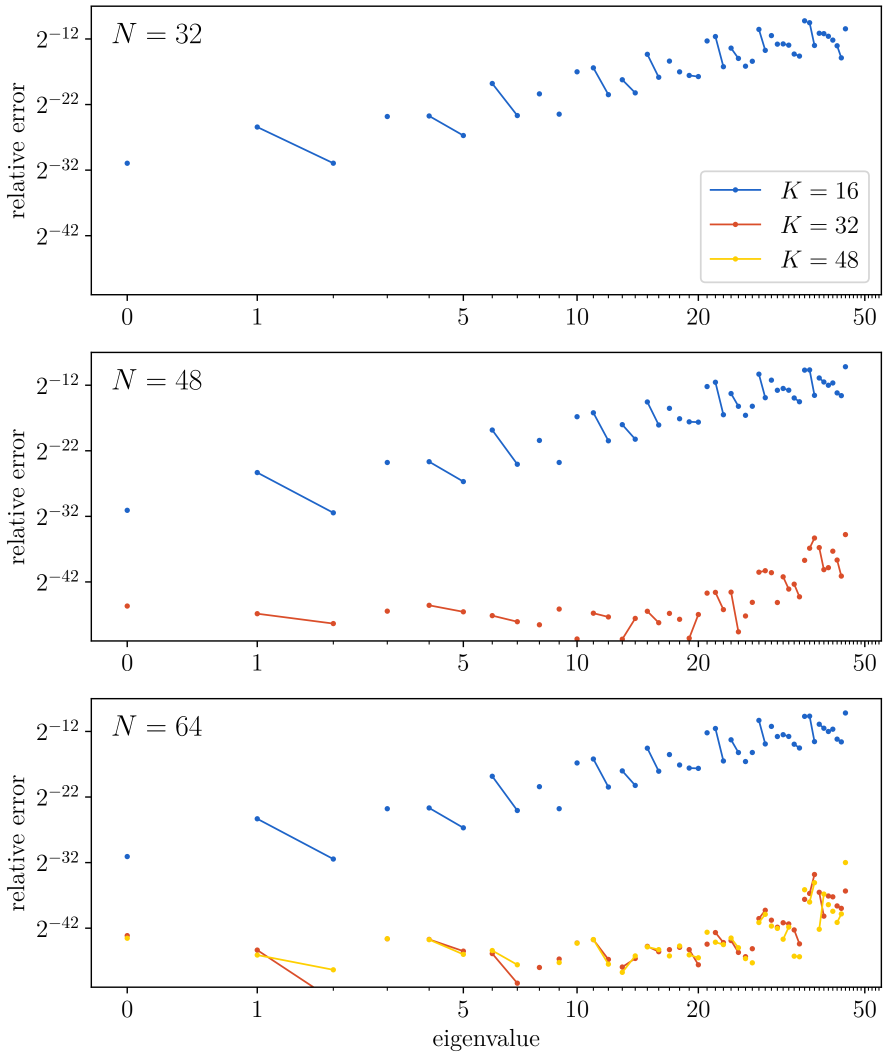


Figure 4.13: These graphs display the relative errors when we compare the results from our method with those from [102] for the Hénon–Heiles system on the domain $[-10, 10] \times [-10, 10]$. For the parameters $N = K = 32$ and $N = K = 48$, our method found some physically nonsensical eigenvalues and therefore are missing from the plots.

Ignoring these special cases allows us to interpret the graphs from figure 4.13. Even on a small grid $N = 48$ we are able to accurately determine the eigenvalues. We also see that for the high eigenvalues increasing the grid size or basis size seems to have no effect. This is unexpected and seems to indicate that maybe the reference values are not perfect. The relative error for these high eigenvalues is $< 2^{-32} \approx 10^{-10}$, which is nonetheless extremely accurate.

4.2.3.4 Ixaru's potential

Next, we will consider the following Schrödinger problem

$$-\nabla\psi + (1 + x^2)(1 + y^2)\psi = E\psi$$

on $[-5.5, 5.5] \times [-5.5, 5.5]$ with homogeneous Dirichlet boundary conditions, as in [47] and section 3.4.3.

The eigenvalues can be approximated with the following code. This program runs in less than a second. The resulting approximations agree up to all digits reported in [47].

```python
from strands import Schrodinger2D, Rectangle

def V(x, y):
    return (x**2 + 1) * (y**2 + 1)

ixaru = Schrodinger2D(V, Rectangle(-5.5, 5.5, -5.5, 5.5),
                      gridSize=(40, 40), maxBasisSize=30)
print(ixaru.eigenvalues(13))
```

In figure 4.14, we study the relative error of the first 25 eigenvalues. Since no sufficiently accurate eigenvalues are available, we have to calculate the reference values ourselves. For this, we use `Strands` with $N = 90$ and $K = 40$. This computation takes a little less than one minute. We provide these reference eigenvalues in table 4.1.

The interpretation of the relative errors from figure 4.14 is similar to previous discussions. Our method is able to reach close to machine precision, with relatively little computational work. The used grids don't need to be dense to obtain these results. Visually, we notice that the relative error for the higher eigenvalues with $N = 50$ and $K = 35$ is close to $10^{-9} \approx 2^{-30}$.

As a last example program, we compute three eigenfunctions corresponding to the highest computed eigenvalues E_{23}, E_{24} and E_{25}. The graphs for these

