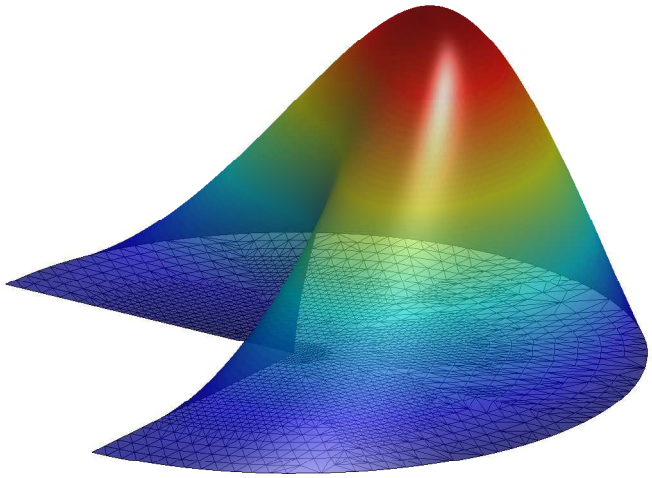
AMP Eigensolver:
Adaptive **M**ultigrid **P**reconditioned **E**igensolver

Users' Guide, Version 1.0

Ming Zhou, Klaus Neymeyr

Institut für Mathematik,
Universität Rostock,
Ulmenstraße 69, Haus 3, 18057 Rostock, Germany.

July 21, 2014.



Preface

The Adaptive Multigrid Preconditioned Eigensolver software (AMPEigensolver) can be used to compute a modest number of the smallest eigenvalues and the associated eigenfunctions of the negative Laplace operator in bounded two-dimensional domains. The AMPE software is written in FORTRAN and uses the BLAS and LAPACK libraries. The users' front-end of the software is a Graphical User Interface (GUI) of Matlab. The program is implemented in a storage efficient way and can solve problems up to 50 million degrees of freedom on a personal computer with 32GiB RAM (without disk swapping).
The eigensolver software uses the finite element method (FEM) and includes fast adaptive grid refinement strategies, which are based on residual estimators using linear and quadratic finite elements. The iterative solver rests on multigrid preconditioned gradient iterations for the minimization of the Rayleigh quotient. A report on numerical experiments with AMPE is contained in [19].

Contents

1 Getting started

The program package AMPEigensolver can be downloaded from

`http://www.math.uni-rostock.de/ampe/`

Then unzip the file `ampe.zip` and open a Matlab command window or desktop window in the directory `ampe`. The Matlab GUI of AMPE can be started by calling the function `ampe.m`. The Matlab-GUI contains two panels, namely the left panel "Initialization & Computation", see Figure 1, and the right panel "Results", see Figure 3.

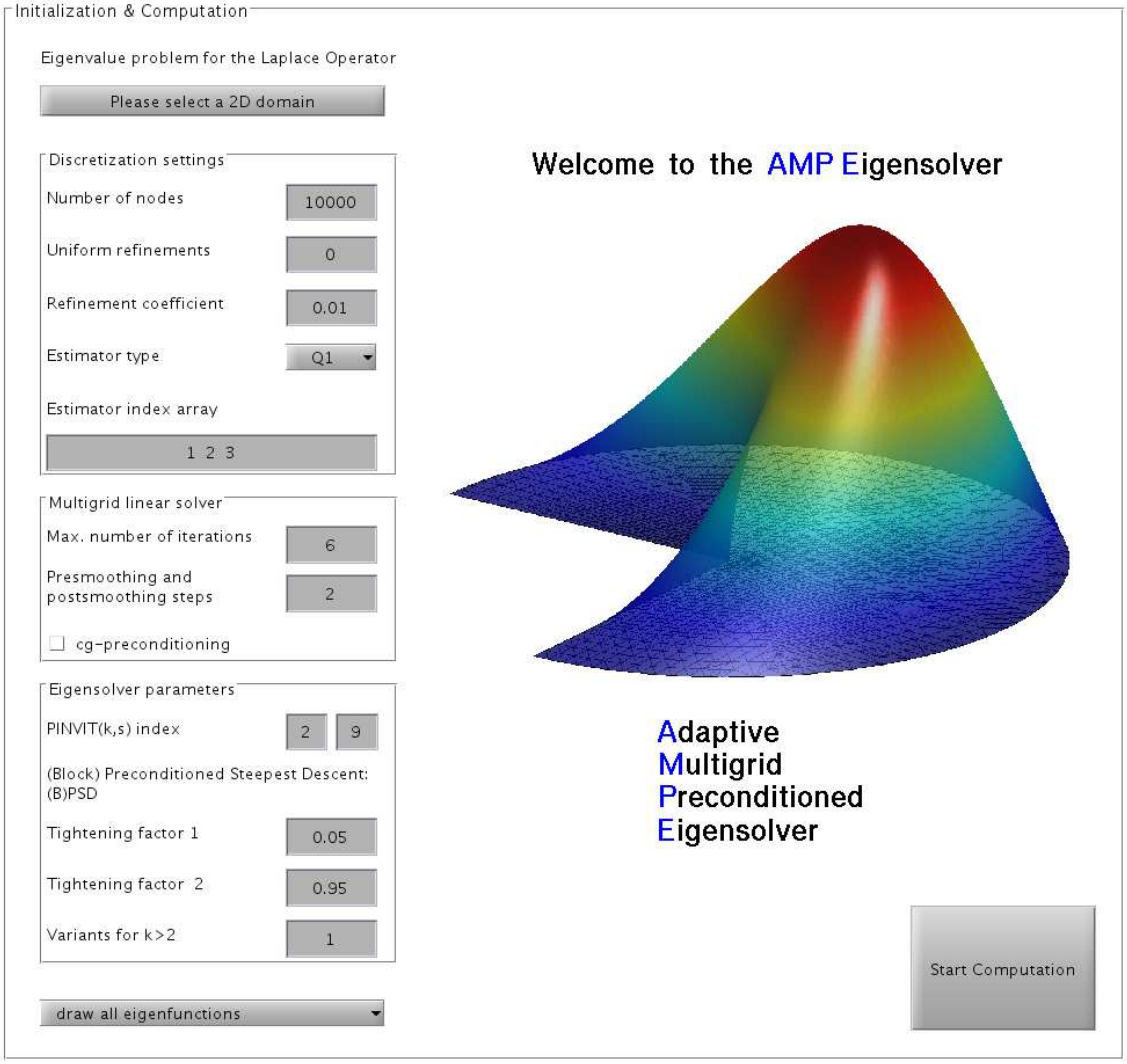


Figure 1: The left panel "Initialization & Computation" at the start of AMPE.

For a quick test of AMPE the user only has to select a domain in the domain selection menu (in the top of the left panel) and then click the button "Start computation" in the lower right corner of the left panel. With the default setting of the parameters the computation is completed almost instantaneously on a standard office PC. Then the left panel shows not only the initial triangulation, but also the computation times and the number of triangles, edges and nodes in the final triangulation, see Figure 2.

