FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

JÜLICH
Forschungszentrum

Fachhochschule Aachen - Campus Jülich
Fachbereich 9 - Medizintechnik und Technomathematik
Studiengang Technomathematik

Eigenvalue Optimization for Acoustic Scattering Problems

Masterarbeit von Daniel Abele
Jülich, August 2019

Diese Arbeit wurde betreut von:
1. Prüfer: Prof. Dr. Johannes Grotendorst
2. Prüfer: Dr. Andreas Kleefeld

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die Masterarbeit mit dem Thema

„Eigenvalue Optimization for Acoustic Scattering Problems“

selbstständig angefertigt und verfasst habe. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Daniel Abele

_____

Ort Datum Unterschrift

Abstract

This master thesis is concerned with the optimization of eigenvalues of the Laplace differential operator, specifically interior Neumann eigenvalues, with respect to the shape of the domain. Such eigenvalue problems arise in the study of acoustic scattering, which has applications in sonar or radar detection and medical imaging. The shape of the space significantly affects the eigenvalues. Improved optimal values for some of them are reported.

The main focus of the thesis is finding a description of the shape that is well suited for optimization. The number of parameters should be low to keep the optimization space simple. At the same time, the range of representable shapes should be large enough to improve upon previous results. Inspired by physics, equipotentials are used to model the knobbly objects found by previous researchers in a simple way.

The work discusses a method of solving the eigenvalue problem. The Boundary Element Method for boundary value problems is combined with Beyn's method for nonlinear eigenvalue problems. The implementation of these methods is another central issue. As the optimizer requires many evaluations, high speed is desired. The code is parallelized for efficient computation on a large cluster.

The implemented solvers are tested for convergence. The parameter space is thoroughly numerically explored to facilitate optimization. Finally the results of the optimization are presented. The shape description shows a lot of promise but is not yet general enough to optimize every eigenvalue.

This work has been produced in cooperation with *Jülich Supercomputing Center* at *Forschungszentrum Jülich*. Numerical computation was performed on the *JURECA* Supercomputer.

Contents

List of Figures

List of Tables

1. Introduction

Acoustic scattering theory is the study of how waves in an inelastic medium like a fluid or gas are scattered by obstacles. The obstacle has different wave propagation properties than the medium and thus may partially or fully absorb and reemit or reflect the wave. This has applications for example in radar or sonar detection or medical imaging. The theory for time harmonic waves inside a closed two-dimensional space or *domain* $D \subset \mathbb{R}^2$ states that the field u (e.g. pressure of the medium) must satisfy the partial differential equation (PDE)

$$\triangle u + \kappa^2 u = 0 \quad \text{in } D\,.$$

This equation is called *Helmholtz* equation. The second order differential operator $\triangle u = \partial^2 u/\partial x_1^2 + \partial^2 u/\partial x_2^2$ with spatial coordinates x_1 and x_2, is called *Laplace* operator. The parameter $\kappa \in \mathbb{R}$ is the *wavenumber* or spatial frequency of the wave. The propagation properties of the obstacle (e.g. the walls of a room) are specified by boundary conditions (BC). Sound-hard surfaces allow no transfer of pressure across the surface. This is described by the Neumann BC

$$\frac{\partial u}{\partial \nu} = 0 \quad \text{on } \partial D$$

where ν is the normal vector onto the boundary ∂D directed into the exterior $\mathbb{R}^2 \setminus D$. PDE and BC together form a boundary value problem (BVP).

The BVP can also be considered an eigenvalue problem. A solution u is an eigenvector for the eigenvalue $\lambda = \kappa^2$ of the Laplace operator. For this particular problem, they are also referred to as interior Neumann eigenvalues and -vectors. The goal of this work is to find shapes of the domain D of constant area so that the eigenvalues are maximal.

There has already been some study in this area. It is well known that the eigenvalues are real and discrete as the Laplace operator is self-adjoint. Szegö [25] and later Weinberger [28] have shown that the first eigenvalue (numbered ascendingly and ignoring the trivial case $\lambda = 0$) is maximized by a disk. Girouard et al. [10] have proven that the second eigenvalue is maximized by two disjoint disks of the same size. For higher eigenvalues, theoretical results are sparse. It was conjectured for a time that unions of disjoint disks maximize all eigenvalues. However, recently Poliquin and Roy-Fortin [23] have shown that this cannot be true. There is also numerical evidence for connected shape maximizers for some low eigenvalues. Numerical solutions depend on a

good parameterization of the shape. Antunes and Oudet [4] have used truncated Fourier series and found connected shapes that produce eigenvalues higher than disjoint unions of disks for most eigenvalues below ten. Based on their results, Kleefeld [15] developed a description of the shapes with only two parameters and used it to improve the third and fourth eigenvalue. In the work on hand, Kleefeld's description will be tested on the higher eigenvalues and extended and hopefully improved by adding additional parameters.

The chapter immediately following this introduction will discuss methods for the numerical solution of the BVP and eigenvalue problem. It will also give a detailed description of the optimization problem and of the parametrization of the shapes. Then follows a discussion of the practical implementation with a special focus on parallelization. The solution of this problem is very time consuming. Efficient computation is necessary for the rapid testing of different parametrizations. Chapter 4 will discuss the numerical results. The work closes with a summary and an outlook towards possible future avenues of exploration.

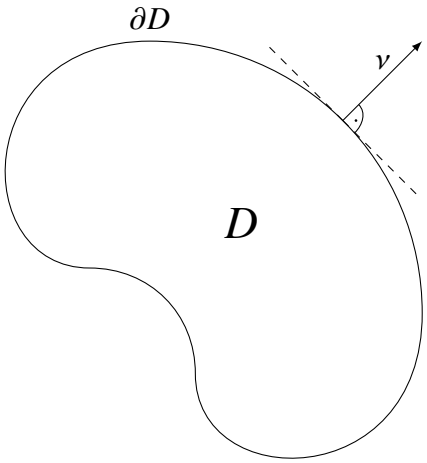
2. Numerical Methods

As has been stated in the introduction, the goal is to find a shape so that the interior Neumann eigenvalues are maximal. This chapter presents the numerical methods used for each step of the computation. First the BVP must be discretized. Given a domain, the Boundary Element Method transforms it into a homogeneous linear system whose system matrix depends nonlinearly on the parameter κ. Thus the linear system is also a nonlinear eigenvalue problem. Eigenvalues of this system are computed using Beyn's method. An objective function for optimization is constructed by combining these methods with a description of the shape of the domain using as few parameters as possible.

2.1. Numerical Solution of Boundary Value Problems

This section describes a method of solving BVPs. First the problem is stated in detail. Let the domain $D \subset \mathbb{R}^2$ be an open and bounded set and ∂D its boundary. For the described method to be applicable, the boundary is allowed to be disconnected but must be smooth on each subset, i.e. ∂D is of class C^2. The normal vector at some point $x \in \partial D$ towards the outside of the domain is denoted by $\nu := \nu(x)$ (see Figure 2.1).

Figure 2.1.: Exemplary domain D, its boundary ∂D and the normal ν directed into the exterior.



The BVP consists of finding a function u that satisfies the Helmholtz equation and the Neumann boundary condition

$$\triangle u + \kappa^2 u = 0 \quad \text{in } D \tag{2.1a}$$

$$\frac{\partial u}{\partial \nu} = f \quad \text{on } \partial D \tag{2.1b}$$

for some wave number $\kappa \in \mathbb{C}$ and some function $f \in C(\partial D)$. Here $\triangle u = \partial^2 u/\partial x_1^2 + \partial^2 u/\partial x_2^2$ with x_1, x_2 the Cartesian coordinates of $\mathbb{R}^2$ denotes the Laplace operator and $\partial u/\partial \nu$ is the normal derivative of u in the direction of ν. Note that the method for the solution of eigenvalue problems discussed later requires considering wavenumbers from the set of complex numbers even though only real eigenvalues exist.

2.1.1. The Boundary Element Method

The BVP is solved using the boundary element method. This reduces the dimension of the problem and the number of unknowns as only the boundary needs to be discretized instead of the whole domain. For numerical stability, an ansatz based on the single layer potential is chosen that yields an integral equation of the second kind. The theory behind this method is covered extensively in [17, Chapter 6]. Note Example 12.14 regarding the application to the Helmholtz equation.

As easily verified by application, the function $\Phi_\kappa : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{C}$

$$\Phi_\kappa(x, y) := \frac{\mathrm{i}}{4} H_0^{(1)}(\kappa \|x - y\|), \tag{2.2}$$

with i the imaginary unit, $H_0^{(1)}$ the Hankel function of the first kind of order 0 and $\|\cdot\|$ the Euclidian norm is a solution to the Helmholtz equation (2.1a) with respect to x for any y. It is called a fundamental solution of (2.1a). It follows that the so-called single layer potential

$$u(x) := \int_{\partial D} \Phi_\kappa(x, y)\psi(y)\, \mathrm{d}s(y)\,, \quad x \in D \tag{2.3}$$

is also a solution to (2.1a) for any density function $\psi(x) \in C(\partial D)$. Note that the order of differentiation and integration can be switched for any $x \in D$ when applying (2.1a) to (2.3). The density must be determined such that (2.3) also satisfies the boundary condition (2.1b). Approaching the boundary from inside the domain, the normal derivative of the single layer potential satisfies the jump condition

$$\frac{\partial}{\partial \nu(x)} u(x) = \frac{1}{2}\psi(x) + \int_{\partial D} \frac{\partial}{\partial \nu(x)} \Phi_\kappa(x, y)\psi(y)\, \mathrm{d}s(y)\,, \quad x \in \partial D\,. \tag{2.4}$$

Applying the boundary condition (2.1b) yields the integral equation

$$\frac{1}{2}\psi(x) + \int_{\partial D} \frac{\partial}{\partial \nu(x)} \Phi_\kappa(x, y)\psi(y)\, \mathrm{d}s(y) = f(x) \quad \forall x \in \partial D\,. \tag{2.5}$$

For a numerical solution, (2.5) must be discretized, which yields an approximation of the density. Inserting this density into a discretization of the single layer potential (2.3) yields an approximation of the solution u.

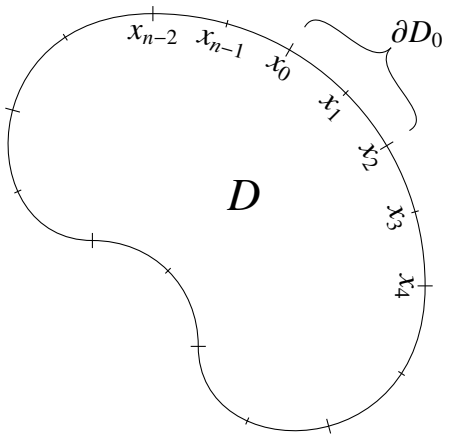
2.1.2. Discretization

This section describes the discretization of the integral equation (2.5) following this rough outline of the procedure. The boundary ∂D is separated into multiple intervals supported by nodes on the boundary. Quadratic interpolation is used to approximate the boundary and the density ψ on the intervals. In order to determine the unknown density values at the nodes the integral equation that is the result of discretizing the boundary is required to be satisfied exactly at each node. This yields a linear system.

The discretization of the single layer potential (2.3) is not discussed here in detail as it is fundamentally similar to the discretization of (2.5). Also the linear system resulting from the latter is already sufficient for calculating eigenvalues κ^2 of the PDE (2.1a), which is the main purpose of this work.

The first step is to discretize the boundary. Nodes $x_i, i = 0, \ldots, n-1$ on the boundary are chosen as in Figure 2.2. The number of nodes n must be even so that the nodes separate the boundary into intervals $\partial D_k, k = 0, \ldots, n/2 - 1$ with start node x_{2k}, middle node x_{2k+1}, and end node x_{2k+2}. The boundary being closed, the end node of the last interval x_n is identical to the start node of the first interval x_0.

Figure 2.2.: Discretization of the boundary of domain D with n points and $n/2$ intervals.



By this separation into intervals the integral over the boundary in (2.5) is transformed into the sum

$$\frac{1}{2}\psi(x) + \sum_{k=0}^{n/2-1} \int_{\partial D_k} \frac{\partial}{\partial \nu(x)} \Phi_\kappa(x, y)\psi(y)\, \mathrm{d}s(y) = f(x) \quad \forall x \in \partial D\,. \tag{2.6}$$

For each interval ∂D_k there exists a unique smooth and bijective function $g_k(t)$ that maps any value in the unit interval onto a point on ∂D_k, specifically $g_k(0) = x_{2k}$, $g_k(1/2) = x_{2k+1}$, and $g_k(1) = x_{2k+2}$. Therefore we can substitute the integration limits and variable in (2.6). Applying the substitution rule for integrals yields

$$\frac{1}{2}\psi(x) + \sum_{k=0}^{n/2-1} \int_0^1 \frac{\partial}{\partial \nu(x)} \Phi_\kappa(x, g_k(t)) \psi(g_k(t)) \left\| g_k'(t) \right\| \, \mathrm{d}t = f(x) \quad \forall x \in \partial D \,. \tag{2.7}$$

The unknown functions $g_k(t)$ and $\psi(g_k(t))$ must be approximated. Let $f \in C([0,1])$ be any continuous function whose values are known at points $t_i, i = 0, 1, 2$. Then the function can be approximated by quadratic interpolation as

$$\begin{aligned} \tilde{f}(t) &= L_0(t) f(t_0) + L_1(t) f(t_1) + L_2(t) f(t_2) \\ &= \sum_{j=0}^{2} L_j(t) f(t_j) \end{aligned} \tag{2.8}$$

where L_j are the quadratic Lagrange basis polynomials given by the general formula

$$L_j(t) = \prod_{\substack{i=0 \\ i \neq j}}^{2} \frac{t - t_i}{t_j - t_i} \,.$$

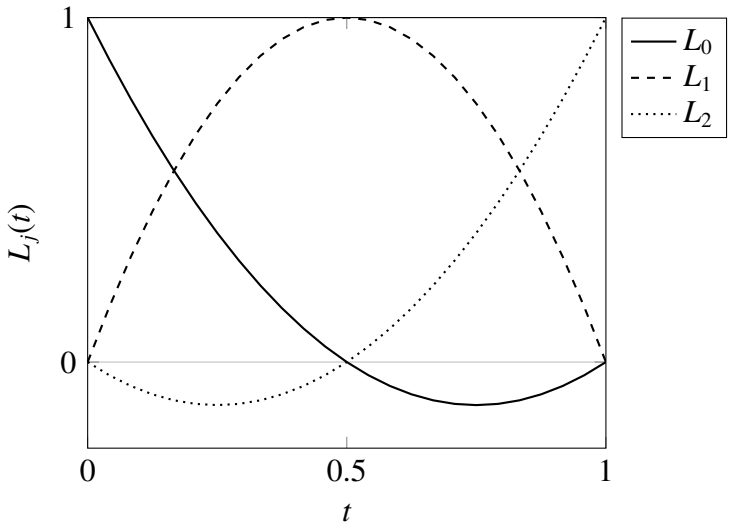
These polynomials have the property that $L_j(t)$ yields 1 at $t = t_j$ and 0 at all other nodes. For interpolation over the unit interval, the nodes $t_0 = 0, t_1 = 1/2, t_2 = 1$ are chosen corresponding to the start, middle, and end node of an interval, respectively. The specific formulas for the polynomials (see also Figure 2.3) then read

$$\begin{aligned} L_0(t) &= 2(t - 1/2)(t - 1) \\ L_1(t) &= -4t(t - 1) \\ L_2(t) &= 2t(t - 1/2) \,. \end{aligned}$$

The function $g_k(t)$ is continuous and known at the discretization nodes, so it satisfies the conditions for quadratic interpolation. Interpolation yields the approximation

$$\begin{aligned} \tilde{g}_k(t) &= L_0(t) g_k(t_0) + L_1(t) g_k(t_1) + L_2(t) g_k(t_2) \\ &= L_0(t) x_{2k} + L_1(t) x_{2k+1} + L_2(t) x_{2k+2} \\ &= \sum_{j=0}^{2} L_j(t) x_{2k+j} \,. \end{aligned} \tag{2.9}$$

Figure 2.3.: Quadratic Lagrange basis polynomials.



Note that $\tilde{g}_k(t_j) = g_k(t_j)$ at each interpolation node. Likewise the density is quadratically interpolated on each interval ∂D_k as

$$\psi(\tilde{g}_k(t)) = \sum_{j=0}^{2} L_j(t)\psi(g_k(t_j)) = \sum_{j=0}^{2} L_j(t)\psi_{2k+j} \tag{2.10}$$

with $\psi_i = \psi(x_i)$ the yet unknown values of ψ at the nodes. Inserting the approximations into the integral equation (2.7) and requiring the integral equation to be solved exactly at the discretization nodes (*collocation method*) yields a linear system

$$\frac{1}{2}\psi_i + \sum_{k=0}^{n/2-1}\sum_{j=0}^{2}\left[\psi_{2k+j}\int_0^1 \frac{\partial}{\partial\nu(x_i)}\Phi_\kappa(x_i, \tilde{g}_k(t))L_j(t)\left\|\tilde{g}_k'(t)\right\| \,\mathrm{d}t\right] = f(x_i) \tag{2.11}$$

for $i = 0, \ldots, n-1$ with n unknowns ψ_i and n equations. The node x_i that corresponds to each single equation is called the *collocation node*.

This linear system can be written in matrix form as

$$\left(\frac{1}{2}I + A_\kappa\right)\vec{\psi} = \vec{f} \tag{2.12}$$

with $\vec{\psi} = (\psi_0, \ldots, \psi_{n-1})^\top \in \mathbb{C}^n$, $\vec{f} = (f(x_0), \ldots, f(x_{n-1}))^\top \in \mathbb{C}^n$, and identity matrix $I \in \mathbb{C}^{n\times n}$. The matrix $A_\kappa \in \mathbb{C}^{n\times n}$ is dense and asymmetric. Because of the overlap of end nodes of adjacent intervals the entries of A_κ in even columns ($j = 0, 2, \ldots, n-2$) consist

of two summands

$$\begin{aligned}(A_\kappa)_{i,j} = &\int_0^1 \frac{\partial}{\partial \nu(x_i)} \Phi_\kappa(x_i, \tilde{g}_{j/2-1}(t)) L_2(t) \left\| \tilde{g}'_{j/2-1}(t) \right\| \, \mathrm{d}t \\ &+ \int_0^1 \frac{\partial}{\partial \nu(x_i)} \Phi_\kappa(x_i, \tilde{g}_{j/2}(t)) L_0(t) \left\| \tilde{g}'_{j/2}(t) \right\| \, \mathrm{d}t \,,\end{aligned} \tag{2.13}$$

wheras the entries in odd columns ($j = 1, 3, \ldots, n-1$) consist of only one summand

$$(A_\kappa)_{i,j} = \int_0^1 \frac{\partial}{\partial \nu(x_i)} \Phi_\kappa(x_i, \tilde{g}_{(j-1)/2}(t)) L_1(t) \left\| \tilde{g}'_{(j-1)/2}(t) \right\| \, \mathrm{d}t \,. \tag{2.14}$$

The integrals must be evaluated numerically as discussed in the following section.

2.1.3. Analysis of the Integrand

In order to analyze the integrand in the previous section, the normal derivative of the fundamental solution (2.2) must be determined. Differentiating the Hankel function of the first kind of order 0 and applying the chain rule yields

$$\begin{aligned}\frac{\partial}{\partial \nu(x)} \Phi_\kappa(x, y) &= \frac{\partial}{\partial \nu(x)} \left[\frac{\mathrm{i}}{4} H_0^{(1)}(\kappa \|x - y\|) \right] \\ &= -\frac{\mathrm{i}\kappa}{4} H_1^{(1)}(\kappa \|x - y\|) \frac{\partial}{\partial \nu(x)} \|x - y\| \\ &= -\frac{\mathrm{i}\kappa}{4} H_1^{(1)}(\kappa \|x - y\|) \frac{1}{\|x - y\|} \langle x - y, \nu(x) \rangle\end{aligned} \tag{2.15}$$

where $\langle \cdot \,, \cdot \rangle$ denotes the standard scalar product and $H_1^{(1)}$ is the Hankel function of the first kind of order 1. This expression contains a discontinuity at $x = y$ as the norm $\|x - y\|$ becomes 0, $H_1^{(1)}$ is singular at the origin of the complex plane and the norm appears again in the denominator. The Hankel function is continuous everywhere but at 0, so this is the only point where the behavior of (2.15) must be explored further.