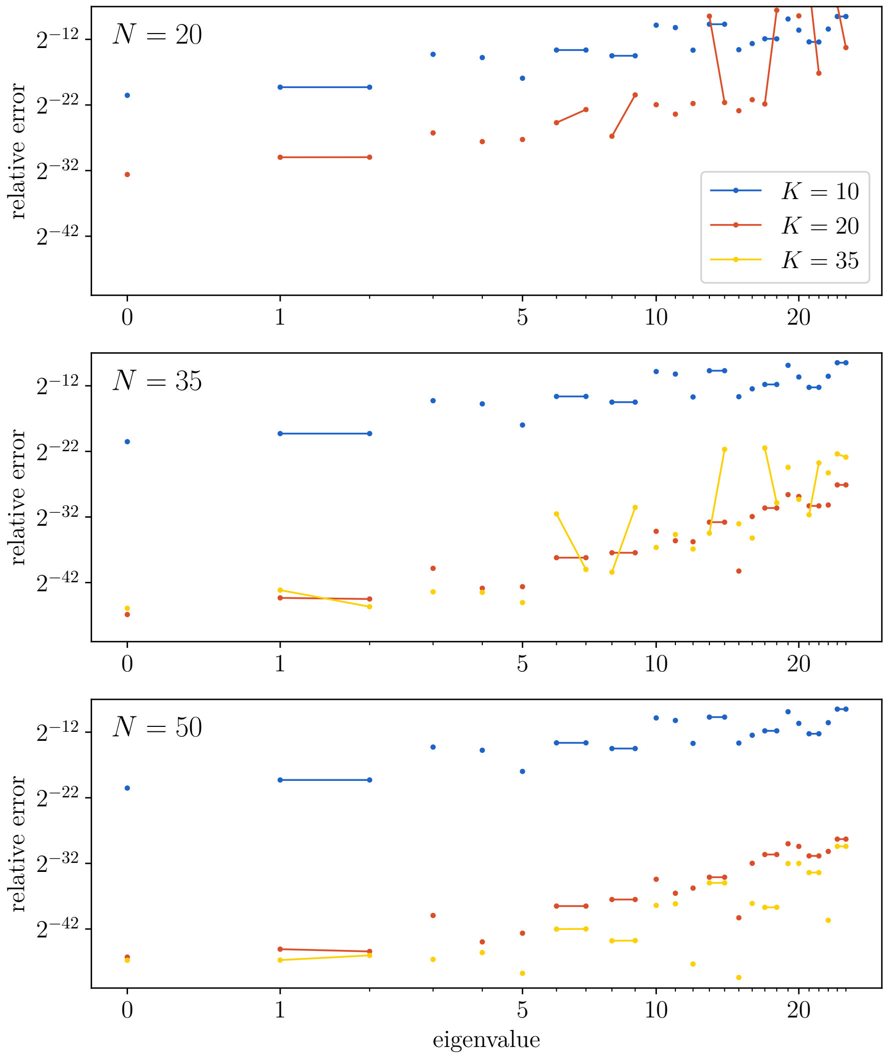


Figure 4.14: These graphs display the relative errors for Ixaru's problem in section 4.2.3.4 for the first 25 eigenvalues. For $N = 20$, the results for $K = 35$ are omitted, as these are exactly the same as for $K = 20$.

$E_0 = 3.195\,918\,085\,200$	$E_{12} = 13.332\,331\,271\,155$
$E_1 = E_2 = 5.526\,743\,874\,395$	$E_{13} = E_{14} = 14.348\,268\,533\,253$
$E_3 = 7.557\,803\,326\,786$	$E_{15} = 14.450\,478\,721\,981$
$E_4 = 8.031\,272\,340\,314$	$E_{16} = 14.580\,556\,315\,644$
$E_5 = 8.444\,581\,361\,570$	$E_{17} = E_{18} = 16.151\,419\,224\,568$
$E_6 = E_7 = 9.928\,061\,056\,952$	$E_{19} = 16.517\,192\,463\,374$
$E_8 = 11.311\,817\,050\,618$	$E_{20} = 16.564\,871\,925\,909$
$E_9 = 11.311\,817\,050\,619$	$E_{21} = E_{22} = 17.894\,578\,279\,407$
$E_{10} = 12.103\,253\,578\,719$	$E_{23} = 18.583\,391\,734\,468$
$E_{11} = 12.201\,178\,967\,971$	$E_{24} = E_{25} = 18.756\,204\,273\,611$

Table 4.1: Reference eigenvalues of the problem from section 4.2.3.4, computed with $N = 90$ and $K = 40$.

eigenfunctions can be found in figure 4.15. We notice the clear oscillatory behavior of these functions. To generate similar graphs yourself the following code can be used.

```python
import numpy as np
import matplotlib.pyplot as plt

xs = np.linspace(-5.5, 5.5, 200)
ys = np.linspace(-5.5, 5.5, 200)
X, Y = np.meshgrid(xs, ys)

for E, f in ixaru.eigenfunctions(26)[23:]:
    plt.pcolormesh(X, Y, f(X, Y))
    plt.show()
```

4.2.4 Runtime analysis

Throughout the development of this new method, we have focused upon keeping the matrices small. The idea is that finding eigenvalues of a (sparse) matrix is quite expensive, definitely more expensive than all other steps. But it would be careless to take this assumption at face value without more thorough analysis. In section 4.2.3, we have already seen that our method can reach the same accuracy as a finite difference scheme with a smaller sparse matrix in the end. So, if all steps to construct the sparse problem are almost negligible in comparison to finding eigenvalues of that sparse system, then our method will be faster for the same accuracies.

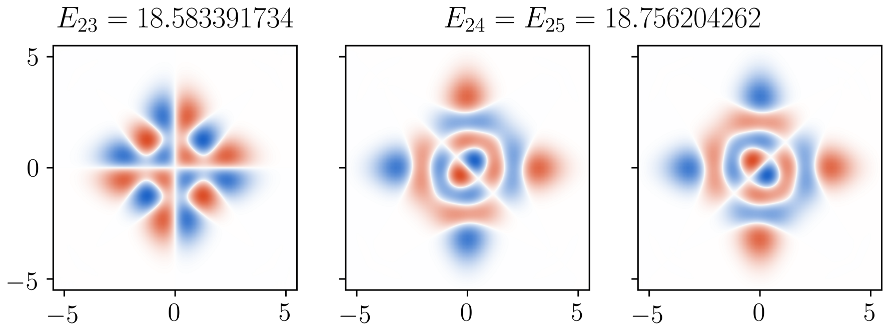


Figure 4.15: Eigenfunctions corresponding to the eigenvalues E_{23}, E_{24} and E_{25}. As $E_{24} = E_{25}$, any linear combination of the two rightmost eigenfunctions is an eigenfunction as well.

As an experiment, we will take a look at the Hénon–Heiles problem from section 4.2.3.3 on the square domain $[-6, 6] \times [-6, 6]$ on a 64×64 grid and a maximum basis size of 48. We have run a profiler on our program to solve this problem. Our program consists of two main parts. First, the grid is constructed and the basis functions on each grid line are computed. Second, we employ `spectra` to find the eigenvalues of the resulting sparse matrix. For the second part, we have two options. We can find the largest eigenvalues of the shifted inverse matrix. Or, with the implicitly restarted Arnoldi method, we can select only the smallest eigenvalues while converging. The first technique has much better convergence, but is more expensive because in each step a linear system has to be solved. For many problems the second technique also converges to the true smallest eigenvalues, but the convergence is less reliable and slower. Yet, the final runtime may be faster because no system of equations has to be solved.

To evaluate these differences, we have solved this problem 20 times with each of the techniques (selecting the smallest eigenvalues versus using the shifted inverse matrix). With the first technique, our program had an average total runtime of 3.295 s and with the second technique the total runtime rose to 8.700 s. Using only these numbers to draw conclusions may be a little premature.