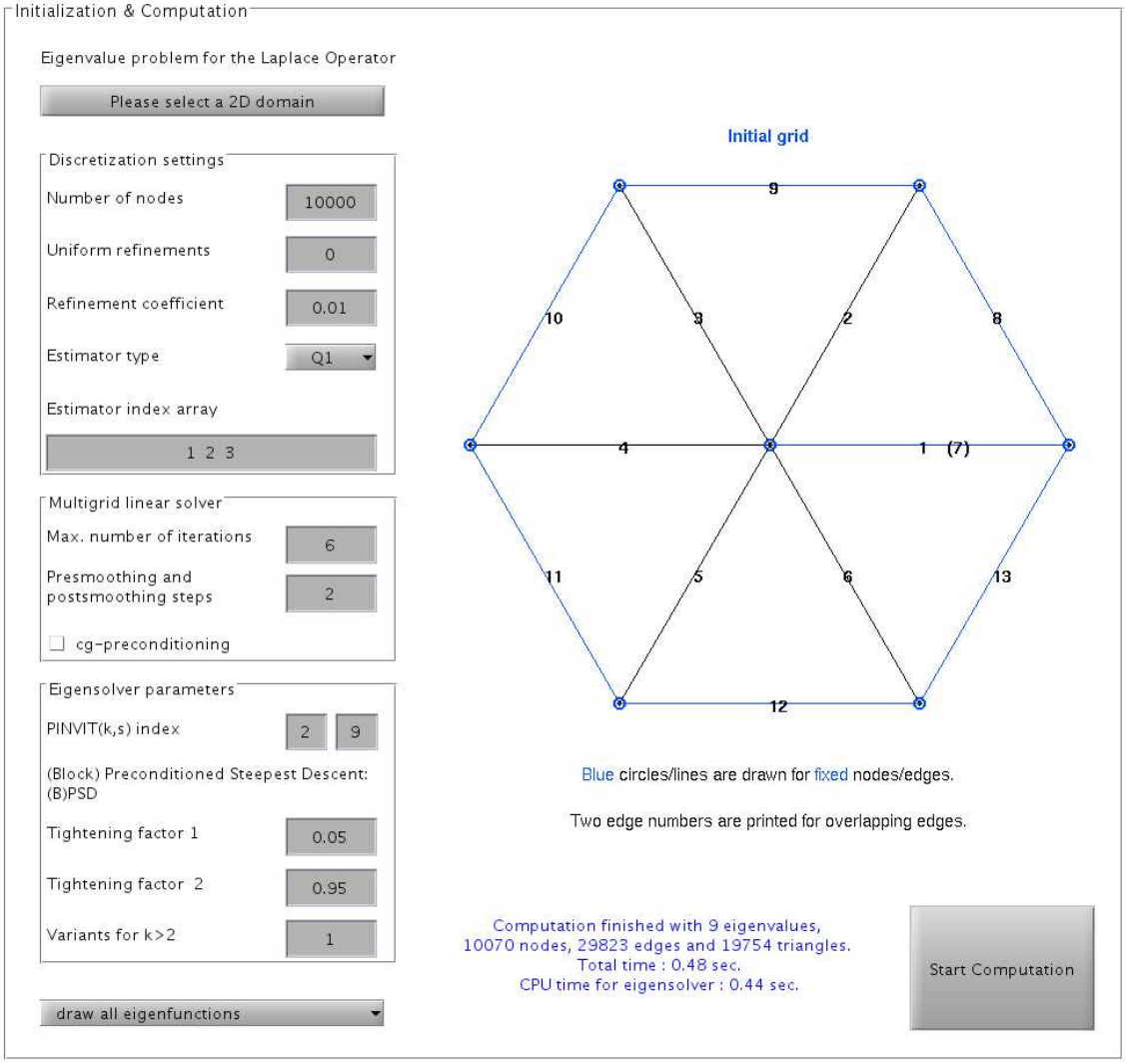


Figure 2: Panel "Initialization & Computation" after completion of the computation.

The right panel "Results" shows the 3D plots of the eigenfunction approximations, see Fig. 3. Additionally the six buttons in the lower part of the right panel allow to draw the

(i) `Computational costs`
(ii) `Triangulation`,
(iii) `Sparsity pattern`,
(iv) `Convergence of the Ritz values`,
(v) `Eigenfunctions`,
(vi) `Residuals`.

By clicking the right mouse button within the axes of one of these figures a separate Matlab figure opens. The figure can now be modified, printed or exported. It should be noted that the option to plot the sparsity pattern does not imply that AMPE uses internally the finite element discretization matrix in an explicit way. Instead the core of the FORTRAN program is a so-called matrix-free finite element multigrid solver for the eigenvalue problem.

1.1 External FORTRAN programs

The adaptive grid refinement process, the preconditioned gradient type eigensolver and the multigrid preconditioning are parts of an external FORTRAN program which is called from the AMPE GUI. Precompiled versions of the FORTRAN program for the following systems are contained in the distribution:

- `ampelinux32` for Linux 32 bit,
- `ampelinux64` for Linux 64 bit,
- `ampemac32` for Mac OS 32 bit,
- `ampemac64` for Mac OS 64 bit,
- `ampewin32.exe` for Windows 32 bit and
- `ampewin64.exe` for Windows 64 bit.

We have used the `gcc` compiler `gfortran`, see

`https://gcc.gnu.org/wiki/GFortranBinaries`.

The executable files can be found in the directory `programs`. AMPE automatically selects the right executable for your hardware.

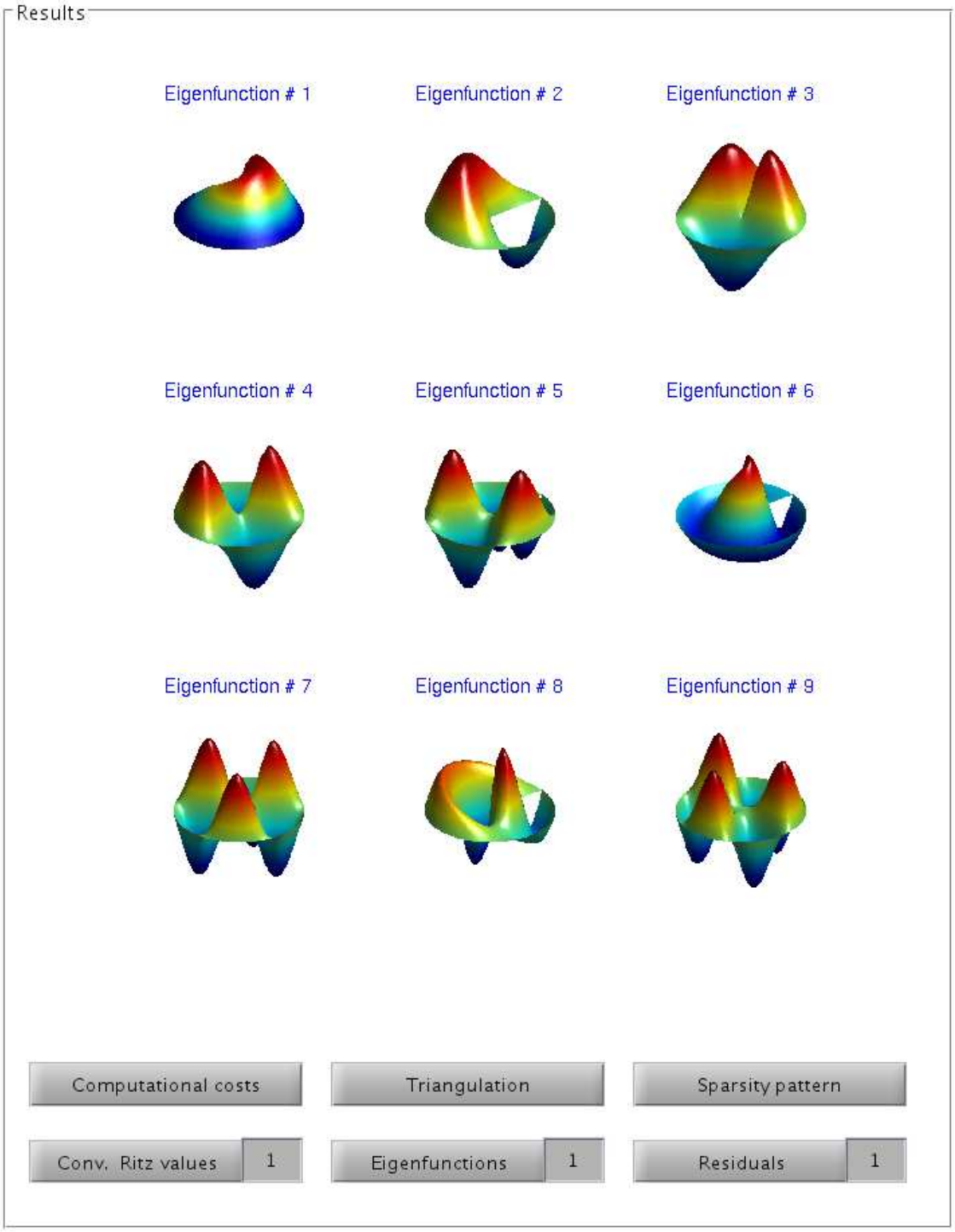


Figure 3: The "Results" panel shows the final eigenfunction approximations.

2 The eigenvalue problem and the eigensolvers

2.1 The elliptic eigenvalue problem

Let $\Omega \subset \mathbb{R}^2$, be a bounded, open, connected set with a Lipschitz continuous boundary $\Gamma = \Gamma_1 \dot\cup \Gamma_2$. The problem is to find a modest number of the smallest eigenvalues λ together with real-valued eigenfunctions $u = u(x)$ satisfying

$$\begin{aligned} -\nabla \cdot (c(x)\nabla u) + q(x)u &= \lambda u, & x &\in \Omega, \\ u &= 0, & x &\in \Gamma_1, \\ \nu \cdot c(x)\nabla u &= 0, & x &\in \Gamma_2. \end{aligned} \tag{1}$$

Therein, ν is the exterior unit normal to Γ_2, $c(x)$ is a symmetric positive definite matrix-valued function and $q(x) \geq 0$. In the present version of AMPE $c(x)$ is the identity matrix and $q(x) = 0$.

The weak formulation of (1) is the key for the application of the mathematically sound spectral theory of self-adjoint compact operators in Hilbert spaces. It guarantees the existence of a countable set of real eigenvalues [1, 21].

The Rayleigh-Ritz discretization of this weak form results in the generalized matrix eigenvalue problem

$$Ax = \lambda Mx. \tag{2}$$

The discretization (or stiffness) matrix $A \in \mathbb{R}^{n\times n}$ and the mass matrix $M \in \mathbb{R}^{n\times n}$ are symmetric and positive (semi)definite matrices, which are typically very large and sparse. The finite element discretization in AMPE uses piecewise linear elements and for the error estimation also piecewise quadratic finite elements.

2.2 Preconditioned gradient eigensolvers

The numerical solver for the eigenvalue problem (2) should exploit the structure of the mesh eigenproblem and its computational costs should ideally increase almost linearly in the dimension n of the matrices. These demands rule out all eigensolvers which are usually used for small and dense matrices, see [2, 7, 20]. Here any factorization of A or M must be avoided.

