Multigrid

Julian Roth, Max Schröder

June 19, 2020

TABLE OF CONTENTS:

CHAPTER

ONE

BASICS OF GEOMETRIC MULTIGRID

1.1 Introduction

In the following, we are describing the geometric multigrid method, which for certain problems yields an iterative solver with optimal cost complexity, i.e. the solver returns a solution to a PDE in $O(n_{\text{DoFs}})$ arithmetic operations. We will show that this can also be achieved for some convection-diffusion equations on uniformly refined triangular meshes, when discretizing with linear finite elements.

1.2 Problem setup

Let $\Omega := (-1,1)^2 \setminus (0,1)^2$ be an L-shaped domain. We decompose the boundary of this domain $\partial\Omega$ into the Neumann boundary $\Gamma_D := (0,1) \times \{0\} \cup \{0\} \times (0,1)$ and the Dirichlet boundary $\Gamma_N := \partial\Omega \setminus \Gamma_D$.

Next, we define the ansatz and test function space $V := \left\{u \in H^1(\Omega) \mid u = 0 \text{ on } \Gamma_D\right\}$, where

$$H^1(\Omega) := W^{1,2}(\Omega) := \left\{u \in L^2(\Omega) \mid |\nabla u| \in L^2(\Omega)\right\}$$

is the Sobolev space containing the weakly differentiable functions in Ω. Note that we haven't explicitly prescribed any boundary conditions on Γ_N in the function space, since the homogeneous Neumann boundary conditions come up naturally when deriving the variational form of the problem. Further, we need a right hand side function

$$f(x) := \begin{cases} -1 & \text{for } x \in (-1,0) \times (0,1) \\ 0 & \text{for } x \in (-1,0) \times (-1,0) \\ 1 & \text{for } x \in (0,1) \times (-1,0) \end{cases}.$$

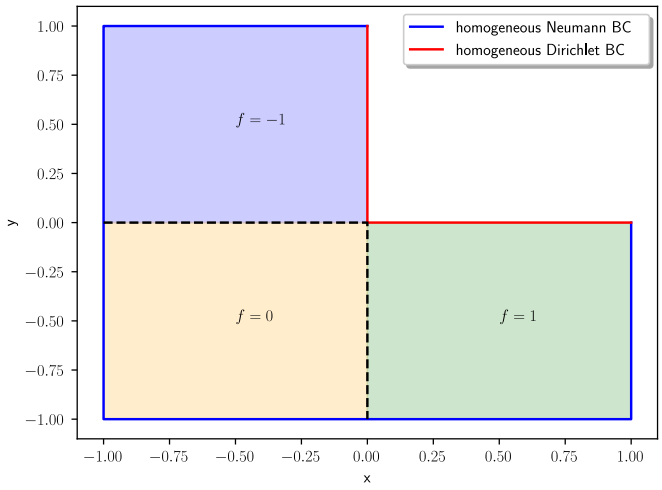


Figure 1: Domain Ω

Using the parameters $a = 1$ and $c = 0$ in Ω, we can now formulate the strong form of our convection diffusion equation:

Strong form

Find $u : \Omega \to \mathbb{R}$ such that

$$\begin{aligned} -\nabla \cdot (a \nabla u) + cu &= f \quad \text{in } \Omega \qquad (1) \\ u &= 0 \quad \text{on } \Gamma_D \\ \partial_n u &= 0 \quad \text{on } \Gamma_N \end{aligned}$$

In the above formulation, we used the notation $\partial n := \nabla \cdot n$. To be able to solve this problem, we need to convert it into its integral form. Therefore we multiply (1) from the right with a test function $v \in V$ and integrate over Ω.

$$-\int_\Omega (\nabla \cdot (a \nabla u)) \cdot v \, dx + \int_\Omega cu \cdot v \, dx = \int_\Omega f \cdot v \, dx \quad \forall v \in V$$

Now integration of parts can be applied to the first integral and we use the fact that $\partial\Omega = \Gamma_D \dot\cup \Gamma_N$.

$$\begin{aligned} \int_\Omega a \nabla u \cdot \nabla v \, dx - \int_{\Gamma_D} a \partial_n u \cdot v \, ds - \int_{\Gamma_N} a \partial_n u \cdot v \, ds \\ + \int_\Omega cu \cdot v \, dx = \int_\Omega f \cdot v \, dx \quad \forall v \in V \end{aligned}$$

Note that the integrals over the boundaries Γ_D and Γ_N vanish, since $\partial_n u = 0$ on Γ_N and, due to $v \in V$, $u = 0$ on Γ_D. Thus we have derived the following integral problem of our problem, which is often referred to as the weak or variational form in the literature.

Weak form

Find $u \in V$ such that

$$a(u, v) = l(v) \quad \forall v \in V$$

where $a : V \times V \to \mathbb{R}$ is the bilinear form defined as

$$a(u, v) := \int_\Omega a\nabla u \cdot \nabla v \, dx + \int_\Omega cu \cdot v \, dx$$

and the right hand side $l : V \to \mathbb{R}$ is a linear form defined as

$$l(v) := \int_\Omega f \cdot v \, dx$$

Furthermore using the fundamental lemma of calculus of variations, it can be shown that the strong and the weak form are equivalent. Hence it suffices to solve the weak form of the problem.

1.3 Finite Element Method

The problem that we are facing is that V is an infinite dimensional function space and we need the ability to solve the weak form with a classical computer. Hence instead we work with a finite dimensional subspace $V_h \subset V$. This will enable us to rewrite the weak form as a linear equation system.