In both cases, the construction of the grid and the determination of the basis functions are identical. In the following table we have summarized the averaged results from the profiler. All reported times are the average of 20 runs, with no deviation of more than 5% between runs.

	calls	runtime
construct grid and solve one-dimensional problems	1	531 ms
finding a basis on a single grid line	128	436 ms
evaluate each eigenfunction in all grid points	6 144	89 ms

The 'calls' column contains the total number of times that function was executed. The 'runtime' column contains the total time that was spent inside this function. So the construction of the grid took 531 ms and was executed once (per run). On each grid line, the basis functions had to be found. Because we have chosen a 64×64 grid, the routine to find this basis was called $128 = 64 + 64$ times. In this routine our program constructs a `Matslise` object to find the first 48 eigenvalues. For each of these eigenvalues the eigenfunction is also calculated, but not yet evaluated. The evaluation of these functions is done a few moments later when constructing the relevant $\mathbf{B_x}$ and $\mathbf{B_y}$ matrices. A total of $6\,144 = 48 \cdot 128$ eigenfunctions were evaluated with a total runtime of 89 ms. The other steps within this construction (such as determining the boundaries of the domain, allocating matrices or keeping track of grid points) only took 6 ms on average. So this table contains the most expensive functions in the construction of the problem.

To compute the eigenvalues themselves, we will first study the technique of the implicitly restarted Arnoldi iteration with selection of the lowest 100 eigenvalues. For this Hénon–Heiles problem, the correct results are found. However, we expect that the convergence may be slow. The following table contains the profiler's summary.

	calls	runtime
computing eigenvalues	1	2 772 ms
SPECTRA compute eigenvalues	1	2 656 ms
perform operation	1 316	1 151 ms

We see that the computation for this part took 2.772 s. Together with the construction, this gives a total runtime of 3.295 s. We also see that `spectra` called our matrix-free operation procedure 1 316 times. The computation of this procedure took less than half the total time, the other time is spent inside the implicitly restarted Arnoldi iteration algorithm of `spectra`.

Let us also consider the following table, where the summary of the profiler is displayed when the eigenvalue algorithm is applied to the shifted inverse matrix. This program uses the pseudocode from algorithm 1.

	calls	runtime
computing eigenvalues (shiftInvert)	1	8 289 ms
sparse LU-decomposition	1	3 731 ms
SPECTRA compute eigenvalues	1	4 529 ms
perform operation	353	4 166 ms

Here we see that the convergence is faster, it is to say, fewer operations are needed. Although, the runtime of 8.289 s is much higher. The cause is twofold. First we see that before calling the eigenvalue routine, we have to compute the sparse LU-decomposition of the involved matrix. This alone takes more time than the computation without inverse matrix. Even though almost four times fewer operations are needed to reach convergence, each operation is almost fourteen times slower. This negates all benefits of the 'faster' convergence.

The main take-away from these tables is that the construction time is dwarfed by the time needed to solve the matrix eigenvalue problem, regardless of the method (with or without shift-invert mode) used. Though, only looking at fixed parameters may be misleading. In figure 4.16 we have plotted the true runtime of the construction time and the computation of the eigenvalues (with both techniques) for different parameters $N \in \{32, 48, 64, 80, 96, 112\}$ and $K = \frac{3}{4}N$.

In this figure we see that the construction of the grid (with solving all one-dimensional problems) has an experimental time complexity of $\mathcal{O}(N^2)$. This is in line with our expectation. We have $2N$ one-dimensional problems to solve, and for each of these problems $\frac{3}{4}N$ eigenvalues are requested. Now, in principle we have a $\mathcal{O}(N^3)$ runtime complexity, because we will evaluate each of the $\frac{3}{2}N^2$ eigenfunctions in N points. However, as seen from the detailed profiler results from before, this evaluation is quite efficient and as such, this theoretic cubic complexity will not be present in practice.

When we are using the shifted inverse matrix to find eigenvalues, we can see an experimental runtime of $\mathcal{O}(N^5)$. A dense LU-decomposition has a cubic complexity in the size of the involved matrix. In our case a sparse LU-decomposition is calculated of the $N^2 \times N^2$ sparse matrix. This decomposition is able to exploit some sparsity and as such, we hope (and expect) a speed-up. Indeed, this is what we see with the measured $\mathcal{O}(N^5)$ complexity.

When the eigenvalues are found by selecting the smallest values during convergence, a theoretical analysis is more difficult. In a single step, our matrix-free operation computes deflated vectors and a matrix-vector product. The former is computed with sparse basis vectors, the latter is computed with a sparse matrix. Let us focus on the main matrix-vector product. This matrix is given in

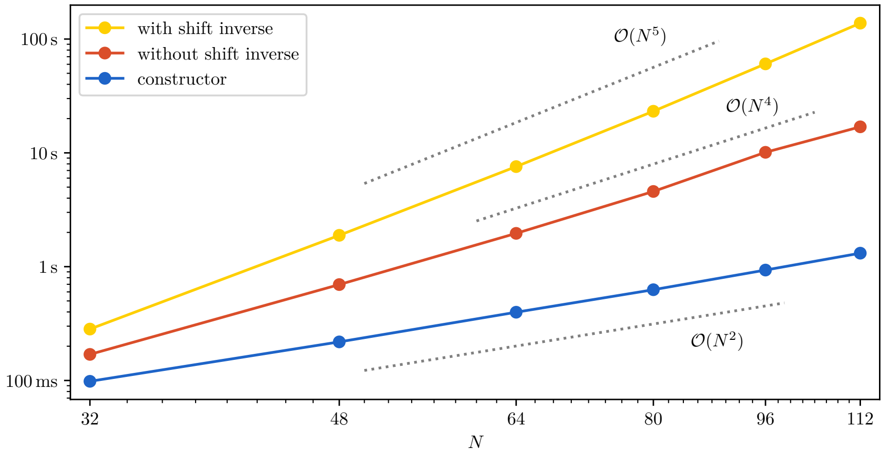


Figure 4.16: The Hénon–Heiles problem on a $[-6, 6] \times [-6, 6]$ square domain is solved for parameters $N \in \{32, 48, 64, 80, 96, 112\}$ and $K = \frac{3}{4}N$. The average runtime (over five runs) of the construction of the grid (with solving all one-dimensional subproblems) and of the computation of the eigenvalues with and without inverting the matrix are reported.

the right-hand side of equation (4.16). The first term is a block diagonal-matrix with N times a $K \times K$-block. The second term has a similar but permuted structure. Therefore, each row contains at most $2K - 1$ non-zero entries, for a total of $\mathcal{O}(NK^2) = \mathcal{O}(N^3)$ non-zero entries. The bases of the deflation spaces are similarly sparse. For determining the total time complexity, 'only' an estimate for the number of operations is needed. However, in most analyses of eigenvalue solvers, it is assumed that the outer boundary of the spectrum is selected in each step, not the smallest values. So finding reliable estimates is quite difficult. Using the experiment from figure 4.16, we can suspect that the number of steps before convergence is $\mathcal{O}(N)$, which is the square root of the matrix size.