A conceptually very simple approach, which allows to construct near-optimal-complexity eigensolvers, uses gradient iterations for the iterative minimization of the Rayleigh quotient

$$\rho(v) = \frac{v^T Av}{v^T Mv}, \tag{3}$$

see [6, 14]. The gradient of the Rayleigh quotient is collinear to $Av - \rho(v)Mv$. A correction of a current iterate in the direction of the negative gradient of (3) can decrease the Rayleigh quotient if a proper step-length is used. The minimum of the Rayleigh quotient is the smallest eigenvalue of (2) and is attained in an associated eigenvector. This simple gradient iteration converges very slowly; the convergence factor grows like $\mathcal{O}(1 - h^2)$ in the discretization parameter h. A change of the underlying geometry can accelerate the convergence considerably. If $T \in \mathbb{R}^{n\times n}$ is an approximate inverse or preconditioner for A, then the gradient vector with respect the geometry induced by T^{-1} is collinear

to $T(Av - \rho(v)Mv)$. In the best case multigrid preconditioned gradient eigensolvers can converge with a grid-independent convergence rate and the computational costs for the approximation of a fixed number of the smallest eigenvalues increases only linearly in the degrees of freedom n [13]. If not only the smallest eigenvalue but a modest number of the smallest eigenvalues are to be computed, then the gradient iteration can easily be generalized to a subspace iteration.
In AMPE the operation $x \mapsto Tx$ is implemented by the approximate solution of a linear system with a multigrid preconditioner [8]. The user can specify the parameters on the number of pre- and postsmoothing Jacobi iterations per grid level and the maximal number of V-cycles in the left panel. The multigrid hierarchy is generated by an adaptive grid refinement strategy, see Section 5.

2.3 Preconditioned eigensolvers in AMPE

The AMPE software includes various implementations of preconditioned gradient solvers for the computation of one or more eigenvalues and associated eigenvectors by means of vector iterations and subspace iterations. These methods can be selected in the left panel of AMPE by choosing the parameters k and s within the PINVIT(k,s) hierarchy of iterative solvers [16]. The integer number s is the subspace dimension and the integer k says that the method uses not only the current preconditioned residual, but also $k-1$ preceding (vector or subspace) iterates in order to determine the next iterate by means of the Rayleigh-Ritz method. Typical selections of these parameters are explained next:

1. **k = s = 1**: This basic preconditioned gradient iteration is a vector iteration ($s = 1$) which maps a current iterate v_i to

$$v_{i+1} = v_i - Tr_i$$

with the residual $r_i = Av_i - \rho(v_i)Mv_i$. The preconditioner T is assumed to fulfill $\|I - TA\|_A < 1$ with respect to the operator norm induced by A. The iteration can be interpreted as a perturbed and/or preconditioned inverse iteration procedure (PINVIT). See [12] for the convergence analysis.

2. **k = 2, s = 1**: The preconditioned steepest descent (PSD) iteration uses the Rayleigh-Ritz procedure in order to extract from the two-dimensional space $\mathrm{span}\{v_i, Tr_i\}$ the Ritz vector of (A, M) which corresponds to the smaller Ritz value

$$v_{i+1} \longleftarrow \mathrm{RR}_{\min}\big(\mathrm{span}\{v_i, Tr_i\}\big).$$

This iteration is faster compared to the case $k = s = 1$ since the Rayleigh-Ritz method implicitly determines an optimal step length so that the Rayleigh quotient of the new iterate v_{i+1} is minimized. See [18] for the convergence analysis of PSD.

3. **k = 3, s = 1**: This iteration is known as the Locally Optimal Preconditioned Conjugate Gradient method (LOPCG), see [10, 11, 13].

$$v_{i+1} \longleftarrow \mathrm{RR}_{\min}\big(\mathrm{span}\{v_{i-1}, v_i, Tr_i\}\big).$$

Asymptotically this iteration has observed to behave like a preconditioned conjugate gradient iteration and includes a three-term recursion. The local optimality is achieved by the Rayleigh-Ritz procedure.

4. **k = 1, s > 1**: This block form of the preconditioned gradient iteration uses a block iterate $V_i \in \mathbb{R}^{n\times s}$ instead of the vector iterate v_i. This matrix contains in its columns the Ritz vectors of (A, M) in the column space of V_i. If Θ_i is the associated diagonal matrix containing the Ritz values on its diagonal, then the Ritz vectors and Ritz values of the next iterate are

$$(V_{i+1}, \Theta_{i+1}) \longleftarrow \mathrm{RR}_{s,\min}\big(\mathrm{span}\{V_i - TR_i\}\big)$$

with $R_i = AV_i - MV_i\Theta_i$. Therein $\mathrm{RR}_{s,\min}$ stands for the Rayleigh-Ritz procedure which extracts a subspace spanned by s Ritz vectors corresponding to the s smallest Ritz values.

5. **k = 2, s > 1**: The block preconditioned steepest descent (BPSD) iteration is an accelerated form of the case $k = 1$ and $s > 1$. Here the Rayleigh-Ritz procedure is applied to the $2s$ dimensional subspace spanned by the columns of V_i and TR_i. It has the form

$$(V_{i+1}, \Theta_{i+1}) \longleftarrow \mathrm{RR}_{s,\min}\big(\mathrm{span}\{V_i, TR_i\}\big).$$

For a convergence analysis see [19].

6. **k = 3, s > 1**: This important preconditioned subspace eigensolver with the form

$$(V_{i+1}, \Theta_{i+1}) \longleftarrow \mathrm{RR}_{s,\min}\big(\mathrm{span}\{V_{i-1}, V_i, TR_i\}\big)$$

is well-known as the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) method [11, 9]. This method has an optimal convergence behavior compared to all the other methods within the (k, s) hierarchy of iterative eigensolvers.

The preconditioned eigensolvers for $k > 3$ and with $s \geq 1$ have the form

$$(V_{i+1}, \Theta_{i+1}) \longleftarrow \mathrm{RR}_{s,\min}\big(\mathrm{span}\{V_{i-k+2}, \ldots, V_i, TR_i\}\big).$$

The efficiency of these iterations seems to fall behind the methods with $k = 3$ and the same s. The reason is that the additional costs for the Rayleigh-Ritz procedure to work with a $k \cdot s$ dimensional space cannot be justified by the marginal convergence acceleration for the iterations with $k > 3$. The reader can validate this statement by numerical experiments with AMPE.

3 The domain library

The AMPE software provides some predefined sample problems. For these test problems the initial grids are coded in a short Matlab program whose form is explained in Section 3.2. These initial grids are

(i) a `Square`,
(ii) an `L-shaped` domain,
(iii) a `Circle` and
(iv) an `Omega-shaped` domain.

For each domain various homogeneous boundary conditions (BC) can be selected, see the directory `domains` of the AMPE software and Section 3.1 for the details.

3.1 Sample domains

Next the initial grids for the four domains `Square`, `L-shaped`, `Circle`, and `Omega-shaped` are introduced. In the graphical representations of these domains blue circles and blue edges are located on the Dirichlet boundary. The Dirichlet boundary conditions for the eigenvalue problem are homogeneous and hence the eigenfunctions in blue nodes are fixed to zero (fixed nodes or edges). All other black edges are either interior edges or edges on the Neumann boundary of the domain (free nodes and edges). The edges of a triangulation are enumerated with black numbers.
Curvilinear parts of the boundary are approximated by sequences of straight edges. If an element is refined within the adaptive grid refinement procedure, then new nodes are projected to the boundary of the domain.

Domain (i): Square $\Omega = [-1,1]^2$ with homogeneous Dirichlet BC. This domain and its initial triangulation is defined in `square1.m` in the directory `domains`, see Figure 4.

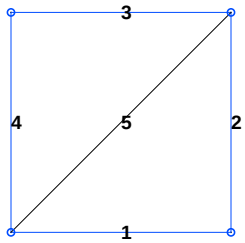


Figure 4: Initial triangulation of $\Omega = [-1,1]^2$

Domain (ii): For the L-shaped domain $\Omega = [-1,1]^2 \setminus [0,1]^2$ three different settings for the boundary conditions are used, cf. Figure 5.

- `lshaped1.m`: Homogeneous Dirichlet BC are used on the entire boundary.
- `lshaped2.m`: Homogeneous Dirichlet BC are assumed on the entire boundary, and a finer (but not refined) initial triangulation compared to `lshaped1.m` is used.
- `lshaped3.m`: Homogeneous Neumann BC are used for the two boundary edges touching the origin and homogeneous Dirichlet BC are assumed for the rest.

Domain (iii): The initial triangulation and the boundary conditions for the unit circle

$$\Omega = \{(r\cos(\varphi), r\sin(\varphi)) \ ; \ r \in [0,1], \ \varphi \in [0,2\pi]\}$$

are shown in Figure 6.

- `circle1.m`: A slit along the positive abscissa is assumed. Homogeneous Dirichlet BC are assumed for $r = 1$ and $\varphi \in (0,2\pi)$ as well as for $\varphi = 0$ and $r \in [0,1]$ (top side of the slit). On the bottom side of the slit with $\varphi = 2\pi$ and $r \in (0,1)$ homogeneous Neumann BC are used. The domain has a slit along the positive abscissa, which leads to overlapping nodes and edges.

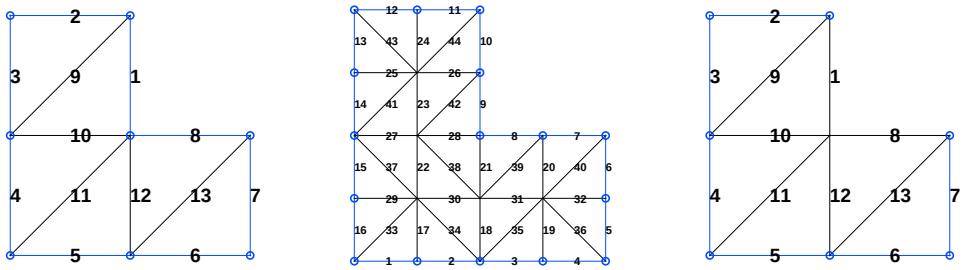


Figure 5: Initial triangulations of the L-shaped domain.

