DELFT UNIVERSITY OF TECHNOLOGY

REPORT 10-19

Fast Iterative Methods for Discontinuous Galerkin Discretizations for Elliptic PDEs

P. van Slingerland, and C. Vuik

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2010

Contents

1 Discontinuous Galerkin in One Dimension (Research Area) 3
- 1.1 Notation & Preliminaries . . . 3
- 1.2 Interior Penalty (IP) methods . . . 4
- 1.3 DG Methods in the Unified Framework . . . 4
- 1.4 Linear System . . . 5
- 1.5 Numerical example . . . 6
- 1.6 Conclusion . . . 6

2 Discontinuous Galerkin in Two Dimensions (Research Area) 9
- 2.1 Notation . . . 9
- 2.2 DG Methods in the Unified Framework . . . 10
- 2.3 Linear System . . . 10
- 2.4 Numerical example . . . 12
- 2.5 Conclusion . . . 12

3 Solution Techniques for Linear Systems (Research Framework) 15
- 3.1 Linear system . . . 15
- 3.2 Conjugate Gradient method . . . 16
- 3.3 Preconditioning . . . 18
- 3.4 Deflation . . . 19
- 3.5 Conclusion . . . 20

4 Existing Preconditioners (Literature Overview) 21
- 4.1 h-Multigrid . . . 21
- 4.2 p-Multigrid . . . 23
- 4.3 Schwarz Domain Decomposition . . . 24
- 4.4 Space Decomposition . . . 26
- 4.5 Conclusion . . . 28

5 BILU Preconditioning (Research Direction) 29
- 5.1 Exact Block LU-decomposition for Block Tridiagonal matrices . . . 29
- 5.2 A Recursive BILU Preconditioner for Block Triangular Matrices . . . 31
- 5.3 A More General BILU Framework . . . 32
- 5.4 Examples . . . 34
- 5.5 Numerical Example . . . 36
- 5.6 Conclusion . . . 36

6 Conclusion & Research questions 39

A Derivation of DG Methods 41
- A.1 Preliminaries . . . 41
- A.2 Interior Penalty Methods . . . 41
- A.3 DG Methods in the Unified Framework . . . 42

B Computing the Coefficient Matrix for an IP scheme **45**
B.1 One-dimensional case 45
B.2 Two-dimensional case 47

C BILU in Detail **53**

Introduction

Discontinuous Galerkin (DG) finite element methods for elliptic problems approximate the solution in the form of piecewise polynomials of degree p. The main advantages of these methods are the flexibility in handling non-matching grids and in designing hp-refinement strategies. An important drawback is that the resulting linear systems are usually large (due to the large number of degrees of freedom), and ill-conditioned.

Therefore, efficient iterative algorithms are required to minimize the computational costs and increase the practical applicability of DG methods. The main goal of this report is to obtain an overview of the current literature regarding these methods for elliptic problems. Based on this overview, research questions will be formulated that indicate how we will seek to improve on the existing iterative methods in the near future.

The outline of this report is as follows. First, DG methods are discussed for one- and two-dimensional elliptic problems in Chapter 1 and Chapter 2 respectively. These methods require the solution to large sparse linear systems, for which efficient solution strategies, such as preconditioning and deflation, are considered in Chapter 3. Chapter 4 provides an overview of existing preconditioners in the context of DG methods. It appears that preconditioners based on a Block Incomplete LU-decomposition (BILU), which have been rather successful for finite difference methods, have received little attention so far in this field. Therefore, existing BILU algorithms are studied in Chapter 5 as a starting point of this research. Finally, a conclusion is given in Chapter 6, together with the resulting research questions.

Chapter 1

Discontinuous Galerkin in One Dimension (Research Area)

This research focuses on linear systems resulting from DG discretisations of elliptic problems. Therefore, this type of discretisation will be discussed first. This chapter considers the one-dimensional case. The two dimensional case is considered in Chapter 2. The main idea is to assume that the solution is a polynomial of degree p within each mesh element. Furthermore, the solution is allowed to be discontinuous at the element boundaries.

The outline of this chapter is as follows. First, Section 1.1 introduces the required notation regarding the mesh and the trace operators for jumps and averages of the discontinuous solution at the element boundaries. After that, Interior Penalty (IP) methods are discussed in Section 1.2. These methods are often studied in current literature on efficient solution techniques, as we will see in Chapter 4 later on. Basically, IP methods penalize inter-element jumps to ensure stability. Besides IP methods, a second class of DG methods exists which are formulated in terms of numerical fluxes, inspired by finite volume techniques for hyperbolic problems. These two families have been developed independently of each other and were presented in a unified framework in [ABCM02]. This unified framework is discussed in Section 1.3. After that, Section 1.4 demonstrates how a DG approximation can be computed in terms of the solution to a linear system. A numerical example that illustrates the performance of DG methods is provided in Section 1.5. Finally, a conclusion is given in Section 1.6.

1.1 Notation & Preliminaries

Consider the one-dimensional Poisson equation:

$$-u'' = f(x), \tag{1.1}$$

on the interval Ω, together with homogeneous Dirichlet boundary conditions. Additionally, consider a mesh $\{[x_{i-1}, x_i]\}_{i=1,...,N}$ with uniform element diameter h. Furthermore, let V denote the test space which contains each (test) function v that is a polynomial of degree p or lower within each mesh element, and that may be discontinuous at the element boundaries. Derivatives of test functions should therefore be interpreted piecewise, existing within the element interiors only.

For any test function $v \in V$, let the function v_i denote the continuous representation of $v|_{[x_{i-1},x_i]}$. More generally, for each piecewise continuous function v and for each element $[x_{i-1}, x_i]$ with $i = 1, ..., N$, define $v_i : [x_{i-1}, x_i] \to \mathbb{R}$ such that $v_i = v$ in the element interior (x_{i-1}, x_i), and v_i is continuous at the boundaries x_{i-1} and x_i. Using this definition, introduce the following so-called trace operators for jumps and averages at each interior element boundary x_i with $i = 1, ..., N-1$:

$$[v]_i := v_i(x_i) - v_{i+1}(x_i), \qquad \{v\}_i := \frac{v_i(x_i) + v_{i+1}(x_i)}{2}. \tag{1.2}$$

Similarly, at the domain boundary, define:

$$[v]_0 := -v_1(x_0), \qquad [v]_N := v_N(x_N), \qquad \{v\}_0 := v_1(x_0), \qquad \{v\}_N := v_N(x_N). \tag{1.3}$$

1.2 Interior Penalty (IP) methods

Now that the required notation is introduced, we can formulate a family of classical Interior Penalty (IP) methods: an IP approximation $u_h \in V$ for the exact solution u for model problem (1.1) satisfies (cf. Appendix A.2 for a derivation):

$$B(u_h, v) = \int_\Omega fv, \qquad \text{for all test functions } v \in V, \tag{1.4}$$

where the bilinear form is defined as:

$$B(u_h, v) := \int_\Omega u_h' v' + \sum_{i=0}^{N} \left(-\{u_h'\}_i [v]_i + \epsilon [u_h]_i \{v'\}_i + \frac{\eta_0}{h} [u_h]_i [v]_i \right). \tag{1.5}$$

Depending on the parameters ϵ and η_0, different types of interior penalty methods are obtained (cf. [Riv08, p. 6] and references therein). Common types are:

ϵ	Method
-1	Symmetric Interior Penalty Galerkin (SIPG)
1	Non-symmetric Interior Penalty Galerkin (NIPG)
0	Incomplete Interior Penalty Galerkin (IIPG)

If both ϵ and η_0 are zero, the method is not convergent, and existence and uniqueness of u_h cannot be shown. This research will mainly focus on the SIPG method. The accuracy of this method is $O(h^{p+1})$ provided that the penalty parameter η_0 is sufficiently large [Riv08, p. 12].

1.3 DG Methods in the Unified Framework

Besides the family of IP methods discussed in the previous section, there is also a large family of DG methods inspired by finite volume techniques for hyperbolic problems, which are formulated in terms of numerical fluxes. These two families have been developed independently of each other and were presented in a unified framework by Arnold, Brezzi, Cockburn, and Marini [ABCM02], by showing that the IP methods can be "obtained as special cases of the second family simply by choosing the proper numerical fluxes".

In this unified framework, a DG approximation $u_h \in V$ for the exact solution u for model problem (1.1) satisfies (cf. Appendix A.3 for derivation):

$$B(u_h, v) = \int_\Omega fv, \qquad \text{for all test functions } v \in V, \tag{1.6}$$

where the bilinear form is defined as:

$$B(u_h, v) := \int_\Omega u_h' v' + \sum_{i=0}^{N} \left([\hat{u} - u_h]_i \{v'\}_i - \{\hat{\sigma}\}_i [v]_i \right) + \sum_{i=1}^{N-1} \left(\{\hat{u} - u_h\}_i [v']_i - [\hat{\sigma}]_i \{v\}_i \right). \tag{1.7}$$

Here, $\hat{u}$ and $\hat{\sigma}$ are numerical fluxes that approximate u and $\sigma := u'$ at the element boundaries. Depending on the choice of these numerical fluxes, different types of DG methods are obtained [ABCM02, Section 3.4].

For example, an IP method as described in Section 1.2 is obtained if the flux functions are chosen such that, for all $i = 0, .., N$:

$$\hat{u}_i(x_i) = \{u_h\}_i + \frac{1+\epsilon}{2}[u_h]_i, \qquad \hat{u}_{i+1}(x_i) = \{u_h\}_i - \frac{1+\epsilon}{2}[u_h]_i,$$
$$\hat{\sigma}_i(x_i) = \{u_h'\}_i - \frac{\eta_0}{h}[u_h]_i, \qquad \hat{\sigma}_{i+1}(x_i) = \{u_h'\}_i - \frac{\eta_0}{h}[u_h]_i,$$

so that:

$$[\hat{u} - u_h]_i = \epsilon [u_h]_i, \qquad \{\hat{\sigma}\}_i = \{u_h'\}_i - \frac{\eta_0}{h}[u_h]_i,$$
$$\{\hat{u} - u_h\}_i = 0, \qquad [\hat{\sigma}]_i = 0.$$

As a consequence, for this choice of numerical fluxes, the bilinear form (1.7) reduces to (1.5).

1.4 Linear System

This section discusses how a DG solution, defined by (1.6) and (1.7) in the previous section, can be computed in terms of the solution to a linear system. The main idea is to first construct a basis for the test space V. Here, we will use monomial basis functions, but other options exist as well. After that, the DG solution u_h is written as a linear combination of these basis functions, of which the coefficients are now the unknowns to be solved for. Then, this expression for u_h is substituted into (1.6), and the result should be satisfied for any basis function v. More specifically, the procedure reads as follows:

Consider a monomial of degree $k \leq p$ that is translated from the interval $[-1, 1]^2$ to the element $[x_{i-1}, x_i]$ with cell center $x_{i-\frac{1}{2}}$:

$$\phi_{k+1}^{(i)}(x) = \left(\frac{x - x_{i-\frac{1}{2}}}{\frac{1}{2}h}\right)^k, \tag{1.8}$$

for all $k = 0, ..., p$, for all $x \in [x_{i-1}, x_i]$, and for all $i = 1, ..., N$. For $x \notin [x_{i-1}, x_i]$, the function is set to zero. Next, write the DG solution $u_h \in V$ as a linear combination of these monomial test functions of degree p and lower:

$$u_h(x) = \sum_{i=1}^{N} \sum_{k=1}^{M:=p+1} \alpha_k^{(i)} \phi_k^{(i)}(x), \tag{1.9}$$

Substitute this expression for u_h, and the monomial basis functions $\phi_\ell^{(j)}$ for v into the DG formulation (1.6):

$$\sum_{i=1}^{N} \sum_{k=1}^{M} \alpha_k^{(i)} B(\phi_k^{(i)}, \phi_\ell^{(j)}) = \int_\Omega f \phi_\ell^{(j)}, \tag{1.10}$$

for all $j = 1, .., N$ and for all $\ell = 1, .., M$. The unknowns $\alpha_k^{(i)}$ determine the final DG solution as defined in (1.9).

To obtain the unknowns $\alpha_k^{(i)}$, the system (1.10) can be rewritten as a linear system $Ay = b$ of the form:

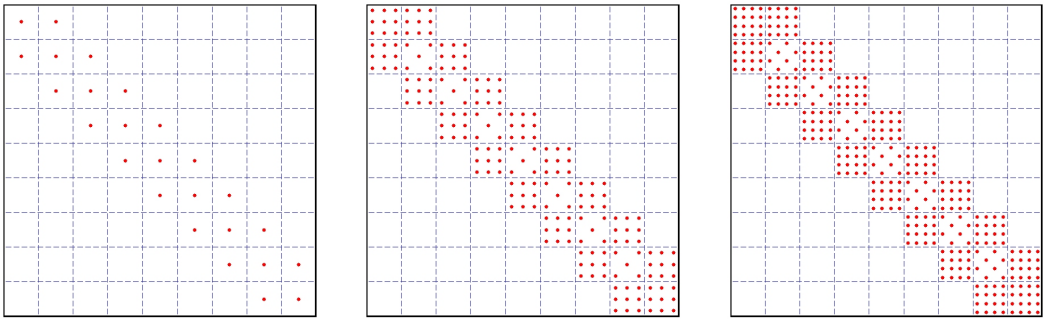
$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & A_{22} & & \vdots \\ \vdots & & \ddots & \\ A_{N1} & \dots & & A_{NN} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}, \tag{1.11}$$

where the blocks all have dimension M, and where, for all $i, j = 1, ..., N$:

$$A_{ji} = \begin{bmatrix} B(\phi_1^{(i)}, \phi_1^{(j)}) & B(\phi_2^{(i)}, \phi_1^{(j)}) & \dots & B(\phi_M^{(i)}, \phi_1^{(j)}) \\ B(\phi_1^{(i)}, \phi_2^{(j)}) & B(\phi_2^{(i)}, \phi_2^{(j)}) & & \vdots \\ \vdots & & \ddots & \\ B(\phi_1^{(i)}, \phi_M^{(j)}) & \dots & & B(\phi_M^{(i)}, \phi_M^{(j)}) \end{bmatrix}, \quad y_i = \begin{bmatrix} \alpha_1^{(i)} \\ \alpha_2^{(i)} \\ \vdots \\ \alpha_M^{(i)} \end{bmatrix}, \quad b_j = \begin{bmatrix} \int_\Omega f\phi_1^{(j)} \\ \int_\Omega f\phi_2^{(j)} \\ \vdots \\ \int_\Omega f\phi_M^{(j)} \end{bmatrix}. \tag{1.12}$$

More details on the computation of the coefficient matrix A can be found in Appendix B.1.

For $p = 0$, the matrix A is equivalent to a matrix resulting from a central finite difference method (aside from a constant that can be included in the right hand side). This means that each off-diagonal element A_{ji} is zero if mesh element i and j are not adjacent. For $p > 0$, the matrix structure is similar, except that the matrix is now a *block* matrix, in which each off-diagonal *block* A_{ji} is zero if mesh element i and j are not adjacent. This is illustrated in Figure 1.1.