First separation into intervals as in (2.6) and the substitution as in (2.7) are applied to obtain for the integrand in the k-th interval:

$$-\frac{\mathrm{i}\kappa}{4} H_1^{(1)}(\kappa \|x - g_k(t)\|) \frac{1}{\|x - g_k(t)\|} \langle x - g_k(t), \nu(x) \rangle \,.$$

If the point x lies outside the k-th interval of the boundary, this expression is again continuous for $t \in [0, 1]$ as $\|x - g_k(t)\|$ cannot be 0. In the following it is assumed that there is a discontinuity in the interval, so $g_k(t_0) = x$ for some $t_0 \in [0, 1]$. The remaining factors of the integrand of (2.7) (the density $\psi(t)$, the Jacobian $\|g'(t)\|$, and the Lagrange polynomials) are continuous and do not need to be included in the analysis.

As $\nu(x) = \nu(g_k(t_0))$ is the outside normal along the boundary, it must be normal to the curve g_k at point t_0. Assuming g_k describes the boundary in anticlockwise direction, the normal can be written as

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \frac{g_k'(t_0)}{\left\| g_k'(t_0) \right\|}$$

where the tangent $g_k'(t_0)$ is normalized and rotated by a right angle. This yields the final expression to be analyzed

$$K(t) := -\frac{\mathrm{i}\kappa}{4} H_1^{(1)}(\kappa \left\| g_k(t_0) - g_k(t) \right\|) \frac{1}{\left\| g_k(t_0) - g_k(t) \right\|} \left\langle g_k(t_0) - g_k(t), \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \frac{g_k'(t_0)}{\left\| g_k'(t_0) \right\|} \right\rangle .$$

The type of the discontinuity depends on the limit of this expression as t approaches t_0 from both sides. The mathematical software system Maple™ [19] (see the attached worksheet in Appendix B) finds the limit

$$\lim_{t \to t_0^+} K(t) = \lim_{t \to t_0^-} K(t) = -\frac{1}{4\pi \left\| g_k'(t_0) \right\|^3} \left\langle g_k'(t_0), \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} g_k''(t_0) \right\rangle . \tag{2.16}$$

As the boundary described by g_k is of class C^2, this limit involving second derivatives is well-defined. Thus the discontinuity is removable by defining

$$K(t_0) = \lim_{t \to t_0} K(t) .$$

As the remaining pieces of the integrand, the density $\psi(g_k(t))$ and the Jacobian $\|g'(t)\|$, are continuous, the whole integrand is made continuous by that definition.

It is necessary to preserve this continuity when discretizing the integrand by approximating $g_k(t)$ using quadratic interpolation. For the limits (2.16) to be correct when the replacing g_k with the approximation $\tilde{g}_k$, the normal vector at the collocation node must be orthogonal to $\tilde{g}_k$. Otherwise the integrand will contain an infinite singularity that is much more difficult to handle numerically. An odd numbered collocation node x_{2i+1} lies in the middle of interval i where $\tilde{g}_i$ is continuously differentiable, so the orthogonality condition is easily satisfied. An even numbered node x_{2i} is both in interval i and $i-1$ as $\tilde{g}_i(0) = \tilde{g}_{i-1}(1) = x_{2i}$. However, the derivatives $\tilde{g}_i'(0)$ and $\tilde{g}_{i-1}'(1)$ are generally not equal. Derivative $\tilde{g}_{i-1}'(1)$ must be used when integrating over interval $i-1$ and $\tilde{g}_i'(0)$ for interval i to satisfy orthogonality. When integrating over the other intervals there are no clear criteria. Alternating every interval is convenient and also smoothes out the error.

Using the limit it is possible to use any established quadrature routine for evaluating the integrals. Due to cancellation in floating point arithmetic, evaluating the integrand close to the singularity is still not recommended (see Appendix B and Table 3.2 for numerical experiments for this effect). If this is required by the quadrature routine, the limit can be used to approximate the exact integrand in an environment around the singularity.

2.1.4. Symmetries

Symmetries of the domain D can be exploited to reduce the computational effort. Note that the integrand contains the discretization points and normal vectors only as part of a norm or scalar product. These operations are mostly invariant under rotation and reflection. If $\hat{x}, \hat{y}$, and $\hat{\nu}$ are the results of rotating vectors x, y, and ν, respectively, then

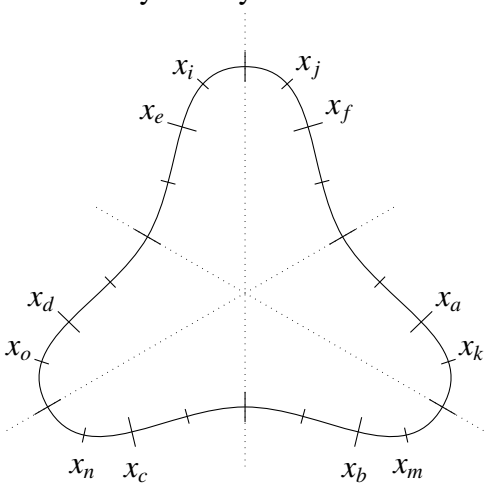
$$\begin{aligned} \|\hat{x} - \hat{y}\| &= \|x - y\| \quad \text{and} \\ \langle \hat{x} - \hat{y}, \hat{\nu} \rangle &= \langle x - y, \nu \rangle \ . \end{aligned}$$

The first equation is also true if $\hat{x}, \hat{y}, \hat{\nu}$ are reflected vectors. The scalar product changes its sign:

$$\begin{aligned} \|\hat{x} - \hat{y}\| &= \|x - y\| \quad \text{and} \\ \langle \hat{x} - \hat{y}, \hat{\nu} \rangle &= -\langle x - y, \nu \rangle \ . \end{aligned}$$

So the integrand changes sign on reflection. However, as the whole integration interval is reflected, the integration boundaries are switched and the sign is compensated. Thus the value of an integral over an interval is identical if the interval and the collocation point are reflected or rotated.

Figure 2.4.: Discretized boundary with six degrees of symmetry, three rotations multiplied by two for axial symmetry.



If the discretization nodes on the boundary are chosen so that the intervals are symmetric and interval end nodes lie on the symmetry axes as in Figure 2.4 many matrix entries will be identical. The computational effort saved depends on the total degree of rotational and reflectional symmetry. The shape in Figure 2.4 has rotational symmetry of degree three, i.e. it can be rotated by 120° and 240° in any direction around the center

of the shape without changing it. It also has reflectional symmetry at each angle. Thus the final degree of symmetry is six (three times two). If as in the figure the pair (x_a, x_i) is reflected and/or rotated onto (x_b, x_j), (x_c, x_k), (x_d, x_m), (x_e, x_n), and (x_f, x_o), then

$$(A_\kappa)_{a,i} = (A_\kappa)_{b,j} = (A_\kappa)_{c,k} = (A_\kappa)_{d,m} = (A_\kappa)_{e,n} = (A_\kappa)_{f,o}$$

of which only one value has to be evaluated. The same is true for all other pairs of points and their images.

Each interval consists of two points, the middle node and one of the end nodes. So in order for intervals to be symmetric, the number of nodes n must be divisible $2s$, where s is the degree of symmetry. Only a segment of the boundary is considered that consists of $n/2s$ intervals. This can be interpreted as either evaluating the whole integral equation (row) for each collocation point in the segment (reducing the number of evaluated rows) or evaluating for every collocation point only those parts of the equation that correspond to an interval inside the segment (reducing the number of evaluated columns). Unlike the whole boundary, the segment is not cyclical. The matrix elements that correspond to both ends of the segment have to be explicitly evaluated. Thus, the number of matrix elements that must be evaluated is $n(n+1)/s$ instead of n^2. Asymptotically for large n the factor of improvement approaches s.

2.2. Nonlinear Eigenvalue Problems

Suppose $T : \mathbb{C} \supset S \to \mathbb{C}^{n\times n}$ is a matrix valued operator that is holomorphic, i.e. infinitely smooth, over a subset S of the complex plane. This is denoted as $T \in H(S, \mathbb{C}^{n\times n})$. Then a nonlinear eigenvalue problem (NLEP) is to find values $\lambda \in \mathbb{C}$ and vectors $v \in \mathbb{C}^n \setminus \{0\}$ such that

$$T(\lambda)v = 0\,. \tag{2.17}$$

A value λ that satisfies (2.17) is called *eigenvalue* of T, the vector v is called *(right) eigenvector*. Note that a linear eigenvalue problem $Ax = \lambda x$ can be transformed into the form (2.17) by introducing the operator $T(\lambda) = (A - \lambda I)$.

As in the linear case eigenvalues are characterized by their algebraic and geometric multiplicity. An eigenvalue is called simple if it satisfies

$$\begin{aligned} &\mathcal{N}(T(\lambda)) = \operatorname{span}\{v\}\,, v \neq 0 \text{ and} \\ &T'(\lambda)v \notin \mathcal{R}(T(\lambda))\,, \end{aligned}$$

i.e. if it has geometric multiplicity one. Here $\mathcal{N}(T(\lambda)) = \{x \in C^n \mid T(\lambda)x = 0\}$ and $\mathcal{R}(A) = \{y \in C^n \mid T(\lambda)x = y, x \in \mathbb{C}^n\}$ denote the nullspace and range of the matrix $T(\lambda)$, respectively. The derivative of T is taken elementwise. In the following only simple eigenvalues will be considered.

It can be proven that eigenvalues of T are isolated, i.e. for every eigenvalue λ there is a neighbourhood $U \subset \mathbb{C}$ of λ that does not contain any other eigenvalues [6, Theorem 2.2].

Given a simple eigenvalue λ and its right eigenvector v there exists a *left* eigenvector $w \in \mathbb{C}^n \setminus \{0\}$ such that

$$\begin{aligned} T(\lambda)^{\mathrm{H}} w &= 0 \text{ and} \\ w^{\mathrm{H}} T'(\lambda) v &\neq 0 \end{aligned}$$

where $T(\lambda)^{\mathrm{H}}$ denotes the conjugate transpose of the matrix $T(\lambda)$. Left and right eigenvectors can always be chosen from the nullspaces of $T(\lambda)$ and $T(\lambda)^{\mathrm{H}}$ as a normalized pair such that

$$\begin{aligned} w^{\mathrm{H}} T'(\lambda) v &= 1 \text{ and either} \\ \|w\| &= 1 \text{ or} \\ \|v\| &= 1\,. \end{aligned} \tag{2.18}$$

2.2.1. Beyn's Contour Integral Method

This section describes a method to find eigenvalues within a contour in the complex plane. The method was derived by W.-J. Beyn [6]. For simplicity only simple eigenvalues are considered. As Beyn shows, the algorithm can be applied to the general case without modification as the structure of multiplicities is preserved. The described method also requires that there are fewer eigenvalues than the dimension of the matrices $T(\lambda)$ and that all eigenvectors are linearly independent. This is usually the case but Beyn also proposes an extension to the method that removes this restriction.

The central theorem of Beyn [6, Theorem 2.9] is the following:

Theorem 2.1. *Let $T \in H(S, \mathbb{C}^{n\times n})$ have no eigenvalues on a contour $C \subset S$ and eigenvalues λ_j, $j = 0, \ldots, k-1$ with corresponding right and left eigenvectors v_j and w_j in the interior of the contour. The eigenvectors have been normalized according to* (2.18)*. Then for every $f \in H(S, \mathbb{C})$*

$$\frac{1}{2\pi \mathrm{i}} \int_C f(z) T(z)^{-1} \,\mathrm{d}z = \sum_{j=1}^{k} f(\lambda_j) v_j {w_j}^{\mathrm{H}}\,. \tag{2.19}$$

From this theorem the method is now derived. The matrices

$$V = (v_0, \ldots, v_{k-1}), W = (w_0, \ldots, w_{k-1}) \in \mathbb{C}^{n\times k}$$

are introduced so the right hand side of (2.19) can be written as a matrix product. As the eigenvectors are required to be linearly independent the rank conditions

$$\begin{aligned} \operatorname{rank}(V) &= k \text{ and} \\ \operatorname{rank}(W) &= k \end{aligned} \tag{2.20}$$

hold. A dimension m with $k \leq m \leq n$ and a matrix $M \in \mathbb{C}^{n \times m}$ is chosen such that

$$\operatorname{rank}(W^{\mathrm{H}} M) = k\,. \tag{2.21}$$

This is almost certainly true if M is chosen at random as $\operatorname{rank}(W) = k$.

Both sides of equation (2.19) are multiplied from the right with M and two different functions $f_0(z) = 1$ and $f_1(z) = z$ are inserted. This yields two equations

$$A_0 = \frac{1}{2\pi \mathrm{i}} \int_C T(z)^{-1} M \, \mathrm{d}z = V W^{\mathrm{H}} M \tag{2.22}$$

$$A_1 = \frac{1}{2\pi \mathrm{i}} \int_C z T(z)^{-1} M \, \mathrm{d}z = V \Lambda W^{\mathrm{H}} M \tag{2.23}$$

with $\Lambda = \operatorname{diag}(\lambda_0, \ldots, \lambda_{k-1})$ the diagonal matrix of eigenvalues. The first equation (2.22) does not include the eigenvalues so it will be used to eliminate the unknown matrices V and W from the second equation (2.23).

The first step is to evaluate the integrals A_0 and A_1. By the rank conditions (2.20) and (2.21) $\operatorname{rank}(A_0) = \operatorname{rank}(V W^{\mathrm{H}} M) = k$, so A_0 has k non-zero singular values σ_j, $j = 0, \ldots, k-1$. The singular value decomposition (*SVD*) of A_0 in reduced form can be computed as

$$V W^{\mathrm{H}} M = A_0 = V_0 \Sigma_0 {W_0}^{\mathrm{H}} \tag{2.24}$$

with orthogonal matrices $V_0 \in \mathbb{C}^{n \times k}$, ${V_0}^{\mathrm{H}} V_0 = I$ and $W_0 \in \mathbb{C}^{m \times k}$, ${W_0}^{\mathrm{H}} W_0 = I$ and diagonal matrix $\Sigma_0 = \operatorname{diag}(\sigma_0, \ldots, \sigma_{k-1})$. There exists a nonsingular matrix $S \in \mathbb{C}^{k \times k}$ such that

$$V = V_0 S\,. \tag{2.25}$$

Thus (2.24) can be transformed into $V_0 S\, W^{\mathrm{H}} M = V_0 \Sigma_0 {W_0}^{\mathrm{H}}$ which finally yields

$$W^{\mathrm{H}} M = S^{-1} \Sigma_0 {W_0}^{\mathrm{H}}\,. \tag{2.26}$$

Inserting (2.25) and (2.26) into (2.23) yields

$$A_1 = V_0 S \Lambda S^{-1} \Sigma_0 {W_0}^{\mathrm{H}}$$

which can be transformed into

$$ {V_0}^{\mathrm{H}} A_1 \Sigma_0^{-1} W_0 = S \Lambda S^{-1}\,. \tag{2.27}$$

So the matrix on the left hand side of the last equation can be diagonalized and its eigenvalues are the same as λ_j, the eigenvalues of T inside the contour C. The nonlinear eigenvalue problem has thus been transformed into a linear eigenvalue problem. The left hand side of (2.27) can be computed solely from A_0 and A_1. The right eigenvectors v_j can be calculated after diagonalization from $V = {V_0}^{\mathrm{H}} S$.

It remains to find the right dimension m so that the rank condition (2.21) is satisfied. If m is less than or equal to k the matrix A_0 will have rank m. So m is continually increased until A_0 is no longer of maximal rank, i.e. $\operatorname{rank}(A_0) < m$. Thus the rank of A_0 is no longer constrained by the dimension of M but by the rank of V and W. The rank of A_0 can be determined by the number of non-zero singular values. Algorithm 1 summarizes the described procedure.

Input: Operator T, contour C, maximum rank n, initial rank m
Output: Eigenvalues of T

while $m \leq n$ **do**
  Initialize random matrix M;
  Evaluate integrals A_0 and A_1 using T, C and M;
  Compute SVD of A_0;
  Let k = number of non-zero singular values;
  if $k < m$ **then**
    Compute $B = {V_0}^{\mathrm{H}} A_1 W_0 \Sigma^{-1}$;
    Compute eigenvalues of B;
    return eigenvalues;
  else
    Increment m;
  end if
end while

Algorithm 1: Computation of all nonlinear eigenvalues of a matrix valued operator $T(z)$ inside a contour C.

2.2.2. Discretization

The contour integrals A_0 and A_1 must be evaluated numerically. The contour C is assumed to be 2π-periodic and can be described by the smooth mapping $h : [0, 2\pi] \to C, h(0) = h(2\pi)$. Thus the integral over the contour can be transformed into an integral over the interval $[0, 2\pi]$. Applying the substitution rule for integrals yields

$$A_0 = \frac{1}{2\pi \mathrm{i}} \int_0^{2\pi} T(h(t))^{-1} M h'(t) \, \mathrm{d}t \, .$$

The simplest useable contour, a circle with radius r and center μ, is given by $h(t) = \mu + r\mathrm{e}^{\mathrm{i}t}$ with derivative $h'(t) = r\mathrm{i}\mathrm{e}^{\mathrm{i}t}$.

The integral is evaluated using the trapezoidal quadrature rule. Beyn [6] shows theoretically and experimentally that using this scheme the numerical error decays exponentially with the number of quadrature intervals, although care must be taken in selecting

the contour as the rate of convergence depends on the distance of the eigenvalues from the contour. Introducing the quadrature nodes $t_j = 2\pi j/N$, $j = 0, \ldots, N-1$ spaced $2\pi/N$ apart yields the approximation

$$A_0 \approx \frac{1}{N\mathrm{i}} \sum_{j=0}^{N-1} T(h(t_j))^{-1} M h'(t_j)\,.$$

Similarly, A_1 can be approximated as

$$A_1 \approx \frac{1}{N\mathrm{i}} \sum_{j=0}^{N-1} h(t_j) T(h(t_j))^{-1} M h'(t_j)\,.$$

Inverting $T(\lambda)$, computing the SVD of A_0 and diagonalization of B can be done by established methods. As zero singular values are not reproduced exactly due to the approximation error of the contour integrals, a tolerance must be applied when selecting the non-zero singular values in order to accurately determine the rank A_0 and therefore the number of eigenvalues. Spurious eigenvalues may be produced if the tolerance is too large.

Evaluating the integrals A_0 and A_1 is the most computationally expensive part of the algorithm, especially if evaluating the operator T is expensive and/or the iteration has to be repeated multiple times due to the rank condition. The matrices on which SVD and diagonalization is performed are expected to be small unless the number of eigenvalues in the contour is very large.

2.3. Shape Optimization

The main challenge of maximizing eigenvalues with respect to the shape of the domain D of the BVP is to find a good parametrization of the shape. This section presents an implicit description of the domain with only a small number of parameters. Then the methods for discretizing the boundary value problem and solving nonlinear eigenvalue problems from previous sections are employed in the construction of an objective function that receives the shape parameters and computes a specific eigenvalue.