- `circle2.m`: Homogeneous Dirichlet BC for $r = 1$ and $\varphi \in [0, 2\pi]$.
- `circle3.m`: Homogeneous Dirichlet BC for $r = 1$ and $\varphi \in [0, 2\pi]$ as well as in the single point $r = 0$.

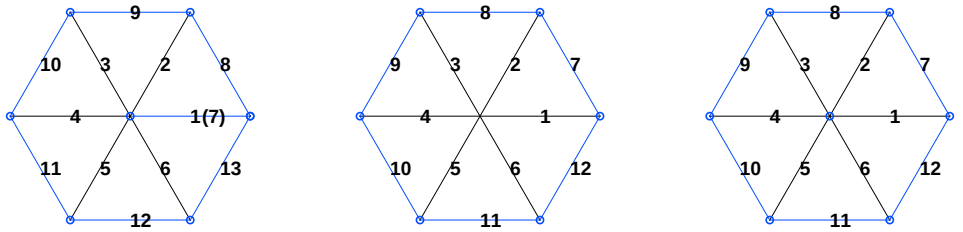


Figure 6: Initial triangulation and boundary conditions for the circle.

Domain (iv) : An Omega-shaped domain including the open ring

$$\Omega = \{(r\cos(\varphi), r\sin(\varphi)) \ ; \ r \in [0.7, 1], \ \varphi \in [\delta, 2\pi - \delta]\}$$

and two rectangles, see Figure 7. For the details see the Matlab code.

- `omega1.m`: Homogeneous Dirichlet BC are assumed on the entire boundary.
- `omega2.m`: Homogeneous Neumann BC are used on the inner circle and homogeneous Dirichlet BC on the remaining part of the boundary.

3.2 User defined domains

The user can define further domains and their boundary conditions by the definition of the initial triangulations. This is explained by using the Matlab subroutine `threequartercircle.m`

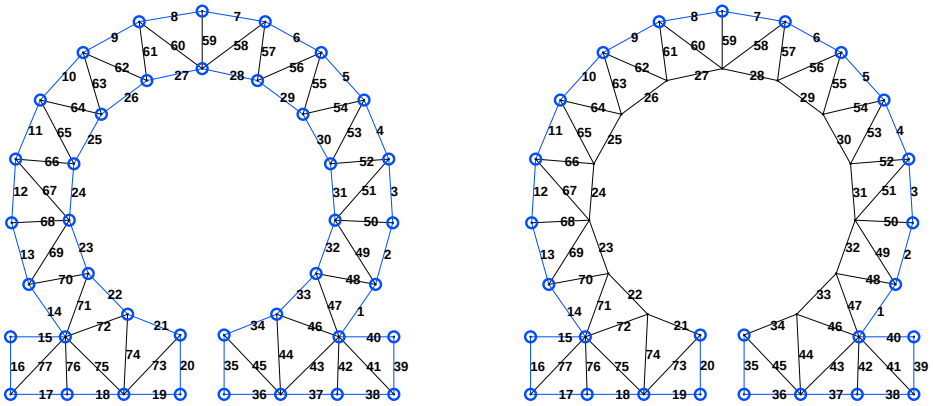


Figure 7: Initial triangulation and boundary conditions for the Ω-shaped domain.

which defines the domain of a three-quarter circle with homogeneous Dirichlet BC. With this domain the AMPEigensolver logo has been computed; this logo is shown on the start screen of the program. The initial triangulation with the enumeration of the edges and the enumeration of the nodes are shown in Figure 8.

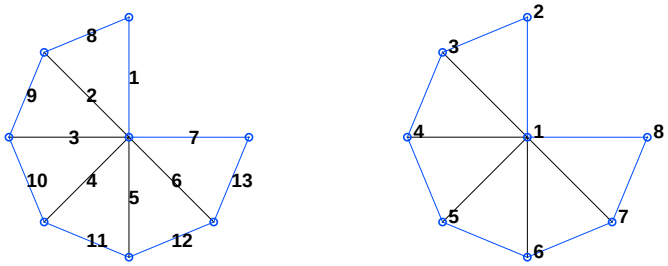


Figure 8: Initial triangulation and boundary conditions for the 3/4 circle (left) and enumeration of the nodes (right).

The file `threequartercircle.m` has the following form:

```
function threequartercircle
x=0; y=0;
t=linspace(pi/2,2*pi,7);
x=[x cos(t)]; y=[y sin(t)]; v=[];
e=[];
for k=2:8, e=[e; 1 k]; end
for k=2:7, e=[e; k k+1]; end
```

```
ep=zeros(size(e));
ep([1 7],1)=1;
ep(8:end,1)=2;
tr=[1 8 2; 2 9 3; 3 10 4; 4 11 5; 5 12 6; 6 13 7];
trp=zeros(size(tr,1),2);
cc=[0 0 1]; ci=2;
save domain.mat
```

The parameters and arrays are explained in the following.

`x,y`: These two vectors contain the planar coordinates of the nodes.

`v`: A vector of integers which are the indices of the free nodes. In the case of the 3/4 circle all initial nodes are located on the Dirichlet boundary so that `v` is empty.

`e`: This integer array contains in the ith row the indices of the nodes which are connected by the ith edge. In each row the indices are written in ascending order.

`ep`: The first column of this integer array contains the edge types.

The type 0 stands for interior edges and for edges on the (affine linear) Neumann boundary.

The type 1 is used for edges on the (affine linear) Dirichlet boundary.

Types "smaller than −1" are used if an edge approximates a region with a curvilinear Neumann boundary.

Types "larger than 1" denote edges which approximate a region with a curvilinear Dirichlet boundary.

Further the type −1 is used for auxiliary edges which are only used in the refined meshes. In the current version of AMPE, curvilinear parts of the boundary are approximated by arc elements, see the array `cc` below. The second column of `ep` contains the status information on the edges. For the initial triangulation these variables are set to 0.

`tr`: An integer array whose ith row contains the edge numbers for the ith triangle. The edge numbers are listed counter-clockwise.

`trp`: The first column contains the triangle types and the second column contains the depth of refinement for a triangle. All these variables are set equal to 0 for the initial triangulation. Section 5.2 explains further types for refined triangles.

`cc`: This variable can be used to describe curvilinear regions of the boundary. If the jth curvilinear region of the boundary is approximated by an arc element whose radius r is centered at (x, y), then the j row reads $[x\ y\ r]$. Thus `cc` is an $\alpha \times 3$ array if a number of α curvilinear regions of the boundary are approximated by arc elements.

`ci`: This $1 \times \alpha$ vector contains the edge types for the curvilinear regions of the boundary. See the variable `cc` for the arc elements and the variable `ep` for the edge types.

Remarks:

- The initial triangulation may have no free node. Then the adaptive eigensolver automatically invokes a uniform grid refinement (as a part of the program core written in FORTRAN).
- The program `domainview.m` in the directory `utilities` can be used in order to check whether or not (in user defined domains) the arrays `e` and `tr` are correctly defined.

4 The graphical user interface (GUI)

The AMPEigensolver combines a user-friendly graphical user interface (GUI) for the program control and for the graphical output with a numerically efficient core of the program written in FORTRAN. The GUI is written in Matlab and can be started by calling `ampe.m` from a Matlab command window or Matlab desktop. The GUI is explained next.

4.1 Problem selection

The first step is to select a 2D domain. See Section 3.1 for pre-defined domains and Section 3.2 for the generation of user defined domains. After the domain selection the initial grid is shown in the left panel, see Figure 9.

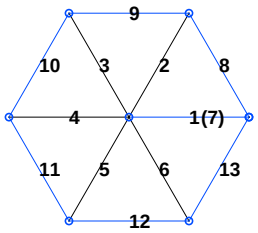


Figure 9: The initial grid `circle1.m`

In three subpanels various control parameters can be selected:

Subpanel 1: Discretization settings, see Figure 10.

`Number of nodes`: Maximal number of nodes which are acceptable with respect to the finest discretization.

`Uniform refinements`: Number of uniform refinement cycles which are to be applied to the initial triangulation before the adaptive grid refinement procedure is started.

`Refinement coefficient`: A real number $\alpha \in (0,1)$. The coefficient α defines a convex combination of the largest squared component and a weighted mean of the

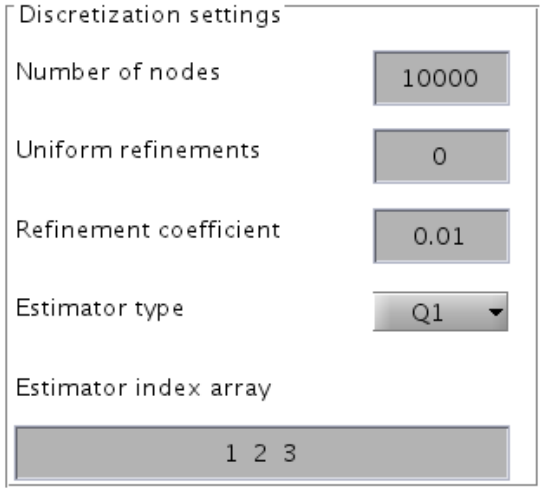


Figure 10: Setting of the parameters for the discretization.

squared components of the residual vector with respect to quadratic finite elements, see Equation 37 in [15]. All the edges whose squared component of the residual is larger than the convex combination are marked for a refinement.