Let a subdivision of Ω into finite elements (K, P_1, Σ) be given, where

- K is a two dimensional right triangle,
- $P_1(K) := \text{span}\{1 - x_1 - x_2, x_1, x_2\}$ is the space of linear functions defined on K,
- $\Sigma := \{a_0, a_1, a_2\}$ is a set of degrees of freedom (DoF), which here are the values of the polynomial at the vertices of K.

Then a $P_1(K)$ function is defined by

$$u(x) = a_0 + a_1x_1 + a_2x_2 \quad \forall x = (x_1, x_2) \in K.$$

To recapitulate: First, we have divided Ω into triangles $K_1, ..., K_m$. Examples for this can be found in the section *"Grid Setup"*. Secondly, we have seen that we have the parameters (DoFs) which can describe any linear function on such a triangle K_k. Now simply define our function space V_h as the space of functions which are continuous on the whole domain Ω, linear on each triangle K_k and satisfy the Dirichlet boundary conditions, i.e.

$$V_h := \{v_h \in C(\Omega) \mid v_h|_{K_k} \in P_1(K_k) \quad \forall 1 \leq k \leq m, \; v_h = 0 \text{ on } \Gamma_D\}.$$

We use the index h to show that we are not longer using the infinite dimensional function space V, but a finite dimensional subspace which is defined on triangles K_k where the short sides have length h. By working with V_h, we now try to find an element-wise linear approximation to the solution of the weak form.

Thus we are now trying to solve the discrete weak form:

Discrete weak form

Find $u_h \in V_h$ such that

$$a(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h$$

Furthermore, we know that V_h is finite dimensional and we can write down its basis, since we know the bases of $P_1(K_k)$. Hence

$$V_h = \text{span}\{\phi_1, \ldots, \phi_{n_{DoFs}}\},$$

where ϕ_i is the basis function corresponding to the i.th degree of freedom, i.e. the i.th grid point. It follows that

$$u_h = \sum_{i=0}^{n_{DoFs}} u_i \phi_i \quad \text{and} \quad v_h = \sum_{i=0}^{n_{DoFs}} v_i \phi_i$$

for some $\left(u_1, \dots u_{n_{DoFs}}\right)^T, \left(v_1, \dots v_{n_{DoFs}}\right)^T \in \mathbb{R}^{n_{DoFs}}$. Therefore the discrete weak form can be written as

$$a\left(\sum_{i=0}^{n_{DoFs}} u_i \phi_i, \sum_{j=0}^{n_{DoFs}} v_j \phi_j\right) = l\left(\sum_{j=0}^{n_{DoFs}} v_j \phi_j\right).$$

Since a is linear in the second argument and l is also linear, it is thus sufficient to solve

$$a\left(\sum_{i=0}^{n_{DoFs}} u_i \phi_i, \phi_j\right) = l\left(\phi_j\right) \quad \forall 1 \leq j \leq n_{DoFs}.$$

The convection-diffusion problem is linear itself, thus a is also linear in the first argument and we get

$$\sum_{i=0}^{n_{DoFs}} u_i \, a\left(\phi_i, \phi_j\right) = l\left(\phi_j\right) \quad \forall 1 \leq j \leq n_{DoFs}.$$

This can also be written as a linear equation system

$$\begin{bmatrix} a\left(\phi_1, \phi_1\right) & \cdots & a\left(\phi_{n_{DoFs}}, \phi_1\right) \\ \vdots & \ddots & \vdots \\ a\left(\phi_1, \phi_{n_{DoFs}}\right) & \cdots & a\left(\phi_{n_{DoFs}}, \phi_{n_{DoFs}}\right) \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_{n_{DoFs}} \end{bmatrix} = \begin{bmatrix} l(u_1) \\ \vdots \\ l(u_{n_{DoFs}}) \end{bmatrix}.$$

To remain consistent with future chapters, we follow the naming convention

$$A_h := \begin{bmatrix} a\left(\phi_1, \phi_1\right) & \cdots & a\left(\phi_{n_{DoFs}}, \phi_1\right) \\ \vdots & \ddots & \vdots \\ a\left(\phi_1, \phi_{n_{DoFs}}\right) & \cdots & a\left(\phi_{n_{DoFs}}, \phi_{n_{DoFs}}\right) \end{bmatrix},$$

$$x_h := \begin{bmatrix} u_1 \\ \vdots \\ u_{n_{DoFs}} \end{bmatrix} \quad \text{and}$$

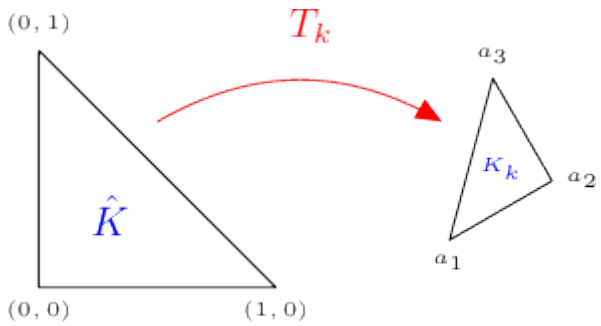
$$b_h := \begin{bmatrix} l(u_1) \\ \vdots \\ l(u_{n_{DoFs}}) \end{bmatrix}.$$

