Rayleigh Quotient Multigrid

Bachelor Thesis

written by
Robert Gantner

supervised by
Prof. Dr. Ralf Hiptmair

Seminar for Applied Mathematics
ETH Zürich

Spring Semester 2011

Abstract

This thesis elucidates the Rayleigh quotient multigrid algorithm and presents an optimal implementation. Basic considerations regarding the smoothing operation, a coordinate relaxation scheme, are made. The asymptotic complexity if found to be $\mathcal{O}(n)$ analytically and is numerically verified, along with the h-independence of convergence.

Various other numerical experiments are conducted, including empirical determination of the rate of convergence and timings for composite multigrid schemes like nested iteration or the FMG cycle. Some pitfalls of computing multiple eigenvector approximations are discussed and numerically analyzed. This is aided by a suboptimal implementation of complexity $\mathcal{O}(n \log n)$, which allows more flexible manipulation of the iterates.

Contents

List of Symbols

1 Introduction

The goal of this thesis is to analyze an optimal $\mathcal{O}(n)$ implementation of the Rayleigh quotient multigrid algorithm. The implementation is based on the existing Matlab framework LehrFEM, developed at the Seminar for Applied Mathematics at ETH Zürich. [Burtscher *et al.* 2009] Various properties of this algorithm, including optimal complexity and h-independence of convergence, are numerically scrutinized.

The model problem used in this thesis is an elliptic eigenvalue problem. This model problem is stated and formulated as a variational problem in Section 2. The discretization by the finite element method is then elucidated and some fundamental properties of the resulting Galerkin matrices are discussed.

At the core of the Rayleigh quotient multigrid algorithm lies a Rayleigh quotient minimization iteration, which is formulated as a multiplicative Schwarz procedure in Section 3. This iteration will be used as a smoothing procedure in the multigrid method. Some analytical considerations are necessary to optimize the smoothing iteration by updating certain quantities. This is explained in Section 3.5.1.

After the prerequisite tools are defined, Section 4 deals with the Rayleigh quotient multigrid algorithm. The algorithm is stated in detail and intergrid transfers of corrections are derived. This decouples the individual levels from any $\mathcal{O}(n)$ operations, further paving the way for an optimal implementation.

The possibility of calculating multiple eigenvectors is mentioned in Section 4.3, while the Ritz projection is introduced in Section 3.6. Some common multigrid cycles are illustrated, as they allow further reduction of the total runtime. The data structures used in the implementation are then presented, along with other implementation details. A theoretical complexity analysis forms the end of Section 4.

In order to test the correctness and optimality of the implementation, various numerical experiments were conducted and are documented in Section 5. A few sample domains are shown, along with some solutions of the model problem on each of them. The asymptotically linear complexity is numerically verified in Section 5.3, along with the h-independence of convergence in Section 5.4. This is compared to a standard Rayleigh quotient minimization iteration without coarse-grid corrections.

Following these experiments, a variety of computations involving multiple eigenvalue approximations are presented (Section 5.5). In Section 5.6, convergence rates are computed for some of the more common situations and compared to each other. Finally, a comparison of the execution time of the nested iteration scheme, the full multigrid cycle and a standard V-cycle is exhibited in Section 5.7.

2 Elliptic Eigenvalue Problems

Eigenvalue Problems are found in many diverse areas of science. From the eigenfrequencies of bridges to the values of quantum mechanical observables, eigenvectors and their corresponding eigenvalues are important concepts for solving modern scientific problems. [Arbenz & Kressner 2010, p. 1]

2.1 Strong Formulation

In this thesis, we will consider eigenproblems arising from elliptic partial differential equations of the form

$$\boxed{\begin{aligned} -\Delta u &= \lambda u \quad && \text{in } \Omega \\ u &= 0 && \text{on } \partial\Omega \end{aligned}} \tag{1}$$

where the Laplace operator is defined as $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ on a two-dimensional domain Ω, λ is the eigenvalue and $u : \Omega \mapsto \mathbb{R}$ its corresponding eigenfunction.

2.2 Variational Formulation

The partial differential equation given in (1) implies the existence of two derivatives of u in Ω (i.e. $u \in C^2(\Omega)$). This is an unnecessarily specific constraint which can be reduced by deriving a so-called variational, or weak, formulation of the problem. This is desirable because one may be interested in solutions of less smoothness than those fulfilling the strong formulation given in (1). [Friese 1998, p. 7]

To this end, we proceed as follows:

- Multiply with a test function $v \in \mathcal{H}_0^1(\Omega)$
- Integrate over Ω and apply Green's formula

where $\mathcal{H}_0^1$ is the Sobolev space as defined in [Braess 2007, p. 27]:

$$\mathcal{H}_0^1(\Omega) = \left\{ f : \Omega \mapsto \mathbb{R} \;\middle|\; f, \partial_i f \in L_2(\Omega), f_{|\partial\Omega} = 0 \right\}.$$

Executing this procedure yields

$$-\int_\Omega \Delta u \cdot v \, \mathbf{dx} \overset{\text{Green}}{=} \int_\Omega \nabla u \cdot \nabla v \, \mathbf{dx} - \int_{\partial\Omega} \nabla u \cdot v \, \mathbf{ds} \overset{v \in \mathcal{H}_0^1}{=} \int_\Omega \nabla u \cdot \nabla v \, \mathbf{dx}$$

$$\Rightarrow \int_\Omega \nabla u \cdot \nabla v \, \mathbf{dx} = \lambda \int_\Omega u \cdot v \, \mathbf{dx}.$$

Defining $\mathsf{a}(u, v) := \int_\Omega \nabla u \cdot \nabla v \, \mathbf{dx}$ and $\mathsf{m}(u, v) := \lambda \int_\Omega u \cdot v \, \mathbf{dx}$, we obtain the variational formulation:

$$\text{find } u \in \mathcal{H}_0^1(\Omega) \text{ such that:} \quad \mathsf{a}(u, v) = \lambda \mathsf{m}(u, v) \quad \forall v \in \mathcal{H}_0^1(\Omega). \tag{2}$$

Note that all classical solutions of (1) automatically fulfill (2), but not vice-versa.

2.3 Finite Element Galerkin Discretization

The discretization of (2) using a standard Galerkin finite elements approach involves replacing the infinite-dimensional function space $\mathcal{H}_0^1$ with a finite-dimensional subspace $\mathsf{V}_{0,N} \subset \mathcal{H}_0^1$ where the subscript N denotes both the dimension of the subspace and its discrete nature.

The discrete variational problem now reads

$$\text{find } u_N \in \mathsf{V}_{0,N} \text{ s.t.:} \quad \mathsf{a}(u_N, v_N) = \lambda \mathsf{m}(u_N, v_N) \quad \forall v_N \in \mathsf{V}_{0,N}. \tag{3}$$

Both u_N and v_N are taken from the space $\mathsf{V}_{0,N}$, i.e. the trial and test spaces are the same.

2.3.1 Basis of Finite Element Space

In order to find a representation of $\mathsf{V}_{0,N}$ suitable for use on a computer, a basis must be chosen. Let $\mathcal{B}_N = \{b_1, \ldots, b_N\}$ be a basis of $\mathsf{V}_{0,N}$. This leads to the following ansatz for u_N, called the *Ritz ansatz* [Arbenz & Kressner 2010, p. 16]:

$$u_N = \sum_{i=1}^{N} \mu_i b_i. \tag{4}$$

Inserting this into (3) results in the following:

$$\sum_{i=1}^{N} \mu_i \int_\Omega \nabla b_i \nabla v \, \mathbf{dx} = \lambda \sum_{i=1}^{N} \mu_i \int_\Omega b_i v \, \mathbf{dx}.$$

Since $\{b_1, \ldots, b_N\}$ form a basis of $\mathsf{V}_{0,N}$, this is equivalent to:

$$\sum_{i=1}^{N} \mu_i \int_\Omega \nabla b_i \nabla b_j \, \mathbf{dx} = \lambda \sum_{i=1}^{N} \mu_i \int_\Omega b_i b_j \, \mathbf{dx}, \quad j = 1, \ldots, N.$$

By interpreting both summations over i as matrix-vector products, this equation can be written as a generalized eigenvalue problem

$$\mathbf{A}\mu = \lambda \mathbf{M}\mu \tag{5}$$

with

$$(\mathbf{A})_{i,j} = \int_\Omega \nabla b_i \nabla b_j \, \mathbf{dx} \qquad \text{“}\textit{stiffness matrix}\text{”} \tag{6}$$

$$(\mathbf{M})_{i,j} = \int_\Omega b_i b_j \, \mathbf{dx} \qquad \text{“}\textit{mass matrix}\text{”}. \tag{7}$$

2.3.2 Properties of Galerkin Matrices

First, let us state the Poincaré-Friedrichs inequality as presented in [Braess 2007, p. 29]:

Poincaré-Friedrichs Inequality: If Ω is contained in a square of side length s, then

$$||v||_0 \leq s|v|_{\mathcal{H}^1} \quad \forall v \in \mathcal{H}_0^1. \tag{8}$$

Using this inequality, we can show that $\mathbf{A}$ is positive definite:

$$\mu^T \mathbf{A} \mu = \sum_i \mu_i \sum_j \mu_j \int_\Omega \nabla b_i \nabla b_j \, \mathbf{dx} = \int_\Omega \nabla\Big(\sum_i \mu_i b_i\Big) \nabla\Big(\sum_j \mu_j b_j\Big) \, \mathbf{dx}$$

$$= \int_\Omega \nabla u_N \nabla u_N \, \mathbf{dx} = |u_N|^2_{\mathcal{H}^1(\Omega)} \overset{(8)}{\geq} \frac{1}{s^2} ||u_N||_0^2.$$

Since the last term is a norm, the positive definiteness of $\mathbf{A}$ follows from the positivity property of the norm. The symmetry of $\mathbf{A}$ follows trivially from the commutative property of the scalar product.

The matrix $\mathbf{M}$ can also be shown to be spd:

$$\mu^T \mathbf{M} \mu = \sum_i \mu_i \sum_j \mu_j \int_\Omega b_i b_j \, \mathbf{dx} = \int_\Omega u^2 \, \mathbf{dx} \geq 0, \; u \neq 0$$

$$= 0, \; u = 0$$

where the symmetry is of course given by the commutativity of scalar multiplication. [Braess 2007]

3 Rayleigh Quotient Minimization

3.1 Definition of Rayleigh Quotient

One method to find the smallest eigenvalue λ_1 and its corresponding eigenvector $\mathbf{v}_1$ is by minimization of the so-called Rayleigh quotient. The Rayleigh quotient of a generalized eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{Mx}$ is defined as

$$RQ(\mathbf{x}) := \frac{\mathbf{x}^T\mathbf{Ax}}{\mathbf{x}^T\mathbf{Mx}}. \tag{9}$$

It can be seen that if $\mathbf{x}$ is an eigenvector, $RQ(\mathbf{x})$ is the corresponding eigenvalue. The following theorem provides the theoretical framework for general Rayleigh quotient minimization methods.

Courant-Fischer Minimax Theorem: If $\mathbf{A} \in \mathbb{R}^{n\times n}$ is symmetric, then

$$\lambda_k(\mathbf{A}) = \min_{\dim(S)=k} \max_{0\neq\mathbf{x}\in S} \frac{\mathbf{x}^T\mathbf{Ax}}{\mathbf{x}^T\mathbf{x}} \tag{10}$$

for $k = 1, \ldots, n$.

In the above theorem, $\frac{\mathbf{x}^T\mathbf{Ax}}{\mathbf{x}^T\mathbf{x}}$ is the Rayleigh quotient of a special eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{x}$ with λ_1 being the smallest eigenvalue of $\mathbf{A}$. Since we have a generalized eigenvalue problem, we must show that this theorem is still applicable. This can be done by assuming that at least one of the two matrices $\mathbf{A}$ or $\mathbf{M}$ is spd. As proven in the previous section, in our case both $\mathbf{A}$ and $\mathbf{M}$ fulfill this condition; without loss of generality we will use $\mathbf{M}$ in the following. This leads to the following reformulation using the well-defined decomposition $\mathbf{M} = \mathbf{M}^{1/2}\mathbf{M}^{1/2}$:

$$\frac{\mathbf{x}^T\mathbf{Ax}}{\mathbf{x}^T\mathbf{Mx}} = \frac{\mathbf{x}^T\mathbf{Ax}}{(\mathbf{M}^{1/2}\mathbf{x})^T(\mathbf{M}^{1/2}\mathbf{x})} \overset{\mathbf{M}^{1/2}\mathbf{x}=\mathbf{y}}{=} \frac{\mathbf{y}^T\mathbf{M}^{1/2}\mathbf{A}\mathbf{M}^{1/2}\mathbf{y}}{\mathbf{y}^T\mathbf{y}} = \frac{\mathbf{y}^T\mathbf{By}}{\mathbf{y}^T\mathbf{y}},$$

which, since $\mathbf{B}$ must be symmetric, allows us to apply (10) to our generalized eigenvalue problem. [Golub & Van Loan 1996, p. 394]

3.2 Subspace Correction

Instead of minimizing the Rayleigh quotient over the entire domain, a decomposition with (overlapping) subdomains can be constructed:

$$\Omega = \cup_i \Omega_i \qquad\qquad \mathsf{V}_{0,N} = \cup_i \mathsf{V}_i$$

In order to map the coefficient vector of a finite element function on a subspace to the coefficient vector on the entire space, so-called *prolongation* operators $\mathbf{P}_i : \mathsf{V}_i \to \mathsf{V}_{0,N}$ are needed. $\mathbf{P}_i$ is the transpose of $\mathbf{P}_i^T : \mathsf{V}_{0,N} \to \mathsf{V}_i$, the orthogonal projection into the subspace V_i.