In any case, our focus on reducing the matrix-size has been worthwhile. The main bottleneck is still the computation of the eigenvalues for which many well-tested extremely-efficient and even parallelly distributed algorithms exist. In comparison to [102], our method is able to reach the same or higher accuracies (as discussed in section 4.2.3) with smaller grid sizes, and thus matrix sizes, which in turn reduces the total runtime.

4.2.5 The name Strands

When developing and publishing a new method maybe one of the most difficult challenges (besides developing, implementing, testing and analyzing the method) is probably finding a fitting name. As mathematicians, we may sometimes forget the importance of a recognizable name. So, let us take a look at some names we encounter in the area of Schrödinger equations. For the one-dimensional method we have for example `SLEDGE`, `SLEIGN`, `SLCPM12` and `Matslise` itself. For the system of coupled equations we have `LILIX` or `MatSCS`. We have not found any packages with memorable names for the specific two-dimensional problem. If we look a bit further to the libraries we used, then we encounter: `Eigen` (`C++`-library for linear algebra), `spectra` (`C++`-library for finding eigenvalues), `PETSc` and `SLEPc` (scalable `C`-libraries for linear algebra). We also came across `NumPy` and `SciPy` as numerical libraries in `python`. Or we can look at the wider scientific computing community for inspiration: `BLAS` and `LAPACK` (interfaces for basic linear algebra), `Armadillo` (`C++`-library for linear algebra), `GNU GSL` (scientific computing in `C`), `SageMath` (a computer algebra system), `Maple` (program for symbolic computation), `FEniCS` with `DOLPHIN` (platform to solve partial differential equations), `OpenFOAM` (for computational fluid dynamics),

A name should be short and be more or less relevant to the method. The names I like the most are the pronounceable fun names (with bonus points if it is a backronym[7]) like `LILIX`, `Spectra`, `Sage` and `FEniCS`.

When first developing the ideas for this method, I always saw it as a loom, with warp and weft threads. The tension ensures that in the grid points these threads intersect. There are many fun words in the weaving-industry for which a backronym could be found, but these are always quite niche and not well-known. After a short conversation with ChatGPT, it proposed 'Stranded Schrödinger Solver' as name. And I liked the word *strand* a lot. It is synonymous with *thread* and *wire*, so it fits definitely with my mental model of the method. But, *strand* is also the Dutch word for *beach*, which brings the 'fun' aspect back to the table. And as our method uses many strands together, this gives us:

`Strands`
: **St**anding waves app**r**oxim**a**tions for the ***n*-d**imensional **S**chrödinger problem (with $n = 2$ and maybe in the future $n = 3$).

4.3 Conclusion

In this chapter we have developed and implemented Strands to approximate eigenvalues and eigenfunctions of the two-dimensional time-independent Schrödinger equation. We drew inspiration from a method solely based upon finite difference approximations. This inspiring method was able to reach extremely accurate results with relatively little computational effort. Our new method was able to reach the same accuracies. In fact, these extremely accurate results were achieved, while using smaller matrix problems. This resulted in an overall significantly faster algorithm.

In this new method, we have strived to avoid or fix some issues with the method from chapter 3. We believe our method to be easier to implement than Ixaru's two-dimensional method. In the numerical experiments, Strands achieved more accurate results with similar computational cost. However, comparing these methods directly is difficult. In chapter 3, we have studied a shooting method which is able to target specific eigenvalues. On the other hand, our method from section 4.2 translates the problem into a direct matrix eigenproblem, similarly to the method from section 4.1.1.

In summary, this new method shows a lot of promise. The numerical experiments illustrate that Strands can reach extreme accuracies with relatively little

[7]Wikipedia tells us: "a backronym is an acronym formed from an already existing word". For example: `Spectra` stands for "*Sp*arse *E*igenvalue *C*omputation *T*oolkit as a *R*edesigned *A*RPACK."

computational effort. Still, we see many ways to further improve upon this work. Non-rectangular domains definitely need more research: our method was able to approximate eigenvalues, however the accuracy was significantly reduced. How we could evaluate eigenfunctions close to the boundary of those non-rectangular domains is also an open question.

The spurious small eigenvalues we found for some parameters in the Hénon-Heiles problem may cause some concern. When trying out our method on other problems, for some sets of parameters similar small wrong values are found. We are not yet certain what causes this phenomenon.

We believe the study of this method to be valuable future work. It combines the strengths from Ixaru's method from chapter 3, and the finite difference scheme discussed in section 4.1.1.

Closing remarks

Looking back at this thesis, and the researching years preceding it, fills me with many emotions: both positive, as well as less positive. First and foremost, I am quite proud of the results we were able to achieve. In chapter 2, we have started from a well-established and thoroughly studied technique for one-dimensional Schrödinger equations, and we improved upon known results. Both theoretical and practical advances were made. In chapter 3, we have taken a relatively new technique and build upon it. Our implementation advances the use of this technique with some new features. For these, we took some theoretical strides. One of the results I am most proud of in this thesis is theorem 3.10. Five years ago, when I implemented this new technique for the first time in MATLAB, I was disappointed that it was impossible to ensure one has found all eigenvalues in a given range. In my master's thesis I have voiced this missing feature as a possible idea for future work. Privately, my promotor and I were rather pessimistic if this issue could ever be fixed. We believed that this method was too complicated to be able to give any guarantees on the index of eigenvalues. Despite this pessimism, I am proud that we have persevered and developed this new theorem 3.10.

During the research for chapters 2 and 3, at times, I was quite demotivated. Although we were advancing and building improvements, both techniques were not *mine*. I had difficulties finding ownership. The real low-point in my PhD-research was in August 2020. I was browsing through the literature and found an article which was able to solve the two-dimensional Schrödinger equations faster and more accurate than we ever could. Unavoidably, many existential questions were raised. Getting through this *crisis of faith* was definitely not easy. It took some time, but a few long months later, I found renewed inspiration and *goesting*[8]. In my archives, the earliest draft I can

[8]This is a very Flemish word. It loosely translates to 'with an enthusiastic motivation'.

find for ideas for a new method for two-dimensional Schrödinger equations dates from February 2021. And in the subsequent years I have learned that, maybe unsurprisingly, developing a new method is hard. We went through many iterations. In chapter 4, you can read the (for now) final version.