`Estimator type`: The five estimators `Q1`, `Q2`, `Q3`, `L1` and `L2` are residual based a posteriori error estimators which use quadratic finite elements or an intermediary local regular refinement. See Section 5.1 for details.

`Estimator index array`: Vector of integers (separated by spaces) which are the indices of the eigenvectors whose residuals are used for the residual based error estimator.

Subpanel 2: Settings for the multigrid solver, see Figure 11.

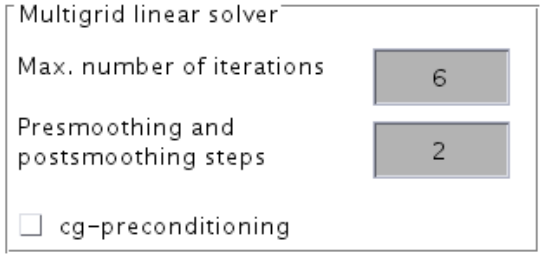


Figure 11: Settings for the multigrid solver.

`Maximal number of iterations`: The maximal number of iterations which can be executed by the approximate linear system solver. This linear solver, which represents the action of the preconditioner T, is the inner iteration within the inner-outer

loop structure. The outer iteration is the preconditioned gradient type eigensolver. Typically only 1 or 2 inner iterations are needed to decrease the residual of the linear systems sufficiently.

`Presmoothing and postsmoothing steps`: The number of the pre- and post-smoothing steps per grid level. A Jacobi smoother with a damping constant $\omega = 2/3$ is used. If a slow convergence of the linear system solver is detected, then the number of smoothing steps is automatically increased.

`cg-preconditioning`: If the cg-preconditioning box is checked, then the inner linear system solver is combined with a conjugate gradient method. This considerably increases the accuracy of the linear system solver. If the `Tightening factor 1` parameter in the subpanel "Eigensolver" is very small (less than 0.01), then the conjugate gradient solver is recommended.

Subpanel 3: Eigensolver, see Figure 12.

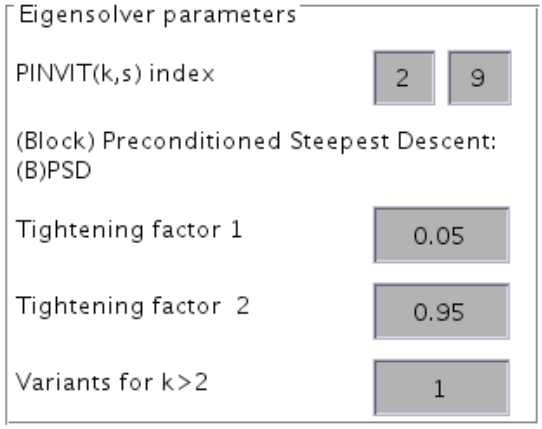


Figure 12: Parameter setting for the eigensolver.

`PINVIT(k,s) index`: The indices k and s are used to specify the gradient eigensolver. The integer value s is the dimension of the iteration subspace and for $k \geq 2$ a number of $k-1$ preceding and cuurent (vector or subspace) iterates together with the preconditioned residual are used to span the subspace from which the Rayleigh-Ritz procedure extracts the new subspace and its basis of Ritz vectors. See Section 2.3 for details. The parameter s is an upper bound for the maximal index for the `Estimator index array` in the subpanel 1.

`Tightening factor 1`: This factor, a real number in $(0,1)$, is used to tighten the stopping condition for the iterative gradient type eigensolver. The upper bound for the residual is multiplied by this factor and the preconditioned gradient iteration is stopped with respect to a certain grid level if the residual falls below this tightened bound. After this the error estimation and grid refinement procedures are called.

`Tightening factor 2`: This factor, a real number in $(0, 1)$, is used only for the finest discretization level as a second multiplicative tightening for the stopping condition. This leads to very accurate Ritz pairs with a small iteration error with respect to the finest level of discretization.

`Procedural variants for k>2`: Three variants for the Rayleigh-Ritz procedure can be selected by the program input 1, 2 or 3. In the case 1 all the vectors to which the Rayleigh-Ritz procedure is applied are orthogonalized with respect to the Euclidean inner product. This improves the stability of the projection step. In the case 2 no preceding orthogonalization is used. In precise arithmetic this case is equivalent to the case 1. However, for high-dimensional problems the usage of case 1 is recommended. In the case 3 a number of $k-1$ previous iteration subspaces, each space has the dimension s, are prolongated after a grid refinement. Then the subspace residual is computed for the prolongation of the last iteration subspace and so the Rayleigh-Ritz procedure can work in each iteration step with a ks-dimensional subspace.

Finally the menu at the bottom of the left panel allows to switch off the time consuming printing of 3D surface plots of the eigenfunctions with a very large number of nodes. All the eigenfunctions are automatically drawn if the final mesh has less than 30000 nodes. If the final mesh has between 30000 and 1 million nodes, then only the one eigenfunction corresponding to the smallest eigenvalue is drawn. For problems with more nodes the plot of the eigenfunctions can be deactivated in order to avoid a cost-intensive graphical representation.

4.2 Program execution

If one of the default parameters is changed, then the message "initial setting updated" is displayed in the left panel. If the initial settings can be accepted or if the necessary changes of the settings have been made, then the button "Start computation" allows to start the FORTRAN core program of the AMPEigensolver. This core program accesses the initial data via the files `domain` and `input` in the directory `fortran_io`.
During the program execution the mouse cursor in the Matlab GUI changes to a watch. Simultaneously a short transcript of the convergence history of the eigensolver appears on the standard output of the FORTRAN program (usually on command window or a xterm window), see Figure 13.
The first column contains the level indices, the second column the total number of nodes and the third column the degrees of freedom. Then the approximations of the smallest eigenvalue are listed. These data are for the domain circle1.m and can be compared with the analytical eigenvalues, see Section 5 in [17]. The three step numbers stand for the (outer) iterations of the eigensolver, the inner iterations of the multigrid linear systems solver (preconditioner) and for the pre- or postsmoothing steps. Finally the columns `ref`, `evp` and `est` contain the CPU times in seconds for the grid refinement process, for the gradient eigensolver and for the residual based error estimation. After program completion the triangulation is checked in order to verify that no hanging nodes have been generated. The FORTRAN program writes the program output in binary form to the file `output` in

```
program started
level      #nodes      d.o.f.      lambda(1)      #steps      ref      evp      est
    1          65          36    9.904281251                          0.00     0.00
    2         183         140    8.996708271     2  2  2      0.00     0.00     0.00
    3         266         218    8.574820851     1  1  2      0.00     0.00     0.00
    4         599         527    8.288486559     2  1  2      0.00     0.00     0.00
    5         709         633    8.132655237     2  1  2      0.00     0.00     0.00
    6        1004         903    8.001751898     2  1  2      0.00     0.01     0.00
    7        1048         946    7.933179202     2  1  2      0.00     0.01     0.00
    8        1342        1227    7.878231104     2  1  2      0.00     0.01     0.00
    9        1815        1696    7.844654845     3  1  2      0.00     0.02     0.00
   10        2226        2096    7.819098917     2  1  2      0.00     0.02     0.00
   11        3185        3006    7.791113620     2  1  2      0.00     0.02     0.00
   12        3449        3267    7.778759977     2  1  2      0.00     0.02     0.00
   13        4123        3903    7.766060987     2  1  2      0.00     0.03     0.00
   14        4219        3997    7.760100351     2  1  2      0.00     0.03     0.00
   15        4921        4669    7.753190234     2  1  2      0.00     0.04     0.00
   16        6114        5858    7.749948076     2  1  2      0.00     0.05     0.00
   17        7999        7737    7.746953318     2  1  2      0.00     0.06     0.00
   18       10070        9761    7.744099295     2  1  2      0.00     0.08     0.01
total cpu time:      0.44
all edges checked
all triangles checked
program completed
```

Figure 13: Transcript of the convergence history of the eigensolver.

the directory `fortran_io`. For large problems the generation and writing of the output can be time-consuming.
The Matlab GUI of AMPE reads these data and provides a graphical presentation. For instance the eigenfunction approximations are drawn in the right panel, see Figure 3. Further a short summary of the computation is shown left of the "Start computation" button, see Figure 14.

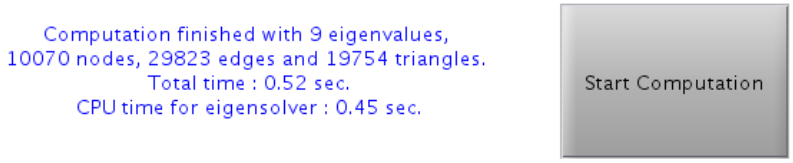


Figure 14: Short summary of the computation.

4.3 Results

The right panel of the GUI provides various options for the visualization of the numerical results, see Figure 15. By clicking the right mouse button within the axes of one of these graphical representations a separate Matlab figure with the same content opens. This figure can now be modified, printed or exported in the way as usual in Matlab.

`Computational costs`: Within a double logarithmic (log-log) plot the CPU times per level and the total CPU times since program start are plotted versus the number

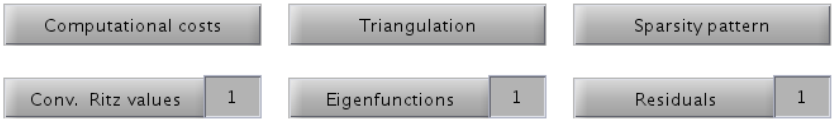


Figure 15: Options of the results menu.

of nodes.