To be able to solve $A_h x_h = b_h$, we need an efficient way to compute $a\left(\phi_i, \phi_j\right)$ and $l\left(\phi_j\right)$. For that we use that $\Omega = \cup_{k=1}^{n_{DoFs}} K_k$ and we thus get

$$\begin{aligned} a(\phi_i, \phi_j) &= \int_\Omega a \nabla \phi_i \cdot \nabla \phi_j \, dx + \int_\Omega c \phi_i \cdot \phi_j \, dx \\ &= \sum_{k=1}^{n_{DoFs}} \left(\int_{K_k} a \nabla \phi_i \cdot \nabla \phi_j \, dx + \int_{K_k} c \phi_i \cdot \phi_j \, dx \right), \end{aligned}$$

similarly we get for the right hand functional

$$\begin{aligned} l(\phi_j) &= \int_\Omega f \cdot \phi_j \, dx \\ &= \sum_{k=1}^{n_{DoFs}} \left(\int_{K_k} f \cdot \phi_j \, dx \right). \end{aligned}$$



Note that many of these integrals are zero, since the basis functions ϕ_i only have support on the triangles that contain the vertex corresponding to the i.th degree of freedom. Furthermore, we use the isoparametric concept that allows us to assemble the system matrix A_h and right side b_h by once computing integrals on a reference element $\hat{K}$ and then transforming the results to the elements K_k.

Figure 2: Transformation from reference element to finite element

We now apply the transformation theorem

$$\int_{\hat{K}} g\left(T_k\left(\hat{x}\right)\right) \left|\det\left(\nabla T_k\left(\hat{x}\right)\right)\right| \, d\hat{x} = \int_{K_k} g\left(x\right) \, dx$$

to all integrals that need to be evaluated in the discrete weak form.

Then the algorithm for the assembly is given by

```
for triangle K in grid.triangles:
  for DoF φ_i in K.dofs:
    b_h[φ_i.globalIndex] += f ∫_K̂ φ̂_i |det(∇̂T_K(x̂))| dx̂
    for DoF φ_j in K.dofs:
      A_h[φ_i.globalIndex, φ_j.globalIndex] += (
        a ∫_K̂ (∇̂T_K(x̂)^{-T} ∇̂φ̂_i) · (∇̂T_K(x̂)^{-T} ∇̂φ̂_j) |det(∇̂T_K(x̂))| dx̂ +
        c ∫_K̂ φ̂_i · φ̂_j |det(∇̂T_K(x̂))| dx̂
      )
```

where $\hat{\phi}_i = \phi_i \circ T_k$ are the basis functions on the reference element.

At the end of the assembly, we need to account for the Dirichlet boundary constraints, e.g. let a constraint $u_\tau = \xi$ be given. Then we would need to make sure that $(A_h)_{\tau,j} = \delta_{\tau,j}$ for all $1 \leq j \leq m$. Here $\delta_{\tau,j}$ denotes the Kronecker delta, which is defined as

$$\delta_{\tau,j} := \begin{cases} 1 & \text{for } j = \tau \\ 0 & \text{for } j \neq \tau \end{cases}.$$

Futhermore, we would need to set $(b_h)_\tau = \xi$. Obviously these steps ensure that when solving the linear system $A_h x_h = b_h$, we get $u_\tau = \xi$. In our model problem, we only have homogeneous Dirichlet constraints. Thus, we only need to find all indices τ at the Dirichlet boundary Γ_D and apply the procedure from above with $\xi = 0$.

Overall, the Finite Element Method enabled us to transform a discrete form of the convection-diffusion equation on a given grid into a linear equation system. In the following, we will investigate how such a linear equation system can be solved iteratively.

1.4 Iterative Methods

We want to construct an iteration, where each iterate x_k^k is a better approximation to the linear equation system $A_h x_h = b_h$. To measure the quality of our solution, we monitor the defect

$$b_h - A_h x_h$$

and try to minimize it. One can try to formulate a fixed point scheme $x_h^{k+1} = g\left(x_h^k\right)$ to solve the system of equations. The goal of the fixed point scheme is to find some input x such that $g(x) = x$. In our case, we want the exact solution x_h to be a fixed point of our iteration. We observe that the defect of the exact solution is zero. Thus one might try to increment the old iterate x_h by some multiple of the defect. This is called the Richardson method. However, the Richardson method is rarely used in practice. Instead we will work with the more general fixed point scheme

$$x_h^{k+1} = x_h^k + \omega C^{-1}\left(b_h - A_h x_h^k\right)$$

where $C \in \mathbb{R}^{n_{DoFs} \times n_{DoFs}}$. Here the type of method depends on the matrix C, e.g. $C = I$ is the Richardson method. In our code, we implemented

- $C = D$ which is the ω-Jacobi method,
- $C = (D + L)$ which is the Forward Gauss-Seidel method for $\omega = 1$,
- $C = (D + U)$ which is the Backward Gauss-Seidel method for $\omega = 1$.

In these definitions, we used the decomposition $A_h = L + D + U$, where L has only nonzero entries below the diagonal (strictly lower triangular matrix), D has only nonzero entries on the diagonal (diagonal matrix) and U has only nonzero entries above the diagonal (strictly upper triangular matrix).

Note that we need to compute the inverse matrix C^{-1}. This can be easily done for ω-Jacobi, since we just invert the diagonal. It wouldn't be efficient to invert $(D + L)$ or $(D + U)$ directly. Hence we use the formula

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij} x_j^k - \sum_{j>i} a_{ij} x_j^{k-1}\right) \quad \text{for } i = 1, \dots n_{DoFs}$$

for Forward Gauss-Seidel and the formula

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij} x_j^{k-1} - \sum_{j>i} a_{ij} x_j^k\right) \quad \text{for } i = n_{DoFs}, \dots, 1$$

for Backward Gauss-Seidel. In these formulas, we used the notation $a_{ij} := (A_h)_{ij}$, $b_i := (b_h)_i$ and $x_i^k := (x_h^k)_i$.