Looking forward

Writing about what the future will bring is, what we would call in Dutch, looking at coffee grounds[9]. Without sounding overly ambitious: I hope our new method and our implementation may be useful for someone somewhere.

As all things in life, our new method is not perfect. One of the more user-friendly features of our improvements to the method from chapter 3, is the ability to automatically select the needed sector sizes to ensure a requested accuracy. In chapter 4, the grid size is still an open parameter. The relation of this size to the numerical accuracy is not yet thoroughly researched. Also, in section 4.2, we were adamant that our new technique works on non-uniform grids with varying basis sizes on each grid line. Yet, we studied this only to a limited extent. Ideally, some automatic grid selection should be implemented. In vague terms, on regions of the domain where the sought eigenfunctions are interesting, the used grid should be denser. Implementing a heuristic for this may be straightforward, but being able to guarantee that a given accuracy is obtained with a particular grid is difficult. Another improvement to this technique may be found in the choice of basis functions on grid lines. On the line $x = x_i$, in chapter 4, we solve a one-dimensional Schrödinger problem with potential $\frac{1}{2}V(x_i, y)$. In each grid point, the value of the potential is split between both intersecting grid lines. In principle, this split does not need to be equal. Maybe other choices can be defended as well.

Besides direct improvements to the method itself, we believe more fundamentally different ideas can be explored. Does the grid need to be square? How can this method be used to solve time-dependent Schrödinger equations[10]? Can this technique be extended to general linear operators? What changes when solving three-dimensional problems?

Concerning the three-dimensional problem, we have studied some preparatory ideas. Following the two-dimensional method, we place a rectangular grid on the domain, with three grid lines per grid point. As a basis, we could

[9]In English, more commonly: reading tea leaves.

[10]For one-dimensional time-dependent problems, `Matslise 2.0` is used in [62].

use eigenfunctions of one-dimensional Schrödinger problems with a three-way split of the potential. For example, on the line $x = x_i \wedge y = y_j$ we solve the one-dimensional Schrödinger problem with potential $\frac{1}{3}V(x_i, y_j, z)$. To find the eigenvalues we have to solve (analogous to (4.12))

$$\mathbf{B_x \Lambda_x c_x} + \mathbf{B_y \Lambda_y c_y} + \mathbf{B_z \Lambda_z c_z} = E\mathbf{B_x c_x} = E\mathbf{B_y c_y} = E\mathbf{B_z c_z}.$$

The technique to solve this equation, as described in section 4.2.1.3, generalizes elegantly for more dimensions. In summary, approximating eigenvalues is quite straightforward. The largest issues we have found in a first exploratory study is the computation of an eigenfunction in arbitrary points. No longer can we use the same trick to solve a much smaller Schrödinger problem with calculated boundary conditions. Eigenfunction solutions are only known upon the grid lines, not on the planer faces of each of the grid cells.

On a more personal note, the researching for and the writing of this thesis has taught me a great deal. Some lessons were a confirmation of things I already knew. For example, I really like solving problems. When a new challenge crosses my path, far too often I think: "It could not be *that* hard, right?" And, after spending anything between a few hours and a few years trying to solve the problem, I have to conclude: "Well, it definitely is that hard."

One of the other lessons I learned about myself is that I like collaborating. In researching this thesis, it was almost always only my promotor and I. Now, I am very fortunate to get along really well[11] with my promotor. Yet, many times I felt that I was missing some more people to collaborate with, to bounce ideas around with, to just talk with...

The last lesson I want to share is that I easily get distracted with interesting problems. During the past five years, far too many times I was very busy not doing my PhD-research. All these distractions are out of the scope of this text. However, if you, the reader, want to talk with me, and don't know what about, then just ask me something about: mathematical origami, plagiarism detection in source code, sunlight on crop fields for agroforestry, basins of convergence and Julia sets, ray tracing in mathematical figures, hyperbolic geometry (in VR), the roots of the Littlewood polynomials, picking numbers in a lottery, dynamical systems in arbitrary precision, geodesics and shortest paths over surfaces, computations in non-commutative quantum algebras, Keith numbers, drawing L-systems...

[11]Of course, I hope (and do believe) that this is symmetric.

As the last paragraph, I want to thank Marnix once again to be my guide during this research, to provide invaluable much appreciated feedback and to give me the freedom to pursue my distractions. But also again, I want to thank Emilie for being my best friend, for listening to all my troubles and for carrying my burdens with me.

Toon Baeyens

April 2023

Bibliography

[1] R. A. Adams and J. J. F. Fournier. *Sobolev Spaces.* Elsevier, 2003. 321 pp.

[2] E. Anderson et al. *LAPACK users' guide.* 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[3] A. L. Andrew. "Correction of finite difference eigenvalues of periodic Sturm-Liouville problems". In: *The anziam journal* 30.4 (1989), pp. 460–469.
DOI: 10.1017/S0334270000006391.

[4] A. L. Andrew and J. W. Paine. "Correction of Numerov's eigenvalue estimates". In: *Numerische mathematik* 47.2 (1985), pp. 289–300.
DOI: 10.1007/BF01389712.

[5] T. Archibald, C. Fraser, and I. Grattan-Guinness. "The History of Differential Equations, 1670–1950". In: *Oberwolfach reports* 1.4 (2005), pp. 2729–2794.
DOI: 10.4171/owr/2004/51.

[6] W. E. Arnoldi. "The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem". In: *Quarterly of applied mathematics* 9.1 (1951), pp. 17–29.

[7] N. H. Asmar. *Partial Differential Equations with Fourier Series and Boundary Value Problems.* Pearson Education, 2005. 690 pp.

[8] T. Baeyens and M. Van Daele. "Improvements to the computation of eigenvalues and eigenfunctions of two-dimensional Schrödinger equations by constant perturbation based algorithms". In: *Journal of computational and applied mathematics* (2022), p. 114292.
DOI: 10.1016/j.cam.2022.114292.

[9] T. Baeyens and M. Van Daele. "The fast and accurate computation of eigenvalues and eigenfunctions of time-independent one-dimensional

Schrödinger equations”. In: *Computer physics communications* (2020), p. 107568.
DOI: `10.1016/j.cpc.2020.107568`.

[10] P. B. Bailey, W. N. Everitt, and A. Zettl. “Algorithm 810: The SLEIGN2 Sturm-Liouville Code”. In: *Acm transactions on mathematical software* 27.2 (2001), pp. 143–192.
DOI: `10.1145/383738.383739`.

[11] C. Ballester and V. Pereyra. “On the Construction of Discrete Approximations to Linear Differential Expressions”. In: *Mathematics of computation* 21.99 (1967), 297–s19.
DOI: `10.2307/2003232`.