`Triangulation`: The final and finest triangulation is drawn.

`Sparsity pattern`: The sparsity pattern of the stiffness matrix A with respect to the final triangulation is plotted. However, the eigensolver is a so-called matrix-free finite element multigrid solver which uses only the operation $x \mapsto Ax$. The sparsity pattern is generated by analyzing the index arrays for the edges. For high-dimensional problems this plot can take a very long time.

`Conv. Ritz val.`: The convergence of Ritz values is presented by drawing the differences of the Ritz values (for all grid levels aside from the last level and for each iterate) and the final Ritz values with respect to the finest and final triangulation. These errors are displayed in groups of three Ritz values, i.e. the first group includes the three smallest Ritz values and so on. The group index is located directly right to the button.

`Eigenfunction`: The eigenfunction approximations whose index appears directly to the right of the button is plotted. The default value is 1.

`Residual`: This plot shows for the active eigenvalue/function index (see directly right to the button) an upper estimate for the residual, the stopping criterion and the current computed residual versus the level index. See [15] for mathematical details.

Remarks:

- The font size is automatically adapted to the size of the GUI window.
- A Matlab menu bar and a toolbar can be added to the GUI window by clicking the right mouse button on the outer boundary of the GUI. This allows for instance to print and to export the window. By clicking the left mouse button in the margin these additional bars disappear if tools like “zoom” or “rotate” are not active.

5 Error estimation and adaptive grid refinement

5.1 Residual based error estimation

For an introduction to a posteriori error estimation see [22] and also [3, 4, 5, 23] for the theory on and for applications of residual based error estimators for the solution of boundary value problems. Similar concepts can be applied to the adaptive multigrid solution

of eigenvalue problems [17]. An interesting feature of multigrid subspace eigensolvers is that the error estimation and adaptive grid refinement can be coupled only to a specific eigenfunction or to groups of eigenfunctions with potentially different regularity.
The AMPEigensolver includes five a posteriori error estimators. The estimators `Q1`, `Q2` and `Q3` are edge oriented estimators which use *quadratic* finite elements. The estimators `L1` and `L2` are based on an intermediary local regular refinement of triangles with *linear* finite elements. By this intermediary refinement residual based error indicators can also be computed for all edges of a triangle. Short explanations are given below; for the details on the error estimation with quadratic finite elements in AMPE see [15, 17].

`Q1:` The estimator `Q1` compares the sum (per element) of squares of the components of the eigenvector residual for the midpoints of the edges (these error indicators can be computed by using quadratic finite elements) with a certain bound which is computed from a convex combination of the average value and of the maximal value of the squared components of the residual. *All* edges of the triangle are marked for a refinement if the local error indicator is larger than the critical bound.

`Q2:` The estimator `Q2` is very similar to `Q1` but analyzes the residual error indicators for each edge individually. In contrast to `Q1` this allows that only single edges of a triangle are marked for the refinement.

`Q3:` The estimator `Q3` is a modification of the estimator `Q2`. If an edge is marked for the refinement all other edges of the same triangle are also marked (but no marking recursion follows in order to avoid a uniform refinement).

`L1:` The estimator `L1` uses an intermediary local regular refinement of a triangle with *linear* finite elements. This allows to compute for midpoints of edges of a triangle the associated components of the eigenvector residual with respect to linear elements. The decision on marking for refinement is made as in the case `Q2`.

`L2:` The estimator `L2` works with the concept of `L1` and uses the marking strategy of `Q3`.

5.2 Triangle subdivision and grid refinement

Our adaptive grid refinement procedure works with four different classes of triangles. These classes are represented by the four colors white, green, red and cyan. The rules for the triangle subdivision are as follows.

1) All the triangles of the initial triangulation are white triangles.

2) If one or more edges of a white triangle are marked for a refinement, then the subdivision can result in triangles of other types, see Figure 16.

   2a) If only a single edge of a triangle is marked, then its subdivision results in a red and a cyan triangle.

   2b) If exactly two edges are marked for a refinement, then the subdivision results in a white triangle, a green triangle and a cyan triangle.

   2c) If all the three (midpoints of the) edges of a triangle are marked, then the uniform subdivision of the triangle results in four white triangles.

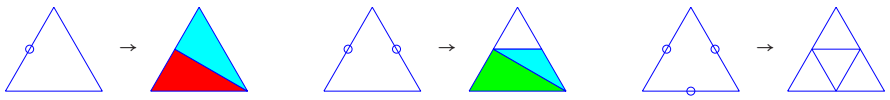


Figure 16: Possible refinements of a white triangle depending on the number of marked edges.

In all subsequent triangle subdivisions white triangles are treated as explained above. The remaining triangles are always considered as pairs "red + cyan" or "green + cyan"; thus a cyan triangle is always accompanied by a green or by a red triangle. All further triangle subdivisions result only in the following pairs of triangle types

"white", "red + cyan", "green + cyan".

Together with proper rotations no further triangle types are to be defined.
To this end the common edge of such colored pairs of triangles is always considered as passive, i.e. this edge can be removed or reoriented. For the remaining four active edges of the pair of triangles there are $2^4 - 1 = 15$ possibilities for the further subdivision. In order to avoid acute-angled triangles the algorithm automatically marks edges belonging to a triangle with a smaller depth within the tree of triangles (these are usually longer edges compared to the remaining edges), whenever an edge belonging to a triangle with a higher depth has been marked. All possible subdivisions of such pairs of triangles a shown in Figure 17 for the pair "red + cyan" and in Figure 18 for the pair "green + cyan". In these figures the markings of edge midpoints by the error estimator are shown by small circles and the additional markings due to the refinement rules are represented by asterisks. More details are to be explained in a forthcoming paper.

5.3 Program execution without using a Matlab GUI

The FORTRAN core of AMPE can be invoked without using the Matlab GUI. To this end two simple Matlab programs are provided in the directory `utilities`. The program `amped.m` produces plots of the sequence of triangulations of the adaptive grid refinement process. The triangles are colorized according to the type definitions in Section 5.2 and the gray value of the colors increases with the depth of a triangle in the triangle tree. A sequence of six triangulations for the unit circle domain with a slit along the positive axis, see domain `circle1.m`, is shown in Figure 19. Finally, the program `ampeo.m` extracts the sequence of eigenvalue approximations from the FORTRAN output file and draws the final eigenfunction corresponding to the smallest eigenvalue.

6 A test problem with 85 million nodes

For a short demonstration of the AMPEigensolver with a large number of nodes we select the domain `Circle 1`. We use the preconditioned steepest descent subspace iteration with

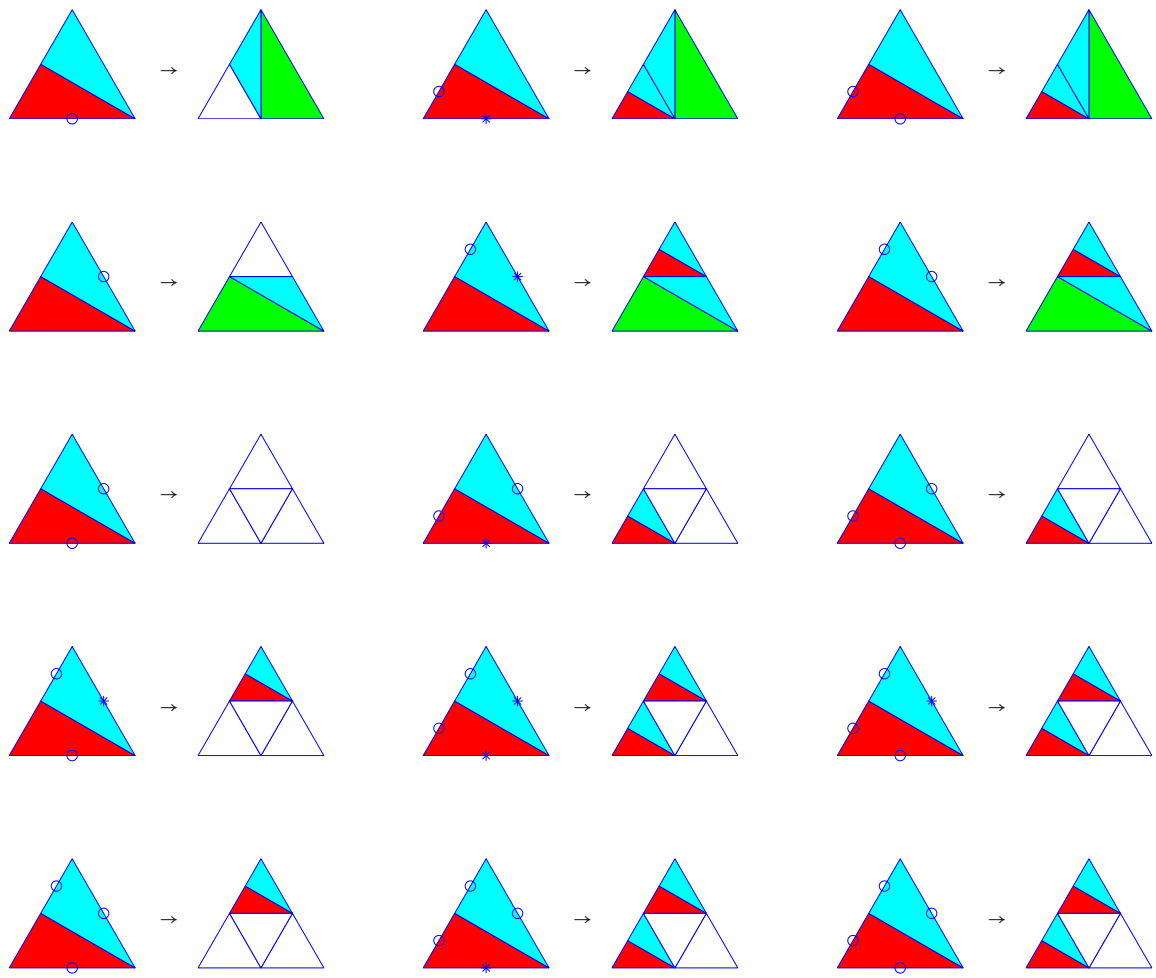


Figure 17: All possible triangle subdivisions of the triangle pair “red + cyan”.