Using such a decomposition, a multiplicative Schwarz procedure can be used as an iterative method to determine the solution on the entire domain. This will allow us to solve multiple small problems instead of one very large one, which is computationally less intensive. The multiplicative Schwarz procedure is summarized as presented in [Chan & Sharapov 1998]:

Algorithm 1 *Subspace Correction*

Input: Initial guess $\mathbf{x}^0$, discrete subspaces $\{\mathsf{V}_i\}_{i=1}^N$, prolongation operators $\mathbf{P}_i$
Output: Approximation $\mathbf{x}$ on $\mathsf{V}_{0,N}$

1: **repeat**
2: **for** $i = 1$ to N **do**
3: $\mathrm{RQ}(\mathbf{x}^{k+\frac{i}{N}}) = \min_{d_i \in \mathsf{V}_i} \mathrm{RQ}(\mathbf{x}^{k+\frac{i-1}{N}} + \mathbf{P}_i d_i)$
4: **end for**
5: **until** convergence

In practice, the subspace iteration in Algorithm 1 is not computed until convergence. Instead, the iteration on lines 2 to 4 is performed a given number of times (called ν in the following). This method will be used as a smoother in the RQMG algorithm.

As shown in [Chan & Sharapov 1998], the subspace problem is also an eigenvalue problem. Therefore, the solution on the subspace can be recursively calculated. As can be seen in line 3 of Algorithm 1, the subspace problem is the minimization of:

$$\mathrm{RQ}(\mathbf{x} + \mathbf{P}_i d_i) = \frac{(\mathbf{x} + \mathbf{P}_i d_i)^T \mathbf{A} (\mathbf{x} + \mathbf{P}_i d_i)}{(\mathbf{x} + \mathbf{P}_i d_i)^T \mathbf{M} (\mathbf{x} + \mathbf{P}_i d_i)}.$$

With $(\mathbf{x} + \mathbf{P}_i d_i) = \begin{pmatrix} \mathbf{P}_i & \mathbf{x} \end{pmatrix} \cdot \begin{pmatrix} d_i \\ 1 \end{pmatrix} = \widetilde{\mathbf{P}_i} \cdot \widetilde{d_i}$, we can write:

$$\mathrm{RQ}(\mathbf{x} + \mathbf{P}_i d_i) = \frac{\widetilde{d_i}^T \widetilde{\mathbf{P}_i}^T \mathbf{A} \widetilde{\mathbf{P}_i} \widetilde{d_i}}{\widetilde{d_i}^T \widetilde{\mathbf{P}_i}^T \mathbf{M} \widetilde{\mathbf{P}_i} \widetilde{d_i}} = \frac{\widetilde{d_i}^T \widetilde{\mathbf{A}} \widetilde{d_i}}{\widetilde{d_i}^T \widetilde{\mathbf{M}} \widetilde{d_i}},$$

which corresponds to the Rayleigh quotient of the following problem:

$$\widetilde{\mathbf{A}} \widetilde{d_i} = \widetilde{\lambda} \widetilde{\mathbf{M}} \widetilde{d_i}.$$

This subspace problem has the dimension $n_i + 1 \times n_i + 1$, where n_i is the dimension of the i-th subspace. As mentioned above, this problem is also an eigenvalue problem,

so by decomposing the chosen subdomain again a recursive algorithm can be formulated to iteratively solve the problem on the whole domain. [Chan & Sharapov 1998]

In this thesis, however, the so-called coordinate relaxation method is used. This is equivalent to using one-dimensional subspaces, which of course cannot be further decomposed. The benefit of this approach is the simplicity of the prolongation operation and of the minimization of the Rayleigh quotient on the one-dimensional subdomains, which corresponds to the solution of a 2×2 generalized eigenvalue problem. Additionally, this solution can be found very efficiently by determining the roots of a quadratic polynomial in δ. This special case is now further investigated.

3.3 Line Search

The approach we will use to find an approximation to the smallest eigenvalue is to find an iteration that fulfills the condition $RQ(\mathbf{x}_{k+1}) < RQ(\mathbf{x}_k)$. If this iteration converges, it will give us a sequence of eigenvalues which converge to λ_1, as well as a sequence of eigenvectors $\mathbf{x}_k$ which converge to $\mathbf{v}_1$.

To realize such an iteration, we modify the current iterate $\mathbf{x}_k$ by adding a search direction $\mathbf{p}_k$ scaled with a "step length" δ_k:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k \cdot \mathbf{p}_k, \tag{11}$$

where δ_k is chosen such that $\mathbf{x}_{k+1}$ minimizes the Rayleigh quotient in the direction $\mathbf{p}_k$:

$$RQ(\mathbf{x}_{k+1}) = \min_{\delta_k} RQ(\mathbf{x}_k + \delta_k \cdot \mathbf{p}_k) \tag{12}$$

The value for δ_k can be computed by rewriting the Rayleigh quotient of the generalized eigenvalue problem (5) as the Rayleigh quotient of a small 2×2 eigenvalue problem:

$$\begin{aligned} RQ(\mathbf{x}_k + \delta\mathbf{p}_k) &= \frac{(\mathbf{x}_k + \delta\mathbf{p}_k)^T \mathbf{A}(\mathbf{x}_k + \delta\mathbf{p}_k)}{(\mathbf{x}_k + \delta\mathbf{p}_k)^T \mathbf{M}(\mathbf{x}_k + \delta\mathbf{p}_k)} = \frac{\mathbf{x}_k^T\mathbf{A}\mathbf{x}_k + 2\delta\mathbf{x}_k^T\mathbf{A}\mathbf{p}_k + \delta^2\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}{\mathbf{x}_k^T\mathbf{M}\mathbf{x}_k + 2\delta\mathbf{x}_k^T\mathbf{M}\mathbf{p}_k + \delta^2\mathbf{p}_k^T\mathbf{M}\mathbf{p}_k} \\ &= \frac{\begin{pmatrix} 1 & \delta \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x}_k^T\mathbf{A}\mathbf{x}_k & \mathbf{x}_k^T\mathbf{A}\mathbf{p}_k \\ \mathbf{x}_k^T\mathbf{A}\mathbf{p}_k & \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \delta \end{pmatrix}}{\begin{pmatrix} 1 & \delta \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x}_k^T\mathbf{M}\mathbf{x}_k & \mathbf{x}_k^T\mathbf{M}\mathbf{p}_k \\ \mathbf{x}_k^T\mathbf{M}\mathbf{p}_k & \mathbf{p}_k^T\mathbf{M}\mathbf{p}_k \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \delta \end{pmatrix}}. \end{aligned}$$

The last term corresponds to the Rayleigh quotient of the eigenvalue problem

$$\begin{pmatrix} \mathbf{x}_k^T\mathbf{A}\mathbf{x}_k & \mathbf{x}_k^T\mathbf{A}\mathbf{p}_k \\ \mathbf{x}_k^T\mathbf{A}\mathbf{p}_k & \mathbf{p}_k^T\mathbf{A}\mathbf{p}_k \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{x}_k^T\mathbf{M}\mathbf{x}_k & \mathbf{x}_k^T\mathbf{M}\mathbf{p}_k \\ \mathbf{x}_k^T\mathbf{M}\mathbf{p}_k & \mathbf{p}_k^T\mathbf{M}\mathbf{p}_k \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{13}$$

with two eigenvalues $\tilde{\lambda}_1 < \tilde{\lambda}_2$. Since the Rayleigh quotients are equal, we can solve the minimization problem (12) by computing the eigenvector of (13) corresponding to the smaller eigenvalue $\tilde{\lambda}_1$. By scaling the eigenvector $(\alpha \ \beta)^T$ such that $\alpha = 1$, the second component becomes the unknown value δ_k. [Arbenz & Kressner 2010, p. 213]

Algorithm 2 `minRQ`$(\mathbf{A}, \mathbf{M}, \mathbf{x}, \mathbf{p})$ *Minimization of Rayleigh Quotient*

Input: System matrices $\mathbf{A}, \mathbf{M}$, initial guess $\mathbf{x}$, search direction $\mathbf{p}$
Output: Step size δ

1: $\widetilde{\mathbf{A}} = \begin{pmatrix} \mathbf{x}^T\mathbf{A}\mathbf{x} & \mathbf{x}^T\mathbf{A}\mathbf{p} \\ \mathbf{x}^T\mathbf{A}\mathbf{p} & \mathbf{p}^T\mathbf{A}\mathbf{p} \end{pmatrix}$
2: $\widetilde{\mathbf{M}} = \begin{pmatrix} \mathbf{x}^T\mathbf{M}\mathbf{x} & \mathbf{x}^T\mathbf{M}\mathbf{p} \\ \mathbf{x}^T\mathbf{M}\mathbf{p} & \mathbf{p}^T\mathbf{M}\mathbf{p} \end{pmatrix}$
3: $\mathbf{V}, \Lambda = \texttt{EIG}(\widetilde{\mathbf{A}}, \widetilde{\mathbf{M}})$
4: **if** $\Lambda_{1,1} < \Lambda_{2,2}$ **then**
5: **return** $\mathbf{V}_{2,1}/\mathbf{V}_{1,1}$
6: **else**
7: **return** $\mathbf{V}_{2,2}/\mathbf{V}_{1,2}$
8: **end if**

3.3.1 Analytical Solution

The unknown δ_k in (11) can be found by solving the minimization problem (12) analytically. In other words, we seek the δ that minimizes the Rayleigh quotient

$$RQ(\mathbf{x} + \delta\mathbf{p}) = \frac{\begin{pmatrix} 1 & \delta \end{pmatrix} \cdot \widetilde{\mathbf{A}} \cdot \begin{pmatrix} 1 \\ \delta \end{pmatrix}}{\begin{pmatrix} 1 & \delta \end{pmatrix} \cdot \widetilde{\mathbf{M}} \cdot \begin{pmatrix} 1 \\ \delta \end{pmatrix}},$$

where $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{M}}$ are the 2×2 matrices from (13). Referring to the elements of $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{M}}$ as a_{ij} and m_{ij} and executing the matrix multiplications yields

$$RQ(\mathbf{x} + \delta\mathbf{p}) = \frac{a_{11} + 2\delta a_{12} + \delta^2 a_{22}}{m_{11} + 2\delta m_{12} + \delta^2 m_{22}}. \tag{14}$$

A necessary condition for the minimizer $\delta^\star$ is $\frac{\partial}{\partial\delta} RQ(\mathbf{x} + \delta\mathbf{p})|_{\delta=\delta^\star} = 0$, which leads to the following quadratic equation:

$$\delta^2 \underbrace{(a_{22}m_{12} - a_{12}m_{22})}_{\alpha} + \delta \underbrace{(a_{22}m_{11} - a_{11}m_{22})}_{\beta} + \underbrace{a_{12}m_{11} - a_{11}m_{12}}_{\gamma} = 0.$$

The two solutions of this equation are of course

$$\delta_{\pm} = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha}.$$

By assuming that the current iterate $\mathbf{x}$ has a smaller Rayleigh quotient than any single basis function, i.e.

$$RQ(\mathbf{x}) = \frac{a_{11}}{m_{11}} < \frac{a_{22}}{m_{22}} = RQ(\mathbf{P}_i d_i), \tag{15}$$

it follows that $\beta = a_{22}m_{11} - a_{11}m_{22} > 0$. Inserting into the second derivative of (14), it is evident that the solution δ_- always yields a maximum:

$$\frac{\partial^2}{\partial\delta^2} RQ(\mathbf{x}+\delta\mathbf{p})|_{\delta=\delta_-} = 2\alpha\delta_- - \beta = 2\alpha\frac{-\beta - \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha} - \beta = -2\beta - \sqrt{\beta^2 - 4\alpha\gamma} < 0.$$

Therefore, $\delta_+ \leq \delta_-$, which makes it clear that δ_+ is the minimizer of the Rayleigh quotient. In practice, assumption (15) is always true. [Mandel & McCormick 1989]

3.4 Intergrid Transfer Operators

An important component of every multigrid algorithm is a method of prolongating coefficient vectors from coarse grids to finer grids. This should not be confused with the prolongation operator defined in Section 3.2, which, in the case of coordinate relaxation, maps one-dimensional subspaces V_i onto the entire discrete finite element space $\mathsf{V}_{0,N}$.

Intergrid transfer operators map the coefficient vector of a finite element function on a coarse grid (n_c components) to the coefficient vector of the function on a fine grid ($n_f > n_c$ components). They are also referred to as prolongation and restriction, where restriction is the adjoint of prolongation, mapping residuals from the fine grid to residuals on the coarse grid.

In the following, the intergrid prolongation and restriction operations are denoted by the operator $\mathbf{I}_i^j$ which describes prolongation from level i to level j for $i < j$ and restriction between the corresponding levels for $j < i$. For the sake of completeness, $\mathbf{I}_i^i$ corresponds to the $n_i \times n_i$ identity matrix. The finest level is described by the index m. Using this convention, $\mathbf{I}_j^m$ prolongates the coefficient vector on level j to the finest level. [Mandel & McCormick 1989, 445]

3.5 Coordinate Relaxation

Coordinate relaxation on a level l involves applying the Rayleigh quotient minimization procedure from Algorithm 2 using the basis vectors d_i^l as the search directions.

On the finest level, this corresponds to an iteration over the euclidean basis vectors. In each iteration, the iterate $\mathbf{x}$ is corrected according to the minimization of the Rayleigh quotient in the direction of the current basis vector as represented on the fine grid ($\mathbf{I}_l^m d_i^l$). This is a one-dimensional subspace correction in the sense of the approach used in Algorithm 1.

Algorithm 3 `CoordRelax`$(\mathbf{A}, \mathbf{M}, \mathbf{x}, l)$ *Coordinate Relaxation*

Input: System matrices $\mathbf{A}, \mathbf{M}$ on finest level, initial guess $\mathbf{x}$, level l.
Output: A better approximation of the eigenvector to the smallest eigenvalue.