[12] P. Bérard and B. Helffer. “Nodal sets of eigenfunctions, Antonie Stern’s results revisited”. In: *Séminaire de Théorie Spectrale et Géométrie* 32 (2014), pp. 1–37.
DOI: `10.5802/tsg.302`.

[13] P. Binding and H. Volkmer. “A Prüfer angle approach to semidefinite Sturm-Liouville problems with coupling boundary conditions”. In: *Journal of differential equations* 255 (2013), pp. 761–778.
DOI: `10.1016/j.jde.2013.04.033`.

[14] P. Binding. “PRUFER’S TRANSFORMATION (Global qualitative theory of ordinary differential equations and its applications)”. In: 数理解析研究所講究録 1838 (2013), pp. 149–160.

[15] P. Binding and H. Volkmer. “A Prüfer Angle Approach to the Periodic Sturm-Liouville Problem”. In: *The american mathematical monthly* 119.6 (2012), pp. 477–484.
DOI: `10.4169/amer.math.monthly.119.06.477`.

[16] G. Boole. *Calculus Of Finite Differences Fourth Edition*. In collab. with Osmania University and Digital Library Of India. Chelsea Publishing Company, 1860. 370 pp.

[17] Boost. *Float128 - 1.81.0*. boost::multiprecision::float128.

[18] M. Braun, S. A. Sofianos, D. G. Papageorgiou, and I. E. Lagaris. “An Efficient Chebyshev-Lanczos Method for Obtaining Eigensolutions of the Schrödinger Equation on a Grid”. In: *Journal of computational physics* 126.2 (1996), pp. 315–327.
DOI: `10.1006/jcph.1996.0140`.

[19] J. Canosa and R. G. De Oliveira. “A new method for the solution of the Schrödinger equation”. In: *Journal of computational physics* 5.2 (1970), pp. 188–207.
DOI: `10.1016/0021-9991(70)90059-8`.

[20] J. C. Chaves et al. “Octave and Python: High-Level Scripting Languages Productivity and Performance Evaluation”. In: *2006 HPCMP Users Group Conference (HPCMP-UGC’06).* 2006 HPCMP Users Group Conference (HPCMP-UGC’06). 2006, pp. 429–434.
DOI: `10.1109/HPCMP-UGC.2006.55`.

[21] D. Conte and B. Paternoster. “Modified Gauss-Laguerre Exponential Fitting Based Formulae”. In: *Journal of scientific computing* 69.1 (2016), pp. 227–243.
DOI: `10.1007/s10915-016-0190-0`.

[22] R. Courant and D. Hilbert. *Methods of Mathematical Physics.* John Wiley & Sons, 2008. 579 pp.

[23] B. Dacorogna. *Introduction to the Calculus of Variations.* 2nd ed. Imperial College Press, 2008.
DOI: `10.1142/p616`.

[24] M. J. Davis and E. J. Heller. “Semiclassical Gaussian basis set method for molecular vibrational wave functions”. In: *The journal of chemical physics* 71.8 (1979), pp. 3383–3395.
DOI: `10.1063/1.438727`.

[25] T. A. Davis. “Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization”. In: *Acm transactions on mathematical software* 38.1 (2011), 8:1–8:22.
DOI: `10.1145/2049662.2049670`.

[26] J. W. Demmel et al. “A Supernodal Approach to Sparse Partial Pivoting”. In: *Siam journal on matrix analysis and applications* 20.3 (1999), pp. 720–755.
DOI: `10.1137/S0895479895291765`.

[27] *NIST digital library of mathematical functions.* 2023.

[28] M. S. P. Eastham, C. T. Fulton, and S. Pruess. “Using the SLEDGE package on Sturm-Liouville problems having nonempty essential spectra”. In: *Acm transactions on mathematical software* 22.4 (1996), pp. 423–446.
DOI: `10.1145/235815.235819`.

[29] V. Fack and G. V. Berghe. “A finite difference approach for the calculation of perturbed oscillator energies”. In: *Journal of physics a: mathematical and general* 18.17 (1985), p. 3355.
DOI: `10.1088/0305-4470/18/17/017`.

[30] V. Fack and G. Vanden Berghe. “A program for the calculation of energy eigenvalues and eigenstates of a schrödinger equation”. In: *Computer physics communications* 39.2 (1986), pp. 187–196.
DOI: `10.1016/0010-4655(86)90130-X`.

[31] B. Fornberg. “Generation of Finite Difference Formulas on Arbitrarily

Spaced Grids". In: *Mathematics of computation* 51.184 (1988), pp. 699–706.
DOI: 10.2307/2008770.

[32] G. H. Golub and C. F. V. Loan. *Matrix Computations.* JHU Press, 2013. 781 pp.

[33] G. Guennebaud, B. Jacob, et al. *Eigen v3.* 2010.

[34] R. B. Guenther and J. W. Lee. *Sturm-Liouville Problems: Theory and Numerical Implementation.* Boca Raton: CRC Press, 2018. 420 pp.
DOI: 10.1201/9780429437878.

[35] M. A. Al-Gwaiz. *Sturm-Liouville Theory and its Applications.* 1st ed. Springer Undergraduate Mathematics Series 1615-2085. Springer London, 2008. 264 pp.

[36] J. K. Hale. "Eigenvalues and Perturbed Domains". In: *10 Mathematical Essays on Approximation in Analysis and Topology.* Ed. by J. Ferrera, J. López-Gómez, and F. R. Ruiz del Portal. Amsterdam: Elsevier Science, 2005, pp. 95–123.
DOI: 10.1016/B978-044451861-3/50003-3.

[37] M. T. Heath. *Scientific Computing.* 2nd edition. Boston: The McGraw-Hill Companies, Inc., 2002. 563 pp.

[38] V. Hernandez, J. E. Roman, and V. Vidal. "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems". In: *Acm transactions on mathematical software* 31.3 (2005), pp. 351–362.
DOI: 10.1145/1089014.1089019.

[39] M. Hořeňovský. *Catch2.* 2023.

[40] Y. Hua and T. Sarkar. "On SVD for estimating generalized eigenvalues of singular matrix pencil in noise". In: *Ieee transactions on signal processing* 39.4 (1991), pp. 892–900.
DOI: 10.1109/78.80911.

[41] W. R. Inc. *Mathematica, Version 13.1.*

[42] L. G. Ixaru. "An Algebraic Solution of the Schrodinger Equation". In: *Internal report IC/69/6.* ICTP Trieste. 1969.

[43] L. G. Ixaru. "The Algebraic Approach to the Scattering Problem". In: *Internal report IC/69/7.* ICTP Trieste. 1969.