First the problem is stated in detail. As has been noted in the introduction, the spectrum of interior Neumann eigenvalues for a domain D is real and discrete:

$$\sigma(D) = \{\lambda_i\}_{i\in\mathbb{N}_0}\,, \lambda_0 = 0 \leq \lambda_1 \leq \lambda_2 \leq \ldots\,.$$

The eigenvalues λ_i depend on the shape and area of the domain. For a maximum to exist, the area of D must be constant because the eigenvalues are inversely proportional

to the area [4, Proposition 1.1]. Using a constant area of one produces a constrained optimization problem

$$\begin{aligned} &\max_D \{\lambda_k(D)\} \\ &|D| = 1 \end{aligned} \tag{2.28}$$

with a fixed index $k \geq 1$, and $|D|$ denoting the area of the domain D. The eigenvalues now only depend on the shape of the domain. Using the homogeneity relations [4, Proposition 1.1]

$$\begin{aligned} \lambda_k(\alpha D) &= \alpha^{-2}\lambda_k(D) \\ |\alpha D| &= \alpha^2 |D| \end{aligned}$$

where D is scaled uniformly by some factor α, the constrained problem (2.28) can be transformed into the unconstrained problem

$$\max_D \{\lambda_k(D)\,|D|\}\,. \tag{2.29}$$

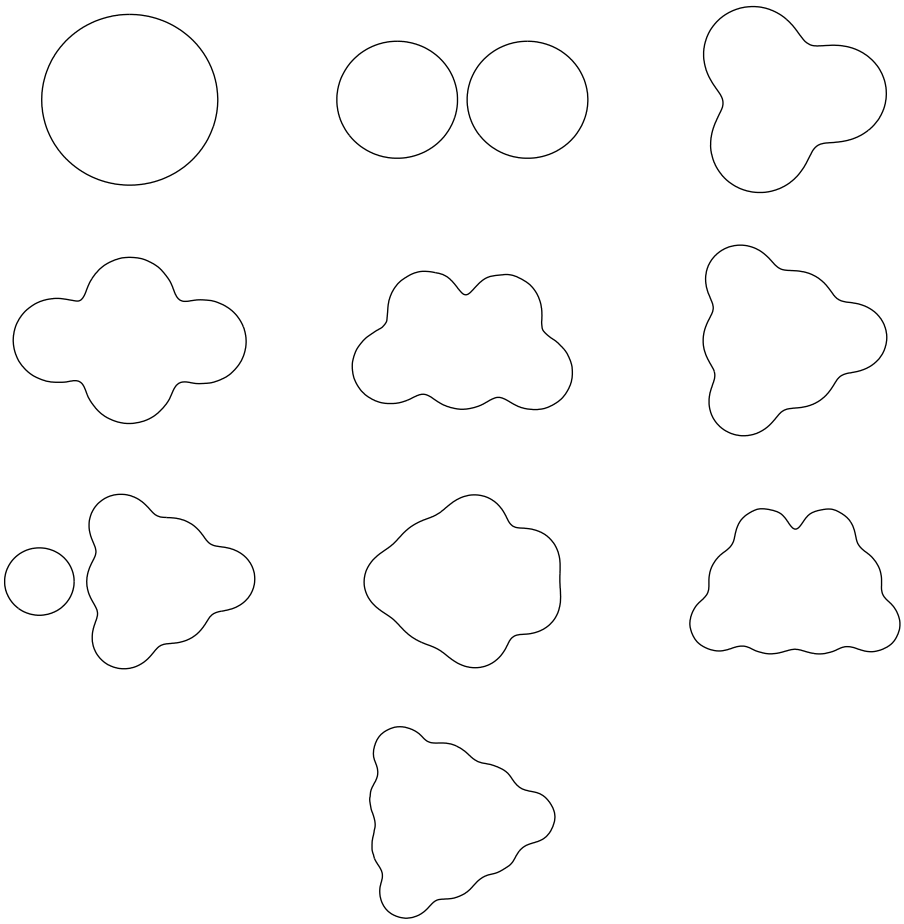
In the rest of this work, unless specially noted, the area is ignored and the term *eigenvalue* refers to the normalized eigenvalue $\lambda_k\,|D|$ that is equal to the eigenvalue λ_k for a domain of the same shape of area one.

The domain is generally allowed to be disconnected. However, the spectrum of a composite domain $\sigma(D_1 \cup D_2)$ where $D_1 \cap D_2 = \emptyset$ (disjoint union) is the ordered union of the spectrums of its component domains $\sigma(D_1)$ and $\sigma(D_2)$. Additionally, if a domain consists of m disjoint connected components, then the eigenvalues $\lambda_0 = \lambda_1 = \cdots = \lambda_{m-1} = 0$. As a consequence of these two facts, the maximum eigenvalues that can be produced using a disjoint union can be easily calculated from the maximums over connected domains [23]. So it is sufficient to consider only connected domains.

As noted in the introduction, the shape maximizers for some eigenvalues have been proven theoretically. The first eigenvalue is maximized by a disk [25, 28]. The second eigenvalue is maximized by two disjoint disks of the same size [10]. There are no such results for higher eigenvalues. It has been shown however, that not all eigenvalues are maximized by disjoint unions of disks [23]. There is ample numerical evidence for connected shape maximizers for many eigenvalues [15, 4].

For problem (2.29) to be numerically solvable, the domain D must be parametrized. A trivial parameterization can be achieved by taking the coordinates of points on the boundary as parameters. This obviously leads to a large number of degrees of freedom and therefore to large computation costs. Antunes and Oudet [4] reduce the number of parameters using truncated Fourier series. The Fourier coefficients are the parameters for optimization. Kleefeld [15] uses an implicit contour with just two parameters. This method shall be described here.

Figure 2.5.: Shape maximizers for the first ten interior Neumann eigenvalues found by Antunes and Oudet [4].



Based on the shape maximizers found by Antunes and Oudet that resemble merged disks (see Figure 2.5), Kleefeld proposes to use equipotentials. These are contours defined by the implicit function

$$\sum_{i=0}^{n_p-1} \frac{1}{\|x - p_i\|} = c \tag{2.30}$$

with base points $p_i \in \mathbb{R}^2, i = 0, \ldots, n_p - 1$ and parameter $c \in \mathbb{R}$ that is satisfied by any point $x \in \partial D$. The shape maximizer for the third eigenvalue of (2.1) consists of three circles, so three base points are used that form an equilateral triangle. Four base points that form two such equilateral triangles are used to approximate the shape maximizer for the fourth eigenvalue. The pattern of adding a point that forms another equilateral triangle can be continued to create maximizers for the fifth, sixth, and tenth eigenvalue. Figures 2.6 and 2.7 show these configurations. The shapes for the other eigenvalues are not used in this work, but they follow largely the same pattern. Some are disjoint unions of a circle and a lower equipotential. The (theoretically proven) maximizer for the first eigenvalue is a disk and is described by the equipotential with one base point. The parameter c constricts the shapes as shown in Figure 2.8. It depends on the otherwise arbitrary size of the triangles. In this work just as in Kleefeld's work, triangles with a side length of $\sqrt{3}$ are used.

Kleefeld further modified the equipotentials in order to increase flexibility, i.e. to be able to model more different shapes and to more accurately approximate the shape maximizers. The effect of additional parameter $\alpha \in \mathbb{R}$ in

$$\sum_{i=0}^{n_p-1} \frac{1}{\|x - p_i\|^{2\alpha}} = c\,. \tag{2.31}$$

can be described as sharpening the indentations without influencing the other areas of the shape. Figure 2.9 shows the effect of this parameter. The factor two before the exponent α is introduced in order to avoid computation of the square root when computing the norm.

With modified equipotentials using these two described parameters, Kleefeld was able to improve upon the results for the third and fourth eigenvalue from Antunes and Oudet. Whether the scheme can be succesfully applied to higher eigenvalues remains to be tested. It is also possible that Kleefeld's results can be improved upon by introducing more degrees of freedom. So far, the base points of the equipotentials have been arranged on a completely regular triangular grid and all base points are weighted equally. As there is no particular reason for this regularity other than visual intuition, breaking it might prove beneficial. So the base points will be allowed to deviate slightly from their regular position. The weight for the base points in the sum of potentials will be allowed

Figure 2.6.: Base points for equipotential shapes maximizing the third to sixth eigenvalue. A cross marks the origin which is also the center of symmetry.

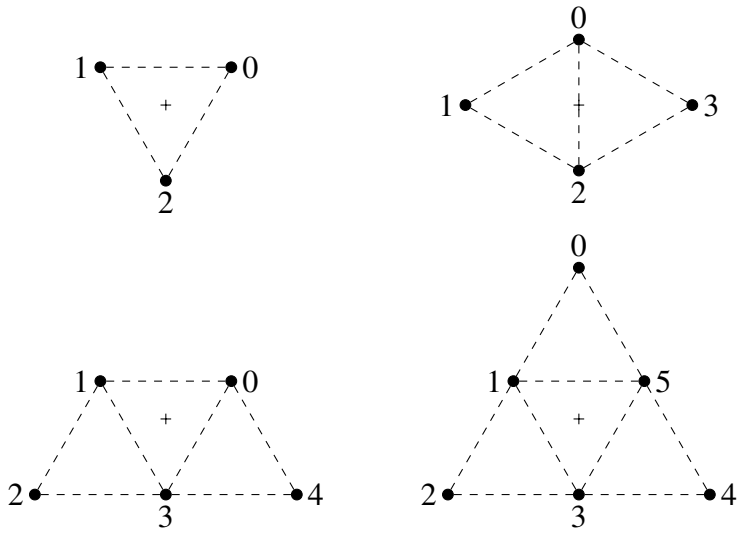


Figure 2.7.: Base points for equipotential shape maximizing the tenth eigenvalue. The origin and rotation center coincide with the central base point.

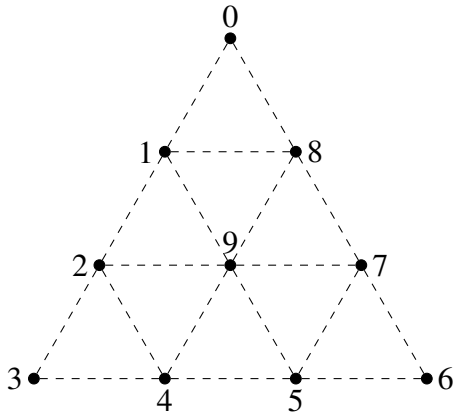


Figure 2.8.: Influence of parameter $c = 1.75, 2.00, 2.25, 2.50, 2.75, 3.00$ on the equipotential shape with triangle base.

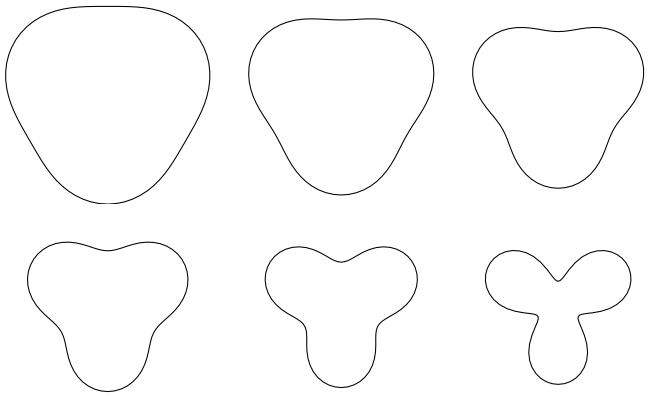
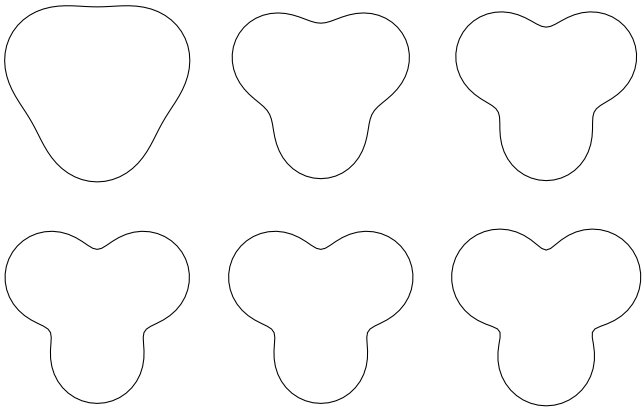


Figure 2.9.: Influence of parameter $\alpha = 0.5, 1.0, 1.5, 2.0, 2.5, 3.0$ with fixed $c = 2.0$ on the modified equipotential shape with triangle base.



to deviate from the regular weight of one. The imagined balls around base points expand as their weight increases. In general the boundary of the shape moves away from such points. This leads to the final equation

$$\sum_{i=0}^{n_p-1} \frac{1+\hat{\delta}_i}{\|x-(p_i+\hat{\epsilon}_i)\|^{2\alpha}} = c \tag{2.32}$$

where $\hat{\epsilon}_i \in \mathbb{R}^2, i = 0, \ldots, n_p - 1$ is the irregularity of position and $\hat{\delta}_i \in \mathbb{R}, i = 0, \ldots, n_p - 1$ is the irregularity of weight of base point i (c.f. indices in Figure 2.6). The eigenvalues do not depend on the absolute position of the domain in space, only on the relative position of its base points, so at least one base point should remain fixed during optimization. One weight should remain fixed to avoid a dependency between the weights and parameter c. It is always possible to normalize one weight to a value of one without changing the shape by dividing all weights and c by that weight. The degrees of freedom added by this irregularity are further reduced by a requirement that no rotation or reflection symmetries of the regular base points are broken. It must be said that at this point the conjecture that the symmetries are meaningful is unproven. However, based on the results of Antunes and Oudet, the conjecture seems reasonable and it keeps the number of parameters low.

Table 2.1 lists the remaining numbers of degrees of freedom for weights f_ϵ and position f_δ after dependencies and symmetries have been accounted for. Which base points and weights are fixed and which are free is arbitrary as long as symmetry is not broken. Figures 2.10 and 2.11 show the assignment of non-zero irregularities graphically. For one base point, there are no degrees of freedom as one posititon and one weight are fixed. For three base points, as the points are all symmetric images of each other and as one of the points is fixed, so are the others. For four points, the left and right points are not images of the top and bottom points. But the left point is a mirror image of the right point, so they must have the same weight. They can only move along the x axis so as to not break rotation symmetry and must move the same amount but in opposite directions. In total there is one degree of freedom each for weights and positions. The degrees of freedoms for the other shapes are determined in the same way.

The family of shapes with k base points described by Equation (2.32) will be referred to as E_k. In order to form an instance of the family, the parameters will be given in braces following the name of the family, e.g. $E_5\{c, \alpha, \epsilon, \delta\}$ or $E_5\{c = 1.0, \alpha = 2.0, \epsilon = (0.07, -0.1, -0.05), \delta = (0.2, -0.1)\}$ for specific numerical values. Irregularities $\epsilon = (\epsilon_0, \ldots, \epsilon_{f_\epsilon - 1})$ and $\delta = (\delta_0, \ldots, \delta_{f_\delta - 1})$ will be omitted if $f_\epsilon = 0$ or $f_\delta = 0$, respectively. Sometimes the specific parameter values of an instance will be irrelevant. In this case, the list of parameters may be omitted. So E_k can refer both to the family and an instance of the family if the context is unambiguous. The shape E_k will be used to maximize the eigenvalue λ_k. So the optimization problem (2.29) can be more accurately written as

$$\max_{c,\alpha,\epsilon,\delta} \{\lambda_k(E_k\{c, \alpha, \epsilon, \delta\})\, |E_k\{c, \alpha, \epsilon, \delta\}|\}\,. \tag{2.33}$$

Figure 2.10.: Assignment of free irregularities of position ϵ_i and weight δ_i to base points of shapes maximizing eigenvalues three through six. A cross marks the origin which is also the center of symmetry.

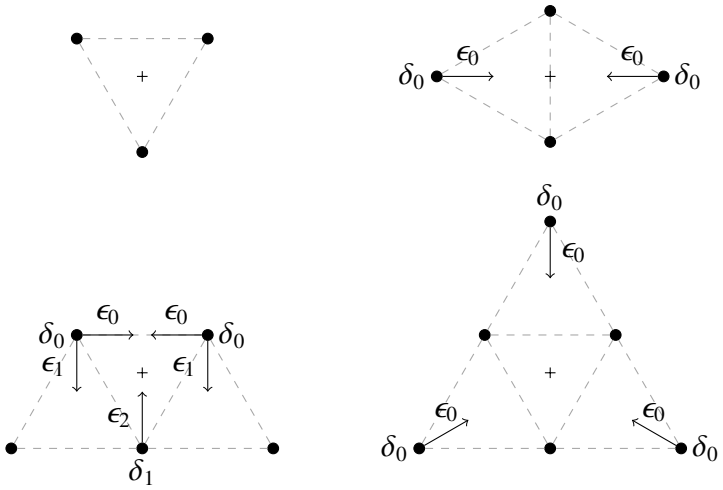


Figure 2.11.: Assignment of free irregularities of position ϵ_i and weight δ_i to base points of the shape maximizing the tenth eigenvalue. The origin and rotation center coincide with the central base point.

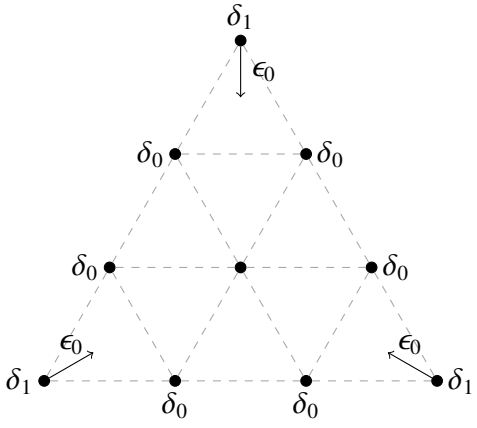


Table 2.1.: Degrees of freedom of weight f_ϵ and position f_δ of equipotential base points per number of base points k without breaking symmetries. Non-zero and symmetrically independent irregularities $\hat{\epsilon}_i$ and $\hat{\delta}_i$ derived from free irregularities $\epsilon_i, i = 0, \ldots, f_\epsilon - 1$ and $\delta_i, i = 0, \ldots, f_\delta - 1$ are given in brackets, the rest are zero or determined through symmetry.

k	f_ϵ	f_δ
1	0 [−]	0 [−]
3	0 [−]	0 [−]
4	$1\left[\hat{\epsilon}_1 = \begin{pmatrix}\epsilon_0 & 0\end{pmatrix}^\top\right]$	$1\left[\hat{\delta}_1 = \delta_0\right]$
5	$3\left[\hat{\epsilon}_1 = \begin{pmatrix}-\epsilon_0 & -\epsilon_1\end{pmatrix}^\top, \hat{\epsilon}_3 = \begin{pmatrix}0 & \epsilon_2\end{pmatrix}^\top\right]$	$2\left[\hat{\delta}_1 = \delta_0, \hat{\delta}_3 = \delta_1\right]$
6	$1\left[\hat{\epsilon}_0 = \begin{pmatrix}\epsilon_0 & 0\end{pmatrix}^\top\right]$	$1\left[\hat{\delta}_0 = \delta_0\right]$
10	$1\left[\hat{\epsilon}_0 = \begin{pmatrix}\epsilon_0 & 0\end{pmatrix}^\top\right]$	$2\left[\hat{\delta}_0 = \delta_1, \hat{\delta}_1 = \delta_0\right]$

The method for discretizing the BVP presented in Section 2.1 requires an even number n of points on the boundary. In order to generate these points, equation (2.31) is transformed into polar coordinates. Equidistant angles $\phi_i = 2\pi i/n, i = 0, \ldots, n-1$ are chosen and a root finding algorithm is applied to find the unique radius r_i for each ϕ_i such that the transformed equation is satisfied. The boundary integral method, specifically the interpolation of the boundary, works more conveniently with Cartesian coordinates, so the pairs (r_i, ϕ_i) are transformed back as $x_i = (r_i \cos(\phi_i), r_i \sin(\phi_i))$. It would be possible to calculate the normals at these points exactly by differentiating the implicit equation as done in [15]. But as described in Section 2.1.3, it is necessary to approximate the normals so they are orthogonal to the approximated boundary.

In order to calculate the area of the domain it is approximated as a polygon with $n_q \gg n$ points $q_i, i = 0, \ldots, n_q - 1$ that are generated using quadratic interpolation of the n boundary discretization points. Note again that the contour is closed, so $q_0 = q_{n_q}$. The area of a non-self-intersecting polygon is the sum of the area of the triangles spanned by two adjacent points and the origin. The area of each triangle is added or subtracted based on the sign of the angle at the origin so the formula works for general polygons

anywhere in the plane. Thus the area is given by the formula [29, p. 206]

$$\begin{aligned} A &\approx \frac{1}{2}\left|\sum_{i=0}^{n_q}\begin{vmatrix} q_{i,1} & q_{i+1,1} \\ q_{i,2} & q_{i+1,2}\end{vmatrix}\right| \\ &= \frac{1}{2}\left|\sum_{i=0}^{n_q}(q_{i,1}q_{i+1,2} - q_{i,2}q_{i+1,1})\right| \end{aligned}$$

which uses the determinant of the matrix of coordinates to calculate the area of a triangle and also applies the correct sign.

With this, all components are available that are necessary to construct the objective function for the optimization of eigenvalue λ_k. The parameters $c, \alpha, \epsilon, \delta$ are inputs to the function. They fully define the boundary of the domain $D = E_k\{c, \alpha, \epsilon, \delta\}$. The first step of evaluating the function is to generate n points on the boundary according to the process that has just been described. With the domain D so fixed, the discretization of the BVP is a holomorphic function $T(\kappa)$ that takes the wave number $\kappa \in \mathbb{C}$ as input and returns an $n \times n$ complex matrix. Thus Beyn's method as described in Section 2.2 can be applied to compute a subset of the spectrum of T. It is not necessary to compute all or even many eigenvalues of T. The results of previous works or experiments can be used to determine a circle with small radius (the contour C for Beyn's algorithm) so that κ_k is contained in the computed subset. Using a similar heuristic, κ_k can be selected from the subset. The area of the domain is computed. The eigenvalue $\lambda_k |D| = \kappa_k^2 |D|$ is the result of the objective function. It may be necessary to multiply with -1 as most generic optimization algorithms search for a minimum instead of a maximum.