(a) $p = 0$ (b) $p = 1$ (c) $p = 2$

Figure 1.1: SIPG method for a 1D Poisson problem: for a mesh with $N = 9$ elements, the coefficient matrix A consist of $N \times N$ blocks, where each block is of size $p + 1 \times p + 1$

1.5 Numerical example

Now that the computational details are discussed, we will consider a numerical example to illustrate the performance of the SIPG method.

Consider the one-dimensional Poisson equation:

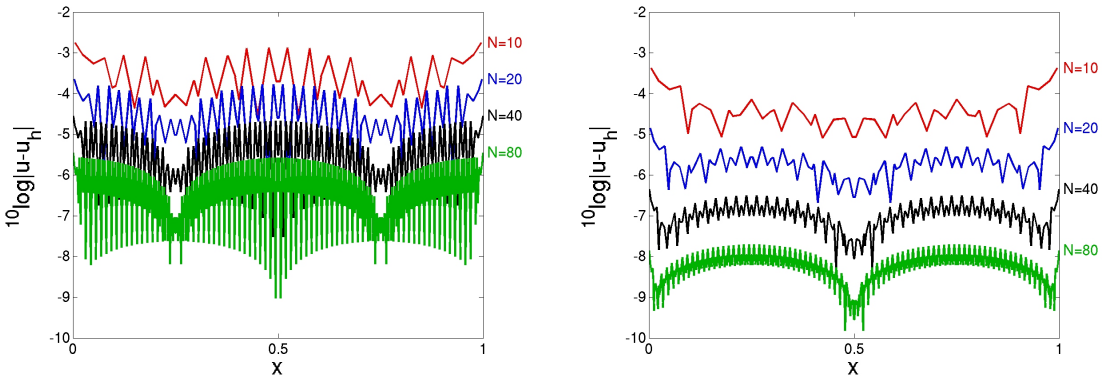
$$-u''(x) = (2\pi)^2 \sin(2\pi x),$$

on the interval $[0, 1]$ together with homogeneous Dirichlet boundary conditions. For this particular problem, the exact solution reads $u(x) = \sin(2\pi x)$.

Figure 1.2 displays the outcome of the SIPG method (cf. Section 1.2) for this problem, using polynomial degree $p = 2, 3$ and penalty parameter $\eta_0 = 10$. Observe that the errors decrease rapidly as the number of elements increases. Table 1.1 illustrates that the L^2-errors are $O(h^{p+1})$ for this problem, as predicted by theory.

1.6 Conclusion

This chapter considers DG methods elliptic problems, which form the main subject of this research, in one dimension. The main idea is to assume that the solution is a polynomial of degree p within each mesh element. Furthermore, the solution is allowed to be discontinuous at the element



(a) $p = 2$ (b) $p = 3$

Figure 1.2: SIPG method (with $\eta_0 = 10$) for a 1D Poisson problem: the errors decrease rapidly as the number of elements increases.

Table 1.1: SIPG method (with $\eta_0 = 10$) for a 1D Poisson problem: the L^2-errors are $O(h^{p+1})$ for this problem.

mesh	p=1		p=2		p=3	
	error	order	error	order	error	order
10	2.47846e-02	-	6.80413e-04	-	9.68405e-05	-
20	6.32866e-03	1.96947	8.37268e-05	3.02265	3.10837e-06	4.96138
40	1.59013e-03	1.99275	1.04326e-05	3.00458	1.50392e-07	4.36936
80	3.98017e-04	1.99825	1.30359e-06	3.00054	8.99025e-09	4.06422
160	9.95340e-05	1.99957	1.62969e-07	2.99983	5.58708e-10	4.00819

boundaries. A DG approximation can be computed in terms of the solution to a linear system. For $p = 0$, the corresponding coefficient matrix is equivalent to a matrix resulting from a central finite difference method. For $p > 0$, the matrix structure is similar, except that the matrix is now a *block* matrix. In general, the matrix is an $N \times N$ block matrix, where N is the number of mesh elements, and each block in turn is an $M \times M$ matrix, where $M = p + 1$ is the dimension of the polynomial test space (within one mesh element). As a consequence, the size of the matrix grows rapidly with both the number of elements and the polynomial degree. Since many practical applications are not formulated as a one-dimensional problem, the next chapter will extend the ideas in this chapter to the two-dimensional case.

Chapter 2

Discontinuous Galerkin in Two Dimensions (Research Area)

This chapter extends the DG formulations of the previous chapter to two-dimensional elliptic problems. Similar to the one-dimensional case, the main idea is to assume that the solution is a polynomial of degree p within each mesh element. Furthermore, the solution is allowed to be discontinuous at the element boundaries. The main difference is that two-dimensional problems usually require the solution of much more unknowns.

The outline of this chapter is as follows. First, Section 2.1 introduces the required notation regarding the mesh and the trace operators for jumps and averages of the discontinuous solution at the element boundaries. After that, Section 2.2 formulates DG methods in the unified framework in [ABCM02]. As before, the DG approximation can be computed in terms of the solution to a linear system, which is discussed in Section 2.3. A numerical example that illustrates the performance of DG methods is provided in Section 2.4. Finally, a conclusion is given in Section 2.5.

2.1 Notation

Consider the two-dimensional Poisson equation:

$$-\Delta u = f, \tag{2.1}$$

on the domain Ω, together with homogeneous Dirichlet boundary conditions. Furthermore, consider a mesh $\{K_i\}_{i=1,...,N}$ in which each element K_i is a compact polygonal and has an outward normal $\boldsymbol{n}_i$. Let E° denote the collection of all interior edges $e = \partial K_i \cap \partial K_j$ in the mesh shared by two distinct elements. Furthermore, let E^∂ denote the collection of all boundary edges $e = \partial K_i \cap \partial\Omega$ shared by an element and the domain boundary. Finally, let $E := E^\circ \cup E^\partial$ denote the collection of all edges.

Next, consider a test space V that contains each function v that is a polynomial of degree p or lower within each mesh element, and that may be discontinuous at the element boundaries. Derivatives of test functions should therefore be interpreted piecewise, existing within the element interiors only.

For any test function $v \in V$, let the function v_i denote the continuous representation of $v|_{K_i}$. More generally, for any piecewise continuous function $v \in L^2(\Omega)$ and for any $i = 1, ..., N$, define $v_i : K_i \to \mathbb{R}$ such that $v_i = v$ in the interior of K_i, and v_i is continuous at the boundary ∂K_i. Using this definition, introduce the following trace operators for jumps and averages at each interior element boundary $\partial K_i \cap \partial K_j \in E^\circ$:

$$[v] := v_i \boldsymbol{n}_i + v_j \boldsymbol{n}_j, \qquad \{v\} := \frac{v_i + v_j}{2}. \tag{2.2}$$

Observe that $[v]$ is a vector, while v is a scalar. Analogously, we define for a vector-valued piecewise continuous function $\boldsymbol{\tau} \in L^2(\Omega) \times L^2(\Omega)$:

$$[\boldsymbol{\tau}] := \boldsymbol{\tau}_i \cdot \boldsymbol{n}_i + \boldsymbol{\tau}_j \cdot \boldsymbol{n}_j, \qquad \{\boldsymbol{\tau}\} := \frac{1}{2}(\boldsymbol{\tau}_i + \boldsymbol{\tau}_j).$$

Observe that $[\boldsymbol{\tau}]$ is a scalar, while $\boldsymbol{\tau}$ is a vector. Similarly, at the domain boundary, we define at each element boundary $\partial K_i \cap \partial\Omega \in E^\partial$:

$$[v] := v_i \boldsymbol{n}_i, \qquad \{v\} := v_i, \qquad [\boldsymbol{\tau}] := \boldsymbol{\tau}_i \cdot \boldsymbol{n}_i, \qquad \{\boldsymbol{\tau}\} := \boldsymbol{\tau}_i, \tag{2.3}$$

2.2 DG Methods in the Unified Framework

Now that the required notation is introduced, we can formulate DG methods for two-dimensional problems. In the unified framework in [ABCM02], a DG approximation $u_h \in V$ for the exact solution u for model problem (2.1) satisfies (the derivation [ABCM02, Section 3.2] is similar to the derivation for the one-dimensional case provided in Appendix A.3):

$$B(u_h, v) = \int_\Omega fv, \qquad \text{for all test functions } v \in V, \tag{2.4}$$

where the bilinear form is defined as:

$$\begin{aligned} B(u_h, v) = &\int_\Omega \nabla u_h \cdot \nabla v + \sum_{e \in E} \int_e \left([\hat{u} - u_h] \cdot \{\nabla v\} - \{\hat{\boldsymbol{\sigma}}\} \cdot [v] \right) \\ &+ \sum_{e \in E^\circ} \int_e \left(\{\hat{u} - u_h\} [\nabla v] - [\hat{\boldsymbol{\sigma}}] \{v\} \right). \end{aligned} \tag{2.5}$$

Here, $\hat{u}$ and $\hat{\boldsymbol{\sigma}}$ are numerical fluxes that approximate u and $\boldsymbol{\sigma} := \nabla u$ at the element boundaries. Depending on the choice of these numerical fluxes, different types of DG methods are obtained [ABCM02, Section 3.4].

For example, similar to the one-dimensional case (Section 1.3), an IP method is obtained if the flux functions $\hat{u}$ and $\hat{\boldsymbol{\sigma}}$ are chosen such that on the edge $e \in E$ of K_i (for all $i = 1, ..., N$):

$$\hat{u}_i = \{u_h\} + \frac{1+\epsilon}{2} [u_h] \cdot \boldsymbol{n}_i, \qquad \hat{\boldsymbol{\sigma}}_i = \{\nabla u_h\} - \frac{\eta_0}{|e|} [u_h].$$

where $|e|$ denotes the length of the edge e. As a consequence:

$$\begin{aligned} [\hat{u} - u_h] &= \epsilon [u_h], & \{\hat{\boldsymbol{\sigma}}\} &= \{\nabla u_h\} - \frac{\eta_0}{|e|} [u_h], \\ \{\hat{u} - u_h\} &= 0, & [\hat{\boldsymbol{\sigma}}] &= 0. \end{aligned}$$

Hence, for this choice of numerical fluxes, the bilinear form (2.5) becomes:

$$B(u_h, v) \quad = \quad \int_\Omega \nabla u_h \cdot \nabla v + \sum_{e \in E} \int_e \left(- \{\nabla u_h\} \cdot [v] + \epsilon [u_h] \cdot \{\nabla v\} + \frac{\eta_0}{|e|} [u_h] \cdot [v] \right). \tag{2.6}$$

2.3 Linear System

This section discusses how a DG solution, defined by (2.4) and (2.5) in the previous section, can be computed in terms of the solution to a linear system. The procedure is quite similar to the one-dimensional case described in Section 1.4.

We will assume that the mesh is uniform and has rectangular elements of size $h_x \times h_y$. Furthermore, for all $i = 1, ..., N$, we define the element $K_i := [x_{i-1}, x_i] \times [y_{i-1}, y_i]$ with cell center $(x_{i-\frac{1}{2}}, y_{i-\frac{1}{2}})$.

Table 2.1: k_x and k_y as function of k

k	1	2	3	4	5	6	7	8	9	10	...
k_x	0	1	0	2	1	0	3	2	1	0	...
k_y	0	0	1	0	1	2	0	1	2	3	...
	$p=0$	$p=1$		$p=2$			$p=3$				...

To construct a basis for the test space, consider a monomial of degree k_x in the first variable and k_y in the second, with $k_x + k_y \leq p$, that is translated from the interval $[-1,1]^2$ to the element K_i:

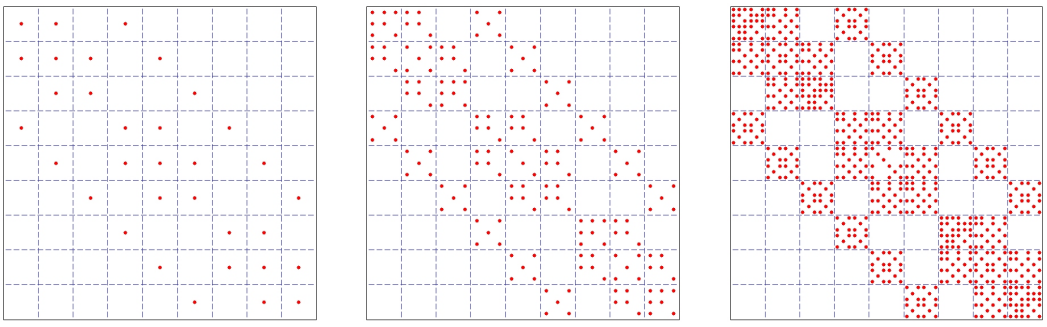
$$\phi_k^{(i)}(x,y) = \left(\frac{x - x_{i-\frac{1}{2}}}{\frac{1}{2}h_x}\right)^{k_x} \left(\frac{y - y_{i-\frac{1}{2}}}{\frac{1}{2}h_y}\right)^{k_y}, \qquad \text{with } k_x \text{ and } k_y \text{ as in Table 2.1,} \tag{2.7}$$

for all $k = 1, ..., M := \frac{(p+1)(p+2)}{2}$, for all $(x,y) \in K_i$, and for all $i = 1, ..., N$. For $(x,y) \notin K_i$, the function is set to zero. Next, write the DG solution $u_h \in V$ as a linear combination of these monomial test functions of degree p and lower:

$$u_h(x,y) = \sum_{i=1}^{N} \sum_{k=1}^{M} \alpha_k^{(i)} \phi_k^{(i)}(x,y), \tag{2.8}$$

Substitute this expression for u_h, and the monomial basis functions $\phi_\ell^{(j)}$ for v into the DG formulation (2.4) to obtain (1.10). As in Section 1.4, the unknowns $\alpha_k^{(i)}$, which determine the final DG solution as defined in (2.8), can be obtained as the solution a linear system $Ay = b$ of the form (1.11), (1.12). More details on the computation of the coefficient matrix A for two-dimensional problems can be found in Appendix B.2.

Similar to the one-dimensional case, for $p = 0$, the matrix A is equivalent to a matrix resulting from a central finite difference method (aside from a constant that can be included in the right hand side). This means that each off-diagonal element A_{ji} is zero if mesh element i and j are not adjacent. For $p > 0$, the matrix structure is similar, except that the matrix is now a *block* matrix, in which each off-diagonal *block* A_{ji} is zero if mesh element i and j are not adjacent. This is illustrated in Figure 2.1.



(a) $p = 0$ (b) $p = 1$ (c) $p = 2$

Figure 2.1: SIPG method for a 2D Poisson problem: for a mesh with $N_x \times N_y = 3 \times 3$ elements, the coefficient matrix A consists of $N_x N_y \times N_x N_y$ blocks, where each block is of size $M \times M$ with $M := \frac{(p+1)(p+2)}{2}$ and p the polynomial degree.

2.4 Numerical example

Now that the computational details are discussed, we will consider a numerical example to illustrate the performance of the SIPG method.