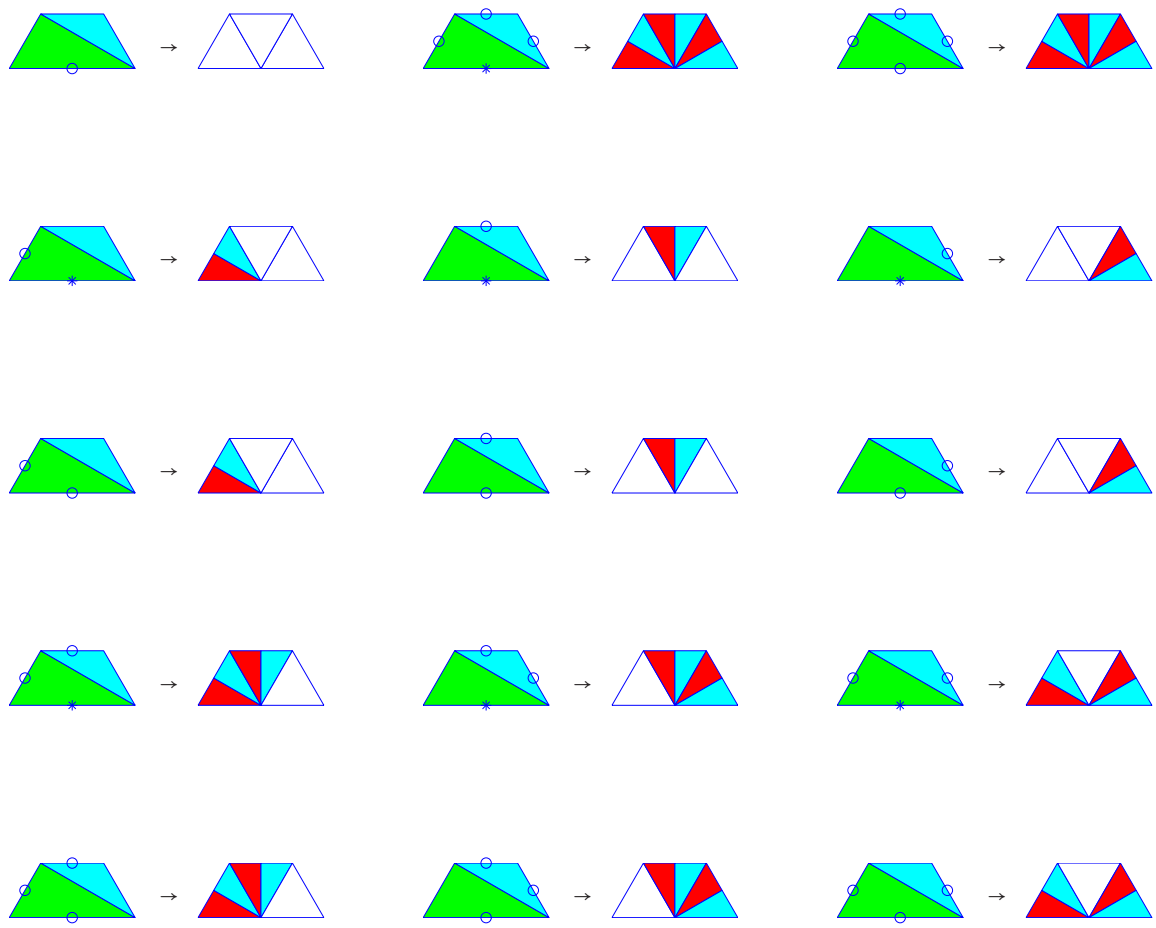


Figure 18: All possible triangle subdivisions of the triangle pair “green + cyan”.

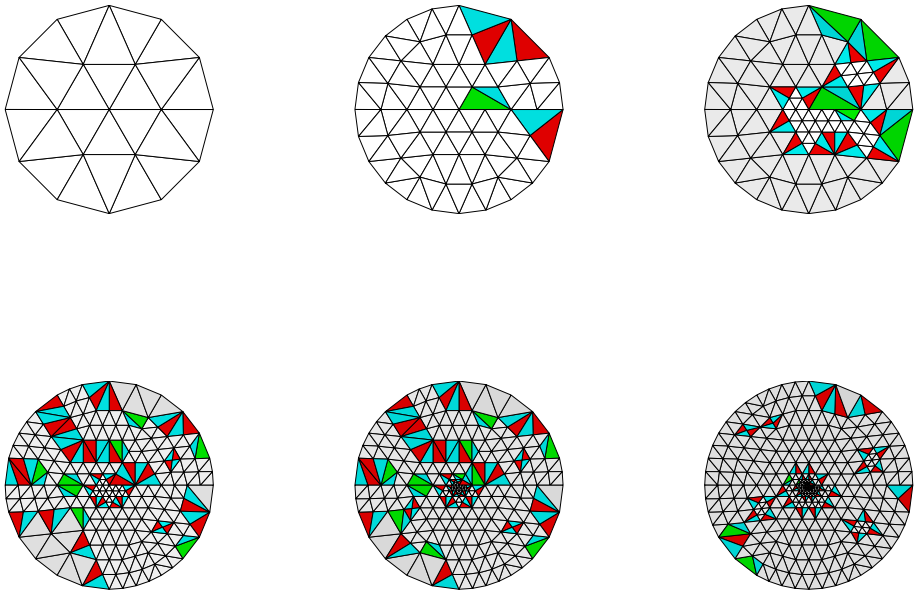


Figure 19: Series of adaptively generated grids for the domain `circle1.m`.

$k = 2$ and $s = 3$ together with the residual based error estimator `Q3` in order to compute the three smallest eigenvalues and associated eigenfunctions. For the convergence analysis of this method see [19]. Further we set the `Estimator index array` equal to 1 in order to get a highly nonuniform grid for this H^1 eigenfunction with an unbounded derivative at the origin. All other initial settings are the default values. The numerical experiments have been performed on a standard PC with an Intel Xeon 3.2GHz CPU and with a RAM of 31.4GiB.
The resulting finest grid has 85 611 460 nodes and 85 584 086 degrees of freedom. Hence the associated 3D subspace has more than $256 \cdot 10^6$ eigenvector components. Up to a discretization level with 53 289 325 nodes the RAM of 31.4GiB is sufficient and after that disk swapping is needed. The total computational time (CPU time) is 582.86 seconds. Further details are shown in Figure 20.
Table 1 shows a table of eigenvalue approximations (Ritz values) versus the level index and number of nodes for the adaptive subspace eigensolver. The three smallest exact eigenvalues corresponding to Bessel eigenfunctions are

$$\lambda_1 \approx 7.7333365335, \;\; \lambda_2 \approx 12.1871394681, \;\; \lambda_3 \approx 17.3507761314. \tag{4}$$

Furthermore, Figure 21 and Figure 22 show sectional enlargements centered at the origin of the computed adaptive grids. The triangulation is highly non-uniform.

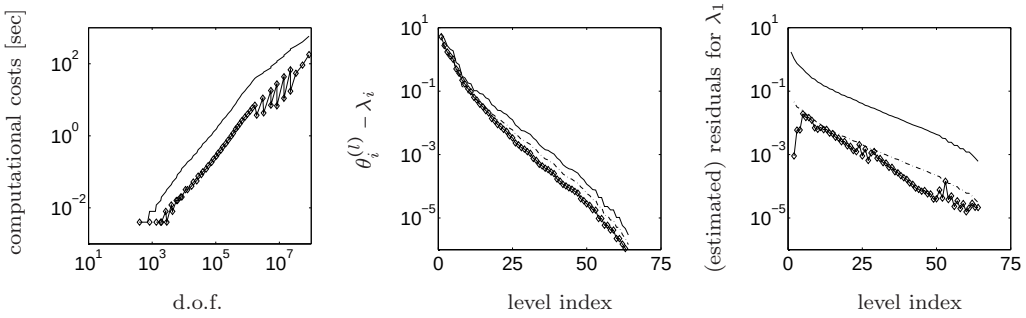


Figure 20: The preconditioned steepest descent iteration (k = 2) with an (s = 3)-dimensional subspace. Left: Total computational costs (solid line) and costs per level (line with markers). Center: Error of the three smallest eigenvalue approximations with regard to the exact eigenvalues given in Equation (4): Line with markers i = 1, broken line i = 2 and solid line i = 3. Right: Residual for i = 1: Estimated residual norm w.r.t. quadratic elements (solid line), modified estimate which is used for the stopping criterion (broken line) and computed residual norm $\|r\|_T^2/\|Tr\|_A$ w.r.t. linear elements (line with markers).

level	nodes	d.o.f.	$\theta_1^{(l)}$	$\theta_2^{(l)}$	$\theta_3^{(l)}$
1	21	6	12.9556062556	16.3582266789	23.5305271202
15	3028	2882	7.7655645173	12.2259533964	17.4312458018
23	13242	12926	7.7387413406	12.1960595144	17.3691880581
31	58353	57682	7.7344307573	12.1892371364	17.3551180201
34	105971	104946	7.7338290185	12.1881857006	17.3528299159
45	665189	662626	7.7334171818	12.1873063161	17.3511134402
49	1416986	1413079	7.7333714847	12.1872193825	17.3509373188
56	8368074	8358333	7.7333424332	12.1871514306	17.3507991426
58	13912669	13901749	7.7333406933	12.1871480617	17.3507940911
60	22248755	22233131	7.7333387701	12.1871445391	17.3507865365
62	33008964	32989529	7.7333380316	12.1871424802	17.3507819249
64	85611460	85584086	7.7333372596	12.1871409134	17.3507790338

Table 1: Eigenvalue approximations computed by the preconditioned steepest descent subspace iteration with k = 2 and s = 3.