1: **for** $i = 1 : n$ **do**
2: $\quad \delta = \texttt{minRQ}(\mathbf{A}, \mathbf{M}, \mathbf{x}, \mathbf{I}_l^m d_i^l)$
3: $\quad \mathbf{x} = \mathbf{x} + \delta \cdot \mathbf{I}_l^m d_i^l$
4: **end for**
5: **return** $\mathbf{x}$

3.5.1 Optimizations

The minimization of the Rayleigh quotient as described above requires $\mathcal{O}(n)$ operations per search direction due to the computation of the elements of $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{M}}$ in the procedure `minRQ` (Algorithm 2). A much more efficient version of this inner-loop function can be obtained by updating the values for $\mathbf{x}^T\mathbf{A}\mathbf{x}$, $\mathbf{x}^T\mathbf{M}\mathbf{x}$, $\mathbf{x}^T\mathbf{A}\mathbf{p}$ and $\mathbf{x}^T\mathbf{M}\mathbf{p}$ after computing each δ on a certain level l. This can be done in constant time, instead of the $\mathcal{O}(n)$ complexity needed for recomputing these values each time. A derivation of the update formulas will now be presented. The derivations are given for the quantities involving the fine-grid matrix $\mathbf{A}$ only; they are identical for the quantities involving $\mathbf{M}$.

Consider a coordinate relaxation iteration on level l. The first correction to $\mathbf{x}_0$ is found by minimizing the Rayleigh quotient in the direction $\mathbf{p}_1 = \mathbf{I}_l^m d_1^l$, i.e. in the direction of the first basis vector on level l. By inserting $\mathbf{x}_1 = \mathbf{x}_0 + \delta^\star \mathbf{p}_1$ into the expressions mentioned above, we obtain the following update formulas.

$\mathbf{x}^T\mathbf{A}\mathbf{x}$:

$$\begin{aligned}\mathbf{x}_1^T\mathbf{A}\mathbf{x}_1 &= (\mathbf{x}_0 + \delta^\star \mathbf{I}_l^m d_1^l)^T \mathbf{A}(\mathbf{x}_0 + \delta^\star \mathbf{I}_l^m d_1^l) \\ &= \mathbf{x}_0^T\mathbf{A}\mathbf{x}_0 + 2\delta^\star \underbrace{\mathbf{x}_0^T\mathbf{A}\mathbf{I}_l^m d_1^l}_{\mathbf{x}_0^T\mathbf{A}\mathbf{p}_1} + \delta^{\star 2} \underbrace{{d_1^l}^T \mathbf{I}_m^l \mathbf{A}\mathbf{I}_l^m d_1^l}_{(\mathbf{A}_l)_{1,1}}\end{aligned}$$

This update requires only the already known quantities $\mathbf{x}^T\mathbf{A}\mathbf{p}$ and $(\mathbf{A}_l)_{i,i}$ and can therefore be executed in constant time.

$\mathbf{x}^T\mathbf{A}\mathbf{p}$:

For the quantity needed in the second iteration, we obtain:

$$\begin{aligned}\mathbf{x}_1^T\mathbf{A}\mathbf{p}_2 &= (\mathbf{x}_0 + \delta_1^\star \mathbf{I}_l^m d_1^l)^T \mathbf{A}(\mathbf{I}_l^m d_2^l) \\ &= \mathbf{x}_0^T \mathbf{A}\mathbf{I}_l^m d_2^l + \delta_1^\star \underbrace{{d_1^l}^T \mathbf{I}_m^l \mathbf{A}\mathbf{I}_l^m d_2^l}_{(\mathbf{A}_l)_{1,2}}.\end{aligned}$$

For the third iteration, we obtain:

$$\begin{aligned}\mathbf{x}_2^T\mathbf{A}\mathbf{p}_3 &= (\mathbf{x}_0 + \delta_1^\star \mathbf{I}_l^m d_1^l + \delta_2^\star \mathbf{I}_l^m d_2^l)^T \mathbf{A}(\mathbf{I}_l^m d_3^l) \\ &= \mathbf{x}_0^T \mathbf{A}\mathbf{I}_l^m d_3^l + \delta_1^\star \underbrace{{d_1^l}^T \mathbf{I}_m^l \mathbf{A}\mathbf{I}_l^m d_3^l}_{(\mathbf{A}_l)_{1,3}} + \delta_2^\star \underbrace{{d_2^l}^T \mathbf{I}_m^l \mathbf{A}\mathbf{I}_l^m d_3^l}_{(\mathbf{A}_l)_{2,3}}.\end{aligned}$$

We see that in the i-th iteration we require a multiplication of the first $i-1$ elements of the i-th column of $\mathbf{A}_l$ with the already determined values $\delta_j^\star$, $j = 1, \ldots, i-1$. This can be realized most efficiently by storing the values $\delta_j^\star$ in a vector, which is initialized to zero. Since only a dot product of this vector with a sparse column of $\mathbf{A}$ is required, the complexity of the second part of this update is constant.

The first part of the update involves the expression $\mathbf{x}_0^T\mathbf{A}\mathbf{I}_l^m$, which can be precomputed before starting the coordinate relaxation loop. Multiplication with d_i^l then simplifies to just accessing the i-th element. In an optimal implementation, this quantity must be passed along from coarser or finer grids. This is explained in detail in Section 4.2.

$\mathbf{p}^T\mathbf{A}\mathbf{p}$:

For $\mathbf{p}^T\mathbf{A}\mathbf{p}$ we need no update – in the i-th iteration, this quantity corresponds to the i-th diagonal element of $\mathbf{A}_l$.

These updates lead to a smoothing procedure requiring only $\mathcal{O}(n)$ operations per sweep over all coordinates. The complexity of the entire algorithm is analyzed in more detail in Section 4.7. [Mandel & McCormick 1989, p. 447]

3.6 Ritz Projection

An effective way of increasing the speed of convergence when computing multiple eigenvectors is by using Ritz projections at some point in the multigrid cycle. After a few smoothing or coarse-grid correction steps, one can assume that the space spanned by $\{\mathbf{x}_i\}_{i=1}^k$ is a good approximation of the eigenspace containing the desired eigenvectors.

Better approximations to the desired eigenvectors can be obtained by writing them as linear combinations of the current iterates:

$$\widetilde{\mathbf{x}}_i = \sum_j \alpha_j \cdot \mathbf{x}_j = \mathbf{X} \cdot \alpha \,, \tag{16}$$

where $\mathbf{X}$ is the matrix containing the orthonormal iterates $\mathbf{x}_i$ as columns and α is the unknown vector of coefficients.

The Rayleigh quotient of this expression is

$$RQ_{\mathbf{A},\mathbf{M}}(\mathbf{X} \cdot \alpha) = \frac{\alpha^T \mathbf{X}^T \mathbf{A} \mathbf{X} \alpha}{\alpha^T \mathbf{X}^T \mathbf{M} \mathbf{X} \alpha} = \frac{\alpha^T \widetilde{\mathbf{A}} \alpha}{\alpha^T \widetilde{\mathbf{M}} \alpha} = RQ_{\widetilde{\mathbf{A}},\widetilde{\mathbf{M}}}(\alpha) \,,$$

which is the Rayleigh quotient of the small $k \times k$ eigenvalue problem

$$(\mathbf{X}^T \mathbf{A} \mathbf{X})\alpha = \tilde{\lambda}(\mathbf{X}^T \mathbf{M} \mathbf{X})\alpha \,. \tag{17}$$

Since (17) is such a small eigenvalue problem, its solutions can be determined efficiently, resulting in the eigenvectors α which give the coefficients of the linear combination in (16). Since the Rayleigh quotients are the same, the sorting of the new iterates $\widetilde{\mathbf{x}}_i$ is determined by the eigenvalues of (17). Implementation details can be found in Section 4.6.3. [Hiptmair *et al.* 2010]

4 Rayleigh Quotient Multigrid

In this section, the Rayleigh quotient multigrid algorithm (RQMG) is presented and details concerning its implementation are described. The algorithm is first presented for computing only the eigenvector to the smallest eigenvalue and then generalized to be able to compute k eigenvectors.

4.1 Algorithm

The first requirement for the RQMG algorithm is a set of nested finite element discretizations of the given elliptic eigenvalue problem. In this thesis, this is done in two dimensions using linear finite elements, with finer levels obtained by regular refinement.

A second requirement is a procedure that reduces the high-frequency components of the error, called a "smoother". In our application, we use coordinate relaxation as elucidated in Section 3.5 and Algorithm 3. This iteration is applied ν_1 times before (presmoothing) and ν_2 times after (postsmoothing) the coarse-grid correction.

In this implementation, a distinction is made between smoothing on the finest grid and on the coarser grids, as additional optimizations are possible on the finest grid. Smoothing on the fine grid is achieved by the procedure `SMOOTH_FINE`, which returns the smoothed solution (see implementation in Section 4.6.2). On the coarser grids, the procedure `SMOOTH_STEP` returns the correction to the current iterate in terms of the basis functions on the given level.

Between the two smoothing steps, a coarse-grid correction is computed by recursively calling the `RQMG_STEP` function. This causes the smoother to be recursively applied to each level until the coarsest level is reached. The corrections are then propagated up to the finest grid, with a smoothing operation being executed on each level after the correction is applied.

Algorithm 4 forms the main body of this function, calling the recursion until a given tolerance (with respect to the error functional $\varepsilon(\mathbf{x})$) is reached. The recursive coarse-grid correction for a single eigenvector is shown in Algorithm 5.

In order to obtain an efficient implementation, various quantities must be updated globally or transferred between the levels. Among these are $\alpha, \beta, \mathbf{q}_l$ and $\mathbf{r}_l$ in Algorithm 5. The global quantities α and β correspond to $\mathbf{x}^T\mathbf{A}\mathbf{x}$ and $\mathbf{x}^T\mathbf{M}\mathbf{x}$, respectively; $\mathbf{q}_l$ and $\mathbf{r}_l$ are explained in the following section. [Mandel & McCormick 1989]

Algorithm 4 RQMG$(\mathbf{A}_l, \mathbf{M}_l, \mathbf{x}_0, m, tol, \nu_1, \nu_2, \gamma)$ *Rayleigh Quotient Multigrid*

Input: System matrices $\mathbf{A}_l, \mathbf{M}_l$ on all levels l; initial guess $\mathbf{x}_0$; finest level m; tolerance tol; number of pre and postsmoothing steps ν_1, ν_2; cycle parameter γ
Output: Better approximation $\mathbf{x}$ of the eigenvector the the smallest eigenvalue λ

$\alpha = \mathbf{x}_0^T \mathbf{A}\mathbf{x}_0; \quad \beta = \mathbf{x}_0^T \mathbf{M}\mathbf{x}_0$
while $\varepsilon(\mathbf{x}) > tol$ **do**
  $\mathbf{x} =$ SMOOTH_FINE$(\mathbf{A}_m, \mathbf{M}_m, \mathbf{x}_0, \nu_1, \alpha, \beta)$
  $\mathbf{q}_{m-1} = \mathbf{I}_m^{m-1} \mathbf{A}_m \mathbf{x}; \quad \mathbf{r}_{m-1} = \mathbf{I}_m^{m-1} \mathbf{M}_m \mathbf{x}$
  for $i = 1, \ldots, \gamma$ **do**
    $\mathbf{c} =$ RQMG_STEP$(\mathbf{A}_{m-1}, \mathbf{M}_{m-1}, m-1, \alpha, \beta, \mathbf{q}_{m-1}, \mathbf{r}_{m-1}, \nu_1, \nu_2, \gamma)$
    $\mathbf{x} = \mathbf{x} + \mathbf{I}_{m-1}^m \mathbf{c}$
  end for
  $\mathbf{x} =$ SMOOTH_FINE$(\mathbf{A}_m, \mathbf{M}_m, \mathbf{x}, \nu_2, \alpha, \beta)$
end while
return $\mathbf{x}_l$

Algorithm 5 RQMG_STEP$(\mathbf{A}_l, \mathbf{M}_l, l, \alpha, \beta, \mathbf{q}_l, \mathbf{r}_l, \nu_1, \nu_2, \gamma)$ *Rayleigh Quotient Multigrid Step*

Input: System matrices $\mathbf{A}_l, \mathbf{M}_l$; level l; dynamic variables $\alpha, \beta, \mathbf{q}_l, \mathbf{r}_l$; number of pre and postsmoothing steps ν_1, ν_2; cycle parameter γ
Output: Correction on current level $\mathbf{x}$

$\mathbf{x} = \mathbf{0}$
for $i = 1, \ldots, \nu_1$ **do**
  $\mathbf{c} =$ SMOOTH_STEP$(\mathbf{A}_l, \mathbf{M}_l, l, \alpha, \beta, \mathbf{q}_l, \mathbf{r}_l)$
  $\mathbf{q}_l = \mathbf{q}_l + \mathbf{A}_l \cdot \mathbf{c}; \quad \mathbf{r}_l = \mathbf{r}_l + \mathbf{M}_l \cdot \mathbf{c}$
  $\mathbf{x} = \mathbf{x} + \mathbf{c}$
end for
if $l \neq 1$ **then**
  $\mathbf{q}_{l-1} = \mathbf{I}_l^{l-1} \mathbf{q}_l; \quad \mathbf{r}_{l-1} = \mathbf{I}_l^{l-1} \mathbf{r}_l$
  for $g = 1, \ldots, \gamma$ **do**
    $\mathbf{c} =$ RQMG_STEP$(\mathbf{A}_{l-1}, \mathbf{M}_{l-1}, l-1, \alpha, \beta, \mathbf{q}_{l-1}, \mathbf{r}_{l-1}, \nu_1, \nu_2, \gamma)$
    $\mathbf{q}_l = \mathbf{q}_l + \mathbf{A}_l \cdot \mathbf{I}_{l-1}^l \mathbf{c}; \quad \mathbf{r}_l = \mathbf{r}_l + \mathbf{M}_l \cdot \mathbf{I}_{l-1}^l \mathbf{c}$
    $\mathbf{x} = \mathbf{x} + \mathbf{I}_{l-1}^l \mathbf{c}$
  end for
  for $i = 1, \ldots, \nu_2$ **do**
    $\mathbf{c} =$ SMOOTH_STEP$(\mathbf{A}_l, \mathbf{M}_l, l, \alpha, \beta, \mathbf{q}_l, \mathbf{r}_l)$
    $\mathbf{q}_l = \mathbf{q}_l + \mathbf{A}_l \cdot \mathbf{c}; \quad \mathbf{r}_l = \mathbf{r}_l + \mathbf{M}_l \cdot \mathbf{c}$
    $\mathbf{x} = \mathbf{x} + \mathbf{c}$
  end for
end if
return $\mathbf{x}$

4.2 Intergrid Transfers

An important aspect of an optimal implementation of the RQMG algorithm is the fact that the computation of corrections on a certain level should only depend on quantities of that level, i.e. one should not have to use the fine-grid matrices $\mathbf{A}_m$ and $\mathbf{M}_m$ on each level. This requires a reformulation of the computation of the quantities $\mathbf{x}_0^T\mathbf{A}_m\mathbf{I}_l^m$ and $\mathbf{x}_0^T\mathbf{M}_m\mathbf{I}_l^m$, mentioned in Section 3.5.1, which involve computations with fine-grid quantities on every level.