Consider the two-dimensional Poisson equation:

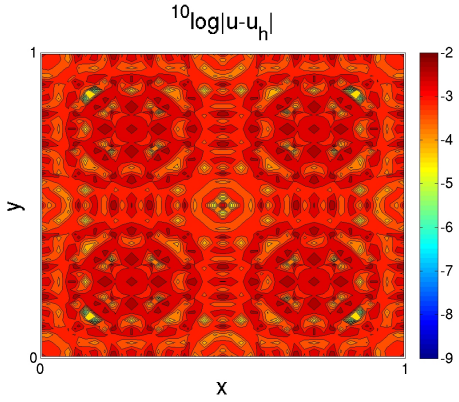
$$-\Delta u(x,y) = 2(2\pi)^2 \sin(2\pi x)\sin(2\pi y),$$

on the domain $[0,1]^2$ together with homogeneous Dirichlet boundary conditions. For this particular problem, the exact solution reads $u(x) = \sin(2\pi x)\sin(2\pi y)$.

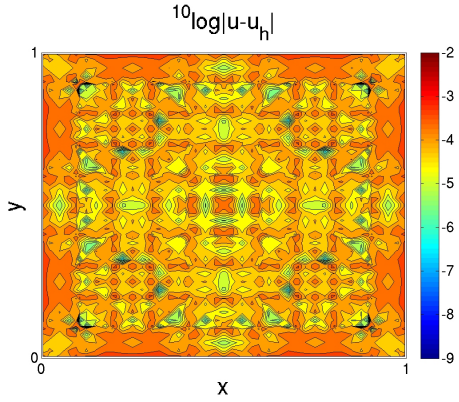
Figure 2.2 displays the outcome of the SIPG method (cf. Section 2.2) for this problem, using polynomial degree $p = 1, 2$ and penalty parameter $\eta_0 = 10$. Observe that the errors decrease rapidly as the number of elements increases.

2.5 Conclusion

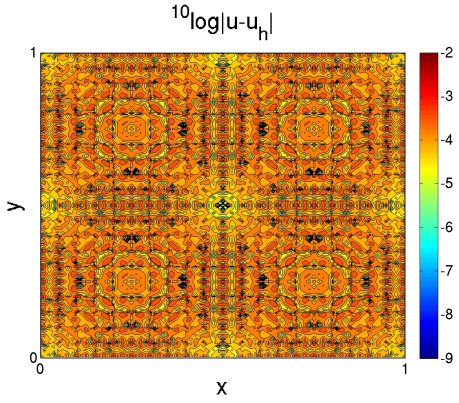
This chapter extends the DG formulations of the previous chapter to two-dimensional elliptic problems. Similar to the one-dimensional case, the main idea is to assume that the solution is a polynomial of degree p within each mesh element. Furthermore, the solution is allowed to be discontinuous at the element boundaries. The main difference is that two-dimensional problems usually require the solution of much more unknowns. As for one-dimensional problems, a DG approximation can be computed in terms of the solution to a linear system. For $p = 0$, the corresponding coefficient matrix is equivalent to a matrix resulting from a central finite difference method. For $p > 0$, the matrix structure is similar, except that the matrix is now a *block* matrix. In general, the matrix is an $N \times N$ block matrix, where N is the number of mesh elements, and each block in turn is an $M \times M$ matrix, where $M = \frac{(p+1)(p+2)}{2}$ is the dimension of the polynomial test space (within one mesh element). As a consequence, the size of the matrix grows rapidly with both the number of elements and the polynomial degree. Therefore, the next chapter will provide an overview of efficient solution techniques for such large sparse linear systems.



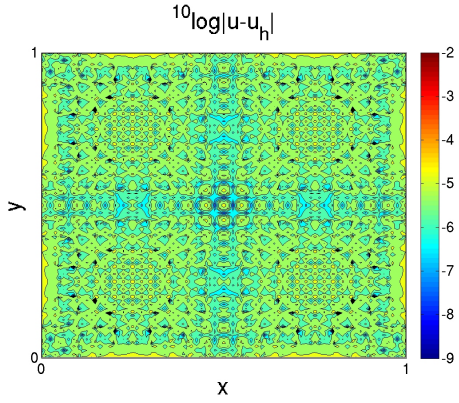
(a) 10×10 elements, $p = 2$



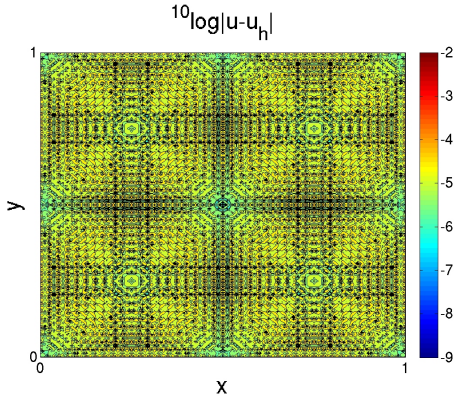
(b) 10×10 elements, $p = 3$



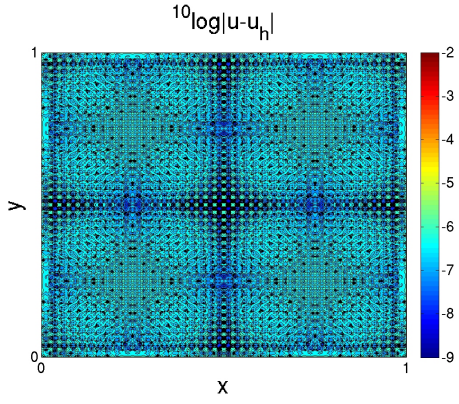
(c) 20×20 elements, $p = 2$



(d) 20×20 elements, $p = 3$



(e) 40×40 elements, $p = 2$



(f) 40×40 elements, $p = 3$

Figure 2.2: SIPG method (with $\eta_0 = 10$) for a 2D Poisson problem: the errors decrease rapidly as the number of elements increases.

Chapter 3

Solution Techniques for Linear Systems (Research Framework)

As we have seenin the previous two chapters, a DG method for an elliptic problem requires the solution to a large sparse linear system. Especially for two-dimensional (and three-dimensional) problems, the computation of the solution can be challenging due to the size and nature of the problem. This chapter provides an overview of efficient solution techniques for such systems, and a framework within which this research will take place. The main idea is to consider the iterative Conjugate Gradient (CG) method, and combine this method with preconditioning and deflation to speed up the convergence.

The outline of this chapter is as follows. First, the type of linear systems under consideration is analysed in Section 3.1. In this research, we will restrict ourselves (initially) to linear systems resulting from SIPG methods for Poisson problems. After that, the CG method is presented as a suitable iterative candidate for solving such systems in Section 3.2. It is also shown that the number of iterations grows linearly with the number of mesh elements since the coefficient matrix is ill-conditioned. Therefore, to enhance the convergence of CG, a preconditioner can be applied. This results in the Preconditioned CG (PCG) method, which is discussed in Section 3.3. Additionally, deflation can be incorporated in the algorithm. This results in the Deflated PCG (DPCG) method, which is considered in Section 3.4. Both strategies seek to improve the spectrum of the coefficient matrix. Finally a conclusion is given in Section 3.5.

3.1 Linear system

A DG discretisation requires the solution of the form $Ax = b$. The coefficient matrix A is typically an $N \times N$ block matrix, where N is the number of mesh elements, and each block in turn is an $M \times M$ matrix, where M is the dimension of the polynomial test space (within one mesh element). As a consequence, the size of the matrix grows rapidly with the number of elements and the polynomial degree (cf. Figure 1.1 and Figure 2.1).

In the remaining of this chapter, we will assume that A is the result of an SIPG discretisation of a Poisson equation. In this case, it can be shown that A is Symmetric and Positive-Definite (SPD), provided that the penalty parameter η_0 is sufficiently large.

The symmetry of A follows from the fact that the bilinear form is symmetric: $B(u, v) = B(v, u)$, with B as in (1.7) and (2.5). That A is positive definite follows from the fact that $B(u, u) > 0$ if the penalty parameter η_0 is sufficiently large (cf. [GK03, p. 531] and [Arn82]).

Finally, it is a well-known result that, for quasi-uniform meshes with element diameter h, the condition number is $O(h^{-2})$:

$$\kappa_2(A) = \|A\|_2 \left\|A^{-1}\right\|_2 \overset{A \text{ is SPD}}{=} \frac{\lambda_{\max}}{\lambda_{\min}} = O(h^{-2}),$$

where $\lambda_{\max} > 0$ and $\lambda_{\min} > 0$ are the largest and smallest eigenvalue of A respectively. It is illustrated in Figure 3.1 that $\kappa_2(A)$ grows rapidly with the penalty parameter and the number of mesh elements. A large condition number generally implies that it is challenging to solve the linear system. This issue will be considered in more detail in the next section.

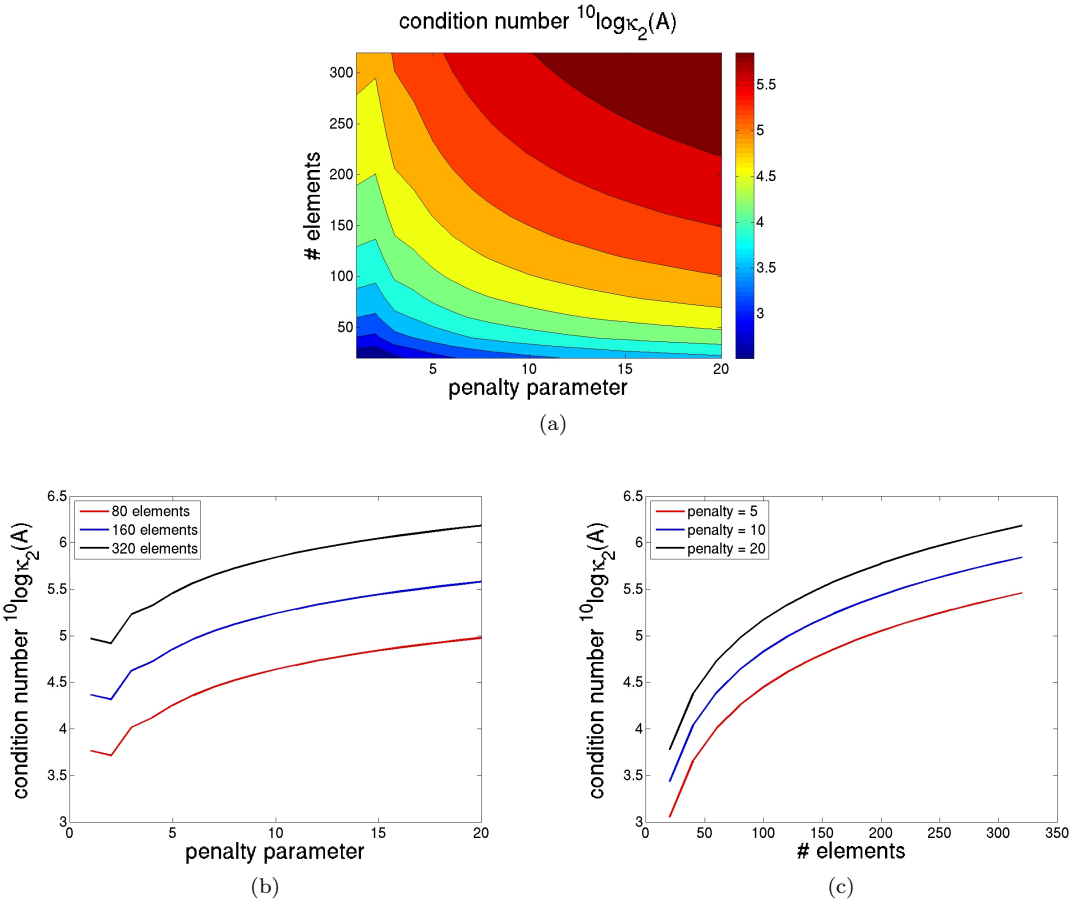


Figure 3.1: SIPG method (with $p = 2$) for a 1D Poisson problem (cf. Section 1.5): the condition number $\kappa_2(A)$ grows rapidly with the number of elements N and the penalty parameter η_0.

3.2 Conjugate Gradient method

As we saw in the previous section, the size of the matrix A grows rapidly with the number of elements and the polynomial degree. For such large sparse systems, iterative methods are usually favoured over direct methods to compute the solution. Because A is SPD, the well-known Conjugate Gradient (CG) method is a suitable iterative candidate (cf. [Saa00, Section 6.7]):

Algorithm 1 (Conjugate Gradient (CG) method)
Computes the solution x to the system $Ax = b$ using an initial guess x_0
1) $r_0 := b - Ax_0$
2) $p_0 := r_0$
3) **for** $j = 0, 1, ...$ until convergence **do**

```
4)      w_j := A p_j
5)      α_j := (r_j, r_j)/(p_j, w_j)
6)      x_{j+1} := x_j + α_j p_j
7)      r_{j+1} := r_j − α_j w_j
8)      β_j := (r_{j+1}, r_{j+1})/(r_j, r_j)
9)      p_{j+1} := r_{j+1} + β_j p_j
10) end
11) x := x_{j+1}
```

The performance of the CG method depends on the condition number $\kappa_2(A)$ [Saa00, p. 193]:

$$\frac{\|x - x_m\|_A}{\|x - x_0\|_A} \leq 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^m.$$

Here, x_m is the m^{th} iterate of the CG algorithm and x is the exact solution of the linear system. Because $\kappa_2(A) = O(h^{-2})$ (for quasi-uniform meshes with element diameter h), it follows from this expression that the number of iterations required for convergence is $O(h^{-1})$. This can be derived in the following manner.

In order for the relative error $\frac{\|x-x_m\|_A}{\|x-x_0\|_A}$ to be smaller than ϵ, the minimum number of iterations m must satisfy, defining $\mu := \sqrt{\kappa_2(A)}$:

$$2\left(\frac{\mu - 1}{\mu + 1}\right)^m \leq \epsilon.$$

Considering the logarithm of this expression yields:

$$m \geq \frac{\log\left(\frac{2}{\epsilon}\right)}{\log\left(\frac{\mu+1}{\mu-1}\right)} = \frac{\log\left(\frac{2}{\epsilon}\right)}{\log\left(\frac{1+\frac{1}{\mu}}{1-\frac{1}{\mu}}\right)} = \frac{\log\left(\frac{2}{\epsilon}\right)}{\log\left(1+\frac{1}{\mu}\right) - \log\left(1-\frac{1}{\mu}\right)}.$$

Constructing the Taylor polynomial of degree 1 near $x = 1$ implies that $\log(x) = (x-1) + O\big((x-1)^2\big)$. Hence, for μ sufficiently large:

$$m \geq \frac{\log\left(\frac{2}{\epsilon}\right)}{\frac{2}{\mu} + O\left(\frac{1}{\mu^2}\right)} \approx \frac{\mu}{2}\log\left(\frac{2}{\epsilon}\right) = \frac{\sqrt{\kappa_2(A)}}{2}\log\left(\frac{2}{\epsilon}\right) = O(h^{-1}),$$

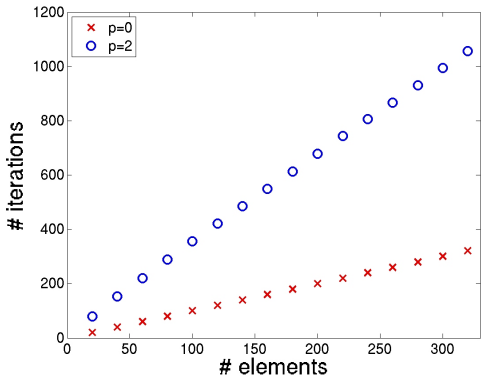
where it was used in the last step that $\kappa_2(A) = O(h^{-2})$.