The gradient

$$\left(\tfrac{\partial}{\partial c}\lambda_k(E_k\{c,\alpha,\epsilon,\delta\}) \quad \tfrac{\partial}{\partial \alpha}\lambda_k(E_k\{c,\alpha,\epsilon,\delta\}) \quad \tfrac{\partial}{\partial \epsilon}\lambda_k(E_k\{c,\alpha,\epsilon,\delta\}) \quad \tfrac{\partial}{\partial \delta}\lambda_k(E_k\{c,\alpha,\epsilon,\delta\})\right)$$

cannot be trivially formed. Thus a method for unconstrained optimization problems that does not require derivatives of the objective function (e.g. the Nelder-Mead simplex method [20]) is employed.

3. Implementation

A program was developed that implements the methods for eigenvalue optimization described in Chapter 2. As the computation of eigenvalues is very expensive especially for the CPU, some effort was put into performance optimization. A few aspects of this work will be highlighted in this chapter. The focus will be on parallelization. As will be shown the numerical methods are well suited for distributed computation. The main target system is the JURECA cluster at Forschungszentrum Jülich [14], also see Appendix C.

Mathematical code is usually implemented most easily in a language like Python or specialized mathematics environments like MATLAB. But one evaluation of eigenvalues to acceptable precision using existing MATLAB code requires many hours or even more than a day. Optimization with two parameters takes weeks, more than two parameters are completely infeasible. For maximum performance, a systems programming language is desirable. The language C was chosen for wide availability of libraries and resources. In principle it would be possible to export the solvers into a library that can be integrated into any other language. Even before parallelization, the new code is faster than the existing MATLAB code by about a factor of 50, which makes up for longer development time.

3.1. Framework

As much work as possible was avoided by employing existing numerical libraries. Specifically robust routines for the following tasks were required:

- Evaluating Hankel function of low integer order for complex arguments (see Section 2.1)
- Quadrature to evaluate the matrix A_k (see Section 2.1)
- Basic and advanced linear algebra for Beyn's method (see Section 2.2)
- Solving nonlinear equations for generating points on the contour given by an implicit function (see Section 2.3)
- Nonconstrained optimization without derivatives (see Section 2.3)

The GNU Scientific Library (GSL) [9, 11] offers numerical routines for a large array of problems and covers all of the points above except for evaluating Hankel functions where it only supports real valued arguments. Due to being open source, it is also easily available and therefore widely used and tested. It has utilities for complex arithmetic and flexible matrix and vector data types. This makes GSL a good framework for the program.

3.2. Hankel Function

The Hankel function in the integrand of (2.11) is the most basic building block of the program. However, robust implementations for complex arguments are rare. The Fortran code `ZBESH` by Amos [2, 21] was compared with function `nag_complex_hankel` from NAG C Library [26]. Both have proven to be reliable in the domain that is relevant for this work. Amos' code is used by SciPy, among others. The routines allow specifying kind and order of the Hankel function. Only $H_1^{(1)}$ is tested as no others are required for this work. Table 3.1 shows the results of a benchmark using Google's benchmark framework [5]. The considerably faster Amos routine was chosen. The reason for the NAG routine being slower despite being based on Amos' code was not further examined. One reason may be that compiling the Amos code into the program allows for better inlining and optimization, specifically whole program optimization where optimization is deferred to the linker stage of the build process.

Table 3.1.: Benchmark of Hankel function evaluation.

Routine	Avg. Time	#Iterations
`ZBESH`	322 ns	2047108
`nag_complex_hankel`	768 ns	912209

3.3. Evaluating the Discrete Integral Operator

A quadrature routine is required to evaluate the elements of matrix A_κ, i.e. the discretized integral operator (see Section 2.1.2). GSL contains integration routines based on QUADPACK. General integrands as well as integrands with singularities or special forms are supported. GSL also provides additional routines to increase the number of integrals that can be handled. As the integrands in this problem are well behaved, the integrals can be solved quickly to within 10^{-10} absolute and relative tolerance by general adaptive 15 point Gauss-Kronrod rule. Complex quadrature routines are not available,

so real and imaginary parts of the integral are evaluated separately. A combined routine might be able to save some evaluations of the integrand.

In the general case, implementing the integrand is straightforward using an available routine for evaluating Hankel functions. However, the integrands contain a singularity for those matrix elements where the collocation point is in the integration interval (see Section 2.1.3). Collocation point x_i causes a singularity in integration interval k ...

- ... at $t = 0$ if i is even and $k = i/2$. (x_i is the start point of the interval)
- ... at $t = 1$ if i is even and $k = i/2 - 1$. (x_i is the end point of the interval)
- ... at $t = 0.5$ if i is odd and $k = (i - 1)/2$. (x_i is the mid point of the interval)

Evaluating the integrand at the singularity is not possible. Close to the singularity cancellation prevents accurate evaluation. Experiments (e.g. Table 3.2) suggest replacing the integrand with a limit approximation if the absolute value of the argument $\kappa \|x_i - \tilde{g}_k(t)\|$ falls below 10^{-6}. Around that value, the real part of the integrand begins to oscillate and diverge.

Table 3.2.: Numerically evaluating the integrand (denoted f) near the singularity t_0. The analytical limit is $f^* := lim_{t \to t_0} f(t) \approx -0.012429678092$. The boundary of the unit disk is discretized with 80 points. For collocation point x_0 the singularity in interval $k = 0$ is at $t_0 = 0$. Wavenumber $\kappa = 1.0$.

$\|t - t_0\|$	$\|x_0 - \tilde{g}_0(t)\|$	$f(t)$	$\|f(t) - f^*\|$
$1.00 \cdot 10^{-1}$	$1.57 \cdot 10^{-2}$	−0.00895519−0.00000174i	$3.47 \cdot 10^{-3}$
$1.00 \cdot 10^{-2}$	$1.57 \cdot 10^{-3}$	−0.01205939−0.00000002i	$3.70 \cdot 10^{-4}$
$1.00 \cdot 10^{-3}$	$1.57 \cdot 10^{-4}$	−0.01239242+0.00000000i	$3.73 \cdot 10^{-5}$
$1.00 \cdot 10^{-4}$	$1.57 \cdot 10^{-5}$	−0.01242595+0.00000000i	$3.73 \cdot 10^{-6}$
$1.00 \cdot 10^{-5}$	$1.57 \cdot 10^{-6}$	−0.01242915+0.00000000i	$5.31 \cdot 10^{-7}$
$1.00 \cdot 10^{-6}$	$1.57 \cdot 10^{-7}$	−0.01248435+0.00000000i	$5.47 \cdot 10^{-5}$
$1.00 \cdot 10^{-7}$	$1.57 \cdot 10^{-8}$	−0.02493252+0.00000000i	$1.25 \cdot 10^{-2}$
$1.00 \cdot 10^{-8}$	$1.57 \cdot 10^{-9}$	0.98552244+0.00000000i	$9.98 \cdot 10^{-1}$
$1.00 \cdot 10^{-9}$	$1.57 \cdot 10^{-10}$	−91.17768000+0.00000000i	$9.12 \cdot 10^{1}$

3.4. Beyn's Integral Method

Apart from evaluating the matrix valued operator that is provided as a function pointer, Beyn's integral method requires solving linear systems and computing matrix products and eigenvalues as well as performing singular value decomposition (SVD). The BLAS

(**B**asic **L**inear **A**lgebra **S**ubprograms) [7] and LAPACK (**L**inear **A**lgebra **Pack**age) [3, 18] APIs are the de-facto standard for these tasks in high performance computing. Implementations of these interfaces include open source libraries like OpenBLAS [22] as well as highly optimized proprietary libraries like the Intel Math Kernel Library (MKL) [13]. Due to the common interface definition, the implementation can be switched freely to use the fastest that is available on the current system (usually MKL on Intel CPUs if available). GSL provides convenient wrappers around BLAS routines based on its own matrix and vector data types. Where GSL is missing such abstractions (e.g. GSL only wraps BLAS, not LAPACK), custom ones can be created easily to improve readability and consistency of the code, e.g. a function `void matrix_complex_eigenvalues (gsl_matrix_complex* A, gsl_vector_complex* s)` that wraps the LAPACK routine `zgeev` (assuming eigenvectors are not required).

The algorithm requires a random matrix. The randomness does not need to be very strong, so basically any random number generator will suffice. A constant seed is used to ease debugging. If the matrix size is limited, the whole matrix could be static, but the performance gains will almost certainly be negligible.

In order to reduce the number of memory allocations and the amount of memory used, matrices are reused between eigenvalue evaluations as well as during the algorithm. Table 3.3 shows the matrices that are involved in the implementation of the algorithm and their lifetime based on the computation steps of the algorithm (additional vectors required for SVD and the final result are handled similarly). The computation of matrix B actually consists of multiple matrix multiplication steps that are ignored here to illustrate the principle. At most one $\mathbb{C}^{n\times n}$ and four $\mathbb{C}^{n\times k}$ matrices are required where n is the dimension of the matrix valued operator and k is the current rank. Smaller matrices are stored in the memory of bigger matrices where necessary as $n > k$. To save memory, it is assumed that the initial rank k is well chosen and no rank iteration is required. Otherwise reallocation with geometrically increased capacity is used like in typical dynamic array implementations to balance the number of allocations and memory usage.

3.5. Generating the Discrete Boundary

A routine is required to solve the modified equipotential equation (2.31) in polar coordinates (r, ϕ) for r at fixed angle ϕ. Instead of a single high level routine, GSL provides the low level building blocks (initialization, iteration, stopping criteria) for custom root solvers. From this an abstract routine was created for the program. GSL implements both bracketing and Newton-type solvers for non-linear equations. It would not be difficult to provide a derivative for the equation in order to use a Newton solver. Bracketing algorithms are generally more simple and reliable.

Assuming the parameters are chosen so that the shape is simply connected and given $r > 0$, an initial bracket can be found naively by probing the positive number line. A

Table 3.3.: Lifetime of matrices during Beyn's algorithm.

		Computation step				
Matrix	Size	$A_{0,1}$	SVD	B	EVs	Note
T	$n \times n$	×				-
M	$n \times k$	×				random matrix
TMP_1	$n \times k$	×				temporary storage for $T^{-1}M$
A_0	$n \times k$	×	×			-
A_1	$n \times k$	×	×	×		-
V/V_0	$n \times n$		×	×		-
W/W_0	$k \times k$		×	×		-
TMP_2	$k \times k$			×		temporary storage for ${V_0}^{\mathrm{H}}A_1$
B	$k \times k$			×	×	-

heuristic like $\max_{0 \le i \le n_p - 1} \|p_i\| + c^{-2\alpha}$ might be used as a decent initial guess. That formula is derived from the implicit function by approximating a point beyond one of the base points where the influence of the other base points is negligible.

The implicit function has a singularity at the base points, i.e. $x = p_i$ for any $i = 0, \ldots, n_p - 1$. The potential is infinite at these points. These poles may be hit by the bracketing algorithm if in polar coordinates the angle of x is equal to the angle of one of the base points. Choosing initial brackets to avoid the poles is not trivial, but as the sign does not change at the poles, infinity can be replaced with any large positive value without compromising the solver.

A few possible performance improvements (Newton solver, heuristic initial guess) have been mentioned. However, unless a scheme with higher algorithmic complexity is used to distribute the points along the boundary, generating the points is a computationally insignificant part of the program, so reliability and simplicity is preferred over faster convergence and potentially better performance. For the same reason, an arbitrary low tolerance of 10^{-10} is used as a stopping criterion instead of a more fine tuned value that is just sufficient for convergence of the final eigenvalues.

3.6. Optimization

The originally constrained optimization problem was transformed into an unconstrained problem as discussed in Section 2.3. The derivative of the objective function is not available. GSL provides an implementation of the Nelder-Mead (or downhill simplex) method for unconstrained optimization without derivatives. As for root solvers, an abstract minimization routine is built from low level building blocks. Both the change of the residual and the size of the simplex are controlled to check for termination.

Parameters that are not inputs to the objective function have to be chosen correctly. From previous works [4, 15] some approximate eigenvalues are known. Others can be found by experimentation. Knowing the approximate normalized eigenvalue $\lambda_k = \kappa_k^2 |D|$ and the area $|D|$ allows choosing the contour for Beyn's algorithm so that the eigenvalue is inside the contour and as close to the center as possible to optimize precision [6, Corollary 4.8]. Beyn's algorithm may still return more than one eigenvalue. The eigenvalues are generally separated well enough so that the correct value can be heuristically selected without computing the whole spectrum by probing the real axis. For these heuristics to work it is necessary to start optimization close enough to the optimum. Some experimentation may be required to find appropriate starting values for the shape parameters.

For example, it is known from previous works that the fifth eigenvalue is almost certainly less than 60. Thus if values for shape parameters are known where the sixth eigenvalue is greater than 60 and assuming the step size for the optimization are not too big, the sixth eigenvalue in each iteration of the optimization can be reliably selected from the output of Beyn's algorithm (after normalization) by choosing the first value above 60. A useable contour for Beyn's algorithm can be determined by reversing normalization. The sixth eigenvalue is at first guess expected to be somewhere between 60 and 67, so a circle that encloses both $\sqrt{60/|D|}$ and $\sqrt{67/|D|}$ could be used.

3.7. Parallelization

A very fine discretization, specifically a high number of points to approximate both the boundary of the domain of the PDE and the contour for Beyn's integral method, is required in order to achieve accurate results. This makes the computation fairly slow and expensive. Previous sections already provided some insight into improving single threaded performance. However, modern systems offer increasing number of CPU cores. This section will describe the parallelization scheme that allows the program to use these resources efficiently.

3.7.1. Performance Analysis

To be able to implement any performance optimization, it is crucial to analyze the runtime profile of the program and identify hotspots. Table 3.4 shows the profile of a single evaluation of eigenvalues, i.e. of the objective function (the optimization algorithm itself is outside the scope of this work). The profile clearly shows that almost all of the time is spent on evaluating contour integrals in Beyn's method. Evaluating the matrix valued operator consists almost exclusively of evaluating the integrals for each matrix element. Therefore, parallelization of these hotspots covers a significant portion of the program.

Table 3.4.: Runtime profile of Eigenvalue computation with Beyn contour integral discretization $N = 48$ and number of collocation points $n = 1152$. Lists the time spent inside each node of the call tree in percentage of total time and percentage of parent node.

Function	% of Total	% of Parent
Compute eigenvalues for one shape	100	100
▷ Discretize shape, solve nonlinear equations	< 0.1	< 0.1
▽ Beyn's method	99.9	99.9
▽ Compute integrals $A_{0/1}$ (loop over N)	99.9	99.9
▽ Evaluate matrix (loop over n^2)	98.8	98.8
▷ Quadrature of matrix elements	98.8	99.9
▷ Memory accesses and symmetries	< 0.1	< 0.1
▷ Invert matrix, multiply and add	1.2	1.2
▷ Other lin. alg. operations (SVD, lin. EVs, etc.)	< 0.1	< 0.1

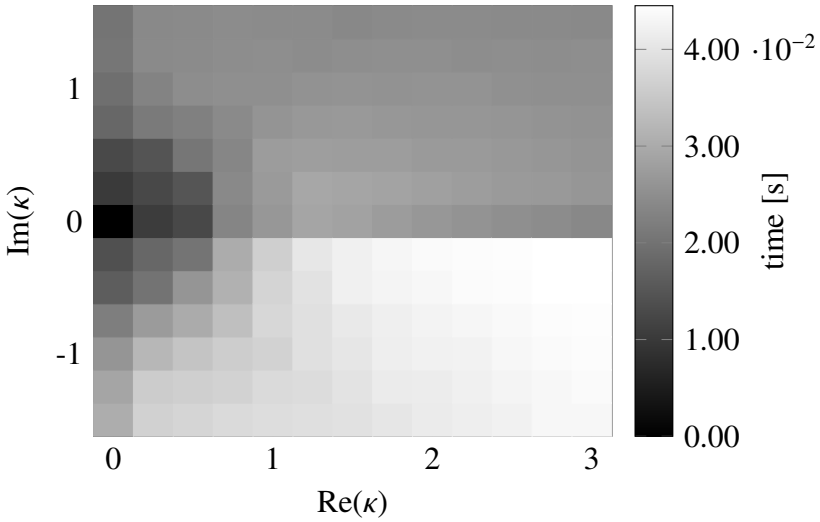
Two approaches will be discussed, parallelization of Beyn's method and parallelization of single matrix evaluations.

3.7.2. Parallelization of Beyn's Contour Integral Method

The contour integrals in Beyn's method are evaluated using a trapezoidal rule with N fixed nodes, which is basically a sum of the matrix valued integrand evaluated at each node (see Section 2.2.2). The evaluations of the summands, which consists mostly of evaluating the matrix valued operator and to a lesser extent of solving linear systems, are completely independent of each other. Thus a large part (>99%) of the program can be trivially parallelized here. Domain decomposition is performed, where each task is assigned some of the nodes, evaluates the corresponding summands and sums them up locally. For this, each task requires the random matrix. This requires no synchronization between tasks as long as each task uses the same seed (can be static) for the random number generator. Each task also requires all the data necessary to evaluate the operator, mostly the discretized boundary. Finally, each task requires memory for the random matrix, the evaluated operator and the local sum. The local results are reduced to get the final result.

For best scaling, each task must be assigned an equal amount of work. During Beyn's method, the matrix is evaluated at different complex values κ along the chosen contour. As discussed previously, the contour used here is a circle centered on the positive real axis of the complex plane. Figure 3.1 shows the time required for one evaluation of the matrix for different κ (0 is omitted as the Hankel function is not defined there). The area

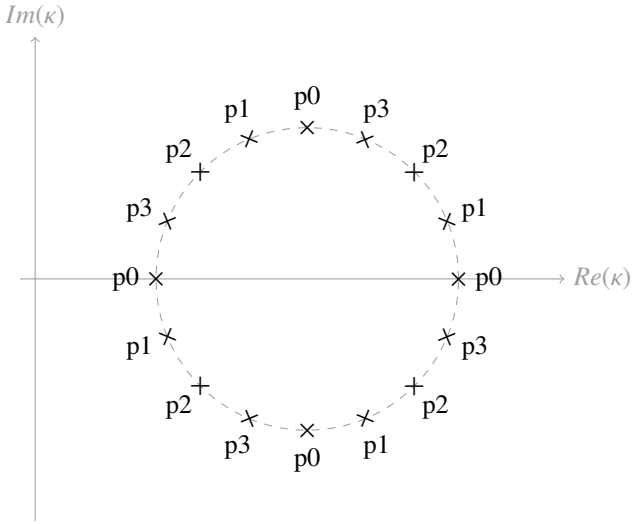
Figure 3.1.: Times required to evaluate the whole matrix for different wavenumbers κ. The origin is omitted because the matrix is undefined for $\kappa = 0$.



around the origin stands out, but it is avoided anyway as it is numerically difficult due to the singularity of the Hankel function and does not contain any eigenvalues of interest. There is a clear jump at the real axis. The required time for κ with negative imaginary part is up to two times greater than for non-negative imaginary parts because evaluating the Hankel function is more expensive. Other than that, the domain around the real axis is quite smooth. Assuming the number of tasks can be chosen so that it divides N, cyclical distribution of summands will balance out the workload well enough. There can still arise a slight imbalance because for even N there are two nodes that lie directly on the real axis where evaluation is cheap. Thus there are more cheap summands than expensive ones. Figure 3.2 shows the cyclical distribution of 16 summands to four tasks. Process 0 is assigned one more cheap summand and one fewer expensive summand. Optimal assignment is supposedly an NP-complete problem. Trying to improving on the cyclical heuristic is unlikely to be worth the effort.

Considering Amdahl's Law [1], where the achievable speedup is constrained by the part of the program that is not parallelized, strong scaling is expected for this first approach. However, workload imbalance will have a negative impact. And the number of cores that can be engaged this way is constrained by the (typically not very high) number of summands N. There is no communication necessary during computation, but the reduce operation at the end introduces some overhead. Due to these restrictions, analysis of the matrix evaluation itself is warranted.

Figure 3.2.: Cyclical distribution of 16 nodes to four parallel tasks (p0-3) for contour integrals in Beyn's method. A circle is used as the contour (dashed line).