[44] L. G. Ixaru. "The error analysis of the algebraic method for solving the Schrödinger equation". In: *Journal of computational physics* 9.1 (1972), pp. 159–163.
DOI: 10.1016/0021-9991(72)90041-1.

[45] L. G. Ixaru. "CP methods for the Schrödinger equation". In: *Journal of computational and applied mathematics.* Numerical Analysis 2000.

Vol. VI: Ordinary Differential Equations and Integral Equations 125.1 (2000), pp. 347–357.
DOI: 10.1016/S0377-0427(00)00478-7.

[46] L. G. Ixaru. "LILIX—A package for the solution of the coupled channel Schrödinger equation." In: *Computer physics communications* 147.3 (2002), pp. 834–852.
DOI: 10.1016/S0010-4655(02)00459-9.

[47] L. G. Ixaru. "New numerical method for the eigenvalue problem of the 2D Schrödinger equation". In: *Computer physics communications* 181.10 (2010), pp. 1738–1742.
DOI: 10.1016/j.cpc.2010.06.031.

[48] L. G. Ixaru. "Operations on oscillatory functions". In: *Computer physics communications* 105.1 (1997), pp. 1–19.
DOI: 10.1016/S0010-4655(97)00067-2.

[49] L. G. Ixaru, H. De Meyer, and G. Vanden Berghe. "CP methods for the Schrödinger equation revisited". In: *Journal of computational and applied mathematics* 88.2 (1998), pp. 289–314.
DOI: 10.1016/S0377-0427(97)00218-5.

[50] L. G. Ixaru, H. De Meyer, and G. Vanden Berghe. "SLCPM12 - A program for solving regular Sturm-Liouville problems". In: *Computer physics communications* 118.2 (1999), pp. 259–277.
DOI: 10.1016/S0010-4655(98)00181-7.

[51] L. G. Ixaru. *Numerical Methods for Differential Equations and Applications*. Dordrecht-Boston-Lancaster: Reidel, 1984.

[52] L. G. Ixaru and G. Vanden Berghe. *Exponential Fitting*. Dordrecht: Springer Netherlands, 2004.
DOI: 10.1007/978-1-4020-2100-8.

[53] W. Jakob, J. Rhinelander, and D. Moldovan. *Pybind11 - Seamless operability between C++11 and Python*. 2017.

[54] C. Jordan and K. Jordán. *Calculus of Finite Differences*. American Mathematical Soc., 1965. 704 pp.

[55] V. J. Katz. *A History of Mathematics*. 3rd edition. Boston: Pearson, 2008. 992 pp.

[56] H. B. Keller and V. Pereyra. "Symbolic generation of finite difference formulas". In: *Mathematics of computation* 32.144 (1978), pp. 955–971.
DOI: 10.1090/S0025-5718-1978-0494848-1.

[57] K. J. Kim, R. Cools, and L. G. Ixaru. "Quadrature rules using first derivatives for oscillatory integrands". In: *Journal of computational*

and applied mathematics. Int. Congress on Computational and Applied Mathematics 2000 140.1 (2002), pp. 479–497.
DOI: 10.1016/S0377-0427(01)00483-6.

[58] K. Klotter. *Technische Schwingungslehre*. Ed. by G. Benz. Berlin, Heidelberg: Springer, 1978.
DOI: 10.1007/978-3-662-40340-2.

[59] R. P. Kuzmina. *Asymptotic Methods for Ordinary Differential Equations*. 2000th edition. Dordrecht ; Boston: Springer, 2000. 376 pp.

[60] P. Lancaster. “On eigenvalues of matrices dependent on a parameter”. In: *Numerische mathematik* 6.1 (1964), pp. 377–387.
DOI: 10.1007/BF01386087.

[61] V. Ledoux and M. Van Daele. “Solving Sturm-Liouville problems by piecewise perturbation methods, revisited”. In: *Computer physics communications* 181.8 (2010), pp. 1335–1345.
DOI: 10.1016/j.cpc.2010.03.017.

[62] V. Ledoux and M. Van Daele. “The accurate numerical solution of the Schrödinger equation with an explicitly time-dependent Hamiltonian”. In: *Computer physics communications* 185.6 (2014), pp. 1589–1594.
DOI: 10.1016/j.cpc.2014.02.023.

[63] V. Ledoux, M. Van Daele, and G. Vanden Berghe. “A numerical procedure to solve the multichannel Schrödinger eigenvalue problem”. In: *Computer physics communications* 176.3 (2007), pp. 191–199.
DOI: 10.1016/j.cpc.2006.09.004.

[64] V. Ledoux, M. Van Daele, and G. Vanden Berghe. “CP methods of higher order for Sturm-Liouville and Schrödinger equations”. In: *Computer physics communications* 162.3 (2004), pp. 151–165.
DOI: 10.1016/j.cpc.2004.07.001.

[65] V. Ledoux, M. Van Daele, and G. Vanden Berghe. “CPM{P,N} methods extended for the solution of coupled channel Schrödinger equations”. In: *Computer physics communications* 174.5 (2006), pp. 357–370.
DOI: 10.1016/j.cpc.2005.10.009.

[66] V. Ledoux, M. Van Daele, and G. Vanden Berghe. “MATSLISE: A MATLAB package for the numerical solution of Sturm-Liouville and Schrödinger equations”. In: *Acm transactions on mathematical software* 31.4 (2005), pp. 532–554.
DOI: 10.1145/1114268.1114273.

[67] V. Ledoux. *Study of special algorithms for solving Sturm-Liouville and Schrodinger equations*. PhD-thesis. Ghent University, 2007.

[68] V. Ledoux and M. Van Daele. “MATSLISE 2.0 : a Matlab toolbox for

Sturm-Liouville computations". In: *Acm transactions on mathematical software (toms)* 42.4 (2016), p. 18.
DOI: 10.1145/2839299.

[69] J. Lee, V. Balakrishnan, C.-K. Koh, and D. Jiao. "From O(k2N) to O(N): A fast complex-valued eigenvalue solver for large-scale on-chip interconnect analysis". In: *2009 IEEE MTT-S International Microwave Symposium Digest*. 2009 IEEE MTT-S International Microwave Symposium Digest. 2009, pp. 181–184.
DOI: 10.1109/MWSYM.2009.5165662.

[70] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998. 150 pp.

[71] X. S. Li. "An overview of SuperLU: Algorithms, implementation, and user interface". In: *Acm transactions on mathematical software* 31.3 (2005), pp. 302–325.
DOI: 10.1145/1089014.1089017.

[72] J. Liouville. "Second mémoire sur le développement des fonctions ou parties de fonctions en séries dont les divers termes sont assujettis à satisfaire à une même équation différentielle du second ordre, contenant un paramètre variable". In: *Journal de Mathématiques Pures et Appliquées* 2 (1837), pp. 16–35.