That the number of iterations required for convergence is $O(h^{-1})$ is also demonstrated numerically in Figure 3.2a. In this figure we consider the model problem defined in Section 1.5, and we apply CG with the stopping criterion:

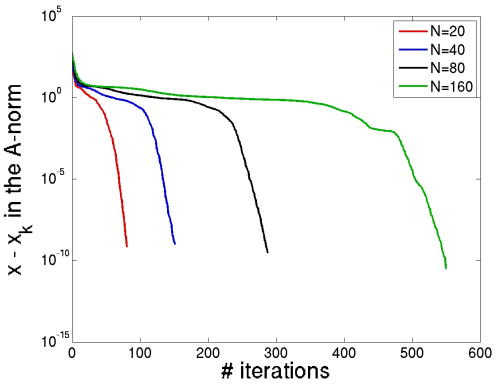
$$\frac{\|r_k\|_2}{\|b\|_2} \leq 10^{-11},$$

where r_k is the residual after the k^{th} iteration. We make use of a random start vector in order to avoid untypically fast convergence due to the fact that the exact solution is a sine function. As expected, the number of iterations grows linearly with the number of mesh elements in Figure 3.2a. However, the number of iterations appears to be larger than the order of the matrix, which is theoretically impossible. This issue will be included in the research questions in Chapter 6.

Finally, in Figure 3.2b, the convergence during the iterative process in the A-norm is shown. A typical acceleration in the convergence speed can be observed.



(a) The number of iterations required for convergence grows linearly with the number of mesh elements.



(b) Convergence in the A-norm $\sqrt{(x-x_k)^T A(x-x_k)}$ (for $p=2$): two accelerations can be observed.

Figure 3.2: SIPG method (with $\eta_0 = 10$) for a 1D Poisson problem (cf. Section 1.5), using the (unpreconditioned) CG method with a random start vector.

3.3 Preconditioning

In the previous section, we found that the number of iterations required for CG to converge grows linearly with the number of mesh elements. In order to keep the computational time at an acceptable level, the efficiency of CG can be enhanced through preconditioning.

To precondition the linear system $Ax = b$, the equation is multiplied by a matrix M^{-1}:

$$M^{-1}Ax = M^{-1}b.$$

The main idea is that the preconditioning operator M^{-1} should be inexpensive to apply to an arbitrary vector. At the same time, the matrix $M^{-1}A$ should have better spectral properties, i.e. eigenvalues clustered around 1. The quality of the spectrum is usually measured in terms of the condition number. In recent literature concerning DG methods for elliptic problems (an overview is given in Chapter 4 later on), much attention is devoted to the construction of preconditioners for which the condition number becomes independent of the number of mesh elements. For an overview of basic general purpose preconditioners, such as Jacobi and Gauss-Seidel, cf. e.g. [Saa00, Chapter 10].

The algorithm for Preconditioned CG (PCG) reads (cf. e.g. [Saa00, Section 9.2.1]):

Algorithm 2 (Preconditioned Conjugate Gradient (PCG) method)
Computes the solution x to the system $Ax = b$ using a left-preconditioner M^{-1} and initial guess x_0

1) $r_0 := b - Ax_0$
2) $z_0 := M^{-1}r_0$ (solve the linear system)
3) $p_0 := r_0$
4) **for** $j = 0, 1, \ldots$ until convergence **do**
5) $\quad w_j := Ap_j$
6) $\quad \alpha_j := (r_j, z_j)/(p_j, w_j)$
7) $\quad x_{j+1} := x_j + \alpha_j p_j$
8) $\quad r_{j+1} := r_j - \alpha_j w_j$
9) $\quad z_{j+1} := M^{-1}r_{j+1}$ (solve the linear system)
10) $\quad \beta_j := (r_{j+1}, z_{j+1})/(r_j, z_j)$
11) $\quad p_{j+1} := z_{j+1} + \beta_j p_j$

12) **end**
13) $x := x_{j+1}$

3.4 Deflation

The previous section discussed how preconditioning can increase the efficiency of CG by enhancing the spectrum of the coefficient matrix. However, even after preconditioning, some unfavorable eigenvalues may still remain in the spectrum, and deteriorate the efficiency of the algorithm. This section discusses the so-called deflation method, which can effectively treat these eigenvalues.

We follow [Tan08, Chapter 3]: consider a linear system $Ax = b$, where the coefficient matrix $A \in \mathbb{R}^{n\times n}$ is SPD. The main idea is to express the original solution x in terms of a deflated solution $\hat{x}$ in the following manner [Tan08, Corollary 3.1]:

$$x = Qb + P^T\hat{x}, \qquad\qquad PA\hat{x} = Pb,$$

where

$$\begin{aligned} P &:= I - AQ \in \mathbb{R}^{n\times n} && \text{is the deflation matrix,} \\ Q &:= ZE^{-1}Z^T \in \mathbb{R}^{n\times n} && \text{is the correction matrix,} \\ E &:= Z^TAZ \in \mathbb{R}^{k\times k} && \text{is the Galerkin or coarse matrix,} \end{aligned}$$

and $Z \in \mathbb{R}^{n\times k}$ with $k < n$ is the so-called deflation-subspace matrix, which we leave unspecified for the moment. It is assumed that Z has full rank and that $\mathcal{N}(A) \nsubseteq \mathcal{R}(Z)$, where $\mathcal{N}(A)$ denotes the null space (kernel) of A, and $\mathcal{R}(Z)$ denotes the range (column space) of Z. The latter ensures that E is non-singular [Tan08, Lemma 3.1].

Note that the deflated solution $\hat{x}$ is the solution to a linear system with coefficient matrix PA. It can be shown (similar to [Tan08, Lemma 3.4]) that the matrix PA is SPD, so that a CG type algorithm can be used to compute $\hat{x}$. Putting it all together, the following algorithm combines CG, preconditioning and deflation to solve $Ax = b$ (cf. [Tan08, Algorithm 4]):

Algorithm 3 (Deflated Preconditioned Conjugate Gradient (DPCG))
Computes the solution x to the system $Ax = b$ using a left-preconditioner M^{-1}, deflation matrix P, correction matrix Q, and initial guess $x_0 \equiv \hat{x}_0$

1) $r_0 := b - Ax_0$
2) $\hat{r}_0 := Pr_0$
3) $z_0 := M^{-1}\hat{r}_0$ (solve the linear system)
4) $p_0 := z_0$
5) **for** $j = 0, 1, ...$ until convergence **do**
6) $\quad w_j := PAp_j$
7) $\quad \alpha_j := (\hat{r}_j, z_j)/(p_j, w_j)$
8) $\quad \hat{x}_{j+1} := \hat{x}_j + \alpha_j p_j$
9) $\quad \hat{r}_{j+1} := \hat{r}_j - \alpha_j w_j$
10) $\quad z_{j+1} = M^{-1}\hat{r}_{j+1}$ (solve the linear system)
11) $\quad \beta_j := (\hat{r}_{j+1}, z_{j+1})/(\hat{r}_j, z_j)$
12) $\quad p_{j+1} := z_{j+1} + \beta_j p_j$
13) **end**
14) $x := Qb + P^T\hat{x}_{j+1}$

It can be shown that the condition number of PA is always smaller than that of A, for all choices of the deflation-subspace matrix Z [Tan08, Theorem 3.4]. Ideally, the columns of Z are the eigenvectors corresponding to the most unfavourable eigenvalues of $M^{-1}A$, which are often the smallest values. Deflation basically takes them out of the process, which results in better convergence.

On the other hand, the computation of eigenvectors can be expensive, and the use of eigenvectors typically leads to a dense matrix Z. Therefore, it can be more effective to approximate the eigenspace in some sense, using information of the application under consideration. Well-known strategies include approximate eigenvector deflation, recycling deflation, subdomain deflation, and multigrid deflation [Tan08, Section 4.2].

3.5 Conclusion

In this research, we will focus on linear systems resulting form the SIPG method for Poisson equations. It can be shown that the corresponding coefficient matrix is large, sparse, ill-conditioned, and Symmetric and Positive-Definite (SPD) (provided that the penalty parameter is sufficiently large). Therefore, the Conjugate Gradient (CG) method is considered a suitable iterative candidate for solving the system. It can be shown that the number of iterations grows linearly with the number of mesh elements since the coefficient matrix is ill-conditioned. Therefore, to enhance the convergence of CG, a combination of preconditioning and deflation can be incorporated in the algorithm, resulting in the Deflated Preconditioned (DPCG) method. The main goal of this research is to develop a new preconditioner for the aforementioned type of systems, and to combine the new preconditioner with a suitable deflation technique. Therefore, the next chapter will provide a literature overview of recent research in this area.

Chapter 4

Existing Preconditioners (Literature Overview)

The CG method discussed in the previous chapter generally converges slowly for problems with a large number of mesh elements. We also considered a brief introduction to preconditioning, which is a common technique to speed up CG. This chapter considers preconditioners that have recently received much attention in the context of DG/IP discretisations for elliptic problems. There are four main classes of preconditioners that are popular in this context, and we devote a section to each of them.

The outline of this chapter is as follows. Section 4.1 considers h-multigrid methods, which make use of coarse grid corrections. p-Multigrid methods operate similarly, except that they use a coarse problem with a lower polynomial degree to perform the correction. These methods are discussed in Section 4.2. Schwarz domain decomposition methods, which are the subject of Section 4.3, solve many small problems on subdomains rather than one huge problem. Section 4.4 studies space decomposition methods, which write the underlying DG finite element space as the direct sum of two subspaces. Finally, a conclusion is given in Section 4.5.

4.1 h-Multigrid

This section discusses h-multigrid preconditioners. The main idea is to make use of global corrections by solving coarser problems. These coarser problems are obtained by considering coarser meshes.

Gopalakrishnan and Kanschat [GK03] present a V-cycle h-multigrid preconditioner for the SIPG method. Their analysis is an application of the abstract theory of multigrid algorithms in [Bra93, BPX91]. For the same type of discretisation, Brenner and Zhao [BZ05] propose V-cycle, F-cycle, and W-cycle h-multigrid algorithms. They show that these algorithms produce uniform preconditioners, if the pre- and post-smoothing steps are sufficiently smooth. Similar methods are studied in [BS06] for C^0 interior penalty methods for fourth-order problems.

The remaining of this section focusses on [GK03].

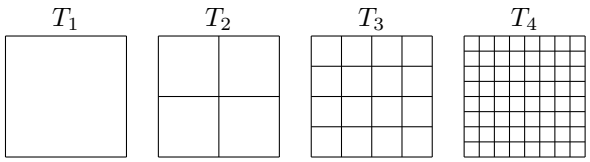


Figure 4.1: h-multigrid meshes

Model Gopalakrishnan and Kanschat [GK03] consider the SIPG method for the two-dimensional Poisson equation (cf. Section 2.2 in this report). They show that the resulting coefficient matrix is symmetric and positive definite, provided that the penalty parameter is sufficiently large. Furthermore, they extend their work for advection-diffusion problems.

Method To solve the linear system resulting from the SIPG scheme, a V-cycle h-multigrid preconditioner is presented. To illustrate this method, consider a spatial domain $T_1 = [-1, 1]^2$ (more general domains are considered in the paper). For all $k = 2, ..., J$, define a mesh T_k as the $2^{k-1} \times 2^{k-1}$ partition of T_1 (cf. Figure 4.1), and the corresponding DG test functions $\{\phi_k^{(i)}\}_{i=1,...,N_k}$. Here, N_k denotes the number of degrees of freedom of the test space (determined by the polynomial degree, the two space dimensions, and the number of elements). Observe that each basis function can be written as a linear combination of basis functions on a finer mesh:

$$\phi_{k-1}^{(i)} = \sum_{j=1}^{N_k} \alpha_{ij}^{(k)} \phi_k^{(j)}.$$

Let C_{k-1} be the matrix whose (i, j)-th entry is $\alpha_{ij}^{(k)}$. Then C_{k-1} is the restriction operator and its transpose C_{k-1}^T is the prolongation operator. Let A_k be the coefficient matrix resulting from applying the SIPG method on the mesh T_k, i.e. its (i, j)-th element is equal to $B(\phi_k^{(j)}, \phi_k^{(i)})$, where B is the bilinear form (cf. Section 2.2). In particular $A := A_J$ is the coefficient matrix of the system we need to solve (corresponding to the finest level). Let R_k denote the block Gauss-Seidel smoother corresponding to A_k (but other smoothers, e.g. block Jacobi could be used as well). Moreover set $R_k^{(\ell)} := R_k$ if ℓ is an odd integer, and $R_k^{(\ell)} := R_k^T$ if ℓ is even. Finally, let $m(k)$ denote a positive integer for all $k = 1, .., J$ (typically $m(k) = 2^{J-k}$).

Then, the result z of applying the preconditioner $B_J \approx A^{-1}$ to a vector r is obtained by recursion in the following manner:

Algorithm 4
Computes $z = B_k r$
1) **if** $k = 1$, **then** $z := B_1 r := A_1^{-1} r$ **else**:
2) $x^0 := 0$
3) % *apply the smoother:*
4) **for** $\ell = 1, ..., m(k)$ **do**
5) $\quad x^\ell := x^{l-1} + R_k^{(\ell+m(k))}(r - A_k x^{\ell-1})$
6) **end**
7) % *coarse grid correction:*
8) $y^{m(k)} := x^{m(k)} + C_{k-1}^T B_{k-1} C_{k-1}(r - A_x x^{m(k)})$, using this algorithm (recursion)
9) % *apply the smoother:*
10) **for** $\ell = m(k), ..., 2m(k)$ **do**
11) $\quad y^\ell := y^{\ell-1} + R_k^{(\ell+m(k))}(r - A_k y^{\ell-1})$
12) **end**
13) $z := B_k r := y^{2m(k)}$

Theoretical results For elliptic problems, the SIPG method yields a coefficient matrix A with a condition number that is $O(h^{-2})$ on quasiuniform grids with element diameter h. Under a certain regularity assumption, the condition number of the preconditioned matrix $B_J A$ becomes $O(1)$, i.e. bounded independently of h (and the element diameters h_k of the subgrids). Furthermore, the preconditioning matrix B_J is symmetric and positive-definite, which makes it particularly suitable for CG, which then "converges in $O(N)$ operations, where N is the number of unknowns".

Numerical results Numerical results that illustrate the theory are provided for the two-dimensional Poisson equation. They consider the domain $[-1,1]^2$ as well as an L-shaped domain. Furthermore, they use $m(k) = 2^{J-k}$ (and other values), polynomial degree $p = 1, 2, 3$, and $J = 2, 3, ..., 8$. Additionally, the block Gauss-Seidel smoother is compared to a block Jacobi smoother (with relaxation). Besides verifying the theory above, they find that the condition number of the preconditioned matrix $B_J A$ grows linearly with the penalty parameter.