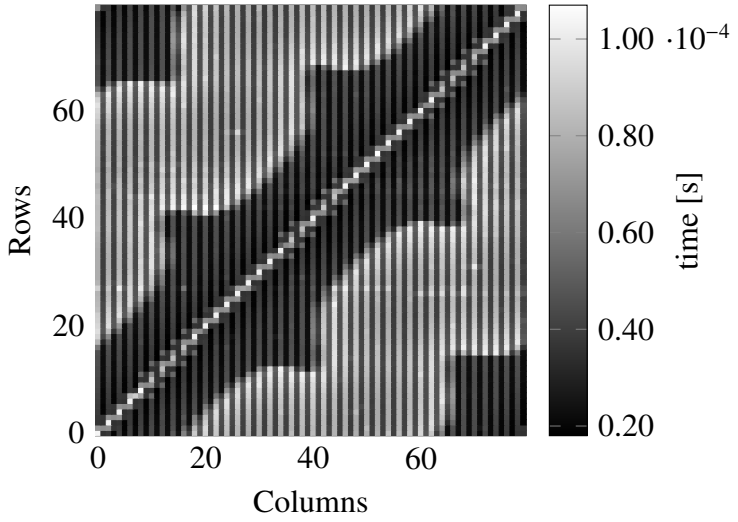


3.7.3. Parallelization of Matrix Evaluation

The evaluations for each matrix element are again completely independent and therefore trivially parallelizable. Each task is assigned a set of matrix elements to evaluate. This requires each task to know the discretized boundary (or at least those points of the boundary that correspond to the assigned matrix elements). The quadrature routine uses some small amount of memory in each task. The part of the program that is not parallelized is only slightly larger than for the first approach, so strong scaling is expected for this approach as well. There is no communication necessary while evaluating each matrix element, but the results have to be gathered into one matrix for further processing.

Figure 3.3 shows the time required to evaluate each element of the matrix. The modified equipotential with three base points was used as the shape of the boundary. A coarser than usual discretization was chosen for the diagram and symmetry was not exploited, so every matrix element was evaluated. Multiple patterns are noticeable. As discussed in Section 2.1.2, elements in columns with even index consist of two integrals whereas elements in columns with odd index consist of only one integral. This causes regular vertical stripes in the diagram. The matrix elements where the integrand contains a singularity lie on the diagonal or the first sub- and superdiagonal. There is a wide band around the three diagonals where computation is faster, although the diagonals themselves are a bit slower due to the treatment of the removeable singularity. By distributing matrix elements by row, both these effects will be balanced out. The shape of the band around the diagonal follows the shape of the boundary of the domain. Points

Figure 3.3.: Times required to evaluate each matrix element.



near the inward bulge of the boundary are closer together as points are generated with equidistant angles. Cyclical row distribution is used to counteract this. A few isolated rows of the matrix contain many elements that are slower than the surrounding rows. This last effect has yet to be fully explained but a cyclical distribution should be able to smooth this out as well. It is trivial to choose a matrix size that is divided by the number of tasks.

Exploiting the symmetries introduces a complication. Only a segment of the boundary is considered now. Unlike the whole boundary, this segment is not cyclical. Matrix elements corresponding to both end points of the segment have to be explicitly evaluated. Symmetry can be used to decrease either the number of rows or the number of columns in the matrix to the number of points in the segment. As the segment must include a whole number of intervals, the number of points in the segment will always be odd. If the number of columns is reduced through symmetry, the rows will be as easy to distribute as before but rows will require vastly different amounts of work as e.g. the diagonal band is not included in some rows. If the number of rows are reduced, balanced row distribution is complicated. In the latter case, a simple fix is available. Cyclical row distribution is used for all but the last row. The last row is distributed by columns. Block-cyclical column distribution is used on the last row. Block size must be even to counteract the effect of alternating between interval mid and end points. Smaller blocks improve workload balance while increasing overhead. Optimal block size depends on the implementation.

The approach of parallelizing matrix evaluations is more fine-grained than the first one. This generally introduces more overhead due to task management. Due to cyclical row distribution, the number of CPUs that can be enganged is theoretically limited by

the number of rows n/s where n is the number of collocation points and s is the degree of symmetry. In case the number of workers crosses that boundary, rows could be partitioned. However, this contingency is not considered in this work as the number of rows is generally large enough. Due to the high degree of parallelism and sparse communication, strong scaling is expected. A high degree of symmetry reduces the overall work to be distributed which has a positive effect on total runtime but should have a negative effect on scaling.

3.7.4. Realization

The two approaches can be combined in order to compensate each approach's disadvantages. This scheme is expected to scale well to a large number of cores. For realizing the scheme, the choice of framework is the most important. Generally there are three different types of parallelization framework available:

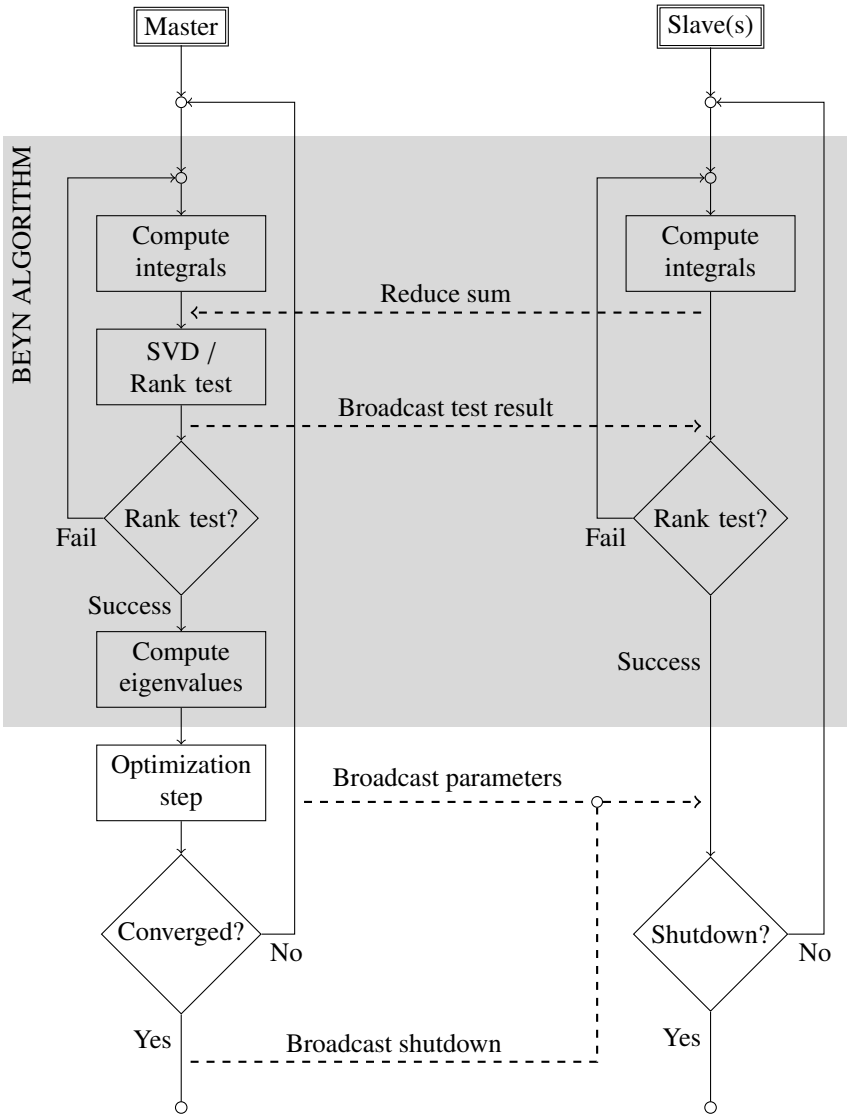
- Shared memory (e.g. threads, OpenMP)
- Distributed memory (usually MPI)
- GPU (e.g. CUDA, OpenCL)

The fine-grained approach of parallelizing the matrix evaluation was implemented using OpenMP. As the matrix is already naturally processed row by row for cache efficiency reasons, implementation is almost as simple as a single `omp parallel for` directive. In case of symmetric shapes, the last row is isolated from the loop and parallelized on its own (but inside the same OpenMP parallel region to reduce overhead). Static scheduling with block size one is chosen for the row distribution to achieve the discussed workload balancing. Static scheduling is used for the column distribution of the last row as well. To avoid false sharing, block size is chosen so that blocks are aligned to cache lines. The 64 B L1 cache lines of the target system fit four double precision complex numbers, so block size should be a multiple of that number. Experiments suggest eight might be optimal but the difference is barely measurable as the computation dominates memory accesses.

A reduction of a status code is performed at the end of the parallel region for handling errors. Other than that no communication or synchronization is necessary, so overhead is low. But the number of workers is limited by the number of cores in a single computer. Note that BLAS/LAPACK operations are already implicitly shared memory parallel in any efficient implementation. Therefore, effectively the whole evaluation of the integrand in Beyn's algorithm for one value κ is parallelized.

In principle, the structure of the computation with a large number of small and completely independent units of work is a good fit for GPUs. However, implementation in

Figure 3.4.: Communication scheme for the MPI parallelization. A master process and one or more slaves cooperate in the evaluation of eigenvalues by distributing the computation of contour integrals in Beyn's method. The processes communicate (dashed lines) using collective operations only.



e.g. CUDA is much more effort, especially because implementations for quadrature and Hankel functions are hard to find.

Beyn's method is parallelized using MPI to increase the number of available cores by using multiple compute nodes of a cluster. This also increases the available memory to offset the fact that each task requires memory for multiple matrices. Figure 3.4 shows the fairly simple communication scheme. There is a master task and a number of slaves. The master performs the actual optimization. Initial parameters are known to each process. Using the parameters, generating the discretized boundary is very quick, so there is no need to broadcast the points. The first rank iteration of Beyn's algorithm is started. Each slave and the master compute the local results for the contour integrals for the values κ it was assigned. Two sum reduce operations produce the final contour integrals. The master performs singular value decomposition and the rank test and signals the slaves to continue with the next rank iteration if necessary. Otherwise the master computes the eigenvalues that are the optimization objective, performs the optimization step and broadcasts the parameter values for the next optimization iteration. These steps repeat until the optimization converges, at which point the master broadcasts special values (e.g. NaN) as parameters to signal the slaves to shut down.

Only collective operations (`MPI_Broadcast`, `MPI_Reduce`) are used in the scheme. Error handling has been omitted from the diagram for simplicity. In the actual implementation, `MPI_Allreduce` with operand `MPI_MAX` is used to exchange a status code instead of broadcasting the rank test result. So almost no additional communication is necessary.

3.7.5. Experiments

Various experiments were performed on the JURECA cluster to analyze the quality of the parallelization schemes and their realization. The two approaches are analyzed individually and in conjunction. All the experiments were performed with discretization parameters $n = 1152$ and $N = 48$. This constitutes a realistic workload with high precision results (see Section 4.1). The parameter n is chosen so that the maximum number of cores of a compute node (24 physical, 48 virtual) divides n/s for all degrees of symmetry $s \in \{1, 2, 4, 6\}$ that are relevant in this work so that results are comparable and the effect of symmetry on scaling can be deduced. The same shape of symmetry 6 is used for all experiments, but symmetry is disabled for some. Each experiment consists of one eigenvalue evaluation, i.e. approximately one objective function evaluation. Each measured time is the average of five measurements performed in a single run of the program. Initial setup for MPI (e.g. `MPI_Init`), OpenMP (e.g. thread creation) and the application (e.g. I/O, allocation of reusable memory) is excluded from measurment. OpenMP threads are warmed up before measurement to avoid the typical jitter in the first parallel region. One time setup is small enough to amortize over a full optimization run

because the objective function is called many times. The standard metrics of speedup

$$S(p) = \frac{\text{Runtime of sequential program}}{\text{Runtime of parallelized program with } p \text{ Threads/Processes}}$$

and efficiency

$$E(p) = \frac{S}{p}$$

are used to interpret the results.

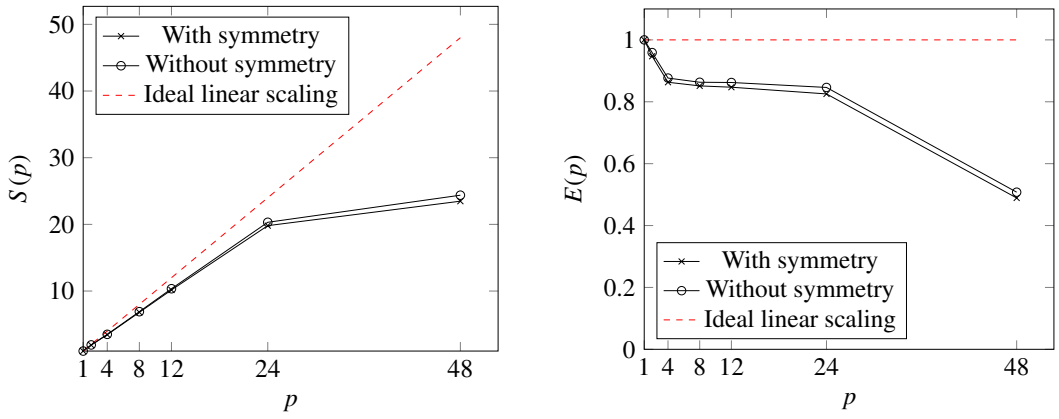
Table 3.5 and Figure 3.5 show the results of measuring the runtime of the OpenMP parallelized program with increasing number of threads. As expected, the program scales well up to the maximum number of threads. Even the use of simultaneous multithreading is beneficial with a speedup of about 1.2 for 48 threads compared to 24 threads. There is an initial drop in efficiency, but scaling is almost linear after that. This suggests that the workload is well balanced. A trace of the program with 12 threads generated using Score-P [16, 24] and analyzed in Vampir [27] shows an average of around 1.6 % of the time inside the parallel region that is spent waiting for the other threads to complete their work. About 1.5 % of the total program runtime are spent outside of parallel regions. If symmetry is not exploited, there is more work to be done, so scaling is slightly better. It is close enough that it is ignored in the analysis of the remaining performance experiments.

The scaling of pure MPI applications (i.e. no multithreading) changes when the processes are run all on the same machine or each on a different node of a cluster. Applications that are heavy on communication can benefit from avoiding the network layer and sending data in memory only. But processes on the same machine contend for resources in the operation system or hardware (e.g. CPU cache). Table 3.6 and Figure 3.7 show the scaling of the MPI parallelized eigenvalue evaluation in both these cases (the number of nodes was limited so as to not use up too many resources on the cluster). It was already mentioned that the scheme is low on communication. Therefore, it is not particularly surprising that scaling is better when each process runs on its own node. Of course a lot of CPU resources are wasted that way, so it is still not recommended. But the very strong scaling gives an indication that the scaling of the program is at this point more limited by the system than by the communication scheme and implementation. At 48 nodes, scaling drops of significantly as one node calculates only one summand and cyclical distribution does not balance the workload anymore. Scaling of multiple processes on one node is very similar to the OpenMP parallelization up to 24 cores. SMT is used much more efficiently by the OpenMP version, likely because data can be shared by the virtual cores but also because each process only evaluates one summand, so expensive and cheap summands cannot balance out. As has been discussed in Section 3.7.2, workload is not perfectly distributed as the first process is assigned one more cheap and one fewer expensive nodes. For 12 processes, Score-P/Vampir shows

Table 3.5.: Scaling of OpenMP parallelized eigenvalue evaluation with the number of threads when using symmetry (first number) or not using symmetry (second number).

#Threads	Time	Speedup	Efficiency
1	519.26 / 3,072.60	1.00 / 1.00	1.00 / 1.00
2	274.12 / 1,601.99	1.89 / 1.92	0.95 / 0.96
4	150.41 / 876.24	3.45 / 3.51	0.86 / 0.88
8	76.24 / 444.99	6.81 / 6.90	0.85 / 0.86
12	51.09 / 296.93	10.16 / 10.35	0.85 / 0.86
24	26.21 / 151.27	19.81 / 20.31	0.83 / 0.85
48	22.12 / 126.05	23.48 / 24.38	0.49 / 0.51

Figure 3.5.: Scaling of OpenMP parallelized eigenvalue evaluation with the number of threads.



(a) Speedup

(b) Efficiency

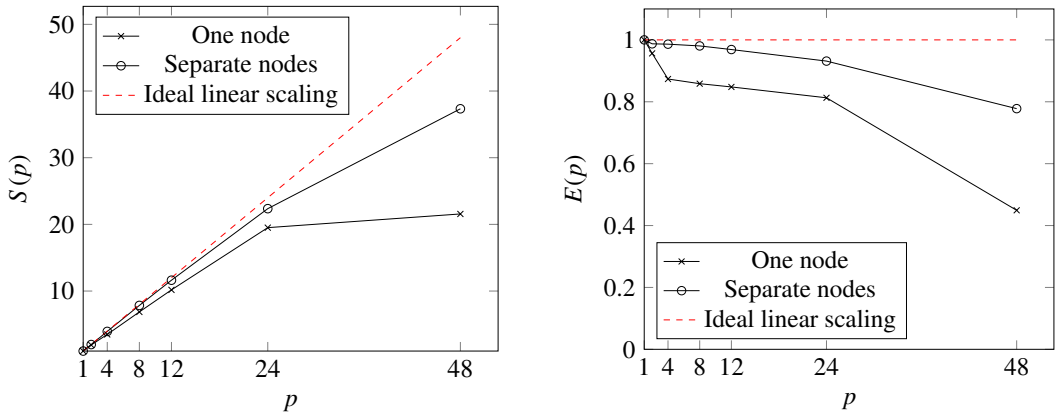
an idle time of around 10 % for that process. The remaining processes are well balanced, however, with an average idle time of around 2.5 % which includes both waiting for the other processes to finish their assigned work before the collective reduce can be performed as well as waiting for the master to perform the rest of the computation.

In hybrid parallel applications, i.e. applications that make use of both MPI and OpenMP, both the number of processes and the number of threads per process can be controlled. Based on the scaling behavior of the OpenMP implementation it is clear that all the available cores of a machine should be used. But these cores can be distributed to any number of MPI processes, e.g. four processes with 12 cores each. In a cluster, there will be an optimal number of processes per node that might depend on the number of overall processes (i.e. the number of nodes). Table 3.7 shows runtimes for different number of nodes and processes per node. When only a few nodes are used there is benefit to running two or even four processes on each node. One process per node increasingly breaks away as the number of total processes (#Nodes $\times$ #Processes Per Node) approaches the maximumg number $N = 48$. Note that e.g. 8 nodes $\times$ 4 processes = 32 does not divide N, so the comparison is not always fair. Figure 3.9 shows speedup and efficiency for increasing number of nodes when using one process per node. There is a slight dip in efficiency at 8 and 16 nodes where workload is less well balanced and just like in the pure MPI experiment, scaling drops off at 48 nodes as cyclical distribution degenerates. But in general the program makes good use of additional resources. A more flexible distribution of work (e.g. two or more processes evaluate the matrix for one κ together) would allow to increase the number of maximum processes beyond N and improve the workload balance for e.g. 8 or 16 nodes. Higher parallelism can also be used to offset higher values of N to keep the time close to constant when results of higher accuracy are required.

Table 3.6.: Scaling of MPI parallelized eigenvalue evaluation with the number of processes when every process runs on the same compute node (first number) or on its own compute node (second number).

#Processes	Time	Speedup	Efficiency
1	519.26 / 519.26	1.00 / 1.00	1.00 / 1.00
2	271.47 / 262.96	1.91 / 1.97	0.96 / 0.99
4	148.60 / 131.63	3.49 / 3.94	0.87 / 0.99
8	75.61 / 66.20	6.87 / 7.84	0.86 / 0.98
12	51.03 / 44.66	10.18 / 11.63	0.85 / 0.97
24	26.62 / 23.23	19.51 / 22.36	0.81 / 0.93
48	24.08 / 13.91	21.56 / 37.34	0.45 / 0.78

Figure 3.7.: Scaling of MPI parallelized eigenvalue evaluation with the number of processes.



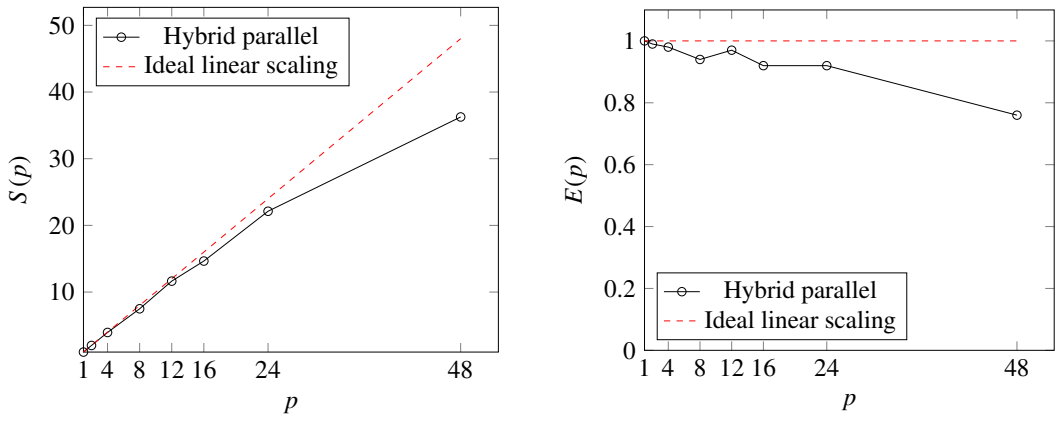
(a) Speedup

(b) Efficiency

Table 3.7.: Scaling of hybrid parallelized eigenvalue evaluation with the number of nodes (rows). The cores of each node (48) are distributed to a varying number of processes per node (columns).