Which of these iterative methods should be used in numerical computations? It depends! ω-Jacobi has the benefit of being fast, since it can be parallelized. Nevertheless, it needs more iterations to converge than Gauss-Seidel and one needs to choose a good value for ω before computation. Although the Gauss-Seidel methods converge in less iterations, they need longer for the computation, since the for loops need to be executed sequentially.

In the next few sections, we will show how ω-Jacobi and Gauss-Seidel can be used in the multigrid method, resulting in a fast solver for the linear equation system derived from a discrete weak form of the convection-diffusion equation.

1.5 Grid Setup

To transform the weak form of our problem into a linear equation system, we first need to discretize our domain. For that purpose, we create an initial triangulation of the domain, i.e. we divide Ω into a set of triangles. We call this triangulation the coarse grid and denote it as $\mathbb{T}_0$. The grid consists of objects of type `Node`, `Edge` and `Triangle`.

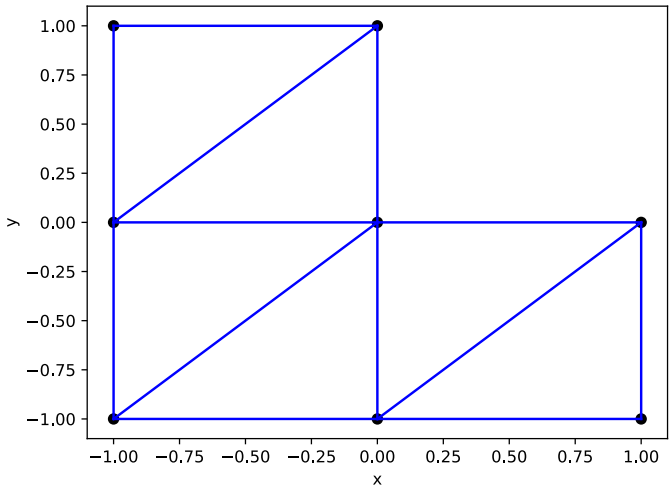


Figure 3: Coarse grid ($\mathbb{T}_0$)

For the multigrid method, we need a sequence of such grids. In this work, we restrict our analysis to uniformly refined meshes. How can we create these refined meshes? We have to loop over all triangles of the grid and then refine them.

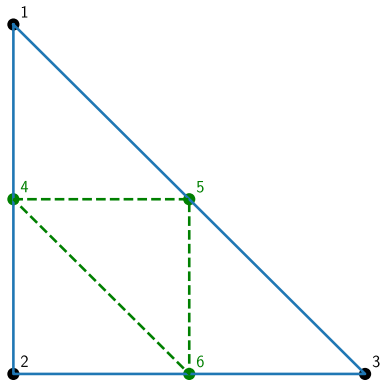


Figure 4: Refining a triangle

To refine a triangle one simply needs to bisect all of its edges and draw a new triangle out of these three new nodes. As shown in figure 4, through the refinement process a triangle is being divided into four smaller triangles. Each `Node` object needs to know its parent nodes. The parents are two end nodes of the edge that has been bisected, e.g. node 1 and node 2 are the parents of node 4. In the literature [1] these relationships are being stored in a father-son list. This is not needed in our case, due to Object Oriented Programming (OOP).

Having refined all triangles of the coarse grid, we get a new triangulation $\mathbb{T}_1$, which is called the grid on level 1. The level of a grid indicates how often we need to (globally) refine the coarse grid to construct that grid.

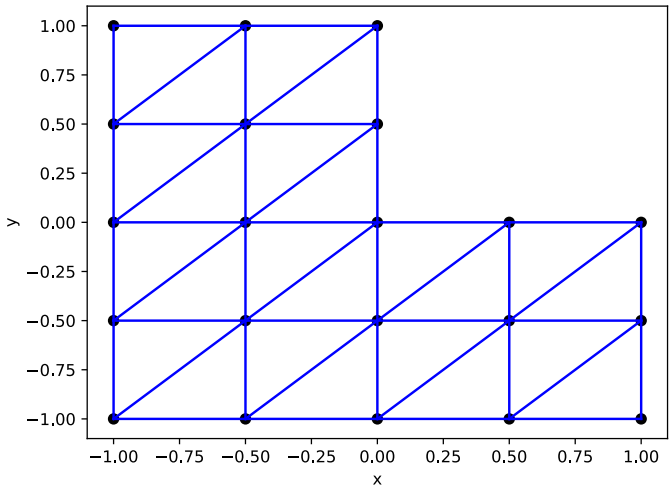


Figure 5: Grid on level 1 ($\mathbb{T}_1$)

We continue the process of refining the grid, until we end up with a grid, which has enough nodes to ensure that a sufficiently good approximation to the exact solution can be computed.