Conclusion The V-cycle h-multigrid preconditioner predicts convergence rates independent of the mesh size. Although the preconditioner is designed for elliptic problems, it can be extended for advection-diffusion problems. It is noted that it should be possible to obtain similar results for a W-cycle preconditioner.

4.2 p-Multigrid

This section discusses p-multigrid preconditioners. Similar to h-multigrid, the main idea is to make use of global corrections by solving coarser problems. The main difference with h-multigrid is that the coarser problem is obtained by reducing the polynomial degree, rather than coarsening the mesh.

The original idea behind p-multigrid is the work by Ronquist and Patera [RP87]. Fidkowski, Oliver, Lu and Darmofal [FOLd05] present a p-multigrid algorithm with an element line Jacobi smoother for the solution of higher-order DG discretizations of the compressible Navier-Stokes equations. Nastase and Mavriplis [NM06] combine h- and p-multigrid into one hp-multigrid scheme. Their method is developed for the two-dimensional non-linear Euler equations on unstructured grids. Persson and Peraire [PP08] consider similar methods, and propose a reordering strategy to enhance the robustness. Hillewaert, Chevaugeon, Geuzaine, and Remacle [HCGR06] study a hierachic multigrid iteration strategy that is "applicable to both p-multigrid and classical h-multigrid for any weak formulation using discontinuous interpolation, even when interpolation spaces are not nested". Bassi, Ghidoni, Rebay, and Tesini [BGRT09] propose a p-multigrid scheme which employs a semi-implicit RK smoother at the finer levels and the implicit backward Euler smoother at the coarsest level with polynomial degree $p = 0$.

The remaining of this section focusses on [PP08].

Model Persson and Peraire [PP08] consider a DG formulation for the incompressible Navier-Stokes equations on triangular meshes. The inviscid flux is determined using Roe's scheme, whereas the viscous flux is calculated using the Compact DG (CDG) method. Since practical applications require a large variation in element size, the time-discretization is implicit (BDF-k scheme). The resulting nonlinear system is solved using a damped Newton method, which requires the solution of several linear systems with coefficient matrix A during each time step.

Method The linear systems are solved using restarted GMRES(20) with a p-multigrid preconditioner [RP87, FOLd05]. They consider two-level variants only, as this gives better overall performance in their experience.

The preconditioner is combined with a smoother based on the well-known BILU(0) decomposition. The latter is constructed by modifying the algorithm for a full block LU decomposition in such a way that no nonzero matrix entries outside the sparsity pattern of the coefficient matrix occur. It is used that this sparsity pattern is given directly by the connectivity between the mesh elements. They simplify the algorithm by neglecting the rare cases that three elements are fully connected.

The performance of the BILU(0) smoother can depend strongly on the numbering of the mesh elements. Therefore, they propose to construct the decomposition in combination with a reordering technique that is inspired by the well-known MDF method. The main idea is to start with the mesh element that produces the least discarded fill-in, and then repeat the process in a greedy way.

Putting it all together, the preconditioner for the coefficient matrix A can be defined as follows. Let $\{\phi_i^{(p)}\}_{i=1,\ldots,N^{(p)}}$ be a basis for the finite element space with polynomial degree p. Similarly, let $\{\phi_i^{(0)}\}_{i=1,\ldots,N^{(0)}}$ be a basis for the coarse space, which operates on the same mesh, but uses a lower polynomial degree that is typically 0 or 1. Similar to h-multigrid (cf. Section 4.1), we can define a prolongation P and restriction operator P^T by observing that

$$\phi_i^{(0)} = \sum_{j=1}^{N^{(p)}} \alpha_{ij} \phi_j^{(p)},$$

and setting $P_{ij} = \alpha_{ji}$. Next, let γ be a smoothing factor that is typically close to 1, and let $\tilde{A}^{-1}$ denote the BILU(0) smoother (but other smoothers, e.g. block Jacobi, block Gauss-Seidel, can be used as well). Now, the result z of applying the preconditioner to a residual r is obtained as follows (cf. [FOLd05] for more details on p-multigrid algorithms):

Algorithm 5
Computes $z \approx A^{-1}r$:
1) Restriction: $A^{(0)} := P^T AP$, $r^{(0)} := P^T b$
2) Solve the coarse scale problem: $A^{(0)} z^{(0)} = r^{(0)}$
3) Prolongation: $z := Pz^{(0)}$
4) Apply the smoother: $z := z + \gamma \tilde{A}^{-1}(r - Az)$

Numerical results For several scalar convection-diffusion equations (including the Poisson equation) and the compressible Navier-Stokes equations, the performance of the following preconditioners is compared: block Jacobi, block Gauss-Seidel, BILU(0), and p-multigrid combined with any of the three aforementioned preconditioners as smoother, using either $p = 0$ or $p = 1$ at the coarse level.

The p-multigrid preconditioner with BILU(0) smoother performs better than the (standalone) BILU(0) preconditioner, especially for diffusion-dominated problems. Furthermore, reordering seems very effective for convection-dominated problems, but hardly for diffusion-dominated problems. For the Navier-Stokes equations, the BILU(0) smoother performed significantly better than the block Gauss-Seidel smoother.

Conclusion Element ordering can be critical for incomplete factorisations. For this reason, an MDF-type algorithm is presented to enhance the performance of a BILU(0) smoother in a p-multigrid preconditioner. This strategy is compared to many other types of preconditioning in the context of advection-diffusion and Navier-Stokes problems.

4.3 Schwarz Domain Decomposition

This section discusses Schwarz domain decomposition preconditioners. The main idea is to subdivide the spatial domain into smaller subdomains, and to solve many small local problems, rather than one huge problem.

Classical theory for Schwarz methods is discussed in [TW05]. Feng and Karakashian [FK01] propose a Schwarz preconditioner for SIPG schemes, which they extend for fourth-order problems in [FK05]. Lasser and Toselli [LT03] present Schwarz preconditioners for advection-reaction-diffusion problems, in which case the resulting matrix is no longer SPD. Sarkis and Szyld [SS07] consider additive Schwarz methods that maintain the Schwarz optimality. Antonietti and Ayuso [AA07] study Schwarz methods in the unified framework in [ABCM02]. They extend their work for non-matching grids in [AA08], and for non-consistent super penalty methods in [AA09].

The remaining of this section focusses on [AA07].

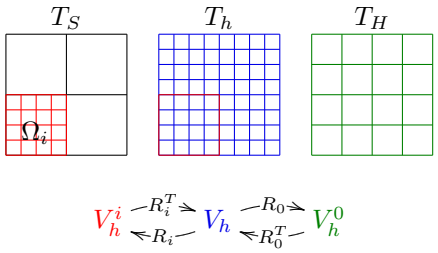


Figure 4.2: Schwarz preconditioning: meshes, test spaces, and restriction and prolongation operators.

Model Anonietti and Ayuso [AA07] study a large class of stable and consistent DG methods in the unified framework in [ABCM02] (cf. Section 1.3 and Section 2.2). Their application is the two- or three-dimensional Poisson equation with homogeneous Dirichlet boundary conditions:

$$-\Delta u = f, \qquad \text{in } \Omega, \qquad\qquad u = 0, \qquad \text{on } \partial\Omega \tag{4.1}$$

Nevertheless, it is claimed that their results also apply for more general second order elliptic problems in divergence form.

Method To solve the linear system resulting from such methods, an additive two-level non-overlapping Schwarz preconditioner is proposed.

Consider three (regular and quasi-uniform) meshes: T_S, T_H and T_h, which are illustrated in Figure 4.2. Here, T_h is the fine mesh with mesh size h on which we wish to obtain a DG approximation. T_H is a coarser mesh with mesh size H such that each element in T_H can be written as the union of a number of elements in T_h. Similarly, T_S is an even coarser mesh (with N_S elements) such that each element in T_S can be written as the union of a number of elements in T_H. The preconditioner is based on the solution of local problems on T_S and a coarse solver on T_H in the following manner.

Let V_h denote the test space consisting of each function on Ω that is a polynomial of degree p_h or lower within each element in T_h. Similarly, for all $i = 1, ..., N_S$, let V_h^i denote the test space consisting of each function on $\Omega_i \in T_S$ that is a polynomial of degree p_h or lower within each element in T_h that is a subset of Ω_i. Moreover, let V_h^0 denote the test space consisting of each function on Ω that is a polynomial of degree $p_H \leq p_h$ or lower within each element in T_H. It should be stressed that, despite the notation, the test space V_h^0 is based on elements in T_H rather than T_h, and, in that sense, has little in common with $V_h^1, ..., V_h^{N_S}$.

Furthermore, for all $i = 1, ..., N_S$, define the restriction operator $R_i : V_h \to V_h^i$ such that $R_i(v) = v|_{\Omega_i}$ for all $v \in V_h$, and define the prolongation operator $R_i^T : V_h^i \to V_h$ as its (unique) transpose with respect to the Euclidean scalar product, i.e. $R_i(v)w = vR_i^T(w)$ for all $v \in V_h$ and all $w \in V_h^i$. Additionally, define the the prolongation operator $R_0^T : V_h^0 \to V_h$, such that $R_0^T(v) = v$ for all $v \in V_h^0$, and define the restriction operator $R_0 : V_h \to V_h^0$ as its transpose (also cf. Section 4.1).

Let $A_h : V_h \times V_h$ denote the bilinear form of the DG method for the global problem (4.1) on the fine mesh T_h. Moreover, for all $i = 1, ..., N_S$ let u_i denote the solution to the local problem:

$$-\Delta u_i = f|_{\Omega_i}, \qquad \text{in } \Omega_i, \qquad\qquad u_i = 0, \qquad \text{on } \partial\Omega_i, \tag{4.2}$$

and let $A_i : V_h^i \times V_h^i \to \mathbb{R}$ denote the bilinear form of the DG method for this local problem. The use of homogeneous Dirichlet boundary conditions in the local problem (4.2) is generally inconsistent with the global model for u and in that sense an approximation. On the other hand, this approach is advantageous in terms of parallelization. Alternatively, the local solver could be obtained by restricting the global DG approximation to the subdomain Ω_i, which is the strategy followed

earlier in [FK01, LT03]. This alternative does not make use of artificial boundary conditions, but is less suitable for parallelization.

Finally, construct the coarse solver $A_0 : V_h^0 \times V_h^0 \to \mathbb{R}$ such that $A_0(u_0, v_0) = A_h(R_0^T u_0, R_0^T v_0) \neq A_H(u_0, v_0)$, where A_H is defined similar to A_h, but on the coarser mesh T_H

Defining $\mathbb{A}$, $\mathbb{A}_i$, $\mathbb{R}_i^T$, and $\mathbb{R}_i$ as the matrix representations of A_h, A_i, R_i^T, and R_i respectively, the Schwarz preconditioning matrix $\mathbb{B} \approx \mathbb{A}^{-1}$ is now defined as follows:

$$\mathbb{B} = \sum_{i=0}^{N_S} \mathbb{R}_i^T \mathbb{A}_i^{-1} \mathbb{R}_i.$$

Theoretical results For symmetric DG methods, such as SIPG, BRMPS, BMMPR, LDG, BZ, BMMPR, it is shown that the condition number of the preconditioned system is $O\left(\frac{H}{h}\right)$, independent of the number of Schwarz subdomains N_S, and linearly dependent on the stabilisation parameter (which is the penalty parameter η_0 for IP methods used in Section 1.3 and Section 2.2).

For nonsymmetric DG methods, such as IIPG and NIPG, it is found that the "Eisenstat et al. (...) GMRES convergence theory, generally used in the analysis of Schwarz methods for non-symmetric problems, cannot be applied even if the numerical results show that the GMRES applied to the preconditioned systems converges in a finite number of steps and the proposed preconditioners seem to be scalable".

Numerical results Numerical results are presented to illustrate the performance of the proposed Schwarz preconditioner on the domain $[0, 1]^2$ for both Cartesian and triangular meshes. Symmetric systems are solved using the preconditioned CG method, nonsymmetric systems are solved using GMRES. Besides a validation of the theoretical results, it is found that larger values of p_H lead to better performance. Furthermore, their results are comparable to those in [FK01] for similar problems. Moreover, it is pointed out that, for smaller H (and fixed h), the coarse system becomes more expensive. Similarly, the local systems corresponding to (4.2) are more costly to solve for smaller N_s (and fixed h).

Conclusion An additive two-level non-overlapping Schwarz preconditioner is proposed for a large class of stable and consistent DG methods in the unified framework in [ABCM02]. Optimal estimates are provided for symmetric systems.

4.4 Space Decomposition

This section discusses methods in the framework of space decomposition and subspace correction methods. The main idea is to write the DG test space as the direct sum of two subspaces. This results in new matrix properties that can be exploited.

By using such a decomposition, Ayuso and Zikatanov [AdDZ09] propose uniform preconditioners for the symmetric SIPG method, and a uniform iterative method for the non-symmetric NIPG and IIPG methods. Their ideas are based on the work in [Xu92, XZ02].

The remaining of this section focusses on [AdDZ09].

Model Ayuso and Zikatanov [AdDZ09] consider symmetric (SIPG) and non-symmetric (NIPG, IIPG) interior penalty methods for the Poisson equation (though it is claimed that the results apply to more general elliptic problems).

A distinction is made between the classical type-1 methods, with bilinear form (cf. (2.6) in this report):

$$\mathcal{A}(u, w) = \int_\Omega \nabla u \cdot \nabla w + \sum_{e \in E} \int_e \left(- \{\nabla u\} \cdot [w] + \epsilon [u] \cdot \{\nabla w\} + \frac{\eta_0}{|e|} [u] \cdot [w] \right),$$

and the type-0 methods, whose bilinear from results by approximating the penalty terms:

$$\mathcal{A}_0(u,w) = \int_\Omega \nabla u \cdot \nabla w + \sum_{e\in E} \int_e \left(- \{\nabla u\} \cdot [w] + \epsilon\, [u] \cdot \{\nabla w\} + \frac{\eta_0}{|e|} \mathcal{P}_e^0\, [u] \cdot \mathcal{P}_e^0\, [w] \right),$$

where $\mathcal{P}_e^0$ denotes the L^2-projection onto the constants on the edge e.

To compute the solution defined by these bilinear forms, they propose to decompose the test space V, which consists of each function that is a polynomial of degree $p = 1$ or lower within each mesh element, into two subspaces V^{CR} and Z. The first subspace V^{CR} consists of all functions $v \in V$ such that the projected *jump* $\mathcal{P}_e^0\,[v] = 0$ for each interior edge e. This space is also referred to as the Crouzeix-Raviart finite element space. The second space Z consists of all functions $v \in V$ such that the projected *average* $\mathcal{P}_e^0\,\{v\} = 0$ for each interior edge e.