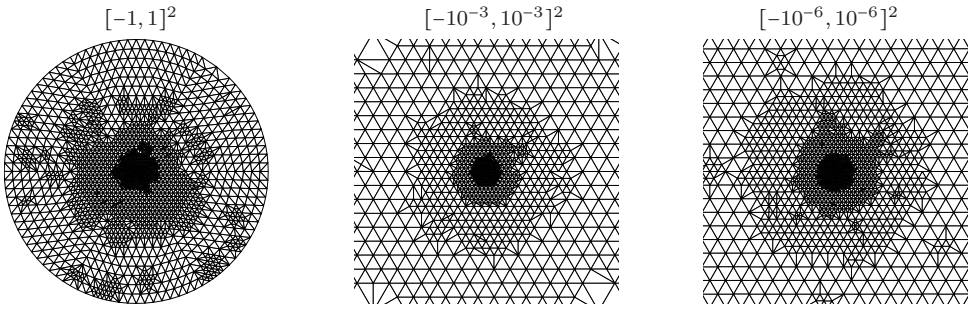


Figure 21: Sectional enlargements of triangle meshes with $n \in \{3028, 13242, 105971\}$ nodes and $\{2882, 12926, 104946\}$ inner nodes. The associated depths of the triangulations are 15, 23 and 34. The positive axis $r \geq 0$ and $\varphi = 0$ is a part of the boundary (and does not include hanging nodes).

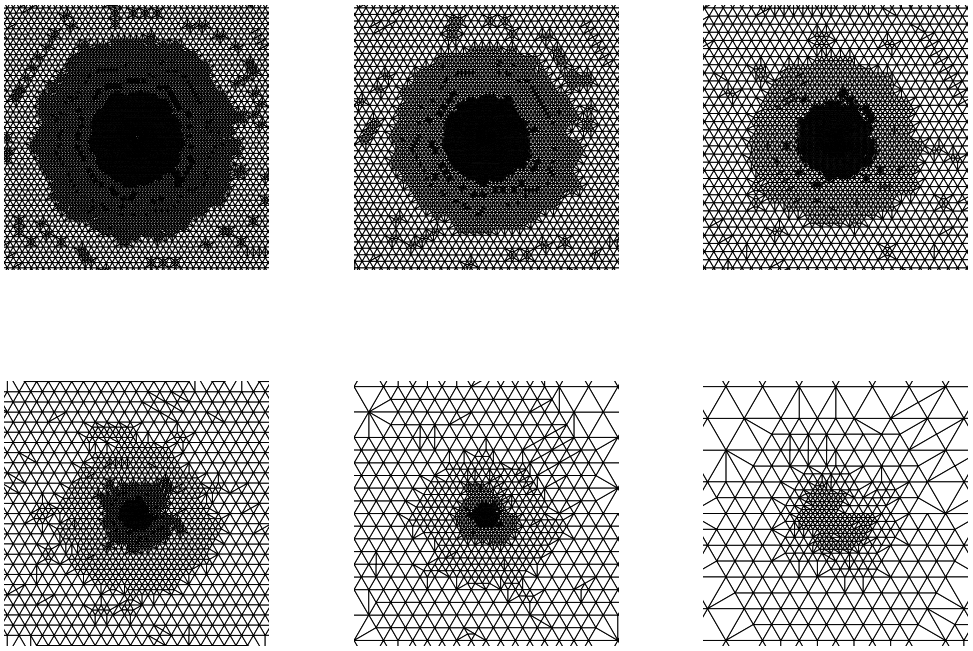


Figure 22: Sectional enlargements of triangle meshes with 13 912 669 nodes and 13 901 749 free nodes. The associated depth of the triangulation is 58. The subfigures show enlargements centered at the origin within $[-10^{-i}, 10^{-i}]$ for $i = 11, \ldots, 16$. The positive axis $r \geq 0$ and $\varphi = 0$ is a part of the boundary (and does not include hanging nodes).

7 Future work

In forthcoming revisions of AMPE we plan to add the following features:

a) Solution of eigenvalue problems for general second order self-adjoint elliptic partial differential operators, see Equation (1).
b) Implementation of general user defined curvilinear domains (not only by arc elements as in the current version).
c) Solution of elliptic eigenproblems in 3D.

References

[1] I. Babuška and J. Osborn, *Handbook of numerical analysis*, vol. II, ch. Eigenvalue problems, Elsevier, North–Holland, 1991.

[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst (eds.), *Templates for the solution of algebraic eigenvalue problems: A practical guide*, SIAM, Philadelphia, 2000.

[3] R.E. Bank, *PLTMG: A software package for solving elliptic partial differential equations. Users' guide 7.0*, SIAM Books, Philadelphia, 1994.

[4] R.E. Bank and A. Weiser, *Some a-posteriori error estimators for elliptic partial differential equations*, Math. Comput. **44** (1985), 283–301.

[5] P. Deuflhard, P. Leinen, and H. Yserentant, *Concepts of an adaptive hierarchical finite element code*, Impact Comput. Sci. Engrg. **1** (1989), 3–35.

[6] E.G. D'yakonov, *Optimization in solving elliptic problems*, CRC Press, Boca Raton, Florida, 1996.

[7] G.H. Golub and C.F. Van Loan, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, 2012.

[8] W. Hackbusch, *On the computation of approximate eigenvalues and eigenfunctions of elliptic operators by means of a multi-grid method*, SIAM J. Numer. Anal. **16** (1979), 201–215.

[9] A. V. Knyazev, M. E. Argentati, I. Lashuk, and E. E. Ovtchinnikov, *Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) in hypre and PETSc*, SIAM J. Sci. Comput. **29** (2007), 1267–1280.

[10] A.V. Knyazev, *A preconditioned conjugate gradient method for eigenvalue problems and its implementation in a subspace*, International Ser. Numerical Mathematics, 96, Eigenwertaufgaben in Natur- und Ingenieurwissenschaften und ihre numerische Behandlung, Oberwolfach, (Basel), Birkhäuser, 1991, pp. 143–154.

[11] ———, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comp. **23** (2001), 517–541.

[12] A.V. Knyazev and K. Neymeyr, *A geometric theory for preconditioned inverse iteration. III: A short and sharp convergence estimate for generalized eigenvalue problems*, Linear Algebra Appl. **358** (2003), 95–114.

[13] ———, *Efficient solution of symmetric eigenvalue problems using multigrid preconditioners in the locally optimal block conjugate gradient method*, Electron. Trans. Numer. Anal. **15** (2003), 38–55.

[14] D.E. Longsine and S.F. McCormick, *Simultaneous Rayleigh-quotient minimization methods for $Ax = \lambda Bx$*, Linear Algebra Appl. **34** (1980), 195–234.

[15] K. Neymeyr, *A posteriori error estimation for a preconditioned algorithm to solve elliptic eigenproblems*, Tech. Report 77, Sonderforschungsbereich 382, Universitäten Tübingen und Stuttgart, 1997.

[16] ———, *A hierarchy of preconditioned eigensolvers for elliptic differential operators*, Habilitationsschrift an der Mathematischen Fakultät, Universität Tübingen, 2001.

[17] ———, *A posteriori error estimation for elliptic eigenproblems*, Numer. Linear Algebra Appl. **9** (2002), 263–279.

[18] ———, *A geometric convergence theory for the preconditioned steepest descent iteration*, SIAM J. Numer. Anal. **50** (2012), 3188–3207.

[19] K. Neymeyr and M. Zhou, *The block preconditioned steepest descent iteration for elliptic operator eigenvalue problems*, Electron. Trans. Numer. Anal. **41** (2014), 93–108.

[20] B.N. Parlett, *The symmetric eigenvalue problem*, Prentice Hall, Englewood Cliffs New Jersey, 1980.

[21] P.A. Raviart and J.-M. Thomas, *Introduction à l'analyse numérique des équations aux dérivées partielles*, Masson, Paris, 1992.

[22] R. Verfürth, *A review of a posteriori error estimation and adaptive mesh-refinement techniques*, Wiley and Teubner, New York and Stuttgart, 1995.

[23] O.C. Zienkiewicz, D.W. Kelley, S.R. Gago, and I. Babuška, *Hierarchical finite element approaches, error estimates and adaptive refinement*, The mathematics of finite elements and applications IV, Academic Press, New York, 1982, pp. 313–346.