Fine to Coarse First consider the transfer from fine to coarse grids. Given the quantity $\mathbf{x}_0^T\mathbf{A}_m\mathbf{I}_l^m$ and the correction $\mathbf{c}_l$ on level l due to the presmoothing step, we seek an expression for $\mathbf{x}_0^T\mathbf{A}_m\mathbf{I}_{l-1}^m$ on level $l-1$. The subscript 0 in the expressions ($\mathbf{x}_0$) signifies the value at the beginning of the smoothing iteration, before any correction to $\mathbf{x}_0$ is computed. After presmoothing on level l, resulting in the correction $\mathbf{c}_l$, we have

$$\mathbf{x}^T\mathbf{A}_m\mathbf{I}_l^m = (\mathbf{x}_0 + \mathbf{I}_l^m\mathbf{c}_l)^T\mathbf{A}_m\mathbf{I}_l^m = \mathbf{x}_0^T\mathbf{A}_m\mathbf{I}_l^m + \mathbf{c}_l^T\underbrace{\mathbf{I}_m^l\mathbf{A}_m\mathbf{I}_l^m}_{\mathbf{A}_l},$$

which, combined with $\mathbf{I}_{l-1}^m = \mathbf{I}_l^m \cdot \mathbf{I}_{l-1}^l$, yields

$$\mathbf{x}_0^T\mathbf{A}_m\mathbf{I}_{l-1}^m = (\mathbf{x}_0^T\mathbf{A}_m\mathbf{I}_l^m + \mathbf{c}_l^T\mathbf{A}_l)\cdot\mathbf{I}_{l-1}^l.$$

In order to simplify the notation, the quantity $\mathbf{x}_0^T\mathbf{A}_m\mathbf{I}_l^m$ will be referred to as $\mathbf{q}_l^T$ and $\mathbf{x}_0^T\mathbf{M}_m\mathbf{I}_l^m$ as $\mathbf{r}_l^T$. Since the derivation with the matrix $\mathbf{M}$ is identical, this yields the following intergrid updates:

$$\begin{aligned}\mathbf{q}_{l-1} &= \mathbf{I}_l^{l-1}(\mathbf{q}_l + \mathbf{A}_l\mathbf{c}_l)\\ \mathbf{r}_{l-1} &= \mathbf{I}_l^{l-1}(\mathbf{r}_l + \mathbf{M}_l\mathbf{c}_l),\end{aligned}$$

which only involve level l quantities. These can be seen in Algorithm 5 on line 8, where the presmoothing corrections $\mathbf{c}_l$ were already applied on line 4.

Coarse to Fine The second type of transfer is needed when coarse-grid corrections are propagated upwards to finer grids. The quantities $\mathbf{x}^T\mathbf{A}_m\mathbf{I}_l^m$ and $\mathbf{x}^T\mathbf{M}_m\mathbf{I}_l^m$ must be updated in order to incorporate the changes made to $\mathbf{x}$ during the coarse-grid operations. Let $\mathbf{x}_b$ be the value of $\mathbf{x}$ before the coarse-grid correction operation on level l. We must add to this the coarse-grid corrections, as represented on the current level l, in order to determine the correct value for $\mathbf{x}^T\mathbf{A}\mathbf{I}_l^m$ before the postsmoothing process. This yields

$$\mathbf{x}^T\mathbf{A}\mathbf{I}_l^m = (\mathbf{x}_b + \mathbf{I}_{l-1}^m\mathbf{c}_{l-1} + \mathbf{I}_{l-2}^m\mathbf{c}_{l-2} + \ldots)^T\mathbf{A}\mathbf{I}_l^m,$$

where $\mathbf{c}_k$, $k < l$ represents the changes due to smoothing on a coarser level k. By defining $\widetilde{\mathbf{c}}_l := \mathbf{c}_l + \mathbf{I}_{l-1}^l \widetilde{\mathbf{c}}_{l-1}$ on each level, the expression simplifies to

$$\mathbf{x}^T \mathbf{A}\mathbf{I}_l^m = (\mathbf{x}_b + \mathbf{I}_{l-1}^m \widetilde{\mathbf{c}}_{l-1})^T \mathbf{A}\mathbf{I}_l^m = \mathbf{x}_b^T \mathbf{A}\mathbf{I}_l^m + (\mathbf{I}_{l-1}^l \widetilde{\mathbf{c}}_{l-1})^T \underbrace{\mathbf{I}_m^l \mathbf{A}\mathbf{I}_l^m}_{\mathbf{A}_l},$$

which, together with the notation adopted above and the fact that $\mathbf{x}_b^T \mathbf{A}\mathbf{I}_l^m$ is already known, yields the updates

$$\begin{aligned} \mathbf{q}_l &= \mathbf{q}_l + \mathbf{A}_l \mathbf{I}_{l-1}^l \widetilde{\mathbf{c}}_{l-1} \\ \mathbf{r}_l &= \mathbf{r}_l + \mathbf{M}_l \mathbf{I}_{l-1}^l \widetilde{\mathbf{c}}_{l-1}. \end{aligned}$$

This is precisely what is done on line 11 of Algorithm 5. If multiple coarse-grid corrections are desired (for example in the case of a W-cycle), they must of course all be included in the update, which is why it is done inside of the loop over γ.

4.3 Multiple Eigenvectors

Multiple eigenvectors can be computed with a subspace iteration. The Rayleigh quotient is minimized for k eigenvector approximations independently as before, with an orthogonalization procedure being executed somewhere in the iteration. This orthogonalization is usually executed on the finest level, for example in Algorithm 4 after the coarse-grid correction is applied on line 7.

However, depending on the cycle parameter γ (particularly for $\gamma > 1$), this orthogonalization is not executed frequently enough to keep the eigenvectors sufficiently separated. Therefore, one must have the possibility of orthogonalizing on coarser grids as well, which involves forming the linear combinations of all basis functions as represented on the finest level. This leads to a destruction of the optimality of the implementation due to the fact that one must perform $\mathcal{O}(n)$ operations on up to $m \propto \log(n)$ coarse levels. For practical applications and $\gamma = 1$ (V-cycle) this problem is avoided.

In order to numerically observe this phenomenon, a suboptimal implementation was created which allows orthogonalization (along with a Ritz projection as discussed in Section 3.6) to be done selectively on a specified level. For details, see Section 5.5.2.

4.4 Multigrid Cycles

In Algorithm 5, the parameter γ denotes the number of coarse-grid corrections applied on each level and is known as the "cycle parameter". The two most common

cycles, the V-cycle and the W-cycle, as well as the full multigrid cycle and the nested iteration technique are now quickly explained.

V-Cycle The V-cycle only involves computing one recursive coarse-grid correction. The diagram of this procedure (see Figure 1) resembles a V, which explains the origin of the name. To obtain this cycle, the cycle parameter γ is set to 1. The V-cycle is more suited to compute multiple eigenvalues, as demonstrated in Section 5.5.2.

W-Cycle The W-cycle applies two coarse-grid corrections on each level. In Algorithm 5, setting the parameter γ to 2 results in a W-cycle. Due to the increased number of coarse-grid operations, the orthogonalization procedure (needed when computing $k > 1$ eigenvectors) must in some cases be applied on coarser grids to ensure continued separation of the eigenvector approximations. The W-cycle has a better rate of convergence than the V-cycle, which is expected since more work is done per iteration. In Section 5.6.1, this is observed numerically.

Nested Iteration Instead of starting with a random initial guess on the finest grid, a better initial guess can be computed by starting with a random guess on the *coarsest* grid and working one's way up to the finest grid by interpolating and smoothing. This is called the *nested iteration* scheme and is formulated in Algorithm 6. An implementation of this method can be found in Section 4.6.4 and an example of its efficacy in Section 5.7. [Saad 2003, p. 424]

Algorithm 6 NESTED$(\mathbf{A}, \mathbf{M})$ *Nested Iteration*

Input: System matrices on all levels $\mathbf{A}_l, \mathbf{M}_l \in \mathbb{R}^{n_l \times n_l}$, Interpolation operators $\mathbf{I}_i^j$
Output: Good initial guess on fine grid.

$\mathbf{x} = \texttt{RAND}(n_1)$
$\mathbf{q}_1 = \mathbf{A}_1\mathbf{x}; \quad \mathbf{r}_1 = \mathbf{M}_1\mathbf{x}$
$\mathbf{c} = \texttt{SMOOTH\_STEP}(\mathbf{A}_1, \mathbf{M}_1, 1, \alpha, \beta, \mathbf{q}_1, \mathbf{r}_1)$
$\mathbf{x} = \mathbf{x} + \mathbf{c}$
for $l = 2, \ldots, m-1$ **do**
  $\mathbf{x} = \mathbf{I}_{l-1}^{l}\mathbf{x}$
  $\mathbf{q}_l = \mathbf{A}_l\mathbf{x}; \quad \mathbf{r}_l = \mathbf{M}_l\mathbf{x}$
  $\mathbf{c} = \texttt{SMOOTH\_STEP}(\mathbf{A}_l, \mathbf{M}_l, l, \alpha, \beta, \mathbf{q}_l, \mathbf{r}_l)$
  $\mathbf{x} = \mathbf{x} + \mathbf{c}$
end for
$\mathbf{x} = \texttt{SMOOTH\_FINE}(\mathbf{A}_m, \mathbf{M}_m, \mathbf{x}, 1)$
return $\mathbf{x}$

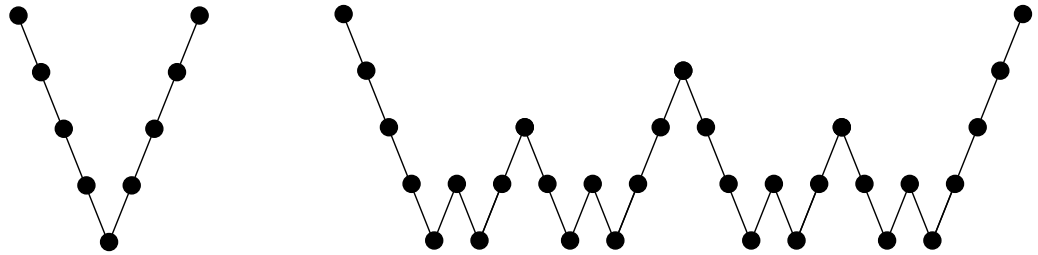


Figure 1: Diagram of a V-cycle (*left*) and a W-cycle (*right*). The finest level is located at the top, connected to coarser levels with a line symbolizing intergrid transfer operations.

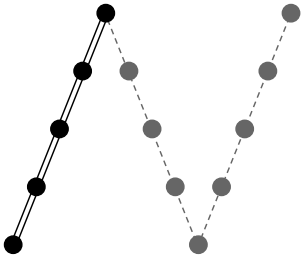


Figure 2: Diagram of a nested iteration. After the finest (top) level is reached, any multigrid cycle can be applied – a V-cycle is shown as an example. An important distinction is the use of the intergrid transfer operators as interpolations of a coarse-grid coefficient vector instead of a coarse-grid correction. This operation is symbolized by the double lines.

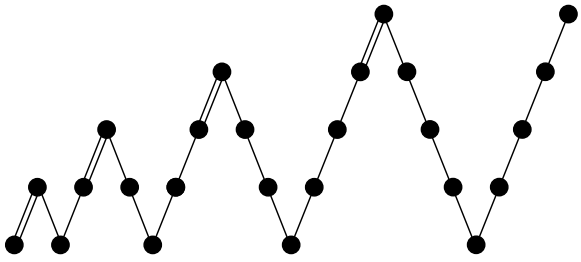


Figure 3: Full multigrid cycle (FMG). Once a level is reached for the first time, a V-cycle is executed. The double lines represent interpolation of a coarse grid coefficient vector to a finer grid, single lines represent interpolation of corrections.

Full Multigrid The full multigrid cycle combines the idea of deducing a good initial guess from the coarse-grid approximations with a multigrid V-cycle. Instead of computing an initial guess on the finest grid by just smoothing as in the nested iteration scheme, V-cycles are performed instead. This is done until the finest level is reached. Typically, only one FMG cycle is needed to obtain a good approximation to the solution. This is verified in Section 5.7. [Saad 2003, p. 424 – 430]

4.5 Data Structures

The data structures used for this implementation are based on the `MG_DATA` struct used in the LehrFEM implementation of geometric multigrid. In order to avoid confusion, the MATLAB structure used in this project will be referred to as `RQMG_DATA`.