	#Processes per Node		
#Nodes	1	2	4
1	22.12	21.20	21.15
2	11.12	10.77	10.66
4	5.62	5.41	5.71
8	2.82	3.01	3.55
12	1.90	2.41	2.33
16	1.51	2.00	
24	1.00	1.59	
48	0.61		

Figure 3.9.: Scaling of hybrid parallelized eigenvalue evaluation with the number of nodes. One process per node uses all available cores for OpenMP.



(a) Speedup

(b) Efficiency

4. Numerical Results

With the program presented in the previous chapter, it is now possible to efficiently compute interior Neumann eigenvalues to a high degree of accuracy. Tests of the program with increasingly fine discretization will demonstrate convergence. This allows estimating the fineness necessary to compute eigenvalues with a desired tolerance. Further experiments will explore the spectrum of eigenvalues and the parameter space. This information will be used to set up and run the optimization with the goal of improving the maximal value and accuracy of previous results.

4.1. Convergence

There are two major discretization parameters that need to be tuned, the number of collocation points n used to discretize the boundary of the domain D and the number of trapzoidal rule nodes N used to calculate the integrals for Beyn's algorithm. Analyzing the error progression of experiments with values for n and N that increase by a factor of two in each step allows estimating the rate of convergence using the formula

$$q = \frac{\log e_n / e_{n/2}}{\log 1/2}$$

where e_n is the error of the numerical value for n point discretization.

The best way to verify correctness of method and implementation and establish the rate of convergence is to compare the numerical results of the program with exact analytical values. Then an absolute error can be computed easily. For the disk with area one the eigenvalues are [12, Section 3.2]

$$\lambda_{pq} = \pi j'^2_{pq}$$

where j'_{pq} is the q-th real positive zero of J'_p, the first derivative of the Bessel function of the first kind of order p. Eigenvalues are simple for order $p = 0$ and have multiplicity 2 for $p > 0$. Numerical values with three decimal digits can be found in [23, Table 1]. Higher precision values can be computed using a root finding algorithm, e.g. Maple™'s `fsolve` function [19]. The first seven eigenvalues in order are 10.64987, 10.64987, 29.30592, 29.30592, 46.12477, 55.44907, 55.44907.

The disk is described by shape E_1. The parameters α and c have no effect on the shape of the disk, only on radius and area. Therefore, the normalized eigenvalues $\lambda_k^2 |E_1|$ are

Table 4.1.: Computed first interior Neumann eigenvalue $\lambda_1^{(n,N)}$ for the disk (E_1), absolute error and estimated convergence rate q with a variable number of collocation points n and fixed Beyn integral discretization $N = 48$. The exact value $\lambda_1 = 10.6498662587$ is used to calculate the absolute error and estimated convergence rate q. The imaginary part converges to zero at estimated rate q_i.

n	$\lambda_1^{(n,N)}$	$\left\lvert\lambda_1^{(n,N)} - \lambda_1\right\rvert$	q	q_i
144	10.649875733785−0.000035410068i	$3.67{\cdot}10^{-5}$		
288	10.649867333064−0.000004363729i	$4.49{\cdot}10^{-6}$	3.03	3.02
576	10.649866354474−0.000000541715i	$5.50{\cdot}10^{-7}$	3.03	3.01
1,152	10.649866260316−0.000000067454i	$6.75{\cdot}10^{-8}$	3.03	3.01
2,304	10.649866256236−0.000000008358i	$8.71{\cdot}10^{-9}$	2.95	3.07
4,608	10.649866257708−0.000000001030i	$1.41{\cdot}10^{-9}$	2.62	3.00
9,216	10.649866258387−0.000000000118i	$3.13{\cdot}10^{-10}$	2.18	

independent of the choice of parameters. Table 4.1 shows the result of computing the first eigenvalue with increasing n. The other parameter N is fixed and high enough so that the absolute error is dominated by n and the computed value actually approaches the exact value. Experiments in [6, Section 4.2] and later in this work suggest that $N = 48$ is sufficient for this. The error starts out quite small even for small n and decreases nicely. Convergence is approximately cubic, although it flattens out at the end. The absolute error cannot be expected to decrease indefinitely as it is limited among others by the quadrature error of the integrals that make up the matrix A_κ. Adaptive quadrature was used to achieve a relative and absolute tolerance of 10^{-10}.

The picture is very similar for other eigenvalues and shapes, so only one more will be discussed in detail. Table 4.2 shows the computed third eigenvalue for shape $E_3\{c = 1.6833, \alpha = 2.0171\}$. No exact eigenvalues are known for this shape, so the most precise computed value was used in place of the exact value to calculate the error and convergence rate. Convergence is again approximately cubic but the error starts out greater than for the disk. This can be explained with the decreased regularity of the shape. The eigenvalue is real, so the exact value for the imaginary part is zero. The imaginary part for the third eigenvalue in Table 4.2 decays at the same rate as for the first eigenvalue in Table 4.1 and as the absolute error.

For fixed n and increasing N, the computed eigenvalues are not expected to converge to the exact eigenvalues listed above of the infinite dimensional differential operator. Rather it should converge to the generalized eigenvalue of the discretized $n \times n$ matrix valued operator. The program currently does not support arbitrarily high values of n so it is not possible to confirm convergence to the exact value. Table 4.3 lists the computed

Table 4.2.: Computed third interior Neumann eigenvalue $\lambda_3^{(n,N)}$ for shape $E_3\{c = 1.6833, \alpha = 2.0171\}$ with a variable number of collocation points n and fixed Beyn integral discretization $N = 48$. The last (and presumably most precise) value is used in place of the unknown exact value to calculate the absolute error and estimated convergence rate q. The imaginary part converges to zero at estimated rate q_i.

n	$\lambda_3^{(n,N)}$	$\left\lvert\lambda_3^{(n,N)} - \lambda_3^{(9216,N)}\right\rvert$	q	q_i
144	32.893775916244−0.001951220319i	$2.01{\cdot}10^{-3}$		
288	32.893365429054−0.000225008217i	$2.33{\cdot}10^{-4}$	3.11	3.12
576	32.893313215505−0.000026937655i	$2.79{\cdot}10^{-5}$	3.06	3.06
1,152	32.893306839241−0.000003293325i	$3.39{\cdot}10^{-6}$	3.04	3.03
2,304	32.893306087231−0.000000407030i	$4.11{\cdot}10^{-7}$	3.05	3.02
4,608	32.893306004644−0.000000050524i	$4.49{\cdot}10^{-8}$	3.19	3.02
9,216	32.893305997190−0.000000006285i	$0.00{\cdot}10^{0}$		3.06

Table 4.3.: Computed first interior Neumann eigenvalue $\lambda_1^{(n,N)}$ for the disk and absolute change from the previous row with variable Beyn integral discretization N and a fixed number of collocation points $n = 9216$.

N	$\lambda_1^{(n,N)}$	$\left\lvert\lambda_1^{(n,N)} - \lambda_1^{(n,N/2)}\right\rvert$
6	10.649662123158−0.000252587437i	
12	10.649866247926+0.000000004369i	$3.25{\cdot}10^{-4}$
24	10.649866258387−0.000000000118i	$1.14{\cdot}10^{-8}$
48	10.649866258387−0.000000000118i	$8.15{\cdot}10^{-15}$
96	10.649866258387−0.000000000118i	$8.41{\cdot}10^{-14}$

Table 4.4.: Computed third interior Neumann eigenvalue $\lambda_3^{(n,N)}$ for the shape $E_3\{c = 1.6833, \alpha = 2.0171\}$ and absolute change from the previous row with variable Beyn integral discretization N and a fixed number of collocation points $n = 9216$.

N	$\lambda_3^{(n,N)}$	$\left\lvert\lambda_3^{(n,N)} - \lambda_3^{(n,N/2)}\right\rvert$
6	32.835359922288+0.031227701024i	
12	32.893095101843+0.000075891936i	$6.56{\cdot}10^{-2}$
24	32.893305995091−0.000000005433i	$2.24{\cdot}10^{-4}$
48	32.893305997190−0.000000006285i	$2.27{\cdot}10^{-9}$
96	32.893304827425−0.000001644377i	$2.01{\cdot}10^{-6}$

first eigenvalue for the disk. It also calculates the absolute difference between the current row N and the previous row $N/2$. Matching the theoretical exponential error decay proven by Beyn [6, Corollary 4.8] the number of digits that are fixed approximately increases with a constant factor. Convergence almost stops at $N = 24$ with only tiny changes afterwards. The finest discretization $N = 96$ shows small signs of cancellation as the change increases slightly in the last row. This effect is more pronounced for the third eigenvalue for shape $E_3\{c = 1.6833, \alpha = 2.0171\}$ (Table 4.4). Here, the error decreases exponentially as well but the error starts out greater than for the disk just as it was with variable discretization parameter n.

This work aims to provide eigenvalues with six digits precision. For this, the experiments suggest discretization parameters of $n = 1152$ and $N = 48$. Both of these values include a small safety cushion. Both for n and N, the error in the third eigenvalue is higher than in the first eigenvalue. It may also depend on the shape parameters. As Beyn [6, Corollary 4.8.] has shown, the error also depends on the distance of the eigenvalue from the chosen contour. Higher N allows some lattitude in that choice. Spurious eigenvalues are also more likely to appear for smaller N. Even with these cautious values, it is advisable to check the precsion of the result after optimization as this does not take much time or effort.

4.2. Spectrum and Parameter Space

For choosing the contour for Beyn's algorithm and selecting the eigenvalue to optimize from the set of eigenvalues returned it is necessary to look at the whole spectrum of eigenvalues and how it changes with the shape parameters.

Table 4.5.: Real interior Neumann eigenvalues for the disk

1	2	3	4	5	6	7
10.6499	10.6499	29.3059	29.3059	46.1248	55.4491	55.4491

For the disk, the parameters have no influence on the normalized eigenvalues. The spectrum is accurately reproduced by the program for any set of parameters as shown in Table 4.5.

For other equipotential shapes, the parameters influence both value and multiplicity of eigenvalues. Two more examples will be discussed, but the picture is similar for all of them. For shape E_3 (Table 4.6), the third, fourth and fifth eigenvalues are noticeably closer in the second column ($c = 2.0$) than in the first column ($c = 1.0$). The third eigenvalue seems to always be at least of multiplicity two. As the third and fourth eigenvalue increase, the fifth eigenvalue decreases. The maximum is expected where the two lines touch or cross (note the switching of multiplicities of seventh, eighth, and

Table 4.6.: Real interior Neumann eigenvalues for shape E_3.

k	$\alpha = 1.0$ $c = 1.0$	$\alpha = 1.0$ $c = 2.0$	$\alpha = 2.0$ $c = 1.0$
1	10.1172	8.5687	8.9838
2	10.1172	8.5687	8.9838
3	30.0851	32.5409	31.6770
4	30.0851	32.5409	31.6770
5	40.5097	33.4469	35.4411
6	54.0537	50.4431	49.9877
7	64.1796	66.3616	68.9967
8	77.4457	66.3616	68.9967
9	77.4457	79.6949	76.5971
10	101.2274	101.7737	105.0557
11	101.2274	101.7737	105.0557

nineth eigenvalue). The third eigenvalue then has multiplicity three, just as reported by Antunes and Oudet [4, Table 2]. For all listed sets of parameters, the third eigenvalue is above 30 and far away both from the current and the maximum second eigenvalue. So if the eigenvalue to be optimized is heuristically selected as the first above 30, any of the sets can be used as starting values. The middle column is already quite close to the maximum.

For shape E_6 (Table 4.7), Antunes and Oudet report the maximum sixth eigenvalue to be of multiplicity four. The sixth, seventh, eighth and nineth eigenvalues are significantly closer together in the second column than in the first and will likely be identical at the maximum. The space between fifth and sixth eigenvalue is again easily wide enough to distinguish them. However the value of 57.4477 in the first column is not far away from the maximum of the fifth eigenvalue. Even though that maximum was achieved with a different shape, a conflict at least cannot be ruled out. Therefore, the first column seems like an unreliable starting value. Either of the second or third column are useable.

In order to find good starting values and predict how well the optimization will perform, the parameter space can be probed. The influence of each parameter isolated on the eigenvalues is fairly simple (e.g. Figure 4.1 for λ_4). All parameters change roughly on the same scale. The slopes of ϵ and δ are slightly but not orders of magnitude steeper. Most parameters exhibit a singe clear peak, at least on the coarse grid used here. Optimization of single parameters is expected to work very well without danger of premature termination or getting stuck in a local optimum. Parameter α is the exception, as depending on the other parameters there may not be a peak or only a very flat one. There is therefore a danger that α runs away and the optimizer gets stuck at extreme values. A

Figure 4.1.: Influence of individual parameters on interior Neumann eigenvalue λ_4 (vertical axis in all subfigures) for shape E_4 with other shape parameters fixed.

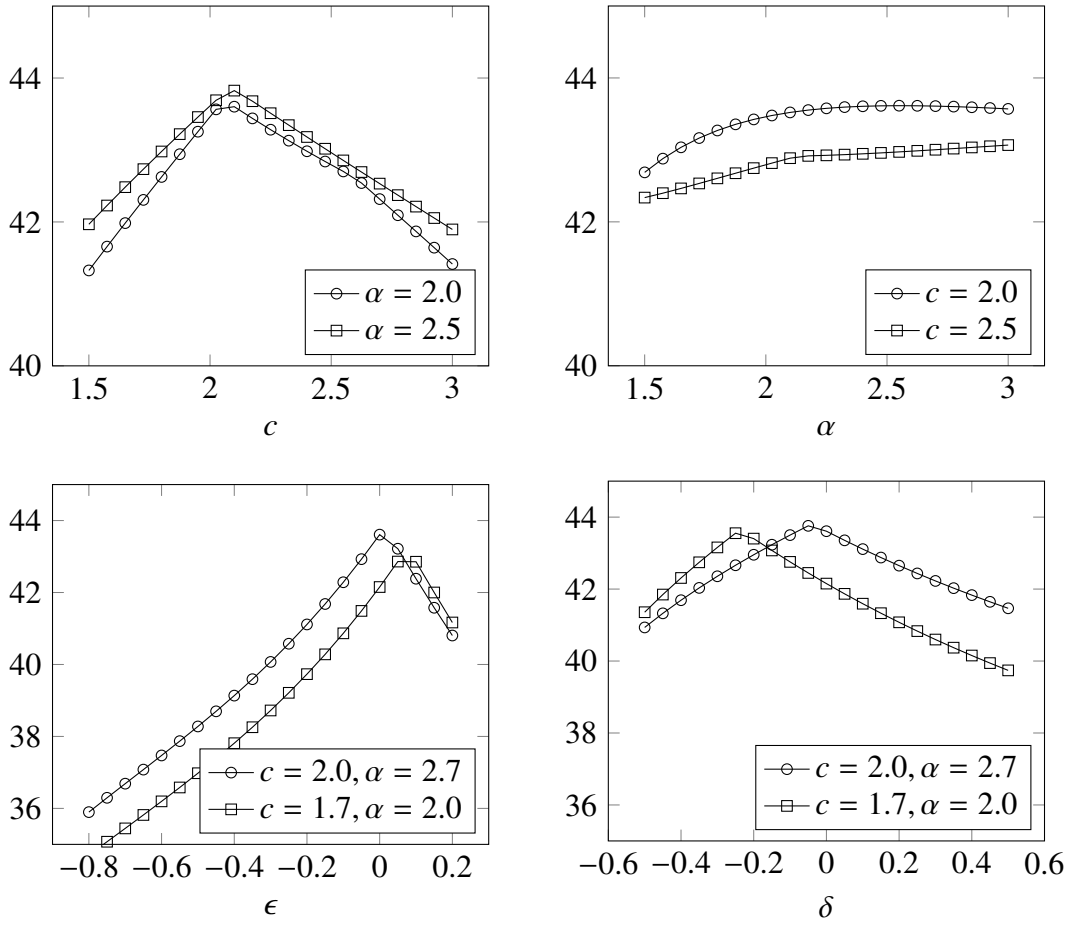


Table 4.7.: Real interior Neumann eigenvalues for shape E_6.

k	$c = 1.0$ $\alpha = 1.0$	$c = 2.0$ $\alpha = 1.0$	$c = 1.0$ $\alpha = 2.0$
1	9.9420	8.7558	8.4829
2	9.9420	8.7558	8.4829
3	30.2906	29.5970	28.7474
4	30.2906	30.7791	31.2093
5	38.2583	30.7791	31.2093
6	57.4477	64.8949	63.4889
7	63.6715	67.4167	66.8065
8	75.4295	67.4167	66.8065
9	75.4295	68.8665	71.2289
10	102.5517	98.0234	93.4450
11	102.5518	98.0234	93.4450

good choice of starting value for the other parameters can prevent this. Parameter ϵ is only valid in a limited range. If the base points move too far apart, the shapes become no longer simply connected which the program currently cannot handle. The result of optimization has to be carefully checked for plausibility.

Even though each single paramater is well behaved, the combined multidimensional parameter space is as usual more complex. The different peaks for single parameters depending on the fixed values of other parameters in Figure 4.1 already hint at this. For the two original parameters c and α (e.g. Figures 4.4 and 4.3), the space is still well behaved in most areas but local optimums where the optimizer gets stuck or flat regions where the optimizer prematurely terminates cannot be ruled out. This is even more true for higher dimensional space (up to seven for shape E_5.) This can be countered, but not completely prevented, by using multiple different starting values and restarting the optimization with larger step size after it terminates. Probing the whole parameter space as in Figures 4.3 and 4.4 to find good starting values is impossible in higher dimensions due to the exponential cost. Random sampling on the other hand is able to cover the space well enough at much lower cost. The result of optimization in lower dimensions is sometimes a good starting value as well.

4.3. Optimization

With the knowledge about convergence and parameter space, the optimization can be set up and performed. Discretization parameters $n = 1152$ and $N = 48$ are used except where specially noted. Multiple different starting values are chosen as close to the max-

Figure 4.3.: Influence of parameters c and α on interior Neumann eigenvalue λ_3 for shape E_3.