[73] L. Mackey. "Deflation methods for sparse PCA". In: *Proceedings of the 21st International Conference on Neural Information Processing Systems*. NIPS'08. Red Hook, NY, USA: Curran Associates Inc., 2008, pp. 1017–1024.

[74] W. E. Milne. "Numerical Integration of Ordinary Differential Equations". In: *The american mathematical monthly* 33.9 (1926), pp. 455–460.
DOI: 10.1080/00029890.1926.11986619.

[75] B. V. Noumerov. "A Method of Extrapolation of Perturbations". In: *Monthly notices of the royal astronomical society* 84.8 (1924), pp. 592–602.
DOI: 10.1093/mnras/84.8.592.

[76] J. W. Paine, F. R. de Hoog, and R. S. Anderssen. "On the correction of finite difference eigenvalue approximations for Sturm-Liouville problems". In: *Computing* 26.2 (1981), pp. 123–139.
DOI: 10.1007/BF02241779.

[77] R. Piessens, E. de Doncker-Kapenga, C. W. Überhuber, and D. K. Kahaner. *Quadpack: A Subroutine Package for Automatic Integration*. Springer Series in Computational Mathematics. Springer-Verlag, 1983.
DOI: 10.1007/978-3-642-61786-7.

[78] S. Pruess. “Estimating the Eigenvalues of Sturm-Liouville Problems by Approximating the Differential Equation”. In: *Siam journal on numerical analysis* 10.1 (1973), pp. 55–68.
DOI: `10.1137/0710008`.

[79] H. Prüfer. “Neue Herleitung der Sturm-Liouvilleschen Reihenentwicklung stetiger Funktionen”. In: *Mathematische Annalen* 95.1 (1926), pp. 499–518.
DOI: `10.1007/BF01206624`.

[80] J. D. Pryce. “A test package for Sturm-Liouville solvers”. In: *Acm transactions on mathematical software* 25.1 (1999), pp. 21–57.
DOI: `10.1145/305658.287651`.

[81] J. D. Pryce. “Error Control of Phase-Function Shooting Methods for Sturm-Liouville Problems”. In: *Ima journal of numerical analysis* 6.1 (1986), pp. 103–123.
DOI: `10.1093/imanum/6.1.103`.

[82] Y. Qiu. *Spectra.* 2022.

[83] M. Reed and B. Simon. *II: Fourier Analysis, Self-Adjointness.* Elsevier, 1975. 380 pp.

[84] M. Reed and B. Simon. *IV: Analysis of Operators.* Elsevier Science, 1978. 424 pp.

[85] M. Reed, B. Simon, M. (U. R. Carolina) North, and B. (U. S. Jersey) New. *I: Functional Analysis.* Gulf Professional Publishing, 1980. 417 pp.

[86] M. Renardy and R. C. Rogers. *An Introduction to Partial Differential Equations.* 2nd ed. Texts in Applied Mathematics. New York: Springer-Verlag, 2004.
DOI: `10.1007/b97427`.

[87] Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2011. 287 pp.
DOI: `10.1137/1.9781611970739`.

[88] H. Sagan. *Boundary and eigenvalue problems in mathematical physics.* Wiley, 1961. 381 pp.

[89] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.6).* manual. 2022.

[90] E. Schrödinger. “Quantisierung als Eigenwertproblem”. In: *Annalen der Physik* 384.4 (1926), pp. 361–376.
DOI: `10.1002/andp.19263840404`.

[91] J. Siedlecki, M. Ciesielski, and T. Błaszczyk. “The Sturm-Liouville eigenvalue problem - a numerical solution using the Control Volume Method”.

In: *Journal of applied mathematics and computational mechanics* 15.2 (2016), pp. 127–136.
DOI: `10.17512/jamcm.2016.2.14`.

[92] G. F. Simmons. *Differential Equations with Applications and Historical Notes*. 3rd edition. Boca Raton London New York: Chapman and Hall/CRC, 2016. 764 pp.

[93] B. Stroustrup. *The C++ Programming Language, 4th Edition*. 4th edition. Westport, Conn: Addison-Wesley Professional, 2013. 1376 pp.

[94] L. M. M. Thomson. *The Calculus Of Finite Differences*. In collab. with Osmania University and Digital Library Of India. Macmillan And Company., Limited, 1933. 590 pp.

[95] E. C. Titchmarsh. *Eigenfunction Expansions - Associated with Second-order Differential Equations. Part I.* 2Rev Ed edition. Oxford: Oxford University Press, 1962. 210 pp.

[96] L. N. Trefethen and D. B. III. *Numerical Linear Algebra*. 1st edition. Philadelphia: SIAM: Society for Industrial and Applied Mathematics, 1997. 184 pp.

[97] J. Unpingco. “Some Comparative Benchmarks for Linear Algebra Computations in Matlab and Scientific Python”. In: *2008 DoD HPCMP Users Group Conference*. 2008 DoD HPCMP Users Group Conference. 2008, pp. 503–505.
DOI: `10.1109/DoD.HPCMP.UGC.2008.49`.

[98] G. Vanden Berghe and H. De Meyer. “A modified numerov method for higher sturm-liouville eigenvalues”. In: *International journal of computer mathematics* 37.1-2 (1990), pp. 63–77.
DOI: `10.1080/00207169008803935`.

[99] G. Vanden Berghe and H. De Meyer. “Accurate computation of higher Sturm-Liouville eigenvalues”. In: *Numerische mathematik* 59.1 (1991), pp. 243–254.
DOI: `10.1007/BF01385778`.

[100] G. Vanden Berghe, M. Van Daele, and H. De Meyer. “A modified difference scheme for periodic and semiperiodic Sturm-Liouville problems”. In: *Applied numerical mathematics* 18.1 (1995), pp. 69–78.
DOI: `10.1016/0168-9274(95)00067-5`.

[101] P. Virtanen et al. “SciPy 1.0: fundamental algorithms for scientific computing in Python”. In: *Nature methods* 17.3 (2020), pp. 261–272.
DOI: `10.1038/s41592-019-0686-2`.

[102] Z. Wang and H. Shao. “A new kind of discretization scheme for solving a

two-dimensional time-independent Schrödinger equation". In: *Computer physics communications* 180.6 (2009), pp. 842–849.
DOI: `10.1016/j.cpc.2008.11.022`.

[103] A. Zettl. *Sturm-Liouville Theory.* American Mathematical Soc., 2012. 346 pp.

[104] G. Zhukova. "Asymptotic methods for solving boundary value eigenvalue problems". In: *E3s web conf.* 164 (2020), p. 09022.
DOI: `10.1051/e3sconf/202016409022`.