The following description of the `RQMG_DATA` structure is an extension of the original LehrFEM documentation found in the files `mg_stima.m`, `mg_error.m`, `mg_mesh.m` and `mg_smooth.m`. The data structure `RQMG_DATA` is a `1×LVL` cell array of structures where `LVL` is the number of levels. Each structure contains the fields listed in Table 1, which will be explained in detail in the following sections.

The initialization of `RQMG_DATA` is done in five steps by the following functions:

1.	`rqmg_mesh.m`	Mesh refinement
2.	`rqmg_stima.m`	Calculate stiffness matrix for each level
3.	`rqmg_mass.m`	Calculate mass matrix for each level
4.	`rqmg_smooth.m`	Set smoother type and properties (ν_1, ν_2, γ)
5.	`rqmg_error.m`	Specify error functionals

4.5.1 Mesh Refinement

The function `rqmg_mesh` generates a sequence of nested meshes for the multigrid iteration. Along with the mandatory argument `RQMG_DATA`, it accepts the parameters listed in Table 2. The most important are `mesh`, the LehrFEM mesh data structure, `ref`, a 1×2 vector specifying the number of refinements for the coarsest and finest grids, and `k`, the number of eigenvectors to compute.

If, for example, a circular domain is to be discretized, one must supply `rqmg_mesh` with a signed distance function in order to be able to project new boundary points onto the actual boundary of the domain. To this end, `rqmg_mesh` accepts the parameter `dist_func`, which specifies a signed distance function, and `dist_args`, which is a cell array of arguments to `dist_func`.

Table 1: Fields of `RQMG_DATA` Structure

`mesh`	LehrFEM mesh data structure
`dofs`	Logical array specifying the non-Dirichlet-boundary degrees of freedom
`n.all`	Total number of degrees of freedom
`n.free`	Number of non-boundary degrees of freedom
`P`	Prolongation matrix from level `lvl-1` to level `lvl`, for functions that vanish on the Dirichlet boundary. Only defined for `lvl>1`
`P_full`	Prolongation matrix for all degrees of freedom. Only defined for `lvl>1`
`P_all`	Cumulative prolongation matrix from level `lvl-1` to the finest level. Only defined for `lvl>1`
`A`	Stiffness matrix
`M`	Mass matrix
`cyc`	Number of coarse-grid corrections γ
`pre`	Number of presmoothing steps ν_1 (see Section 4.5.3)
`post`	Number of postsmoothing steps ν_2 (see Section 4.5.3)
`error`	Structure containing error functional information
`error_ctrl`	Name of error functional to use to control the behavior of the RQMG iteration
`error_rel`	Flag for relative errors. True means relative errors will be used

Table 2: Parameters of `rqmg_mesh`

`mesh`	LehrFEM mesh data structure [default: unit square divided into two triangles]
`ref`	1×2 vector specifying the number of refinements for the coarsest and finest grids, respectively
`full`	Boolean specifying whether or not to add full prolongation matrix. If false, only P(dofs) is stored [default: false]
`all`	Boolean specifying whether or not to add cumulative prolongation matrices ($\mathbf{I}_\ell^m$) [default: false]
`k`	Number of eigenvectors to compute [default: 1]
`dist_func`	Signed distance function to use for refinements
`dist_args`	Arguments to signed distance function (see `dist_func`)

4.5.2 Mass and Stiffness Matrix Computation

The assembly of mass and stiffness matrices is done in the functions `rqmg_mass` and `rqmg_stima`. They accept the structure `RQMG_DATA` and various other (optional) parameters which are explained in detail in the Matlab documentation.

4.5.3 Smoothing Parameters

The function `rqmg_smooth` initializes parameters related to the smoother. `rqmg_smooth` accepts the parameter `m`, which is either a 1×2 vector containing the number of pre and postsmoothing steps or an integer if they are the same, and the parameter `cyc`, which corresponds to the cycle parameter γ.

4.5.4 Error Functionals

The structure `error` contains information on the error functionals that can be computed. The convention, which was adopted from the existing geometric multigrid implementation, is that the structure `error` contains fields named after the error functionals. The values of these fields are function handles which take the arguments `x`, `x0`, `A` and `M` in that order. Absolute (exact) errors are computed by functionals with field names ending in `_exact` while those of relative error functionals end in `_iter`.

Among the error functionals already implemented were standard error norms used in finite element computations. For the application of iteratively computing eigenvectors, however, there are much more well-suited error functionals. These include the deviation of the Rayleigh quotients from the exact eigenvalues as well as the angle between the current approximation of the invariant subspace, i.e. $\mathrm{span}(\mathbf{x}_i)$, and the exact subspace.

Rayleigh quotient Since the RQMG algorithm minimizes the Rayleigh quotient, it is most natural to monitor its convergence to the exact eigenvalue. This "exact" version of the error functional can be written as

$$\varepsilon_{i,exact}^{(k)} = \left| RQ(\mathbf{x}_i^{(k)}) - \lambda_i \right|,$$

where k is the iteration index and i the index of the eigenvector. A drawback of this error functional is the fact that it requires the computation of the desired eigenvalues on the finest mesh. Therefore, a relative version makes sense, measuring the change in the Rayleigh quotient due to one iteration:

$$\varepsilon_{i,iter}^{(k)} = \left| RQ(\mathbf{x}_i^{(k)}) - RQ(\mathbf{x}_i^{(k-1)}) \right|.$$

Since this functional yields one scalar value per desired eigenvector, the maximum over all approximations must be taken to obtain a scalar value.

Angle between subspaces A second important convergence indicator is the angle between the current subspace and the exact invariant subspace [Saad 2010, p.99]. This angle can be easily computed using an SVD-based algorithm implemented in the built-in Matlab function `subspace(U,V)`, which accepts matrices $\mathbf{U}$ and $\mathbf{V}$ containing a basis of the respective subspace in their columns and returns the angle between them in radians. Both exact and iterative versions of this error functional were implemented. For details, see the implementation in `rqmg_error.m`.

4.6 Implementation Details

The implementation of the Rayleigh quotient multigrid algorithm was based on an existing geometric multigrid implementation in the LehrFEM framework. Due to the very different nature of the problem, many changes were necessary to achieve a well-functioning code. First of all, the program should be able to compute multiple eigenvalues. The unknown vectors are therefore stored in the columns of an $n_m \times k$ matrix, where n_m is the number of degrees of freedom on the finest mesh and k is the number of eigenvectors to be computed.

4.6.1 Dynamically Updated Quantities

In Section 3.5.1 and Section 4.2, various dynamic quantities and relations between them were derived. In order to keep track of these in a sensible way, even for multiple iterates and arbitrary cycles, a data structure called `vars` was created. This allows easy management of all dynamic quantities, without the need to pass a large number of arguments to each function.

The `vars` data structure consists of an $m \times k$ cell array, each element of which is a struct containing the fields listed in Table 3.

Table 3: Fields of each element of `vars`

`xAP`	Corresponds to $\mathbf{q}_l = \mathbf{x}^T\mathbf{A}\mathbf{I}_l^m$
`xMP`	Corresponds to $\mathbf{r}_l = \mathbf{x}^T\mathbf{M}\mathbf{I}_l^m$
`xAx`	Corresponds to $\alpha = \mathbf{x}^T\mathbf{A}\mathbf{x}$ (only stored on finest level)
`xMx`	Corresponds to $\beta = \mathbf{x}^T\mathbf{M}\mathbf{x}$ (only stored on finest level)

4.6.2 Coordinate Relaxation

The smoother used in this implementation employs coordinate relaxation as explained in Section 3.5 and detailed in Algorithm 3. Of course, the optimizations detailed in Section 3.5.1 were applied.

The following listing shows the smoother routine for the finest level. In this case, the optimization for $\mathbf{x}^T\mathbf{A}\mathbf{p}_i$ from Section 3.5.1 does not improve the speed, as an additional vector of size n would be needed to store the values for $\delta_i^\star$. It is preferred to update $\mathbf{x}$ in situ and compute $\mathbf{x}^T\mathbf{A}\mathbf{p}_i$ as a scalar product of $\mathbf{x}$ with the i-th column of $\mathbf{A}$, which requires $\mathcal{O}(const)$ operations due to the sparsity of $\mathbf{A}$.

The full implementation for an arbitrary level can be found in `rqmg_step.m`

Listing 1: Optimized Smoother for Finest Level

```
function [x,vars] = smooth_fine(rqmg_data,x,vars)
%SMOOTH_FINE Coordinate relaxation optimized for finest level

% get dynamic variables (potentially faster for some Matlab versions)
A = rqmg_data{end}.A; M = rqmg_data{end}.M;
Adiag = rqmg_data{end}.Adiag; xAx = vars{end}.xAx;
Mdiag = rqmg_data{end}.Mdiag; xMx = vars{end}.xMx;

n = rqmg_data{end}.n.free;
for p = 1:n
  pAp = Adiag(p);  pMp = Mdiag(p);
  xAp = x'*A(:,p); xMp = x'*M(:,p);

  t1 = pMp*xAp - pAp*xMp;
  t2 = xMx*pAp - xAx*pMp;
  t3 = xAx*xMp - xMx*xAp;
  delta = 2*t3 / (t2 + sqrt(t2*t2-4*t1*t3));

  % update x
  x(p) = x(p) + delta;
  % update dynamic variables
  xAx = xAx + 2*delta*xAp + delta*delta*pAp;
  xMx = xMx + 2*delta*xMp + delta*delta*pMp;
end

% store dynamic variables for next relaxation step
vars{end}.xAx = xAx;
vars{end}.xMx = xMx;

end
```

4.6.3 Orthogonalization and Ritz Projection

In order to speed up convergence, one can use a Ritz projection to search for better approximations in the space spanned by the current iterates. This is done after orthogonalizing the iterates, which is accomplished using an economical QR decomposition. Listing 2 summarizes the main steps of this procedure in Matlab code.

Listing 2: Orthogonalization and Ritz Projection

```
function x = ortho( x,A,M )
  [x,dummy] = qr(x,0);
  AA = x'*A*x;  MM = x'*M*x;
  [U,D] = eig(full(AA),full(MM)); % solve eigenvalue problem
  [ev,ind] = sort(abs(diag(D)));
  x = x*U(:,ind); % recover eigenvectors
end
```

Listing 3: Implementation of Nested Iteration Scheme

```
function [ x ] = rqmg_nested( rqmg_data,tol )
  %RQMG_NESTED Nested iteration for RQMG
  LVL = size(rqmg_data,2);
  x = rand(rqmg_data{1}.n.free, 1);
  vars = cell(LVL,1);
  vars{1}.xAP = x'*rqmg_data{1}.A; % q_1
  vars{1}.xMP = x'*rqmg_data{1}.M; % r_1
  vars{end}.xAx = x'*rqmg_data{1}.A*x; % alpha
  vars{end}.xMx = x'*rqmg_data{1}.M*x; % beta
  [c,vars] = smooth_step(rqmg_data,1,vars);
  x = x + c; % correction on coarse grid
  x = rqmg_data{2}.P*x; % interpolation!
  for l=2:LVL-1
    vars{l}.xAP = x'*rqmg_data{l}.A; % q_l
    vars{l}.xMP = x'*rqmg_data{l}.M; % r_l
    [c,vars] = smooth_step(rqmg_data,l,vars);
    x = x + c;
    x = rqmg_data{l+1}.P*x; % interpolation!
  end
  x = rqmg(x,rqmg_data,tol,1); % one complete V-cycle
end
```

4.6.4 Nested Iteration and FMG

Due to the modularity of multigrid implementations, it is very easy to combine various cycles. This allows for implementations of many schemes (e.g. the full multigrid cycle or the nested iteration scheme) based only on the procedures `rqmg_step`, `rqmg` and the smoother `smooth_step`. Listing 3 shows an example of the nested iteration scheme. The implementation for the full multigrid iteration is exactly the same, the only difference being a call to `rqmg_step` instead of `smooth_step` on line 16.

4.7 Complexity

4.7.1 Work on One Level

Before determining the complexity of the entire RQMG algorithm, we need an idea of how much work is executed by each component. As discussed in Section 3.5.1, the smoothing iteration requires $\mathcal{O}(n)$ operations per sweep over all coordinates on a level with n degrees of freedom.

On each level, the smoother is called ν_1 times before and ν_2 times after the coarse-grid correction, resulting in about $(\nu_1+\nu_2)5n_\ell$ operations. Updating the variables $\mathbf{q}_\ell$ and $\mathbf{r}_\ell$ requires approximately $4n_\ell$ operations (two updates before and two after the coarse-grid correction). Assuming that the application of the correction requires n_ℓ operations, this gives a total of approximately $5(1+\nu_1+\nu_2)n_\ell$ operations, resulting in the expression

$$\mathbf{W}_\ell = Cn_\ell \tag{18}$$

for the work on level ℓ, where C is independent of n_ℓ.

4.7.2 Total RQMG Complexity

In order to determine the computational cost of the RQMG algorithm, we are interested in determining the number of operations required to reduce the maximal error in the Rayleigh quotient by a given factor τ:

$$\varepsilon^{(k)} \leq \tau\varepsilon^{(0)} \quad , \quad \varepsilon^{(k)} := \left| RQ(x^{(k)}) - \lambda_i \right|,$$

where $\varepsilon^{(k)}$ is the error of the eigenvector after the k–th iteration. We are currently only interested in analyzing the complexity of calculating one eigenvector.