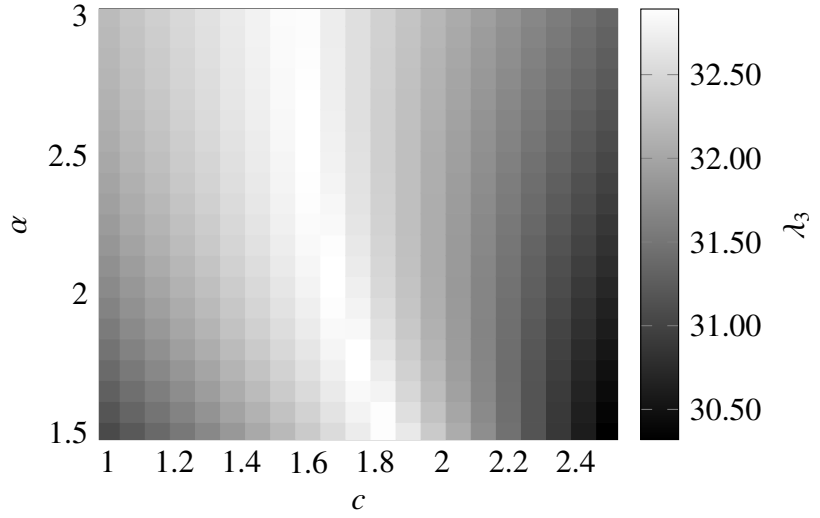
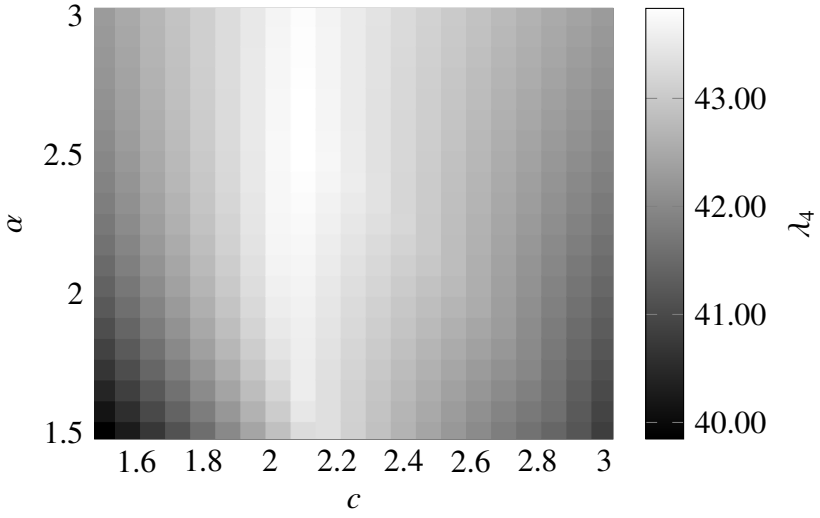


Figure 4.4.: Influence of parameters c and α on interior Neumann eigenvalue λ_4 for shape E_4.



imum as possible by probing the parameter space as in Section 4.2. The optimizer (an implementation of the Nelder-Mead simplex algorithm [20]) is set to terminate when both the size of the simplex and the improvment of the eigenvalue are below 10^{-6}. To rule out premature termination or local optimums, the size of the simplex is reset to its initial value and the optimizer is restarted as long as the maximum improves significantly between restarts. The initial step size of 0.1 in all dimensions should be sufficient to get the optimizer out of difficult regions while avoiding degenerated shapes. For two dimensional parameter space, the restart usually did not improve the result as the parameter space is sufficiently simple. Optimization terminated after only a few hundred iterations. For higher degrees of freedom at least the first restart provided significant gains. Termination often required thousands of iterations. The results are checked for convergence with more collocation points. Optimization is restarted from that result with higher precision in case that convergence check is not succesful. The target is a precision of at least six significant digits for each eigenvalue. The values are truncated after the sixth digit. The parameters are given with a greater number of significant digits so that the results can be reliably reproduced.

This work tries to reproduce or improve the maximal eigenvalues λ_k for $k = 3, 4, 5, 6, 10$ from previous works. Antunes and Oudet [4] give results of (multiplicity in brackets) 32.90 (3), 43.86 (3), 55.17 (3), 67.33 (4), and 114.16 (5) using more than 30 unknown Fourier coefficients. Using equipotentials in parameters α and c, Kleefeld [15] improved the third and fourth eigenvalue to 32.9018 and 43.8694. There is some uncertainty as to the precision of the results given by Antunes and Oudet. The values given in their paper for the first and second eigenvalue deviate by about 10^{-2} from the known theoretical values of 10.65 and 21.30 (rounded to two decimal places). While the second value is too low, which may just be an issue of optimization, the first value is too high. It was not possible to try to reproduce their results as the parameters are not included in the paper. Any comparison to new results can therefore only be tentative.

Using only the two original parameters from Kleefeld, his results for $\lambda_3 = 32.9018$ and $\lambda_4 = 43.8693$ could be reproduced (albeit with slightly different parameters) but not improved. For higher eigenvalues, two parameters proved insufficient. The value of $\lambda_5 = 54.5401$ is still a long way from the optimum of 55.17 found by Antunes and Oudet. The multiplicity three is also not reproduced. On the other hand, $\lambda_6 = 67.0440$ is closer to the old value of 67.33 and the multiplicity is reproduced. But even so the old value could not be reached. The result for the tenth eigenvalue is particularly bad. There is a large gap between the 12th and 13th eigenvalues so that multiplicty five is not reproduced and the value of 109.988 is way below the previous record of 114.16. There is no indication that the parameter space is particularly difficult, so the shape maximizers for the latter three eigenvalues appear to be not modeled well with just two shape parameters. All the results for two parameters are summarized in Table 4.8.

After switching to the complete array of parameters, λ_4 was improved to 43.8700

Table 4.8.: Maximum interior Neumann eigenvalues λ_k for shape E_k with two free shape parameters.

k	c	α	λ_k
3	1.687730810	2.019822714	32.9018 32.9018 32.9018
4	2.084610015	2.541256146	43.8693 43.8693 43.8693
5	2.380671137	3.914738607	54.5401 54.5401 56.0889
6	2.849410261	0.660868556	67.0440 67.0440 67.0440 67.0440
10	1.567009307	5.196376634	109.988 109.988 109.988 118.955 118.955

with only very slight deviations from the regular two parameter equipotential through ϵ and δ. The scheme is similarly succesful for λ_6, where the old value 67.33 was at least reproduced with higher precision in 67.3364, and λ_{10}, with a new value of 114.185 over 114.16. Note that for λ_{10} it was actually necessary to use 2304 collocation points because the convergence check at the end showed less than six digits precision. Also the multiplicity of five is not quite perfect here. The scheme works less well for λ_5. The old value could not be reached. It is not immediately clear whether this is because modified equipotentials cannot represent the actual shape maximizer, whether the higher dimensions of the parameter space caused the optimizer to miss the global optimum, or whether the old value is incorrect. Table 4.9 contains the eigenvalues found by optimization using all parameters. The parameters that achieved these results are listed in Table 4.10.

It is not suprising that the equipotential shape maximizers (see Figure 4.5) are visually similar or even identical to the old shape maximizers by Antunes and Oudet (see Figure 2.5) as the former are heavily inspired by the latter. There is however a clear difference between the equipotential shapes using just two parameters and those using all parameters (dotted and solid lines in Figure 4.5, respectively), especially in the regions around the indentations. Even for the fourth eigenvalue, where the two shapes are almost identical, those regions differ the most. Getting the indentations right appears to be the main challenge.

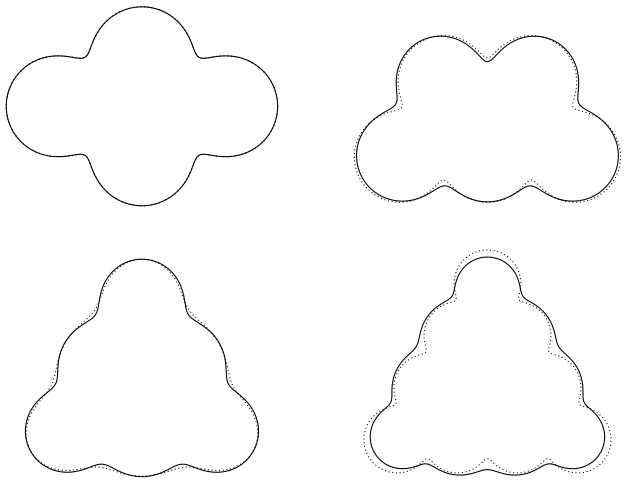
Table 4.9.: Maximum interior Neumann eigenvalues λ_k for shape E_k with all shape parameters free.

k	λ_k
4	43.8700 43.8700 43.8700
5	55.1498 55.1498 55.1498
6	67.3364 67.3364 67.3364 67.3364
10	114.185 114.185 114.185 114.185 114.187

Table 4.10.: Equipotential shape parameters for maximum interior Neumann eigenvalues in Table 4.9. Irregularities ϵ and δ applied as in 2.1.

k	c	α	ϵ	δ
4	1.942568636	2.751523202	$-1.3145316460\cdot10^{-2}$	$-4.6234670530\cdot10^{-2}$
5	1.548694899	2.223124784	$-8.8452303300\cdot10^{-2}$	$-1.9793129920\cdot10^{-1}$
			$-4.5093371990\cdot10^{-2}$	$-1.6718903350\cdot10^{-1}$
			$-4.3547274900\cdot10^{-2}$	
6	2.027170345	1.706097040	$1.5774070160\cdot10^{-1}$	$6.0012147050\cdot10^{-3}$
10	0.899356409	2.706745797	$-4.4817663590\cdot10^{-2}$	$1.1488839400\cdot10^{0}$
				$-2.8284367480\cdot10^{-1}$

Figure 4.5.: Equipotential shape maximizers for interior Neumann eigenvalues λ_4, λ_5, λ_6, λ_{10} using just two parameters (dotted) and using all parameters (solid). The shapes for λ_4 are almost identical. The shapes are scaled so they all have the same area.



5. Conclusions

The computation of interior Neumann eigenvalues required multiple steps. Using the Boundary Element Method, a boundary value problem for the Helmholtz equation was transformed into an integral equation. Discretization yields a homogeneous linear system. Each element of the system matrix is an integral that must be evaluated numerically, although symmetries can be exploited to reduce the amount of work. The singular nature of the integrand required careful analysis. The singularity was shown to be removable. The matrix depends nonlinearly on the wavenumber κ, so the homogeneous linear system can be interpreted as a nonlinear eigenvalue problem. Beyn's method was employed to calculate the eigenvalues. This method involves evaluating the matrix for a number of different values of κ.

The above methods were implemented in the C programming language with the help of different numerical libraries. GSL provided the larger framework and some solvers for nonlinear equations and integrals. BLAS and LAPACK were used for linear algebra. Two different parallelization approaches were included in the program. OpenMP was used to evaluate each element of the matrix in parallel. Additionally, the evaluation of the matrices for Beyn's method was parallelized with the help of MPI to enable computation on the cluster. The distribution of work in both approaches had to be planned carefully and good workload balance was confirmed with the help of tools like Score-P/Vampir. The hybrid parallelization proved to be efficient due to the high degree of parallelism and low communication overhead.

For the shape optimization of the eigenvalues, an existing description of the shape of the domain of the boundary value problem based on equipotentials was extended with additional parameters. With between four and seven degrees of freedom, the range of representable shapes increases significantly while still keeping the number of parameters much lower than more general descriptions. The parameter space is therefore quite simple. In experiments, the modified equipotential shapes have shown a lot of promise. New and improved maximums were found for the fourth, sixth and tenth eigenvalue. Previous optimal values for the third eigenvalue were confirmed. On the other hand, the optimum shape for the fifth eigenvalue appeared not to be modeled well.

So further tweaks are necessary to derive a candidate for the general description of shape maximizers. More modifications are possible to the basic concept of equipotentials like the use of different norms and/or exponents in each potential summand. But it is possible that a completely new idea is required. Results of equipotentials for other eigenvalues should be insightful. Equipotentials also trivially extend to three dimen-

sions, where there is a lot of room for exploration. So far only the two-dimensional problem has been considered. The modular code and high performance of the implementation allows to quickly test such modifications and extensions. The program might benefit from e.g. a Python interface so that these modifications can be made in a simpler environment.

Even though the implementation has demonstrated its quality, some improvements are certainly possible, particulary regarding performance. Tuning of the basic building blocks like the Hankel function and quadrature routines has almost certainly not exhausted its potential. Some researchers have extended Beyn's method by addition of another random matrix that is multiplied from the left in order to reduce the size of the system. Convergence of the Boundary Element Method might be improved by a less naive distribution of collocation points. The workload balance of the parallelization is not quite optimal yet. But the low hanging fruit have probably been picked, so a determination must be made whether incremental performance gains are worth the not insignificant investment of development time when that time could be spent on extending the features of the program instead.

A. The shapeopt Program

This chapter descibes how to build and use the program described in this work. The program must be built from source.

A.1. Dependencies

The following dependencies must be available:

- Intel MKL [13] (tested with 2019.3) or alternatively OpenBLAS with LAPACK [22]
- GNU Scientific Library [11] (tested version 2.5)
- C compiler (tested GCC and Intel C Compiler)
- Fortran compiler (tested GFortran and Intel Fortran Compiler)
- MPI (optional; tested with Intel MPI and OpenMPI)

On JURECA, the dependencies can all be loaded with these two commands:

```
1 module load intel-para
2 module load GSL
```

A.2. Getting the Source Code

The source code is stored in a git repository hosted by the Forschungszentrum Jülich GitLab Server at the URL

`https://gitlab.version.fz-juelich.de/abele2/shapeopt`

Clone the project to get a local copy with e.g.

```
1 git clone https://gitlab.version.fz-juelich.de/abele2/
    shapeopt.git
```

This will create a new directory, *shapeopt*, that contains the project. The project is organized as follows:

- Main directory: build scripts and C source code for program entry point
- Directory *src*: C source code for numerical computation
- Directory *amos*: Fortran source code for Amos' Hankel function implementation [21]
- Directory *doc*: Latex project to build this documentation
- Directory *CuTest*: C unit test framework [8]
- Directory *test*: C source code for unit tests

A.3. Build

The build process is separated into two stages, *configure* and *make*. The configure stage is controlled by the `CONFIGURE` script. Running this script on the command line creates the *Makefile* that controls the make stage that is executed by calling `make`. When changes to the source code are made only the make stage needs to be rerun. Two directories are created: *bin* contains the executable, *obj* contains intermediate compilation artifacts. Besides the default target `all`, the Makefile defines targets `clean` to delete all artifacts (e.g. if a full rebuild is necessary) and `test` to build the unit test executable.

The build process can be customized with a number of options. All options must be given in the format `<key>=<value>`. The format or `<key>+=<value>` is supported to append to existing options. The options can be provided

- as arguments to `CONFIGURE` (recommended, options only need to be set once on the first build)
- as arguments to `make` (options need to be repeated identically on each rebuild)
- as environment variables (not recommended, might cause conflicts, except maybe for `CC` and `FC`).

The supported options are:

- `CBLAS`: the BLAS/LAPACK implementation; `mkl` or `openblas`; default `mkl`
- `ENABLE_MPI`: compile with or without MPI; `0` or `1`; default `1`
- `CC`: the C compiler, must be compatible to the compiler used by MPI; system default
- `FC`: the Fortran compiler; system default

- `CONFIG`: `debug` disables compiler optimizations; default empty
- `CCPREFIX`: prefix C compiler with instrumentation, e.g. Score-P; default empty
- `CFLAGS`: any flags allowed by the C compiler, but some flags are necessary for compilation and are overwritten by the makefile
- `FCFLAGS`: same as `CFLAGS` but for the fortran compiler
- `LDFLAGS`: same as `CFLAGS` but for the linker employed by CC

A typical build on JURECA could look like this:

```
1 ./CONFIGURE CC=icc FC=ifort
2 make
```

A.4. Unit tests

The project includes unit tests with at least basic coverage of all functionality. The unit tests verify correctness of compilation and changes to existing code to a large degree. CuTest [8] is used as a framework to simplify test creation. Build and run the tests from the main directory with

```
1 make test
2 ./bin/shapeopttest
```

The tests can be run in parallel with `mpirun`/`srun` to verify correctness of the MPI parallelization.

A.5. Using the Program

The program is controlled using a command line interface. The basic call signature from the main directory is

```
1 ./bin/shapeopt <mode> [options]
```

where `<mode>` is a single string to set the mode of computation and `[options]` is a sequence of zero or more options (format `-key value`) or flags (format `-flag`). Prefix the call with `mpirun` (or `srun` on JURECA) to run the program in parallel.
Supported modes:

- `opt`: run a full optimization until convergence
- `eval`: evaluate the eigenvalues once

- `matr`: evaluate the discretized matrix operator once
- `contour`: generate the collocation points that define the discretized boundary

Options controlling the precision:

- `-n <val>`: an integer ≥ 4; the number of collocation points on the boundary of the domain; must fit the symmetry of the selected shape; default `96`
- `-N <val>`: an integer ≥ 4; the number of trapezoidal rule nodes for evaluating the integrals in Beyn's algorithm; default `24`

Options controlling the shape of the domain (see Equation (2.31)):

- `-k <val>`: one of $1, 3, 4, 5, 6, 10$; the number of equipotential base points; for optimization also the index of the eigenvalue to be optimized; default `3`
- `-a <val>`: a positive floating point value; the equipotential parameter α; default `2.0171`
- `-c <val>`: a positive floating point value; the equipotential parameter c; default `1.6833`
- `-i <val>`: a space separated list of floating point values; the irregularity of the position of the equipotential base points; default `0`
- `-g <val>`: a space separated list of floating point values; the irregularity of the weights of the equipotential base points; default `0`

Mode specific options:

- `-p <val>`: a string containing one or more of the characters "c", "a", "i", "g"; the shape parameters to be optimized; the other paramaters are fixed; default "caig"
- `-r <val>`: a floating point value; the tolerance for restarting the optimization with initial step size after it terminates; no restart if the change from the last run is less than this tolerance; disable restarts with value <= 0; default `0.0`
- `-w <val>`: a complex number `a(+/-)bi` where `a` and `b` are floating point values; the wave number; mode `matr` only; default `1+0i`

Diagnostic options:

- `-l <val>`: an absolute or relative path; path to the file where log messages are written. The rank of the MPI process is appended to the file name so that each process has its own log file; by default the log messages are written to the terminal

- `-v <val>`: one of `{0, 1, 2, 3, 4}`; the verbosity of the log; `0` = off, `4` = verbose; default `1` = errors
- `-b <val>`: integer; perform benchmark of program with a number of iterations; measures average runtime; benchmark is disable if <= 0; default `0`

Example 1: Evaluate the eigenvalues for the shape $E_3\{c = 1.5, \alpha = 2.0\}$ using a boundary discretization of $n = 240$ (divisible by 12 to fit the symmetry of degree six):

```
./bin/shapeopt eval -n 240 -k 3 -a 2.0 -c 1.5
```

Output:

```
Evaluating eigenvalues with shape 3
Discretization n = 240, N = 24.
Parameter values c = 1.500000, alpha = 2.000000, eps = [],
    delta = []
Found 6 eigenvalues: 32.570245670692-0.000337077156i
   32.570245670959-0.000337074577i
   33.522132466595-0.000145065695i
   48.276808197998-0.000463165976i
   65.624350978805-0.000301023401i
   65.624350994504-0.000301020148i
```

Example 2: Optimize the fourth eigenvalue with respect to shape parameter c with initial value 1.5. The shape parameter α is fixed to 2.0. Discretization parameter $n = 160$ is divisible by eight to fit the symmetry of degree four of shape E_4:

```
./bin/shapeopt opt -n 160 -k 4 -p c -c 1.5 -a 2.0
```

Output:

```
Optimizing eigenvalue 4 w.r.t parameter(s) c.
Discretization n = 160, N = 24.
Parameter values c = 1.500000, alpha = 2.000000, eps =
   [0.000000], delta = [0.000000]
Iteration 1: x = [1.8000000000] y = -42.6245635820
Iteration 2: x = [2.2000000000] y = -43.3904175962
Iteration 3: x = [2.0000000000] y = -43.4593668308
Iteration 4: x = [2.1000000000] y = -43.6042227107
#...
```

B. Analysis of the Integrand in Maple

Analyzing the integrand for Section 2.1 was done in the mathematical software system Maple™, version 2019.0 [19]. For verifiability the worksheet is included on the following pages.

The worksheet consists of two parts. The first part symbolically computes the limit of the integrand at the singularity t_0 where the function $g(t)$ that describes the boundary is equal to the collocation point x so that the argument to the Hankel function becomes zero. The boundary satisfies two conditions that are set in the worksheet: the normal ν is defined to be orthogonal to g at t_0 and $g(t_0) = x$. For greater understanding, the worksheet also computes the matching limits of the integrand where the Hankel function has been replaced by alternative representations, a truncated series and the combination of Bessel functions of the first and second kind.