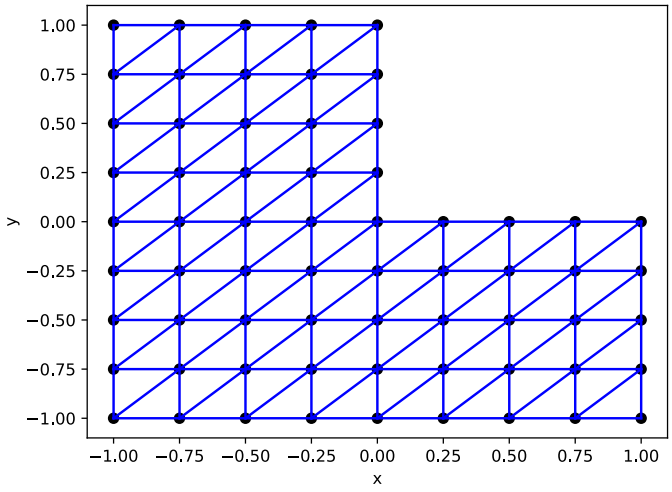


Figure 6: Grid on level 2 ($\mathbb{T}_2$)

The grid on the highest level, in this case $\mathbb{T}_2$ or more generally $\mathbb{T}_L$, is called the finest grid and will be used to assemble the system matrix.

Using the Finite Element Method, we can discretize the weak form of our PDE on each level grid with linear finite elements. For each level $0 \leq l \leq L$, we get a linear equation system

$$A_l x_l = b_l \text{ with } A_l \in \mathbb{R}^{n_l \times n_l}, x_l, b_l \in \mathbb{R}^{n_l},$$

where n_l is the number of degrees of freedom (DoFs), which in our case corresponds to the number of nodes in the grid. Note that the discrete function spaces $(V_l)_{l=0}^{L}$ from the FEM are conforming finite element spaces, i.e. $V_0 \subset V_1 \subset \cdots \subset V_L$. If this wasn't the case, the grid transfer operations, which will be introduced shortly, would need to be modified.

1.6 Two-grid algorithm

To understand the multigrid algorithm we start by looking at the case where we only have two grids $\mathbb{T}_l$ and $\mathbb{T}_{l+1}$. The mulitgrid algorithm is then only a recursive application of the two grid version.

Two-grid algorithm

Let $A_h x_h = b_h$ and $A_{2h} x_{2h} = b_{2h}$ with $A_h \in \mathbb{R}^{n \times n}$, $A_{2h} \in \mathbb{R}^{m \times m}$ and $m < n$ denote the linear equation systems from the grids $\mathbb{T}_{l+1}$ and $\mathbb{T}_l$. Let the k-th iterate x_h^k on the finer grid be given.

```
def TGM(x_h^k):
  # 1. Apply ν_1 smoothing steps of an iterative method S_1.
  x_h^{k,1} = S_1^{ν_1} x_h^k        # PRE - SMOOTHING

  # 2. Restrict defect to coarse grid.
  d_2h = I_h^{2h}(b_h - A_h x_h^{k,1})        # I_h^{2h} := restriction operator

  # 3. Coarse grid correction.
  x_h^{k,2} = x_h^{k,1} + I_{2h}^h(A_{2h}^{-1} d_2h)      # I_{2h}^h := prolongation operator

  # 4. Apply ν_2 smoothing steps of an iterative method S_2.
  x_h^{k,3} = S_2^{ν_2} x_h^{k,2}        # POST - SMOOTHING

  return x_h^{k+1} := x_h^{k,3}
```

Hint: In most cases we want the two-grid method to be a symmetric iteration. Therefore we need $\nu := \nu_1 = \nu_2$ and $S := S_1 = S_2^*$ [4], e.g. choose S_1 as forward Gauss-Seidel and S_2 as backward Gauss-Seidel. Alternatively we have also implemented the ω-Jacobi method which can be used for pre- and post-smoothing. Furthermore $A_{2h}^{-1} d_{2h}$ is not feasible to compute with a direct solver if A_{2h} is too large, which is often the case. Thus $A_{2h}^{-1} d_{2h}$ can be understood as solving the linear equation system and can be done for example by another two-grid method. This recursion then produces the multigrid algorithm.

1.7 Multigrid algorithm

Multigrid algorithm

Let $A_L x_L = b_L$ denote the problem on the finest grid and $A_l x_l = b_l$ the problems on the coarser grids for $0 \leq l \leq L-1$. Let ν denote the number of pre- and post-smoothing steps. Let the k-th iterate x_l^k on the l-th level be given.

```
def MGM(l, x_l^k, b_l):
  # 1. Apply ν smoothing steps of an iterative method S.
  x_l^{k,1} = S^ν x_l^k        # PRE - SMOOTHING

  # 2. Restrict defect to coarse grid.
  d_{l-1} = I_l^{l-1}(b_l - A_l x_l^{k,1})        # I_l^{l-1} := restriction operator

```

```
# 3. Coarse grid solution.
if l == 1:
  y_0 = A_0^{-1} d_0      # direct solver on coarsest grid
else: # l > 1
  y_{l-1} = 0
  for i in range(μ):
    y_{l-1} = MGM(l − 1, y_{l-1}, d_{l-1})

# 4. Coarse grid correction.
x_l^{k,2} = x_l^{k,1} + I_{l-1}^l y_{l-1}        # I_{l-1}^l := prolongation operator

# 5. Apply ν smoothing steps of an iterative method S.
x_l^{k,3} = S^ν x_l^{k,2}        # POST - SMOOTHING

return x_l^{k+1} := x_l^{k,3}
```

The parameter $\mu \in \mathbb{N}^+$ determines the cycle of the multigrid iteration. For $\mu = 1$ we get the V-cycle

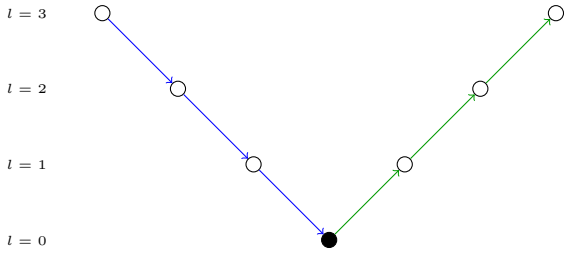


Figure 7: V-cycle

and for $\mu = 2$ we get the W-cycle.