By assuming that the sequence of meshes is obtained by regular refinement, the number of vertices increases by a factor of approximately 4 per refinement step in

two dimensions, where small deviations from this factor are caused by negligible boundary effects. In terms of the number of degrees of freedom on the finest mesh, n_m, the corresponding quantity on level ℓ is given by

$$n_\ell = \frac{n_m}{4^{m-\ell}}, \quad 1 \leq \ell \leq m. \tag{19}$$

In order to estimate the amount of work done in total, $\mathbf{W}^{tot}$, it is helpful to split up the total work on a level ℓ, denoted by $\mathbf{W}_\ell^{tot}$, into the part that is done only on level ℓ and a recursive part, corresponding to the total work done on level $\ell - 1$. This results in the recursion

$$\mathbf{W}_\ell^{tot} = \mathbf{W}_\ell + \gamma \mathbf{W}_{\ell-1}^{tot} \qquad\qquad \mathbf{W}_2^{tot} = \mathbf{W}_2 + \mathbf{W}_1,$$

where $\mathbf{W}_1$ is the work done on the coarsest level and γ is the cycle parameter which describes the number of coarse-grid corrections. Expansion of this recursion for the term $\mathbf{W}_m^{tot}$ yields

$$\mathbf{W}_m^{tot} = \mathbf{W}_m + \gamma(\mathbf{W}_{m-1} + \gamma(\mathbf{W}_{m-2} + \gamma(\ldots))) = \sum_{\ell=2}^{m} \gamma^{m-\ell} \mathbf{W}_\ell + \gamma^{m-2} \mathbf{W}_1.$$

Using (18) for the terms $\mathbf{W}_\ell$, and (19) as an expression for n_ℓ, we obtain

$$\mathbf{W}_m^{tot} \overset{(18)}{=} \sum_{\ell=2}^{m} \gamma^{m-\ell} C n_\ell + \gamma^{m-2} C n_1 \overset{(19)}{=} C n_m \sum_{\ell=2}^{m} \left(\frac{\gamma}{4}\right)^{m-\ell} + \left(\frac{\gamma}{4}\right)^{m-2} 4 C n_m,$$

which, for $\gamma < 4$, results in

$$\begin{aligned} \mathbf{W}_m^{tot} &= C n_m \left(\frac{1 - (\frac{\gamma}{4})^{m-1}}{1 - \frac{\gamma}{4}} \right) + \left(\frac{\gamma}{4}\right)^{m-2} 4 C n_m \\ &\leq C n_m \left(\frac{1}{1 - \frac{\gamma}{4}} \right), \end{aligned}$$

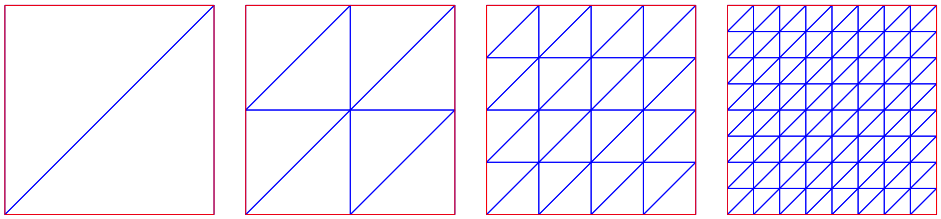
showing that one cycle of the RQMG algorithm requires $\mathcal{O}(n_m)$ operations for $\gamma \leq 3$.

Assuming that the convergence of the RQMG algorithm is independent of h, we only require a constant number of such RQMG cycles (h-independence of the RQMG algorithm is verified empirically in Section 5.4). This assumption, along with the above argumentation, results in the conclusion that the RQMG algorithm requires only $\mathcal{O}(n)$ operations to reduce the error by a constant factor. [Trottenberg *et al.* 2001, p. 50]

5 Numerical Experiments

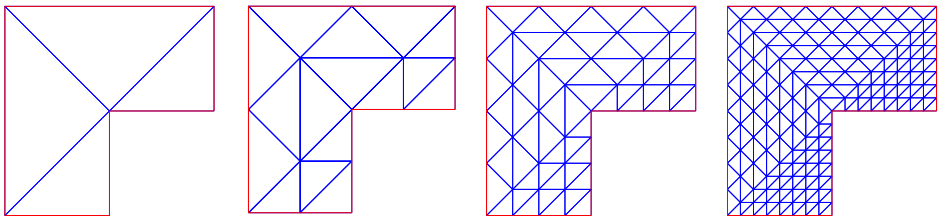
5.1 Domains

In the following, various experimental results are presented. The model problem (1) was solved on three domains: a square domain $\Omega_\square := [-1,1]^2$, an L-shaped domain $\Omega_L := [-1,1]^2 \setminus ([0,1] \times [-1,0])$ and a circular domain $\Omega_\circ := \{\mathbf{x} \in \mathbb{R}^2 : |\mathbf{x}| \leq r\}$.



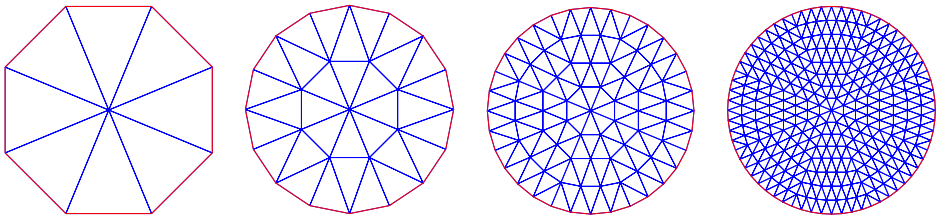
(a) $n_1 = 0$ (b) $n_2 = 1$ (c) $n_3 = 9$ (d) $n_4 = 49$

Figure 4: Meshes $\mathcal{M}_i^\square$ of a square domain $\Omega_\square$ with various levels of refinement.



(a) $n_1 = 0$ (b) $n_2 = 3$ (c) $n_3 = 21$ (d) $n_4 = 105$

Figure 5: Meshes $\mathcal{M}_i^L$ of an L-shaped domain Ω_L with various levels of refinement.

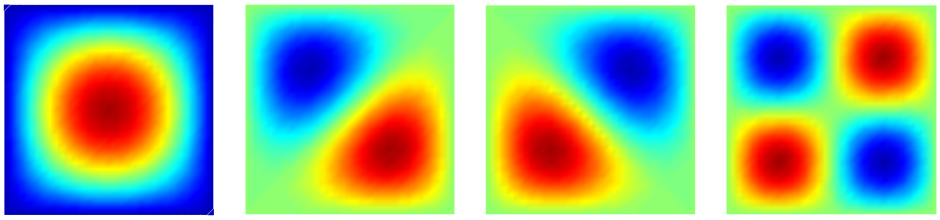


(a) $n_1 = 1$ (b) $n_2 = 9$ (c) $n_3 = 49$ (d) $n_4 = 225$

Figure 6: Meshes $\mathcal{M}_i^\circ$ of a circular domain $\Omega_\circ$ with various levels of refinement.

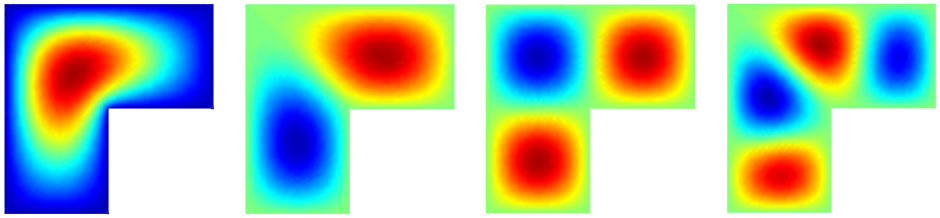
5.2 Exact Solutions

The first four eigenfunctions for each of the domains mentioned above are shown in Figures 7 to 9 with their corresponding eigenvalues.



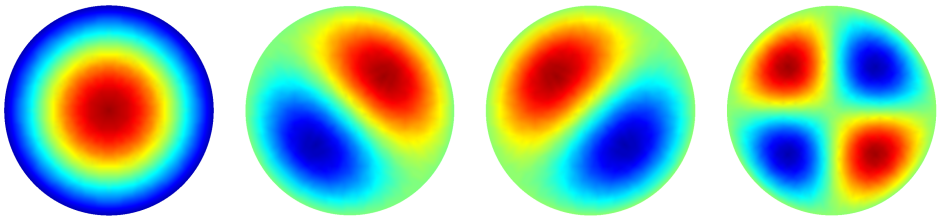
(a) $\lambda_1 \approx 4.9467$ (b) $\lambda_2 \approx 12.3881$ (c) $\lambda_2 \approx 12.4168$ (d) $\lambda_3 \approx 19.9290$

Figure 7: Solutions to the four smallest eigenvalues on $\Omega_\square$. The values for λ_2 are not identical due to an asymmetry in the discretization, see Figure 4.



(a) $\lambda_1 \approx 9.6825$ (b) $\lambda_2 \approx 15.2268$ (c) $\lambda_3 \approx 19.7974$ (d) $\lambda_4 \approx 29.6671$

Figure 8: Solutions to the four smallest eigenvalues on Ω_L.



(a) $\lambda_1 \approx 4.9411$ (b) $\lambda_2 \approx 12.5837$ (c) $\lambda_2 \approx 12.5837$ (d) $\lambda_3 \approx 22.6766$

Figure 9: Solutions to the four smallest eigenvalues on $\Omega_\circ$.

5.3 Asymptotic Complexity

As analyzed in Section 4.7, the time complexity of the RQMG algorithm should scale like $\mathcal{O}(n)$. Figures 10 and 11 show a numerical determination of the complexity with respect to the number of degrees of freedom n and the meshwidth h. This was done by solving the given model problem for the eigenvector to the smallest eigenvalue using successively finer discretizations. In two dimensions, $n \propto h^{-2}$, which is consistent with the observed complexity in Figure 11.

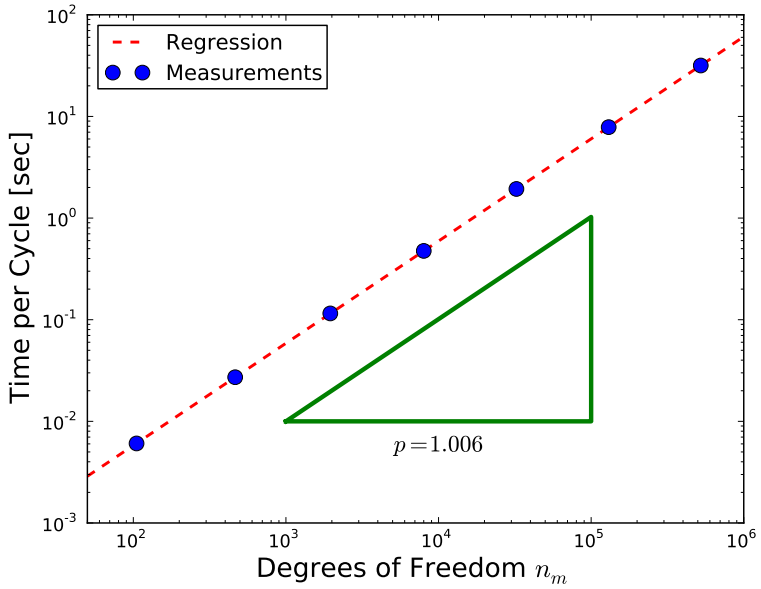


Figure 10: Measurement of the asymptotic complexity of the RQMG algorithm with respect to the number of degrees of freedom on the finest mesh, n_m.

5.4 h-Independence of Convergence

An important property of multigrid algorithms is the independence of convergence on the finest meshwidth h. In Section 4.7 this property was used to derive the $\mathcal{O}(n)$ total complexity of the RQMG algorithm. This property is numerically verified in figures 12 to 14 in order to justify the assumption previously made.

As a contrast, Figure 15 shows the strong dependence of normal Rayleigh quotient minimization (without coarse-grid corrections) on the meshwidth.

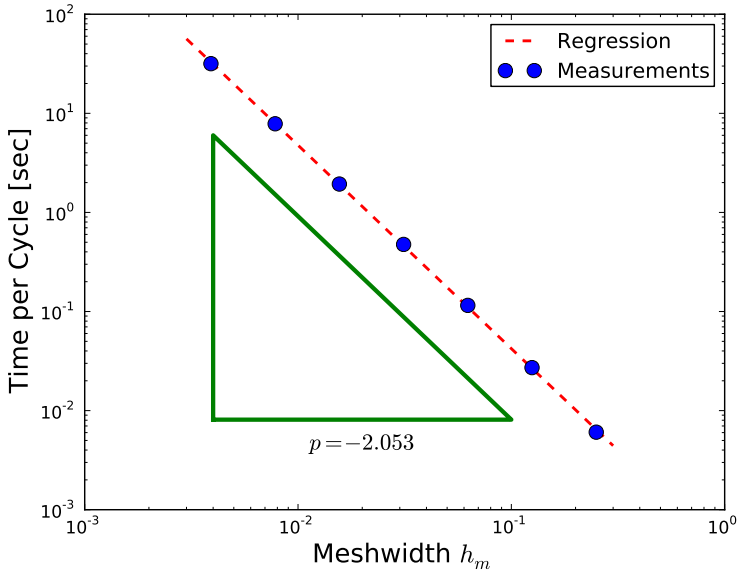


Figure 11: Measurement of the asymptotic complexity of the RQMG algorithm with respect to the meshwidth of the finest mesh, h_m.