It is shown that, for every $u \in V$, there exists a unique $v \in V^{CR}$ and a unique $z \in Z$ such that $u = v + z$. As a consequence, for all $u, w \in V$, we can write $u = v + z$ and $w = \phi + \psi$, with $v, \phi \in V^{CR}$ and $z, \psi \in Z$, so that:

$$\mathcal{A}(u,w) = \mathcal{A}(v,\phi) + \mathcal{A}(v,\psi) + \mathcal{A}(z,\phi) + \mathcal{A}(z,\psi).$$

Furthermore, we can use bases for V^{CR} and Z (rather than one basis for V) to write the IP solution as the solution to a linear system with a coefficient matrix of the form:

$$\mathbb{A} = \begin{bmatrix} \mathbb{A}^{zz} & \mathbb{A}^{zv} \\ \mathbb{A}^{vz} & \mathbb{A}^{vv} \end{bmatrix}.$$

Observe that the matrix is twice as large as usual, but due to the properties of the bases, the number of nonzeros is typically about twice as small. For instance, $\mathcal{A}^{zv} = 0$ for methods of type-0. So the system becomes block lower triangular for those methods. More information on how to solve the linear system for type-0 methods is provided in [AdDZ09, Section 4].

Method To compute the solution to the resulting linear system, two preconditioners for the symmetric SIPG method are proposed, as well as an iterative solver for the non-symmetric NIPG and IIPG methods.

The preconditioners for the symmetric SIPG method are constructed as follows. First, the bilinear form $\mathcal{A}$ is approximated by a bilinear form $\mathcal{B}$. Then, the preconditioning matrix is defined as $\mathbb{B}^{-1}$, where $\mathbb{B}$ is the matrix representation of $\mathcal{B}$ (obtained using the aforementioned bases for V^{CR} and Z). For the bilinear form $\mathcal{B} \approx \mathcal{A}$, two options are suggested:

$$\mathcal{B}(u,v) := \mathcal{A}_0(u,v), \qquad \text{and} \qquad \mathcal{B}(u,w) := \mathcal{A}(v,\phi) + \mathcal{A}(z,\psi).$$

The resulting preconditioners are 2×2 block diagonal matrices, due to the properties of V^{CR} and Z. So in order to apply the preconditioner, two symmetric relatively sparse systems need to be solved.

The iterative solver for the non-symmetric NIPG and IIPG methods is constructed as follows. First, define the bilinear form $\mathcal{B}$ as the symmetric part of $\mathcal{A}$, so that $\mathcal{B}(u,w) = \frac{1}{2}\left(\mathcal{A}(u,w) + \mathcal{A}(w,u)\right)$. Then, the iterative algorithm reads, assuming an initial guess u_0 is specified:

Algorithm 6
1) **for** $k = 0, 1, \ldots$ until convergence
2) Solve $\mathcal{B}(e_k, w) = (f, w) - \mathcal{A}(u_k, w)$ for all $w \in V$
3) update $u_{k+1} = u_k + e_k$
4) **end**

Observe that this algorithm requires the solution to a symmetric system only.

Theoretical results For the preconditioners for the SIPG method, it is shown that $\mathcal{A}$ and $\mathcal{B}$ are spectrally equivalent. This implies that the condition number of the preconditioned system is $O(1)$, depending on the penalty parameter only.

For non-symmetric IP methods, it is shown that the proposed iterative method converges uniformly in the norm defined by $\mathcal{B}$, under certain restrictions on the penalty parameter:

$$\|u - u_{k+1}\|_{\mathcal{B}} \leq \Lambda \|u - u_k\|_{\mathcal{B}},$$

where $\Lambda < 1$ is a positive constant.

Numerical results The main purpose of the numerical results is to validate the theories; required computational times are left out of consideration.

Numerical results on a square two-dimensional domain are shown that validate the theoretical uniform convergence of the proposed methods. The meshes are triangular, and six levels of refinement are used. Indeed, the ℓ^2 condition number of the preconditioned SIPG system is $O(1)$, whereas it scales quadratically with the number of elements in the unpreconditioned case. The latter is a well-known theoretical result.

The iterative method for the non-symmetric IP schemes is tested for several penalty parameters. As predicted, uniform convergence is observed. Furthermore, the convergence rate improves for larger values of the penalty parameter.

Conclusion It is possible to write the DG test space as the internal direct sum of the Crouzeix-Raviart finite element space and a second subspace. This decomposition can be used to obtain uniform preconditioners for the symmetric SIPG method and a uniform iterative method for the non-symmetric NIPG and IIPG methods. According to the authors, this the first time that a uniform iterative scheme is proposed for non-symmetric problems.

4.5 Conclusion

This chapter considers preconditioners that have recently received much attention in the context of DG/IP discretisations for elliptic problems. There are four main classes of preconditioners that are popular in this context: h-multigrid, p-multigrid, Schwarz domain decomposition, and space decomposition. One preconditioning type that is not in this list, but that has been very successful for finite difference methods, is the class of preconditioners that are based on a Block Incomplete LU-decomposition (BILU). Therefore, we will choose this to be our research direction, and we will study existing BILU algorithms in the next chapter.

Chapter 5

BILU Preconditioning (Research Direction)

While preconditioners based on a Block Incomplete LU-decomposition (BILU) have been applied successfully for finite difference discretisations in the past, it was found in the previous chapter that this type of preconditioning has received little attention in the context of DG discretisations. Therefore, this chapter studies existing BILU algorithms as a starting point for our research. An exact Block LU factorisation of a matrix A is of the form $A = LDU$, where D is a block diagonal matrix and L and U are block lower and upper triangular matrices respectively with identity blocks on the diagonal. The main idea behind a BILU preconditioner is to make use of an approximation of such a factorisation.

The outline of this chapter is as follows. The computation of an exact block LU factorisation is discussed in Section 5.1 for block tridiagonal matrices. Section 5.2 discusses a recursive BILU preconditioner, which assumes and exploits a recursive block tridiagonal structure of the matrix. A more general BILU framework is considered in Section 5.3. The recursive BILU decomposition can be obtained as a special case in this framework, which is shown in Section 5.4. Numerical examples that illustrate the definition and performance of the recursive BILU preconditioner are considered in Section 5.5. Finally, a conclusion is given in Section 5.6.

5.1 Exact Block LU-decomposition for Block Tridiagonal matrices

This section discusses the construction of an exact block LU-decomposition for block triangular matrices. Incomplete factorisations that can be used as a preconditioner can be derived by approximating this algorithm.

Consider a matrix with a block tridiagonal structure:

$$A := \begin{bmatrix} A_1 & C_1 & & \\ B_2 & A_2 & \ddots & \\ & \ddots & \ddots & C_{n-1} \\ & & B_n & A_n \end{bmatrix}. \tag{5.1}$$

The exact block-LU decomposition of A reads:

$$A = \underbrace{\begin{bmatrix} I & & & \\ L_2 & I & & \\ & \ddots & \ddots & \\ & & L_n & I \end{bmatrix}}_{=:L} \underbrace{\begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_n \end{bmatrix}}_{=:D} \underbrace{\begin{bmatrix} I & U_1 & & \\ & I & \ddots & \\ & & \ddots & U_{n-1} \\ & & & I \end{bmatrix}}_{=:U}, \tag{5.2}$$

where $D_1 = A_1$, and, for all $j = 2, ..., n$,

$$L_j = B_j D_{j-1}^{-1}, \qquad U_{j-1} = D_{j-1}^{-1} C_{j-1}, \qquad D_j = A_j - B_j D_{j-1}^{-1} C_{j-1}. \tag{5.3}$$

This can be verified by performing the multiplications. In other words, the block LU decomposition can be computed by the following algorithm:

Algorithm 7
Computes the exact LU decomposition $A = LDU$:
1) $D_1 = A_1$
2) **for** $j = 2, ..., n$ **do**
3) $\quad L_j := B_j D_{j-1}^{-1}$
4) $\quad U_{j-1} := D_{j-1}^{-1} C_{j-1}$
5) $\quad D_j := A_j - B_j D_{j-1}^{-1} C_{j-1}$
6) **end**

Observe that this algorithm is expensive in the sense that the inverses D_j^{-1}, which are applied to matrices, need to be computed. Nevertheless, in theory, we could choose $(LDU)^{-1} = A^{-1}$ as a 'preconditioner'. In that case, the result $z = [z_1, ..., z_n]^T$ of applying $(LDU)^{-1} = A^{-1}$ to $r = [r_1, ..., r_n]^T$ could be computed in the following manner:

Algorithm 8
Computes $z = (LDU)^{-1} r$, assuming that $D_1, ..., D_n$ are precomputed.
1) $s_1 := r_1$
2) **for** $j = 2, 3, ..., n$ **do**
3) $\quad s_j := r_j - B_j D_{j-1}^{-1} s_{j-1}$ (by solving the linear system)
4) **end**
5) $z_n = D_n^{-1} s_n$ (by solving the linear system)
6) **for** $j = n-1, n-2, ..., 1$ **do**
7) $\quad z_j = D_j^{-1}(s_j - C_j z_{j+1})$ (by solving the linear system)
8) **end**

Note that only the matrices D_j are required to execute this algorithm; the matrices L_j and U_j are not needed. The reason for this is that we can write:

$$A = LDU = (B + D)D^{-1}(D + C), \tag{5.4}$$

where the matrix B denotes the lower triangular part of A, and the matrix C the upper triangular part.

Finally, observe that Algorithm 8 is actually a direct method to solve the system $Az = r$. Therefore, it is not a suitable preconditioner in practice. However, by using approximations that lower the computational costs of the application of the inverses, an efficient preconditioner can be obtained. An example of such a strategy is discussed in the next section.

5.2 A Recursive BILU Preconditioner for Block Triangular Matrices

This section considers the Block Incomplete LU-decomposition (BILU) studied by Van 't Slot in [Slo08, p. 15-17]. The main idea is to approximate the exact algorithm of the previous section. To this end, both an approximation of both the *decomposition* (cf. Algorithm 7) and the *solver* (cf. Algorithm 8) needs to be constructed. It should be noted that the approximation for the decomposition described below (cf. Algorithm 9) is also proposed in a more general framework in [Mei83], which is discussed in the next section. Furthermore, an algorithm that is similar the approximation for the solver below (cf. Algorithm 10) can be found in [AC83].

We consider a matrix of the form (5.1) with a recursive block tridiagonal structure: each off-diagonal block is assumed to be a regular diagonal matrix, and each diagonal block is assumed to have a recursive block tridiagonal structure similar to A. Here, we restrict ourselves to the case that each diagonal block in A is a (regular) tridiagonal matrix. The preconditioner described below can also be applied to matrices with a deeper nested recursive block tridiagonal structure, such as those resulting from a central difference discretisation for a three-dimensional Poisson problem. The more general case is considered in detail in Appendix C of this report.

A *complete* block-LU decomposition for the matrix A was specified earlier in Algorithm 7. A disadvantage of using this factorisation is that the inverses D_j^{-1} are usually full matrices and too expensive to apply. Therefore, it is proposed to construct a preconditioner by replacing the inverse D_j^{-1} by a certain approximation T_j (specified below) that has the same (regular tridiagonal) structure as A_j. As a consequence, the factorization has now become a Block *Incomplete* LU (BILU) decomposition. Furthermore, because the off-diagonal blocks B_j and C_j are (regular) diagonal matrices, the diagonal factors $D_j = A_j - B_j T_{j-1} C_{j-1}$ will have the same tridiagonal structure as A_j. This property is characteristic for the BILU method, and makes it possible to exploit the recursive block tridiagonal structure of the matrix, as will become clear later on.

The approximation $T_j \approx D_j^{-1}$ is computed as follows:

Algorithm 9
Computes the BILU factors $D_1, ..., D_n$ for (a matrix with a similar structure as) A (cf. (5.1)), as well as an approximation $T \approx A^{-1}$, using the notation:

$$T = \begin{bmatrix} E_1 & G_1 & & \\ F_2 & E_2 & \ddots & \\ & \ddots & \ddots & G_{n-1} \\ & & F_n & E_n \end{bmatrix}. \tag{5.5}$$

1) **if** A is a scalar, compute $T = 1/A$ directly, **else**:
2) $D_1 := A_1$
3) **for** $j = 2, ..., n$ **do**
4) compute $T_{j-1} \approx D_{j-1}^{-1}$ by calling this algorithm recursively
5) $L_j := B_j T_{j-1}$
6) $U_{j-1} := T_{j-1} C_{j-1}$
7) $D_j := A_j - B_j T_{j-1} C_{j-1}$
8) **end**
9) $E_n := T_n$
10) **for** $j = n-1, ..., 1$ **do**
11) $E_j := T_j + U_j E_{j+1} L_{j+1}$
12) **end**
13) **for** $j = 2, ..., n$ **do**
14) $F_j := -T_j L_j$

15) $G_{j-1} := -U_{j-1}T_j$
16) **end**
17) Construct T according to (5.5) and set all of its elements outside the nonzero pattern of A equal to zero.

Observe that the approximation $T \approx A^{-1}$ is constructed using approximations $T_j \approx D_j^{-1}$ at deeper recursion levels (see Appendix C for more details on the relations between recursion levels). Therefore, 'incompleteness' at deeper recursion levels also increases the incompleteness at the higher levels. An example of an outcome of Algorithm 9 is discussed in Section 5.5 later on.

Once the BILU decompositions are obtained at all levels, the result z of applying the preconditioner to r can be obtained similar to Algorithm 8, except that whenever the inverse of a tridiagonal matrix D_{j-1}^{-1} is required, the algorithm is called recursively:

Algorithm 10
Computes the result $z \approx A^{-1}r$ for a matrix (with a similar structure as) A, assuming that a BILU decomposition is precomputed (in terms of D_1,...,D_n) by Algorithm 9
1) **if** A is a scalar, compute $z = A^{-1}r$ directly, **else**:
2) $s_1 := r_1$
3) **for** $j = 2, 3, ..., n$ **do**
4) $s_j := r_j - B_j D_{j-1}^{-1} s_{j-1}$ (by applying this algorithm recursively)
5) **end**
6) $z_{n_k} := D_n^{-1} s_n$ (by applying this algorithm recursively)
7) **for** $j = n-1, n-2, ..., 1$ **do**
8) $z_j := D_j^{-1}(s_j - C_j z_{j+1})$ (by applying this algorithm recursively)
9) **end**

As before in Algorithm 8, only the matrices D_j are required to execute this algorithm; the matrices L_j and U_j are not needed (cf. (5.4)). Furthermore, because the algorithm is applied recursively, the incompleteness at deeper recursion levels increases the incompleteness at the higher levels. An example of an outcome of Algorithm 10 is discussed in Section 2.4.

Finally, it should be stressed that, even if Algorithm 10 is applied using the results of an exact block LU factorisation (cf. Algorithm 7), the preconditioner can *not* be interpreted as a direct solver. This is due to the recursive applications in lines 4, 6 and 8. In those lines, it is assumed that the matrices D_j have a tridiagonal structure. The latter is not the case (in general) for exact block LU decompositions.