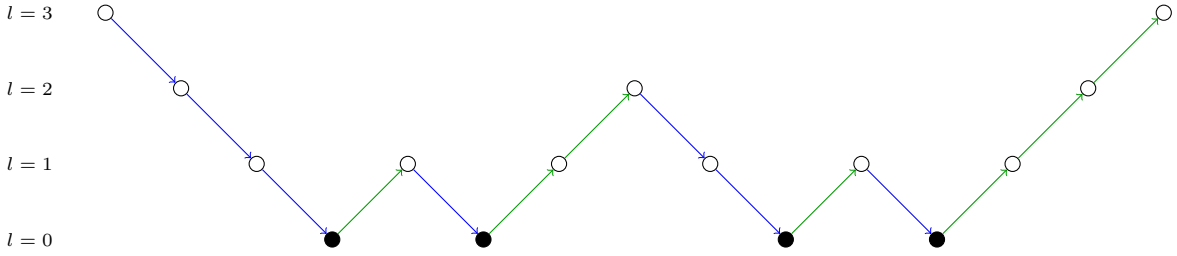


Figure 8: W-cycle

In the figures of these schemes, white circles stand for ν steps of an iterative solver, black circles represent a direct solver, blue arrows illustrate a restriction and green arrows illustrate a prolongation.

1.8 Grid transfer

As we have seen in the previous sections, the multigrid algorithm requires the ability to prolongate vectors from $\mathbb{R}^{n_l}$ to $\mathbb{R}^{n_{l-1}}$. We will only show how the grid transfer operations work for conforming finite elements. For information on how to deal with non-conforming finite element spaces, please refer to [3]. Let $\left\{\varphi_1^l, \ldots, \varphi_{n_l}^l\right\}$ and $\left\{\varphi_1^{l-1}, \ldots, \varphi_{n_{l-1}}^{l-1}\right\}$ be some given bases of V^l and V^{l-1}. Due to the conformity of the finite element spaces, $V^{l-1} \subset V^l$ holds and there

exists a matrix $I_l^{l-1} \in \mathbb{R}^{n_{l-1} \times n_l}$ such that

$$\begin{pmatrix} \varphi_1^{l-1} \\ \vdots \\ \varphi_{n_{l-1}}^{l-1} \end{pmatrix} = I_l^{l-1} \begin{pmatrix} \varphi_1^{l} \\ \vdots \\ \varphi_{n_l}^{l} \end{pmatrix}.$$

The matrix I_l^{l-1} is called the restriction matrix and its transpose $I_{l-1}^l = \left(I_l^{l-1}\right)^T$ is called the prolongation matrix. These matrices are dependent on the finite elements that are being used and on the way that the grids have been refined. They have only very few non-zero entries and thus are stored as sparse matrices. Furthermore, they also have a significant impact on the rate of convergence of the multigrid algorithm [3]]. Additionally, for linear partial differential equations the identity

$$A_{l-1} = I_{l-1}^l A_l I_l^{l-1}$$

is fulfilled. Before taking a look at the actual implementation in our code, it is helpful to a see how the the grid transfer works for one dimensional linear finite elements.

Grid transfer in 1D

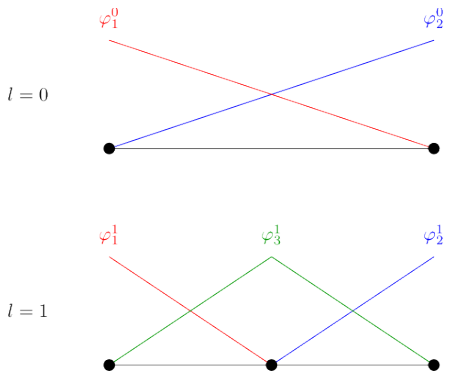


Figure 9: Basis functions on the first two levels

It holds

$$\varphi_1^0 = \varphi_1^1 + \frac{1}{2}\varphi_3^1,$$
$$\varphi_2^0 = \varphi_2^1 + \frac{1}{2}\varphi_3^1.$$

Consequently the restriction matrix reads

$$I_1^0 = \begin{bmatrix} 1 & & \frac{1}{2} \\ & 1 & \frac{1}{2} \end{bmatrix}.$$

We decided to follow these rules to create the interpolation matrix I_{l-1}^l:

- if the i.th node already exists on level $l-1$, then $\left(I_{l-1}^l\right)_{i,i} = 1$
- else get the indices of the parents of the i.th node, then $\left(I_{l-1}^l\right)_{i,\text{parent}_1} = \frac{1}{2}$ and $\left(I_{l-1}^l\right)_{i,\text{parent}_2} = \frac{1}{2}$