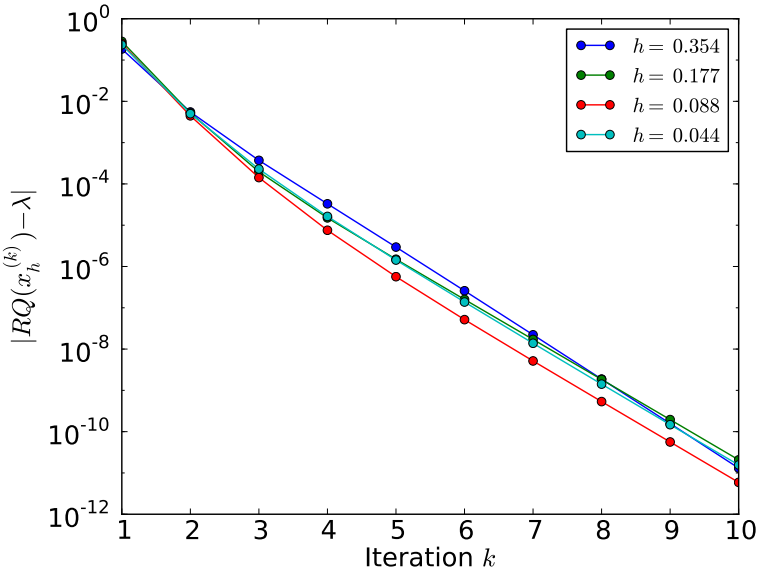


Figure 12: RQMG convergence history using discretizations of $\Omega_\square$ with varying fine meshwidth. The coarsest meshwidth was the same for all trials.

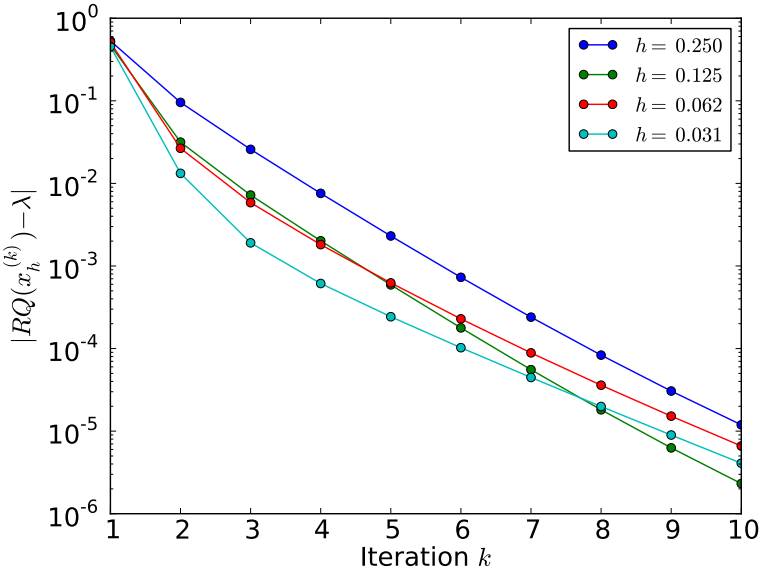


Figure 13: RQMG convergence history using discretizations of Ω_L with varying fine meshwidth. The coarsest meshwidth was the same for all trials.

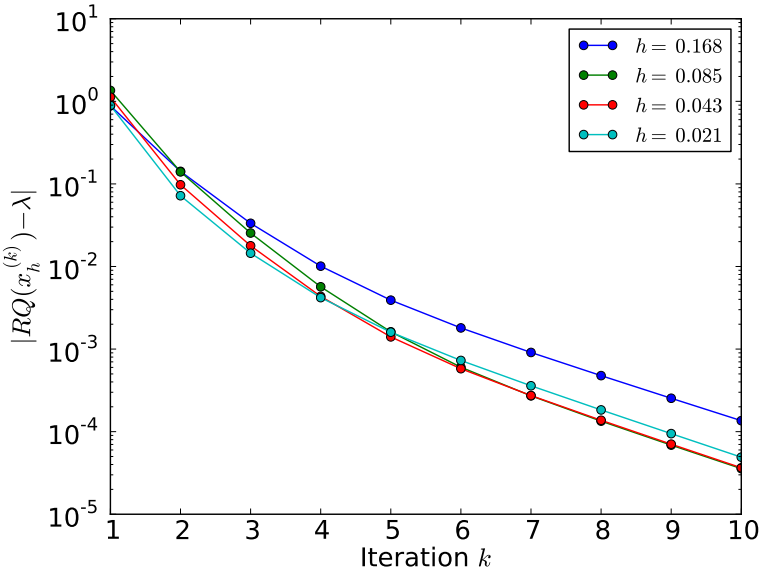


Figure 14: RQMG convergence history using discretizations of $\Omega_\circ$ with varying fine meshwidth. The coarsest meshwidth was the same for all trials.

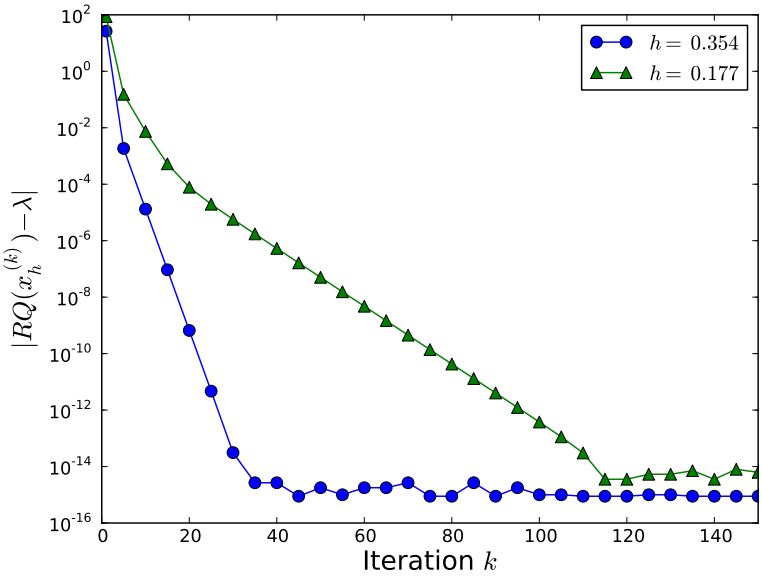


Figure 15: Convergence history using Rayleigh quotient minimization *without* coarse-grid corrections. Convergence to machine precision is observed after about 40 iterations for $h = 0.354$ while taking 120 iterations for $h = 0.177$. One eigenvector was computed for the domain $\Omega_\square$.

5.5 Multiple Eigenvectors

Compared to the case of computing only the eigenvector to the smallest eigenvalue, computing multiple eigenvectors introduces many potential pitfalls. A few of them are now illustrated.

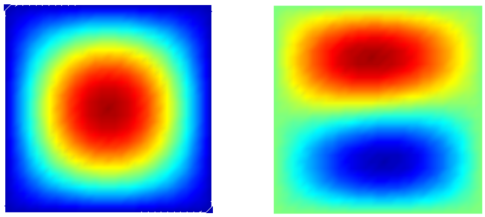
5.5.1 Size of Search Space

Given a domain that leads to the existence of one or more degenerate eigenvalues λ, one must ensure that the search space (i.e. the number of iterates) is large enough to contain the entire eigenspace $\mathbf{E}_\lambda$.

The square domain $\Omega_\square$ can be used to illustrate this point, since the second-smallest analytical eigenvalue is doubly degenerate. By choosing the search space smaller than required, one observes that the solution to the eigenvalue λ is a linear combination of the eigenvectors in $\mathbf{E}_\lambda$ (see Figure 16).

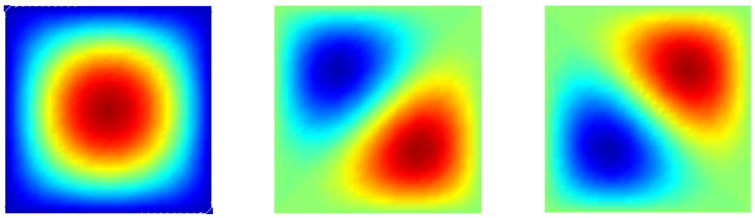
While the solution, as a linear combination of eigenvectors to the same eigenvalue, is of course an eigenvector, an orthonormal basis of the eigenspace is usually sought. This desire can only be fulfilled by choosing the search space large enough to contain the entire eigenspace $\mathbf{E}_\lambda$.

Additionally, in this case the discretization introduces a slight separation of the eigenvalues of the discrete system. Therefore, the eigenvalues are not exactly identical and one cannot settle for a linear combination of the two eigenvectors.



(a) $RQ(\mathbf{x}_1) \approx 4.947$ (b) $RQ(\mathbf{x}_2) \approx 12.400$

Figure 16: Solutions to the two smallest eigenvalues on $\Omega_\square$. Due to the analytical degeneracy of λ_2, solution (b) converges to a linear combination of the two eigenvectors in $\mathbf{E}_{\lambda_2}$ (cf. Figures 7(b) and 7(c)).



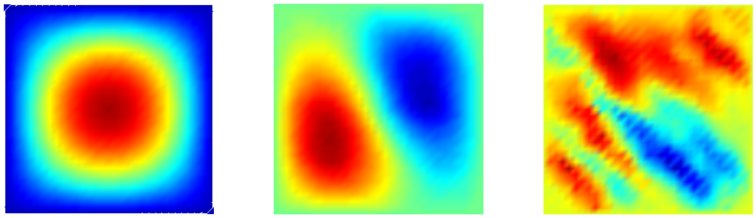
(a) $RQ(\mathbf{x}_1) \approx 4.947$ (b) $RQ(\mathbf{x}_2) \approx 12.388$ (c) $RQ(\mathbf{x}_3) \approx 12.417$

Figure 17: Solutions to the three smallest eigenvalues on $\Omega_\square$, obtained using a V-cycle. The approximations (b) and (c) converge to the correct discrete eigenvectors if the search space is large enough (cf. Figures 7, 16).

5.5.2 Deficiencies of W-Cycle

The W-cycle suffers from a deficiency resulting from the fact that in an optimal implementation one can only orthogonalize on the finest level. A result of this is that when using a W-cycle, the iterates tend to loose orthogonality if the number of levels is sufficiently large, as the finest grid is not visited often enough. This can be seen in Figure 18.

Figure 19 shows the convergence history of a W-cycle in which orthogonalization is done on all levels except the coarsest. This introduces sufficient orthogonality between the iterates, but relies on a suboptimal implementation of complexity $\mathcal{O}(n \log n)$. As can be seen in Figure 17, the V-cycle does not suffer from this problem. It is therefore much more suited for computing multiple solutions to the eigenvalue problem in $\mathcal{O}(n)$ time than the W-cycle.



(a) $RQ(\mathbf{x}_1) \approx 4.947$ (b) $RQ(\mathbf{x}_2) \approx 12.904$ (c) $RQ(\mathbf{x}_3) \approx 71.766$

Figure 18: Solutions to the three smallest eigenvalues on $\Omega_\square$, obtained using a W-cycle. Both (b) and (c) are not correct, which can be blamed on insufficient orthogonality of the iterates during the W-cycle.

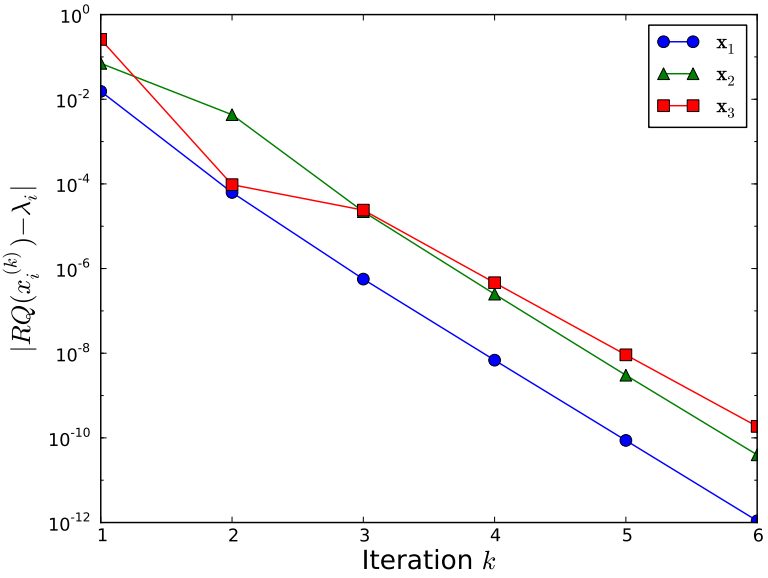


Figure 19: Convergence history of a W-cycle used to solve the eigenvalue problem on $\Omega_\square$. Convergence can be clearly seen, thanks to orthogonalization on the three finest levels (out of four levels in total).

5.6 Rate of Convergence

The rate of convergence of the RQMG algorithm was measured in various error functionals for a few different cycles and domains. Figures 20 to 23 show various combinations of cycles and domains and what convergence rates were measured in each situation.

In Section 5.6.2, multiple eigenvectors are computed and the rate of convergence determined for different search space sizes.

5.6.1 V- and W-Cycles

As one expects, the W-cycle performs much better, especially for the domains Ω_L and $\Omega_\circ$. The angle between the space spanned by the iterates $\mathbf{x}_j$ and the exact invariant subspace $\mathbf{U}_k$, denoted by $\angle(\mathbf{X}_i, \mathbf{U})$[1] in figures 20 and 22, converges more slowly than the modulus of the difference of the Rayleigh quotient and the correct eigenvalue. The two rates seem to be roughly related by the proportionality $r_{RQ} \propto (r_\angle)^2$.