Remark 5.2.1 (Modified BILU)
Inspired by the latter, Algorithm 10 could be modified such that the inverses D_j^{-1} in lines 4, 6 and 8 are handled by performing the multiplication with $T_j \approx D_j^{-1}$ rather than the recursive approach. The resulting modified BILU preconditioner can be interpreted as a direct solver, if the block LU factorisation is exact, and if the inverses $T_j = D_j^{-1}$ are exact as well. However, if the factorisation and the inverses T_j are not exact, the accuracy of the modified BILU algorithm can be worse than the original BILU algorithm. This is numerically validated in Section 5.5 later on. ⌟

5.3 A More General BILU Framework

The BILU preconditioner discussed in the previous section leans heavily on the recursive block tridiagonal structure of the matrix. Unfortunately, the coefficient matrix resulting from an SIPG discretisation of the Poisson equation generally only has this structure for polynomial degree $p = 0$

and certain types of meshes only. In order to modify the BILU algorithm to make it suitable for more general discretisations, this section discusses the work of Meijerink [Mei83], which can be seen as a generalisation of the ideas in [MvdV77].

Consider a matrix of the form:

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & A_{22} & \dots & A_{2N} \\ \vdots & & & \\ A_{N1} & A_{N2} & \dots & A_{NN} \end{bmatrix}.$$

The diagonal submatrices A_{ii} are square, but they do not have to be of equal size. It is assumed that A is an *M-matrix* (cf. e.g. [HJ91, p. 114] for equivalent definitions).

Next, we construct a splitting

$$A = LU - R, \tag{5.6}$$

such that

i) L is a block lower-triangular matrix with identity matrices on the diagonal;

ii) U is a block upper-triangular matrix;

iii) $(LU)^{-1} \geq 0$ and $R \geq 0$, so that the splitting is regular (by definition, cf. e.g. [Saa00, definition 4.1]).

The matrices L, U and R are of the form:

$$\begin{aligned} L &:= (L^{N-1}L^{N-2}...L^1)^{-1}, \\ U &:= A^N, \\ R &:= R^1 + Q^1 + R^2 + Q^2 + ... + Q^{N-1} + R^N, \end{aligned}$$

where the matrices A^k, L^k, R^k and Q^k are formed in stage k of the following process. The main idea is to gradually transform the matrix A into a block upper-triangular matrix. To this end, initially, we define $A^0 := A$. In stage $k \in \{1, 2, ..., N\}$, the following two steps are taken to define A^k, L^k, R^k and Q^k:

1) Consider the M-matrix A^{k-1}. Choose a matrix $\widetilde{A}^k$ that approximates A^{k-1} such that

   i) $A^{k-1}_{kk} = \widetilde{A}^k_{kk} - R^k_{kk}$ is a regular splitting, and $\widetilde{A}^k_{kk}$ can easily be 'inverted' by solving a linear system.

   ii) $A^{k-1}_{ik} \leq \widetilde{A}^k_{ik} \leq 0$ for all $i > k$ (in this chapter, inequalities for matrices should be interpreted elementwise);

   iii) $A^{k-1}_{kj} \leq \widetilde{A}^k_{kj} \leq 0$ for all $j > k$;

   iv) $\widetilde{A}^k_{ij} = A^{k-1}_{ij}$ for all other blocks.

   It can be verified that $\widetilde{A}^k$ is an M-matrix and that $R^k := \widetilde{A}^k - A^{k-1} \geq 0$.

2) Next, each row $i > k$ in $\widetilde{A}^k$ is eliminated by premultiplying row k in $\widetilde{A}^k$ with $\widetilde{A}^k_{ik}\left(\widetilde{A}^k_{kk}\right)^{-1}$ and subtracting the result from row i:

$$\widetilde{A}^k_{ij} := \widetilde{A}^k_{ij} - \widetilde{A}^k_{ik}\left(\widetilde{A}^k_{kk}\right)^{-1}\widetilde{A}^k_{kj}, \qquad \text{for all } j \geq i > k.$$

(for $j < i$ the result would be zero, which is actually the point of this step). To avoid fill-in, the term $\widetilde{A}^k_{ik}\left(\widetilde{A}^k_{kk}\right)^{-1}\widetilde{A}^k_{kj}$ shall be approximated by a sparse matrix M^k_{ij}:

$$\widetilde{A}^k_{ij} :\approx \widetilde{A}^k_{ij} - \underbrace{\widetilde{A}^k_{ik}\left(\widetilde{A}^k_{kk}\right)^{-1}\widetilde{A}^k_{kj}}_{\approx M^k_{ij}} + \underbrace{\widetilde{A}^k_{ik}\left(\widetilde{A}^k_{kk}\right)^{-1}\widetilde{A}^k_{kj} - M^k_{ij}}_{=:Q^k_{ij}} =: A^k_{ij}, \qquad \text{for all } j \geq i > k.$$

In other words:

$$A^k := L^k \widetilde{A}^k + Q^k,$$

where L^k is the identity matrix (of the same size as A), except for column k:

$$L^k := \begin{bmatrix} I & & & & & & \\ & \ddots & & & & & \\ & & I & & & & \\ & & & I & & & \\ & & & -\widetilde{A}^k_{k+1,k}\left(\widetilde{A}^k_{kk}\right)^{-1} & I & & \\ & & & \vdots & & \ddots & \\ & & & -\widetilde{A}^k_{Nk}\left(\widetilde{A}^k_{kk}\right)^{-1} & & & I \end{bmatrix}.$$

Observe that L^k is exact in the sense that it is not expressed in terms of M^k. Furthermore, the nonzero values of Q^k read, for $j \geq i > k$:

$$Q^k_{ij} := \widetilde{A}^k_{ik}\left(\widetilde{A}^k_{kk}\right)^{-1}\widetilde{A}^k_{kj} - M^k_{ij}.$$

Furthermore, the matrix M^k_{ij} approximates $\widetilde{A}^k_{ik}\left(\widetilde{A}^k_{kk}\right)^{-1}\widetilde{A}^k_{kj}$ such that

i) 'too much' fill-in is avoided

ii) A^k is an M-matrix (in order to obtain this property, it can be used that $\widetilde{A}^k_{ik}\left(\widetilde{A}^k_{kk}\right)^{-1}\widetilde{A}^k_{kj} \geq 0$)

Finally, the following remarks can be made:

- In order to use the splitting (5.6) as a preconditioner, it is often not necessary to perform all the computations above. For instance, for block triangular matrices (with arbitrary sparsity pattern), (an approximation of) Algorithm 8 could be used, which only requires the diagonal elements of U (cf. (5.4)).
- The method "can be modified such that either the row-sums or the column-sums of the error matrix $R = A - LU$ are equal to zero". The resulting splitting is not regular anymore, but the "condition of $(LU)^{-1}A$" may be better, e.g. scale with $\frac{1}{h}$ rather than $\frac{1}{h^2}$, depending on the model problem.
- Clearly, the choices for $\widetilde{A}^k$ (in step 1), and M^k (in step 2), determine the efficiency and accuracy of the decomposition. Examples are considered in the next section.

5.4 Examples

The recursive BILU decomposition discussed in Section 5.2 can be formulated in terms of the more general BILU framework that was formulated in Section 5.3. As a matter of fact, the recursive decomposition defined by Algorithm 9 is one of the main examples considered in [Mei83].

To indicate the link between the two approaches, suppose that A is a block tridigonal matrix:

$$A := \begin{bmatrix} A_{11} & A_{12} & & \\ A_{21} & A_{22} & \ddots & \\ & \ddots & \ddots & A_{N-1,N} \\ & & A_{N,N-1} & A_{N,N} \end{bmatrix}.$$

As in Section 5.2, the diagonal blocks are assumed to be (regular) tridiagonal matrices, and the off-diagonal blocks are assumed to be diagonal matrices. We will now discuss specific choices for $\widetilde{A}^k$, and M^k in the algorithm in Section 5.3, so that the method becomes equivalent to Algorithm 9.

To this end, note that for this block tridiagonal matrix, the matrix L^k contains only one non-trivial element:

$$L^k := \begin{bmatrix} I & & & & & & & \\ & \ddots & & & & & & \\ & & I & & & & & \\ & & & I & & & & \\ & & & -\widetilde{A}^k_{k+1,k}\left(\widetilde{A}^k_{kk}\right)^{-1} & I & & \\ & & & 0 & & I & & \\ & & & \vdots & & & \ddots & \\ & & & 0 & & & & I \end{bmatrix}.$$

As a consequence, only the diagonal element $A^k_{k+1,k+1}$ needs to be computed in step 2 of the algorithm; all other elements are either zero or unchanged. Thus, only $M^k_{k+1,k+1}$ needs to be chosen. We choose:

$$\widetilde{A}^k = A^{k-1}, \qquad M^k_{k+1,k+1} = T\big(\widetilde{A}^k_{k+1,k}(\widetilde{A}^k_{kk})^{-1}\widetilde{A}^k_{k,k+1}\big),$$

where the notation $T(B)$ denotes the tridiagonal matrix of which the three diagonals are equal to the corresponding diagonals of B. Since the off-diagonal elements of $\widetilde{A}^k$ are diagonal matrices, we may write:

$$M^k_{k+1,k+1} = \widetilde{A}^k_{k+1,k}T\big((\widetilde{A}^k_{kk})^{-1}\big)\widetilde{A}^k_{k,k+1},$$

It becomes clear from the claim in [Mei83, Appendix 1] that $T\big((\widetilde{A}^k_{kk})^{-1}\big)$ is equivalent to the result T of applying Algorithm 9, using the input $\widetilde{A}^k_{kk}$ for A.

Other choices for $M^k_{k+1,k+1}$ that are given in [Mei83] for the tridiagonal matrix above read:

- $M^k_{k+1,k+1} = 0$ (in this case, the block upper-diagonal elements are not updated in the second step of the algorithm);
- $M^k_{k+1,k+1} = \widetilde{A}^k_{k+1,k}D\big((\widetilde{A}^k_{kk})^{-1}\big)\widetilde{A}^k_{k,k+1}$, where $D(B)$ indicates the diagonal matrix of which the diagonal is equal to B. Note that a strategy to compute $D(B^{-1})$ can be derived from the strategy to compute $T(B^{-1})$ by simply not computing the off-diagonal elements, and setting them equal to zero instead.
- $M^k_{k+1,k+1} = \widetilde{A}^k_{k+1,k}\big(D^{-1}_{kk}(\widetilde{A}^k_{kk} - D_{kk})D^{-1}_{kk}\big)\widetilde{A}^k_{k,k+1}$, where $D_{kk} = D(\widetilde{A}^k_{kk})$ and $D^{-1}_{kk}(\widetilde{A}^k_{kk} - D_{kk})D^{-1}_{kk}$ is an approximation for $\left(\widetilde{A}^k_{kk}\right)^{-1}$ consisting of the first two terms of the Taylor series expansion of $\left(\widetilde{A}^k_{kk}\right)^{-1}$ around D_{kk}.

5.5 Numerical Example

In this section, we investigate the performance of the BILU preconditioner formulated in Section 5.2 numerically. To this end, we consider the SIPG method for the two-dimensional Poisson problem introduced in Section 2.4.

For $p = 0$, the matrix A has the required recursive tridiagonal structure, since it is equivalent to the matrix resulting from a central difference scheme in that case (cf. Figure 2.1). An illustration of the outcome of the BILU decomposition defined by Algorithm 9 (with $n = N_x = N_y$) for this case can be found in Figure 5.1 for 3×3 elements and 4×4 elements. The following definitions and conclusions can be stated.

First, note that the matrices A_j (only the cases with $j = 1$ and $j = n$ are displayed) are the familiar tridiagonal matrices corresponding to the diagonal blocks of the coefficient matrix resulting from a central difference discretisation. Next, observe that the matrices D_j and $T_j \approx D_j^{-1}$ are tridiagonal matrices, just like A_j. Indeed, this was how the algorithm was designed.

Next, we are interested in the errors in D_j and T_j, which are denoted by ΔD_j and ΔT_j respectively. The matrix ΔD_j denotes the difference between D_j, as defined by Algorithm 9, and the 'exact version' of D_j, as would be the result of Algorithm 7. The error $\Delta D_1 = 0$, since $D_1 = A_1$ is not approximated. The errors ΔD_n appear to be relatively small.

The matrix ΔT_j is the difference between T_j, as computed by Algorithm 9, and D_j^{-1}, as would be the result of computing the inverse of the approximation (!) D_j exactly. It can be observed that $\Delta T_j = 0$ on the three 'main' diagonals, as predicted in [Mei83] (cf. Section 5.4). Outside these three diagonals, the errors appear to be relatively small.

Finally, we consider the performance of the preconditioner defined by Algorithm 10. We use the CG method with a random start vector and the stopping criterion (note that we used a different tolerance in earlier CG examples):

$$\frac{\|r_k\|_2}{\|b\|_2} \leq 10^{-3}.$$

Figure 5.2 compares the convergence during the iterations without preconditioning, with the BILU preconditioner, and with a modified version of the BILU preconditioner (defined in Remark 5.2.1). The (original) BILU preconditioner reduces the number of iterations by about a factor 7 (compared to the unpreconditioned case). Furthermore, it can be seen that the modified BILU algorithm performs worse than the original algorithm.