The rest of this matrix is filled with zeros. Doing this for $l = 1$ yields the prolongation matrix

$$I_0^1 = \begin{bmatrix}
1 & & & & & & & & \\
& 1 & & & & & & & \\
& & 1 & & & & & & \\
& & & 1 & & & & & \\
& & & & 1 & & & & \\
& & & & & 1 & & & \\
& & & & & & 1 & & \\
& & & & & & & 1 & \\
\frac{1}{2} & \frac{1}{2} & & & & & & & \\
& \frac{1}{2} & \frac{1}{2} & & & & & & \\
\frac{1}{2} & & & \frac{1}{2} & & & & & \\
& \frac{1}{2} & & & \frac{1}{2} & & & & \\
& & & \frac{1}{2} & \frac{1}{2} & & & & \\
& & \frac{1}{2} & & & \frac{1}{2} & & & \\
& & & & \frac{1}{2} & \frac{1}{2} & & & \\
& & & \frac{1}{2} & & & \frac{1}{2} & & \\
& & & & \frac{1}{2} & & & \frac{1}{2} & \\
& & & & & & \frac{1}{2} & \frac{1}{2} & \\
\frac{1}{2} & & & & \frac{1}{2} & & & & \\
& & & \frac{1}{2} & & & & \frac{1}{2} & \\
& \frac{1}{2} & & & & \frac{1}{2} & & &
\end{bmatrix}.$$

For our kind of prolongation matrix, the restriction matrix is given by

$$I_l^{l-1} = \frac{1}{4} \left(I_{l-1}^l \right)^T.$$

At this point, we should also mention how the boundary conditions can be applied in the multigrid algorithm. The start vector of this method should be the zero vector on which the Dirichlet boundary conditions should be applied. Note that Dirichlet boundary conditions need to be applied to the output vectors of the prolongation and restriction, but here all Dirichlet boundary conditions are set to be homogeneous.

1.9 References

1. Sven Beuchler. *Lecture notes in 'Multigrid and domain decomposition.'* April 2020.
2. Thomas Wick. *Numerical Methods for Partial Differential Equations.* 2020. URL: https://doi.org/10.15488/9248.
3. Dietrich Braess. *Finite Elemente.* Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-34797-9. URL: https://doi.org/10.1007%2F978-3-642-34797-9.
4. Chao Chen. "Geometric multigrid for eddy current problems". PhD thesis. 2012.
5. Julian Roth. "Geometric Multigrid Methods for Maxwell's Equations". Bachelor thesis. 2020.
6. Thomas Richter and Thomas Wick. Einführung in die numerische Mathematik - Begriffe, Konzepte und zahlreiche Anwendungsbeispiele. Springer, 2017.

CHAPTER

TWO

RESULTS OF MULTIGRID FOR CONVECTION DIFFUSION

2.1 Imports

[1]:
```
from main import inputParameters, runDemo
import logging
logging.basicConfig(level=logging.INFO , format='[%(asctime)s] - [%(levelname)s] -
↪%(message)s')
```

2.2 Set parameters

[2]:
```
inputParameters()
```

```
==================
* DEFAULT VALUES *
==================
LEVELS               = 5
SHOW_GRIDS           = False
DEGREE               = 1
CYCLE                = W
MAX_ITER             = 100
SMOOTHING_STEPS      = 2
SMOOTHER             = Jacobi
OMEGA                = 0.8

Would you like to change some values? (Y/N): Y
Number of MG levels (default: 5) = 10
Plot grids (default: False) =
Degree of FE (default: 1) =
Multigrid cycle (default: W; supported: 'V','W') =
Maximum number of MG iterations (default: 100) =
Number of smoothing steps (default: 2) =
Smoother type (default: Jacobi; supported: 'Jacobi','GaussSeidel') =
Relaxation parameter omega (default: 0.8) =

==================
* CUSTOM VALUES *
==================
LEVELS               = 10
SHOW_GRIDS           = False
DEGREE               = 1
CYCLE                = W
```

(continues on next page)

(continued from previous page)

```
MAX_ITER             = 100
SMOOTHING_STEPS      = 2
SMOOTHER             = Jacobi
OMEGA                = 0.8
```

2.3 Demo

Now that the user has inserted the parameters of the multigrid algorithm, we start by solving the two-grid method, refine the grid, solve the three-grid method, etc.

What we would expect from the theory is that the number of mulitgrid iterations needed should be constant and thus independent of the number of levels. This property is approximately fulfilled in our simulation. Furthermore, the total computation time should depend linearly on the number of DoFs. This can't be observed here, since for us the assembly of the level matrices is a big bottleneck and consumes the most time. When taking a closer look at the time stamps of the logs and comparing that to where the program currently is in our code, one might notice that after the assembly the time for the actual process of solving the linear equation system approximately doubles when we add a new level. The number of DoFs quadruples with every refinement. Thus the time of the multigrid solver linearly depends on the number of degrees of freedom.