The second part tests the limit numerically. All symbols must be assigned numerical values. The function g is now defined as the quadratic interpolation of three points $x_i, i \in 0, 1, 2$. Multiple sets of values for the points x_i and the singularity t_0 are included to test the effect of finer discretization and verify that the limit works for singularities at different positions. The results that are printed at the end use to the last choice for points and singularity. The table lists the normalized distance $|t - t_0|$ from the singularity, the real distance $\|x - g(t)\| = \|g(t_0) - g(t)\|$ from the singularity, the value of the integrand, and the absolute difference between integrand and the limit. The integrand approaches the limit as t approaches t_0 up to $\|x - g(t)\| \approx 10^{-7}$. Numerical cancellation prevents precise evaluation of the integrand very close to the singularity.

```
> restart;
```

Analysis

```
> #norm(x - g(t))
  N := t -> sqrt((x1 - g1(t))^2 + (x2 - g2(t))^2);
```

$$N := t \to \sqrt{(x1 - g1(t))^2 + (x2 - g2(t))^2} \tag{1}$$

```
> #prod(x - g(t), nu)
  K := t -> (x1 - g1(t))*n1 + (x2 - g2(t))*n2;
```

$$K := t \to (x1 - g1(t))\, n1 + (x2 - g2(t))\, n2 \tag{2}$$

```
> #jacobian norm(g'(t))
  J := t -> sqrt((D(g1)(t))^2 + (D(g2)(t))^2);
```

$$J := t \to \sqrt{\mathrm{D}(g1)(t)^2 + \mathrm{D}(g2)(t)^2} \tag{3}$$

```
> #define nu so it is normal to g at t0
  n1 := D(g2)(t0) / J(t0);
  n2 := -D(g1)(t0) / J(t0);
```

$$n1 := \frac{\mathrm{D}(g2)(t0)}{\sqrt{\mathrm{D}(g1)(t0)^2 + \mathrm{D}(g2)(t0)^2}}$$

$$n2 := -\frac{\mathrm{D}(g1)(t0)}{\sqrt{\mathrm{D}(g1)(t0)^2 + \mathrm{D}(g2)(t0)^2}} \tag{4}$$

```
> #define singularity g(t0) = x
  g1(t0) := x1:
  g2(t0) := x2:
> #complete general integrand before discretization
  F := - I * kappa / 4 * HankelH1(1, kappa * N(t)) * K(t) / N(t);
```

$$F := -\frac{1}{\sqrt{(x1 - g1(t))^2 + (x2 - g2(t))^2}} \left(\frac{1}{4}\, \mathrm{I}\, \kappa\, \mathrm{HankelH1}\left(1, \kappa \sqrt{(x1 - g1(t))^2 + (x2 - g2(t))^2}\right) \left(\frac{(x1 - g1(t))\, \mathrm{D}(g2)(t0)}{\sqrt{\mathrm{D}(g1)(t0)^2 + \mathrm{D}(g2)(t0)^2}} - \frac{(x2 - g2(t))\, \mathrm{D}(g1)(t0)}{\sqrt{\mathrm{D}(g1)(t0)^2 + \mathrm{D}(g2)(t0)^2}} \right) \right) \tag{5}$$

```
> #integrand using series expansion of HankelH1 at t0
  H11_series := series(HankelH1(1, z), z = 0, 1);
  F_series := simplify(-I * kappa / 4 * subs(z = kappa * N(t),
  convert(H11_series, polynom)) * K(t) / N(t));
```

$$H11\_series := -\frac{2\,\mathrm{I}}{\pi} + \mathrm{O}(z)$$

$$F\_series := -\frac{1}{2} \left((-x2 + g2(t))\, \mathrm{D}(g1)(t0) + \mathrm{D}(g2)(t0)\, (x1 - g1(t)) \right) \Big/ \left(\sqrt{\mathrm{D}(g1)(t0)^2 + \mathrm{D}(g2)(t0)^2}\, \pi \left(g1(t)^2 - 2\, x1\, g1(t) + g2(t)^2 - 2\, x2\, g2(t) + x1^2 + x2^2 \right) \right) \tag{6}$$

```
> #integrand using the relation HankelH1 = BesselJ + I * BesselY
  F_Y := - I * kappa / 4 * BesselY(1, kappa * N(t)) * K(t) / N(t);
  F_J := - I * kappa / 4 * BesselJ(1, kappa * N(t)) * K(t) / N(t);
```

$$F\_Y := -\frac{1}{\sqrt{(x1-g1(t))^2+(x2-g2(t))^2}}\left(\frac{1}{4}\,I\kappa\,\mathrm{BesselY}\left(1,\kappa\sqrt{(x1-g1(t))^2+(x2-g2(t))^2}\right)\left(\frac{(x1-g1(t))\,D(g2)(t0)}{\sqrt{D(g1)(t0)^2+D(g2)(t0)^2}}-\frac{(x2-g2(t))\,D(g1)(t0)}{\sqrt{D(g1)(t0)^2+D(g2)(t0)^2}}\right)\right)$$

$$F\_J := -\frac{1}{\sqrt{(x1-g1(t))^2+(x2-g2(t))^2}}\left(\frac{1}{4}\,I\kappa\,\mathrm{BesselJ}\left(1,\kappa\sqrt{(x1-g1(t))^2+(x2-g2(t))^2}\right)\left(\frac{(x1-g1(t))\,D(g2)(t0)}{\sqrt{D(g1)(t0)^2+D(g2)(t0)^2}}-\frac{(x2-g2(t))\,D(g1)(t0)}{\sqrt{D(g1)(t0)^2+D(g2)(t0)^2}}\right)\right) \quad (7)$$

```
> #determine general limits
  #all methods should provide the same result
  limFJ := limit(F_J, t = t0);
  limFY := limit(I * F_Y, t = t0);
  limFseries := limit(F_series, t=t0);
  limF := limit(F, t = t0);
```

$$limFJ := 0$$

$$limFY := \frac{1}{4}\,\frac{-D^{(2)}(g2)(t0)\,D(g1)(t0)+D^{(2)}(g1)(t0)\,D(g2)(t0)}{\left(D(g1)(t0)^2+D(g2)(t0)^2\right)^{3/2}\pi}$$

$$limFseries := \frac{1}{4}\,\frac{-D^{(2)}(g2)(t0)\,D(g1)(t0)+D^{(2)}(g1)(t0)\,D(g2)(t0)}{\left(D(g1)(t0)^2+D(g2)(t0)^2\right)^{3/2}\pi}$$

$$limF := -\frac{1}{4}\,\frac{D^{(2)}(g2)(t0)\,D(g1)(t0)-D^{(2)}(g1)(t0)\,D(g2)(t0)}{\left(D(g1)(t0)^2+D(g2)(t0)^2\right)^{3/2}\pi} \quad (8)$$

Numeric Test with Discretization

```
> #g discretized: quadratic interpolation using lagrange basis
  polynomials
  g1 := t-> 2 * (t - 1/2) * (t - 1) * x01 - 4 * t * (t - 1) * x11 +
  2 * t * (t - 1/2) * x21;
  g2 := t-> 2 * (t - 1/2) * (t - 1) * x02 - 4 * t * (t - 1) * x12 +
  2 * t * (t - 1/2) * x22;
  limF;
```

$$g1 := t \to 2\left(t-\frac{1}{2}\right)(t-1)\,x01-4\,t\,(t-1)\,x11+2\,t\left(t-\frac{1}{2}\right)x21$$

$$g2 := t \to 2\left(t-\frac{1}{2}\right)(t-1)\,x02-4\,t\,(t-1)\,x12+2\,t\left(t-\frac{1}{2}\right)x22 \quad (9)$$

$$-\frac{1}{4}\left((4\,x02-8\,x12+4\,x22)\left(2\,(t0-1)\,x01+2\left(t0-\frac{1}{2}\right)x01-4\,(t0-1)\,x11\right.\right. \tag{9}$$
$$-4\,t0\,x11+2\left(t0-\frac{1}{2}\right)x21+2\,t0\,x21\Big)-(4\,x01-8\,x11+4\,x21)\left(2\,(t0-1)\,x02\right.$$
$$\left.\left.+2\left(t0-\frac{1}{2}\right)x02-4\,(t0-1)\,x12-4\,t0\,x12+2\left(t0-\frac{1}{2}\right)x22+2\,t0\,x22\right)\right)\Big/$$
$$\left(\left(\left(2\,(t0-1)\,x01+2\left(t0-\frac{1}{2}\right)x01-4\,(t0-1)\,x11-4\,t0\,x11+2\left(t0\right.\right.\right.\right.$$
$$\left.\left.-\frac{1}{2}\right)x21+2\,t0\,x21\right)^2+\left(2\,(t0-1)\,x02+2\left(t0-\frac{1}{2}\right)x02-4\,(t0-1)\,x12\right.$$
$$\left.\left.\left.\left.-4\,t0\,x12+2\left(t0-\frac{1}{2}\right)x22+2\,t0\,x22\right)^2\right)^{3/2}\pi\right)$$

```
> #points on a circle, fine discretization
  x01 := cos(Pi/12):
  x02 := sin(Pi/12):
  x11 := cos(Pi/12 + 1/100):
  x12 := sin(Pi/12 + 1/100):
  x21 := cos(Pi/12 + 2/100):
  x22 := sin(Pi/12 + 2/100):
```

```
> #points on a circle, coarse discretization
  x01 := cos(Pi/12):
  x02 := sin(Pi/12):
  x11 := cos(Pi/6):
  x12 := sin(Pi/6):
  x21 := cos(Pi/4):
  x22 := sin(Pi/4):
```

```
> #singularity at end point 1
  t0 := 0:
  x1 := x01:
  x2 := x02:
  limF;
```

$$-\frac{1}{4}\left(\left(4\sin\left(\frac{1}{12}\pi\right)-4+2\sqrt{2}\right)\left(-3\cos\left(\frac{1}{12}\pi\right)+2\sqrt{3}-\frac{1}{2}\sqrt{2}\right)-\left(4\cos\left(\frac{1}{12}\pi\right)\right. \tag{10}$$
$$\left.-4\sqrt{3}+2\sqrt{2}\right)\left(-3\sin\left(\frac{1}{12}\pi\right)+2-\frac{1}{2}\sqrt{2}\right)\Big)\Big/$$
$$\left(\left(\left(-3\cos\left(\frac{1}{12}\pi\right)+2\sqrt{3}-\frac{1}{2}\sqrt{2}\right)^2+\left(-3\sin\left(\frac{1}{12}\pi\right)+2-\frac{1}{2}\sqrt{2}\right)^2\right)^{3/2}\pi\right)$$

```
> #singularity at mid point
  t0 := 1/2:
  x1 := x11:
  x2 := x12:
  limF;
```

$$-\frac{1}{4}\left(\left(4\sin\left(\frac{1}{12}\pi\right)-4+2\sqrt{2}\right)\left(-\cos\left(\frac{1}{12}\pi\right)+\frac{1}{2}\sqrt{2}\right)-\left(4\cos\left(\frac{1}{12}\pi\right)-4\sqrt{3}\right. \tag{11}$$
$$\left.+2\sqrt{2}\right)\left(-\sin\left(\frac{1}{12}\pi\right)+\frac{1}{2}\sqrt{2}\right)\Big)\Big/\left(\left(\left(-\cos\left(\frac{1}{12}\pi\right)+\frac{1}{2}\sqrt{2}\right)^2+\right.\right.$$

$$-\sin\left(\frac{1}{12}\pi\right)+\frac{1}{2}\sqrt{2}\Big)^{2}\Big)^{3/2}\pi\Big)$$

```
> #singularity at end point 2
  t0 := 1:
  x1 := x21:
  x2 := x22:
  limF;
```

$$-\frac{1}{4}\left(\left(4\sin\left(\frac{1}{12}\pi\right)-4+2\sqrt{2}\right)\left(\cos\left(\frac{1}{12}\pi\right)-2\sqrt{3}+\frac{3}{2}\sqrt{2}\right)-\left(4\cos\left(\frac{1}{12}\pi\right)-4\sqrt{3}+2\sqrt{2}\right)\left(-2+\sin\left(\frac{1}{12}\pi\right)+\frac{3}{2}\sqrt{2}\right)\right)\Big/\left(\left(\left(\cos\left(\frac{1}{12}\pi\right)-2\sqrt{3}+\frac{3}{2}\sqrt{2}\right)^{2}+\left(-2+\sin\left(\frac{1}{12}\pi\right)+\frac{3}{2}\sqrt{2}\right)^{2}\right)^{3/2}\pi\right) \quad (12)$$

```
> #check the limit numerically
  #note elimination of precision close to the singularity
  kappa := 5.2312 + I * 12.323: #random, kappa is eliminated from
  the limit anyway
  Digits := 16:
  evalLimF := evalf(limF):
  printf("limit: %.12f+%.12fi\n\n", Re(evalLimF), Im(evalLimF));
  printf("%-8s  %-8s  %-31s  %-8s\n", "|dt|", "|dn|", "F", "|dF|");
  for i from 1 to 8 do
    dt := - 10.0^(-i): #normalized distance from singularity
    dn := evalf(sqrt((x1 - g1(t0 + dt))^2 + (x2 - g2(t0 + dt))^2)):
  #real distance from singularity
    evalF := evalf(eval(F, t = t0 + dt)): #function value
    dF := abs(evalF - evalLimF):
    printf("%.2e  %.2e  %+.12f%+.12fi  %.2e\n", abs(dt), abs(dn),
  Re(evalF), Im(evalF), dF):
  end:
limit: -0.073212572834+0.000000000000i

|dt|      |dn|      F                                  |dF|
1.00e-01  5.32e-02  -0.055848932774-0.009768531796i  1.99e-02
1.00e-02  5.35e-03  -0.072828156001-0.000387880560i  5.46e-04
1.00e-03  5.35e-04  -0.073214267235-0.000006987004i  7.19e-06
1.00e-04  5.35e-05  -0.073213414125-0.000000101006i  8.47e-07
1.00e-05  5.35e-06  -0.073212666462-0.000000001321i  9.36e-08
1.00e-06  5.35e-07  -0.073212582491-0.000000000016i  9.66e-09
1.00e-07  5.35e-08  -0.073212579296-0.000000000000i  6.46e-09
1.00e-08  5.35e-09  -0.073212633988-0.000000000000i  6.12e-08
```

C. The JURECA Supercomputer

Numerical computations in this work were performed on the supercomputer JURECA [14] at Forschungszentrum Jülich.

JURECA (Jülich Research on Exascale Cluster Architectures) is a modular system consiting of a cluster module and a booster module. The characteristics of the cluster module are listed below. The booster module with Intel Xeon Phi many-core CPUs was not used for the computations in this work. The standard compute nodes provide sufficient parallelism and memory.

Hardware Characteristics of the Cluster Module

- 1872 compute nodes
  - Two Intel Xeon E5-2680 v3 Haswell CPUs per node
    - 2×12 cores, 2.5 GHz
    - Intel Hyperthreading Technology (Simultaneous Multithreading)
    - AVX 2.0 ISA extension
  - 75 compute nodes equipped with two NVIDIA K80 GPUs (four visible devices per node)
    - 2×4992 CUDA cores
    - 2×24 GiB GDDR5 memory
  - DDR4 memory technology (2133 MHz)
    - 1605 compute nodes with 128 GiB memory
    - 128 compute nodes with 256 GiB memory
    - 64 compute nodes with 512 GiB memory
- 12 visualization nodes
  - Two Intel Xeon E5-2680 v3 Haswell CPUs per node
  - Two NVIDIA K40 GPUs per node
    - 2×12 GiB GDDR5 memory
  - 10 nodes with 512 GiB memory

  - 2 nodes with 1024 GiB memory
- Login nodes with 256 GiB memory per node
- 45,216 CPU cores
- 1.8 (CPU) + 0.44 (GPU) Petaflop per second peak performance
- Based on the T-Platforms V-class server architecture
- Mellanox EDR InfiniBand high-speed network with non-blocking fat tree topology
- 100 GiB per second storage connection to JUST

Software Characteristics

- CentOS 7 Linux distribution
- Parastation Cluster Management
- Slurm batch system with Parastation resource management
- Intel Professional Fortran, C/C++ Compiler
  - Support for OpenMP programming model for intra-node parallelization
- Intel Math Kernel Library
- ParTec MPI (Message Passing Interface) Implementation
- Intel MPI (Message Passing Interface) Implementation
- IBM General Parallel Filesystem (GPFS) 4.1

D. References

[1] Gene M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities." In: *AFIPS Joint Computer Conferences*. 1967, pp. 483–485.

[2] Donald E. Amos. "Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order". In: *ACM Transactions on Mathematical Software* 12.3 (Sept. 1986), pp. 265–273. ISSN: 0098-3500. DOI: 10.1145/7921.214331. URL: http://doi.acm.org/10.1145/7921.214331.

[3] Edward Anderson et al. *LAPACK Users' Guide*. Third Edition. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8.

[4] Pedro R. S. Antunes and Edouard Oudet. "Numerical results for extremal problem for eigenvalues of the Laplacian". In: *Shape optimization and spectral theory*. Ed. by Antoine Henrot. Berlin, Boston: DeGruyter, May 2017, pp. 398–412. ISBN: 978-3-11-055088-7. DOI: 10.1515/9783110550887-011.

[5] *benchmark*. Google LLC. URL: https://github.com/google/benchmark.

[6] Wolf-Jürgen Beyn. "An integral method for solving nonlinear eigenvalue problems". In: *Linear Algebra and its Applications* 436.10 (2012). Special Issue dedicated to Heinrich Voss's 65th birthday, pp. 3839–3863. ISSN: 0024-3795. DOI: 10.1016/j.laa.2011.03.030. URL: http://www.sciencedirect.com/science/article/pii/S0024379511002540.

[7] *BLAS Technical Forum Standard*. URL: https://www.netlib.org/blas/blast-forum.

[8] *CuTest: C Unit Testing Framework*. URL: http://cutest.sourceforge.net.

[9] Mark Galassi et al. *GNU Scientific Library Reference Manual*. 3rd Edition. Network Theory Ltd., 2009. ISBN: 978-0954612078.

[10] Alexandre Girouard, Nikolai Nadirashvili, and Iosif Polterovich. "Maximization of the second positive Neumann eigenvalue for planar domains". In: *Journal of Differential Geometry* 83 (Jan. 2008). DOI: 10.4310/jdg/1264601037.

[11] *GNU Scientific Library*. URL: https://www.gnu.org/software/gsl.

[12] Denis S. Grebenkov and Binh-Thanh Nguyen. "Geometrical structure of Laplacian eigenfunctions". In: *arXiv e-prints* (June 2012). arXiv: 1206.1278 [math.AP]. URL: https://arxiv.org/abs/1206.1278.

[13] *Intel Math Kernel Library (MKL)*. Santa Clara, CA, USA: Intel Corporation. URL: https://software.intel.com/en-us/mkl.

[14] Jülich Supercomputing Centre. "JURECA: Modular supercomputer at Jülich Supercomputing Centre". In: *Journal of large-scale research facilities* 4.A132 (2018). DOI: 10.17815/jlsrf-4-121-1. URL: https://dx.doi.org/10.17815/jlsrf-4-121-1.

[15] Andreas Kleefeld. "Shape optimization for interior Neumann and transmission eigenvalues". In: *Integral Methods in Science and Engineering*. Ed. by Christian Constanda and Paul Harris. Basel: Springer Nature Switzerland AG, 2019. Chap. 15. DOI: 10.1007/978-3-030-16077-7_15.

[16] Andreas Knüpfer et al. "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir". In: *Tools for High Performance Computing 2011*. Ed. by Holger Brunst et al. Berlin, Heidelberg: Springer, 2012, pp. 79–91. ISBN: 978-3-642-31476-6. DOI: 10.1007/978-3-642-31476-6_7.

[17] Rainer Kress. *Linear Integral Equation*. 3rd ed. Vol. 82. Applied Mathematical Sciences. New York et al.: Springer, 2014. ISBN: 978-1-4614-9592-5. DOI: 10.1007/978-1-4614-9593-2.

[18] *LAPACK*. URL: https://www.netlib.org/lapack.

[19] *Maple 2019.0*. Waterloo, Ontario: Maplesoft, a division of Waterloo Maple Inc. URL: https://www.maplesoft.com/products/Maple/.

[20] John A. Nelder and Roger Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (1965), pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

[21] *Netlib Repository Amos*. URL: https://netlib.org/amos.

[22] *OpenBLAS*. URL: https://www.openblas.net.

[23] Guillaume Poliquin and Guillaume Roy-Fortin. "Wolf-Keller theorem for Neumann eigenvalues". In: *arXiv e-prints* (July 2010). arXiv: 1007.4771 [math.SP]. URL: https://arxiv.org/abs/1007.4771v1.

[24] *Score-P*. Virtual Institute - High Productivity Supercomputing (VI-HPS). URL: https://www.vi-hps.org/projects/score-p/.

[25] Gábor Szegö. "Inequalities for Certain Eigenvalues of a Membrane of Given Area". In: *Journal of Rational Mechanics and Analysis* 3 (1954), pp. 343–356. ISSN: 19435282, 19435290. URL: http://www.jstor.org/stable/24900293.

[26] *The NAG Library for C*. Oxford, United Kingdom: The Numerical Algorithms Group (NAG). URL: https://www.nag.com.

[27] *Vampir*. Center for Information Services and High Performance Computing (ZIH), Technische Universität Dresden and Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich. URL: https://vampir.eu.

[28] Hans F. Weinberger. "An Isoperimetric Inequality for the N-Dimensional Free Membrane Problem". In: *Journal of Rational Mechanics and Analysis* 5.4 (1956), pp. 633–636. ISSN: 19435282, 19435290. URL: http://www.jstor.org/stable/24900219.

[29] Daniel Zwillinger. *Standard mathematical tables and formulae*. Boca Raton: CRC Press, 2012. ISBN: 978-1439835487.