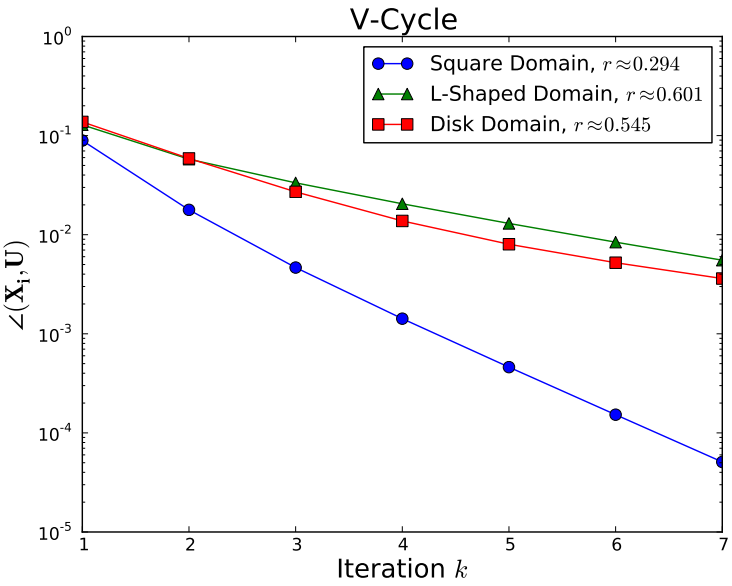


Figure 20: Convergence of the angle between the iterates and the exact eigenvectors using a V-cycle to solve the eigenvalue problem on various domains.

[1] The index i in the quantity $\angle(\mathbf{X}_i, \mathbf{U})$ denotes the domain type.

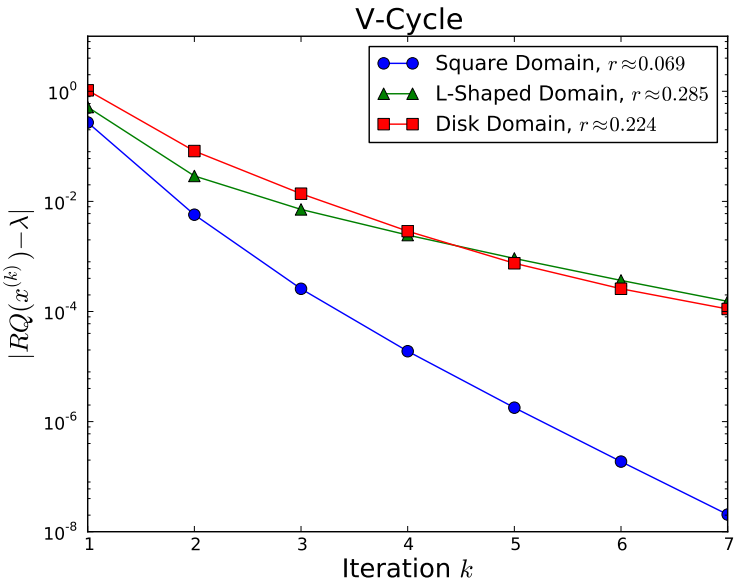


Figure 21: Convergence of the Rayleigh quotient using a V-cycle to solve the eigenvalue problem on various domains.

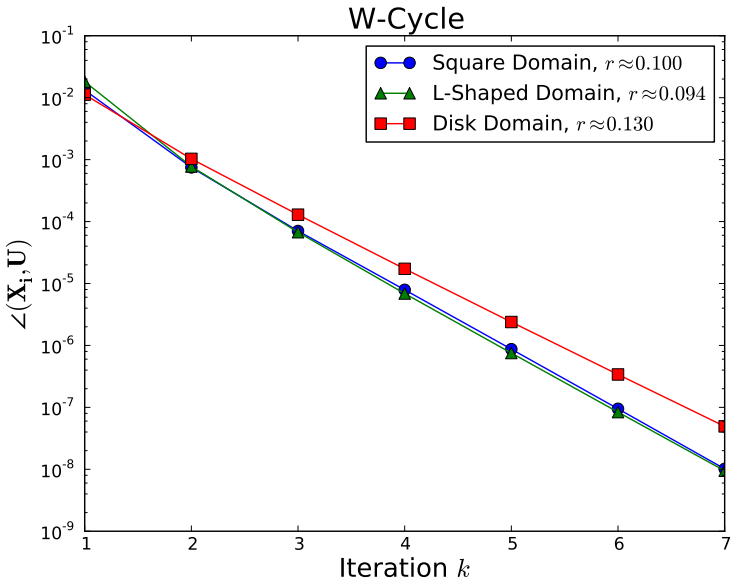


Figure 22: Convergence of the angle between the iterates and the exact eigenvectors using a W-cycle to solve the eigenvalue problem on various domains.

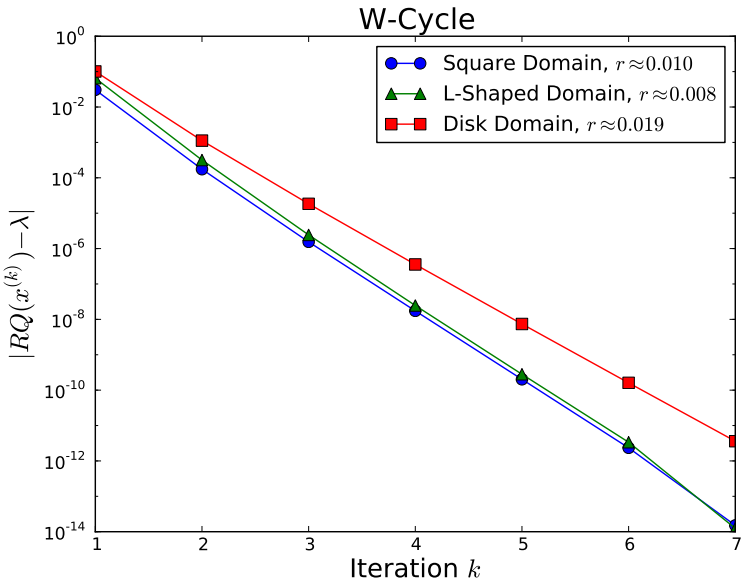


Figure 23: Convergence of the Rayleigh quotient using a W-cycle to solve the eigenvalue problem on various domains.

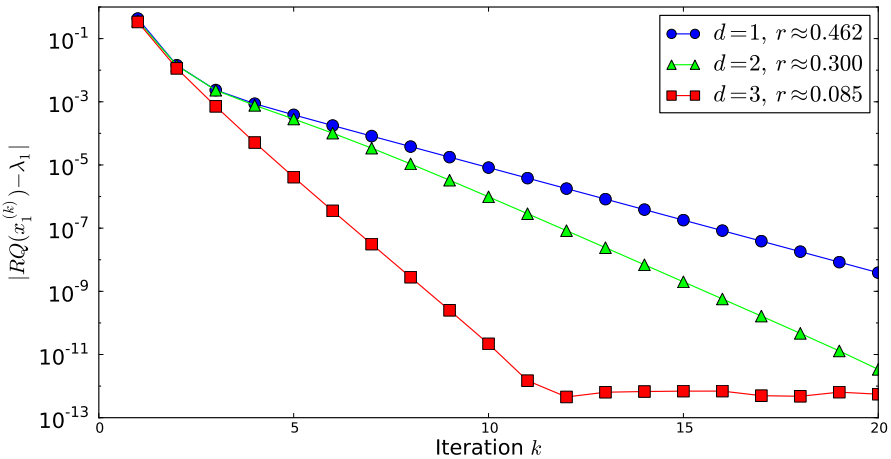


Figure 24: Convergence of the Rayleigh quotient of the eigenvector to the smallest eigenvalue using a V-cycle to solve the eigenvalue problem on Ω_L for various search space sizes. Here, d corresponds to the number of iterates.

5.6.2 Multiple Eigenvectors

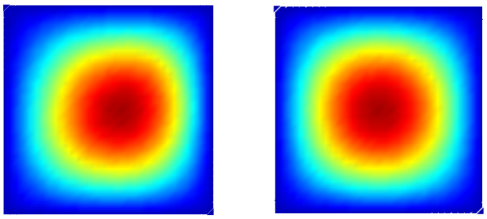
One possibility of increasing the rate of convergence of the iteration is to enlarge the search space. Due to the Ritz projection executed after the orthogonalization procedure, this can lead to a reduction in the number of iterations to achieve a given accuracy on an eigenvector to one of the lower eigenvalues.

In Figure 24, the convergence of the eigenvector to the smallest eigenvalue is shown for different sizes of the search space. The dimension of the search space, d, corresponds to the number of eigenvector approximations used by RQMG. By computing two eigenvectors more than required, the observed rate of convergence drops from 0.46 to 0.09, which corresponds to a factor of about 5.

5.7 Nested Iteration and FMG Cycle

Nested iteration provides a method of quickly determining a relatively good approximation to be used as an initial value. Figure 25(a) shows the first eigenvector after a trial run of the implementation in Section 4.6.4. Comparing with Figure 7, it can be seen that the eigenvector is already a very good approximation to the solution.

Figure 26 shows timings of nested iteration, FMG and standard RQMG with a random initial guess used to solve the model problem for one eigenvector up to a tolerance of 10^{-5}. It can be seen that nested iteration provides a slight improvement, however FMG is clearly the winner. Of course, all three algorithms scale with $\mathcal{O}(n)$.



(a) $RQ(\mathbf{x}) \approx 5.028$ (b) $RQ(\mathbf{x}) \approx 4.947$

Figure 25: (a) Approximation obtained after one application of nested iteration on the model problem. (b) Approximation obtained after one application of Full Multigrid on the model problem ($|RQ(x) - \lambda| \approx 5 \cdot 10^{-4}$).

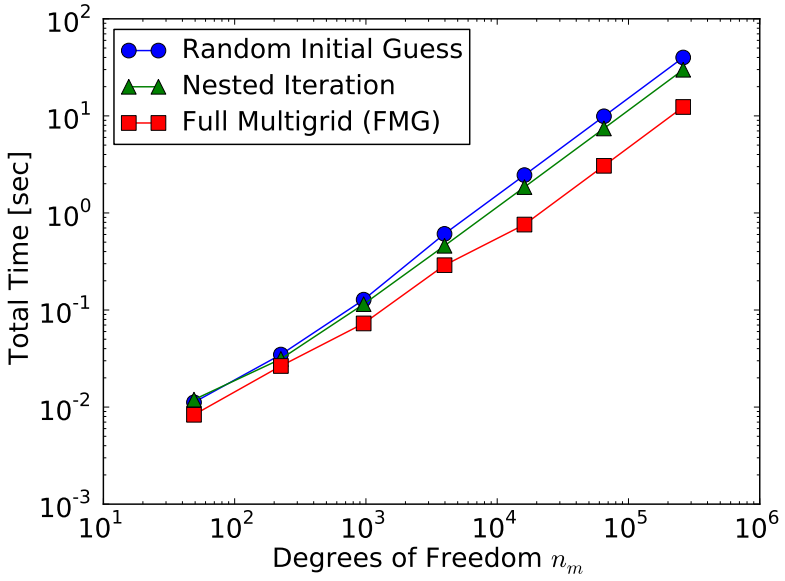


Figure 26: Timings of the nested iteration scheme, FMG (full multigrid), and the standard V-cycle with a random initial guess. One eigenvector was computed until the iterate $\mathbf{x}$ fulfilled $|RQ(\mathbf{x}) - \lambda| < 10^{-5}$.

6 Conclusion

In this thesis, the Rayleigh quotient multigrid algorithm was elucidated and an optimal implementation constructed. The $\mathcal{O}(n)$ asymptotic complexity was analytically derived and numerically verified, along with the h-independence of convergence, a characteristic property of multigrid methods.

Some pitfalls of computing multiple eigenvector approximations were discussed and numerically analyzed. This was aided by a suboptimal implementation of complexity $\mathcal{O}(n \log n)$, which allows more flexible manipulation of the iterates.

The execution time of finding a solution of the model problem to a certain precision was measured for the standard V-cycle and more advanced cycles like the nested iteration scheme and the full multigrid cycle, showing their superiority.

Possible improvements over the current implementation include the use of a more efficient smoother, for example by choosing the search direction $\mathbf{p}_k$ in (11) as the negative gradient of the Rayleigh quotient, resulting in a steepest descent iteration. Conjugate gradient approaches are also possible. [Friese 1998, p. 47]

References

[Arbenz & Kressner 2010] Peter Arbenz and Daniel Kressner. *Lecture Notes on Solving Large Scale Eigenvalue Problems*. 2010. (Cited on pages 2, 3 and 8.)

[Braess 2007] Dietrich Braess. *Finite Elemente*. 2007. (Cited on pages 2 and 4.)

[Burtscher *et al.* 2009] Annegret Burtscher, Patrick Meury and Eivind Fonn. *LehrFEM - A 2D Finite Element Toolbox*. 2009. (Cited on page 1.)

[Chan & Sharapov 1998] T. Chan and I. Sharapov. *Subspace Correction Multilevel Methods for Elliptic Eigenvalue Problems*. 1998. (Cited on pages 6 and 7.)

[Friese 1998] T. Friese. *Eine Mehrgitter-Methode Zur Lösung des Eigenwertproblems der Komplexen Helmholtzgleichung*. PhD thesis, 1998. (Cited on pages 2 and 40.)

[Golub & Van Loan 1996] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. October 1996. (Cited on page 5.)

[Hiptmair *et al.* 2010] R. Hiptmair, C. Schwab, H. Harbrecht, V. Gradinaru and A. Chernov. *Lecture Notes: Numerical Methods for Partial Differential Equations*. October 2010. (Cited on page 12.)

[Mandel & McCormick 1989] Jan Mandel and Stephen F. McCormick. *A Multilevel Variational Method for $Au = \lambda Bu$ on Composite Grids*. Journal of Computational Physics, vol. 80, no. 2, pages 442 – 452, 1989. (Cited on pages 9, 11 and 13.)

[Saad 2003] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. 2003. (Cited on pages 17 and 19.)

[Saad 2010] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. 2010. (Cited on page 22.)

[Trottenberg *et al.* 2001] U. Trottenberg, C. W. Oosterlee and A. Schuller. *Multigrid*. 2001. (Cited on page 26.)