[3]:
```
runDemo()
```

```
[2020-06-17 14:32:55,887] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:55,888] - [INFO] - +    MULTIGRID (LEVELS =  2)             +
[2020-06-17 14:32:55,888] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:56,049] - [INFO] - Number of DoFs: 21 (by level: 8,21)
[2020-06-17 14:32:56,094] - [INFO] - Defect:         5.43e-13
[2020-06-17 14:32:56,094] - [INFO] - GMG iterations: 14
[2020-06-17 14:32:56,095] - [INFO] - Total time:     209 ms
[2020-06-17 14:32:56,096] - [INFO] -
[2020-06-17 14:32:56,096] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:56,097] - [INFO] - +    MULTIGRID (LEVELS =  3)             +
[2020-06-17 14:32:56,097] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:56,262] - [INFO] - Number of DoFs: 65 (by level: 8,21,65)
[2020-06-17 14:32:56,347] - [INFO] - Defect:         4.54e-13
[2020-06-17 14:32:56,347] - [INFO] - GMG iterations: 15
[2020-06-17 14:32:56,348] - [INFO] - Total time:     251 ms
[2020-06-17 14:32:56,348] - [INFO] -
[2020-06-17 14:32:56,349] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:56,349] - [INFO] - +    MULTIGRID (LEVELS =  4)             +
[2020-06-17 14:32:56,349] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:56,614] - [INFO] - Number of DoFs: 225 (by level: 8,21,65,225)
[2020-06-17 14:32:56,789] - [INFO] - Defect:         8.66e-13
[2020-06-17 14:32:56,790] - [INFO] - GMG iterations: 14
[2020-06-17 14:32:56,792] - [INFO] - Total time:     443 ms
[2020-06-17 14:32:56,792] - [INFO] -
[2020-06-17 14:32:56,793] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:56,793] - [INFO] - +    MULTIGRID (LEVELS =  5)             +
[2020-06-17 14:32:56,794] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:57,189] - [INFO] - Number of DoFs: 833 (by level: 8,21,65,225,833)
[2020-06-17 14:32:57,531] - [INFO] - Defect:         6.6e-13
[2020-06-17 14:32:57,532] - [INFO] - GMG iterations: 14
[2020-06-17 14:32:57,536] - [INFO] - Total time:     744 ms
[2020-06-17 14:32:57,537] - [INFO] -
```

(continues on next page)

(continued from previous page)

```
[2020-06-17 14:32:57,537] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:57,538] - [INFO] - +    MULTIGRID (LEVELS =  6)              +
[2020-06-17 14:32:57,538] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:58,253] - [INFO] - Number of DoFs: 3201 (by level: 8,21,65,225,833,
↪3201)
[2020-06-17 14:32:58,953] - [INFO] - Defect:         6.57e-13
[2020-06-17 14:32:58,954] - [INFO] - GMG iterations: 13
[2020-06-17 14:32:58,967] - [INFO] - Total time:     1 s 430 ms
[2020-06-17 14:32:58,968] - [INFO] -
[2020-06-17 14:32:58,969] - [INFO] - +----------------------------------------+
[2020-06-17 14:32:58,969] - [INFO] - +    MULTIGRID (LEVELS =  7)              +
[2020-06-17 14:32:58,970] - [INFO] - +----------------------------------------+
[2020-06-17 14:33:01,179] - [INFO] - Number of DoFs: 12545 (by level: 8,21,65,225,833,
↪3201,12545)
[2020-06-17 14:33:02,991] - [INFO] - Defect:         4.27e-13
[2020-06-17 14:33:02,992] - [INFO] - GMG iterations: 13
[2020-06-17 14:33:03,068] - [INFO] - Total time:     4 s 100 ms
[2020-06-17 14:33:03,069] - [INFO] -
[2020-06-17 14:33:03,069] - [INFO] - +----------------------------------------+
[2020-06-17 14:33:03,070] - [INFO] - +    MULTIGRID (LEVELS =  8)              +
[2020-06-17 14:33:03,070] - [INFO] - +----------------------------------------+
[2020-06-17 14:33:11,378] - [INFO] - Number of DoFs: 49665 (by level: 8,21,65,225,833,
↪3201,12545,49665)
[2020-06-17 14:33:15,631] - [INFO] - Defect:         9.59e-13
[2020-06-17 14:33:15,632] - [INFO] - GMG iterations: 12
[2020-06-17 14:33:15,873] - [INFO] - Total time:     12 s 804 ms
[2020-06-17 14:33:15,874] - [INFO] -
[2020-06-17 14:33:15,874] - [INFO] - +----------------------------------------+
[2020-06-17 14:33:15,875] - [INFO] - +    MULTIGRID (LEVELS =  9)              +
[2020-06-17 14:33:15,875] - [INFO] - +----------------------------------------+
[2020-06-17 14:33:49,566] - [INFO] - Number of DoFs: 197633 (by level: 8,21,65,225,
↪833,3201,12545,49665,197633)
[2020-06-17 14:34:01,356] - [INFO] - Defect:         4.53e-13
[2020-06-17 14:34:01,356] - [INFO] - GMG iterations: 12
[2020-06-17 14:34:02,231] - [INFO] - Total time:     46 s 356 ms
[2020-06-17 14:34:02,232] - [INFO] -
[2020-06-17 14:34:02,232] - [INFO] - +----------------------------------------+
[2020-06-17 14:34:02,232] - [INFO] - +    MULTIGRID (LEVELS = 10)              +
[2020-06-17 14:34:02,233] - [INFO] - +----------------------------------------+
[2020-06-17 14:36:10,700] - [INFO] - Number of DoFs: 788481 (by level: 8,21,65,225,
↪833,3201,12545,49665,197633,788481)
[2020-06-17 14:36:45,769] - [INFO] - Defect:         8.76e-13
[2020-06-17 14:36:45,770] - [INFO] - GMG iterations: 11
[2020-06-17 14:36:49,207] - [INFO] - Total time:     2 min 46 s 974 ms
[2020-06-17 14:36:49,207] - [INFO] -
```

CHAPTER

THREE

LICENSE

MIT License

Copyright (c) 2020 Julian Roth, Max Schröder

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

FOUR

HELP

If you have any questions, feel free to contact us via mathmerizing@gmail.com.