5.6 Conclusion

This chapter studies existing BILU algorithms as a starting point for our research. An exact Block LU factorisation of a matrix A is of the form $A = LDU$, where D is a block diagonal matrix and L and U are block lower and upper triangular matrices respectively with identity blocks on the diagonal. The main idea behind a BILU preconditioner is to make use of an approximation of such a factorisation. An example of such a preconditioner is the recursive BILU preconditioner. It was demonstrated numerically that this algorithm can effectively enhance the convergence of CG for matrices with a recursive block tridiagonal structure. Unfortunately, the coefficient matrix resulting from an SIPG discretisation of the Poisson equation generally only has this structure for polynomial degree $p = 0$ and certain types of meshes only. Furthermore, this chapter discusses a general framework that includes the recursive BILU preconditioner. It is the main challenge of this research to find a generalisation of the recursive BILU algorithm within this framework that is suitable for our application.

$$A_1 = \begin{bmatrix} 4.000 & -1.000 & 0.000 \\ -1.000 & 4.000 & -1.000 \\ 0.000 & -1.000 & 4.000 \end{bmatrix} \qquad A_3 = \begin{bmatrix} 4.000 & -1.000 & 0.000 \\ -1.000 & 4.000 & -1.000 \\ 0.000 & -1.000 & 4.000 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 4.000 & -1.000 & 0.000 \\ -1.000 & 4.000 & -1.000 \\ 0.000 & -1.000 & 4.000 \end{bmatrix} \qquad D_3 = \begin{bmatrix} 3.705 & -1.093 & 0.000 \\ -1.093 & 3.677 & -1.093 \\ 0.000 & -1.093 & 3.705 \end{bmatrix}$$

$$T_1 = \begin{bmatrix} 0.268 & 0.071 & 0.000 \\ 0.071 & 0.286 & 0.071 \\ 0.000 & 0.071 & 0.268 \end{bmatrix} \qquad T_3 = \begin{bmatrix} 0.299 & 0.097 & 0.000 \\ 0.097 & 0.330 & 0.097 \\ 0.000 & 0.097 & 0.299 \end{bmatrix}$$

$$\Delta D_1 = \begin{bmatrix} 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 \end{bmatrix} \qquad \Delta D_3 = \begin{bmatrix} 0.000 & 0.001 & 0.028 \\ 0.001 & 0.000 & 0.001 \\ 0.028 & 0.001 & 0.000 \end{bmatrix}$$

$$\Delta T_1 = \begin{bmatrix} 0.000 & 0.000 & -0.018 \\ 0.000 & 0.000 & 0.000 \\ -0.018 & 0.000 & 0.000 \end{bmatrix} \qquad \Delta T_3 = \begin{bmatrix} 0.000 & 0.000 & -0.029 \\ 0.000 & 0.000 & 0.000 \\ -0.029 & 0.000 & 0.000 \end{bmatrix}$$

(a) 3×3 elements

$$A_1 = \begin{bmatrix} 4.000 & -1.000 & 0.000 & 0.000 \\ -1.000 & 4.000 & -1.000 & 0.000 \\ 0.000 & -1.000 & 4.000 & -1.000 \\ 0.000 & 0.000 & -1.000 & 4.000 \end{bmatrix} \qquad A_4 = \begin{bmatrix} 4.000 & -1.000 & 0.000 & 0.000 \\ -1.000 & 4.000 & -1.000 & 0.000 \\ 0.000 & -1.000 & 4.000 & -1.000 \\ 0.000 & 0.000 & -1.000 & 4.000 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 4.000 & -1.000 & 0.000 & 0.000 \\ -1.000 & 4.000 & -1.000 & 0.000 \\ 0.000 & -1.000 & 4.000 & -1.000 \\ 0.000 & 0.000 & -1.000 & 4.000 \end{bmatrix} \qquad D_4 = \begin{bmatrix} 3.701 & -1.099 & 0.000 & 0.000 \\ -1.099 & 3.665 & -1.110 & 0.000 \\ 0.000 & -1.110 & 3.665 & -1.099 \\ 0.000 & 0.000 & -1.099 & 3.701 \end{bmatrix}$$

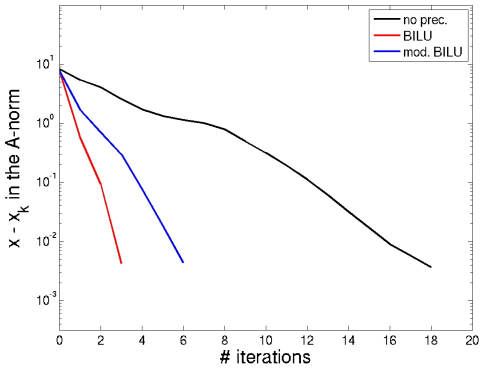
$$T_1 = \begin{bmatrix} 0.268 & 0.072 & 0.000 & 0.000 \\ 0.072 & 0.287 & 0.077 & 0.000 \\ 0.000 & 0.077 & 0.287 & 0.072 \\ 0.000 & 0.000 & 0.072 & 0.268 \end{bmatrix} \qquad T_4 = \begin{bmatrix} 0.300 & 0.100 & 0.000 & 0.000 \\ 0.100 & 0.337 & 0.112 & 0.000 \\ 0.000 & 0.112 & 0.337 & 0.100 \\ 0.000 & 0.000 & 0.100 & 0.300 \end{bmatrix}$$

$$\Delta D_1 = \begin{bmatrix} 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix} \qquad \Delta D_4 = \begin{bmatrix} 0.001 & 0.002 & 0.037 & 0.012 \\ 0.002 & 0.002 & 0.003 & 0.037 \\ 0.037 & 0.003 & 0.002 & 0.002 \\ 0.012 & 0.037 & 0.002 & 0.001 \end{bmatrix}$$

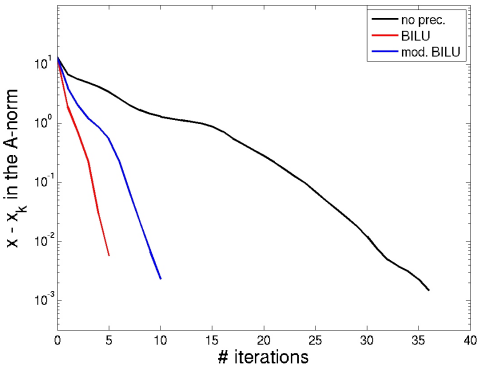
$$\Delta T_1 = \begin{bmatrix} 0.000 & 0.000 & -0.019 & -0.005 \\ 0.000 & 0.000 & 0.000 & -0.019 \\ -0.019 & 0.000 & 0.000 & 0.000 \\ -0.005 & -0.019 & 0.000 & 0.000 \end{bmatrix} \qquad \Delta T_4 = \begin{bmatrix} 0.000 & 0.000 & -0.033 & -0.010 \\ 0.000 & 0.000 & 0.000 & -0.033 \\ -0.033 & 0.000 & 0.000 & -0.000 \\ -0.010 & -0.033 & 0.000 & 0.000 \end{bmatrix}$$

(b) 4×4 elements

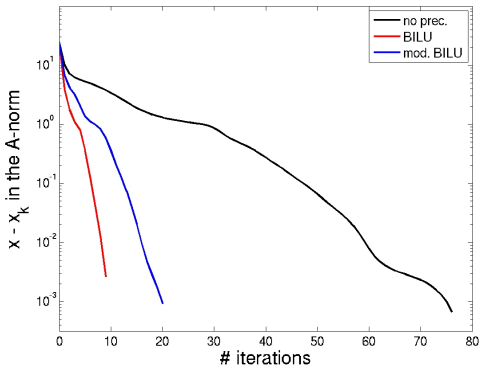
Figure 5.1: BILU decomposition (Algorithm 9) for an SIPG discretisation with $p = 0$ and $\eta_0 = 1$ for the two-dimensional Poisson equation



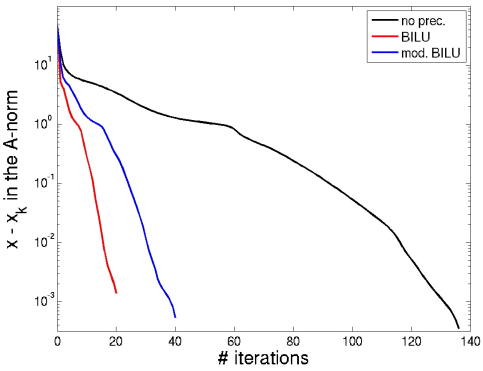
(a) 10×10 mesh elements



(b) 20×20 mesh elements



(c) 40×40 mesh elements



(d) 80×80 mesh elements

Figure 5.2: Convergence of CG in A-norm $\sqrt{(x - x_k)^T A(x - x_k)}$: BILU preconditioning reduces the number of iterations.

Chapter 6

Conclusion & Research questions

Discontinuous Galerkin (DG) finite element methods for elliptic problems approximate the solution in the form of piecewise polynomials of degree p. The main advantages of these methods are the flexibility in handling non-matching grids and in designing hp-refinement strategies. An important drawback is that the resulting linear systems are usually large (due to the large number of degrees of freedom), and ill-conditioned. Therefore, efficient iterative algorithms are required to minimize the computational costs and increase the practical applicability of DG methods. Two main strategies to tackle this problem are preconditioning and deflation.

Recently, new preconditioning techniques have been proposed for DG methods, such as multigrid, Schwarz domain decomposition, and subspace decomposition. It appears that BILU preconditioners, which have been rather successful for finite difference methods, have received little attention so far in this field. Therefore, we choose this to be our research direction. Furthermore, we will restrict ourselves (initially) to matrices resulting from SIPG discretisations for Poisson equations.

An example of an efficient BILU method is the recursive BILU preconditioner. Unfortunately, this method is only applicable to matrices with a recursive block tridiagonal structure. As a consequence, it can not be used readily for our application, because the corresponding coefficient matrix does not have the required structure in general.

Instead, the matrix has the following properties: it is an $N \times N$ block matrix, where N is the number of mesh elements, and each block in turn is an $M \times M$ matrix, where M is the dimension of the polynomial test space (within one mesh element). For polynomial degree $p = 0$, the matrix is equivalent to a matrix resulting from a central finite difference method. For $p > 0$, the nonzero pattern is similar, except that it now consists of (full) $M \times M$ blocks (cf. Figure 1.1 and Figure 2.1). It can be shown that the matrix is Symmetric and Positive-Definite (SPD) (provided that the penalty parameter η_0 is sufficiently large). Furthermore, it is well-known that the condition number is $O(h^{-2})$ for quasi-uniform meshes with element diameter h. The latter is the main reason why CG (without preconditioning) requires $O(h^{-1})$ iterations for convergence.

> *The main goal of this research is to enhance the efficiency of CG for linear systems resulting from SIPG discretisations for Poisson equations through an optimal combination of a generalisation of the recursive BILU preconditioner and deflation.*

To this end, we will seek an answer to the following research questions:

1) How can we generalise the decomposition defined in Algorithm 9?

   i) We could simply nullify all elements outside the required sparsity pattern, or lump them onto the diagonal. What is the (expected) performance of the resulting two algorithms?
   ii) Another idea is to first apply a p-multigrid preconditioner, so that the system at the coarsest level has the required sparsity pattern. What is the performance of the resulting combination of preconditioners?

iii) We could implement another version of the algorithm defined in Section 5.3. Possibilities are specified in Section 5.4. What is their (expected) performance?
iv) What are other possibilities for the algorithm defined in Section 5.3 (in terms of the matrices $\widetilde{A}^k$, and M^k)? What is their (expected) performance?
v) What other literature is related to [Mei83]?
vi) Incomplete Cholesky factorisations (IC, MIC, RIC) have been used often as a preconditioner for the CG method. Can we make use of the underlying ideas of these factorisations?
vii) It is stated in [Mei83] that the BILU decomposition "can be modified such that either the row-sums or the column-sums of the error matrix $R = A - LU$ are equal to zero". The resulting splitting is not regular anymore, but the "condition of $(LU)^{-1}A$" may be better, e.g. scale with $\frac{1}{h}$ rather than $\frac{1}{h^2}$ (where h is the mesh element diameter), depending on the model problem. For our application, do we prefer zero row-sums (or column-sums) to regularity?

2) How can we generalise the solver defined in Algorithm 10?

i) We could simply nullify all elements outside the required sparsity pattern, or lump them onto the diagonal. What is the (expected) performance of the resulting two algorithms?
ii) Appleyard [AC83] also studies recursive solvers for BILU decompositions: What is the precise implementation proposed in that reference? How does it relate to Algorithm 10?
iii) What other literature is related to [AC83]?

3) What type of deflation is suitable to complement the new preconditioner resulting from the answers to the previous questions?

Here, we will measure the 'performance' in terms of:

- the condition number,
- the number of iterations required for PCG to convergence (compared to the unpreconditioned case, or to a standard preconditioner (e.g. Jacobi)),
- the variations in the convergence speed per iteration (stagnation and accelaration during the convergence process),
- the CPU time (or flops) required to construct the decomposition,
- the CPU time (or flops) required to apply the solver,
- and the number of nonzero elements in the decomposition (fill-in).

Other questions are:

- What are the main drawbacks of the existing preconditioners for DG discretisations for elliptic problems?
- The SIPG method only leads to a positive definite matrix provided that the penalty parameter is sufficiently large [GK03, p. 531]. A similar condition is required for convergence of the SIPG solution [Riv08, Chapter 1, Chapter 2]. What is an explicit (computable) expression for a sufficiently high lower bound for the penalty parameter?

Appendix A

Derivation of DG Methods

This chapter provides the derivations for the DG methods formulated in Chapter 1. First, some required preliminaries are derived in Section A.1. Section A.2 considers interior penalty methods. Section A.3 discusses DG methods in the unified framework [ABCM02].

A.1 Preliminaries

In order to derive the DG formulations, we need the following preliminaries (see Section 1.1 for notational aspects):

$$\sum_{i=0}^{N} [uv]_i = \sum_{i=0}^{N} [u]_i \{v\}_i + \sum_{n=1}^{N-1} \{u\}_i [v]_i \,, \qquad \text{or equivalently:} \tag{A.1}$$

$$\sum_{i=0}^{N} [uv]_i = \sum_{i=1}^{N-1} [u]_i \{v\}_i + \sum_{n=0}^{N} \{u\}_i [v]_i \,. \tag{A.2}$$

Notice the difference in the bounds of the sum.

These preliminaries can be derived as follows. First, observe that if follows from (1.2) that, at the interior element boundaries x_i with $i = 1, ..., N-1$:

$$[uv]_i = [u]_i \{v\}_i + \{u\}_i [v]_i \,.$$

Similarly, at the domain boundary, we obtain from (1.3):

$$[uv]_0 = -u_1(x_0)v_1(x_0) = \begin{cases} [u]_0 \{v\}_0 \,, & \text{or equivalently,} \\ \{u\}_0 [v]_0 \,, \end{cases}$$

$$[uv]_N = u_N(x_N)v_N(x_N) = \begin{cases} [u]_N \{v\}_N \,, & \text{or equivalently,} \\ \{u\}_N [v]_N \,. \end{cases}$$

Summing over all element boundaries with index $i = 0, ..., N$, we obtain (A.1) and (A.2).

A.2 Interior Penalty Methods

This section derives the IP formulation defined by (1.4) and (1.5). We follow [Riv08, Chapter 1]: first, multiply the model equation (1.1) by a test function v, integrate over a mesh element, and apply integration by parts:

$$\int_{x_{i-1}}^{x_i} u'v' - u_i'(x_i)v_i(x_i) + u_i'(x_{i-1})v_i(x_{i-1}) = \int_{x_{i-1}}^{x_i} fv.$$

Sum over all mesh elements:

$$\int_\Omega u'v' - \sum_{i=0}^{N} [u'v]_i = \int_\Omega fv. \tag{A.3}$$

Rewrite the second term using (A.1) and the fact that $[u']_i = 0$ for all $i = 1, ..., N-1$ (due to the continuity of u'):

$$\int_\Omega u'v' - \sum_{i=0}^{N} \{u'\}_i [v]_i = \int_\Omega fv,$$

Because the boundary conditions are homogeneous, and because u is continuous, it follows that $[u]_i = 0$ for all $i = 0, ..., N$. Thus, we may write for any $\epsilon \in \mathbb{R}$:

$$\int_\Omega u'v' - \sum_{i=0}^{N} \{u'\}_i [v]_i + \epsilon \sum_{i=0}^{N} [u]_i \{v'\}_i = \int_\Omega fv.$$

The final IP formulation is obtained by substituting the approximation $u_h \in V$ for u and adding terms that penalize the jumps in the functions u_h and v:

$$\int_\Omega u_h'v' - \sum_{i=0}^{N} \{u_h'\}_i [v]_i + \epsilon \sum_{i=0}^{N} [u_h]_i \{v'\}_i + \sum_{i=0}^{N} \frac{\eta_0}{h} [u_h]_i [v]_i = \int_\Omega fv.$$

This completes the derivation of the IP formulation defined by (1.4) and (1.5). It should be noted that there are also IP methods that penalize the jumps in the derivatives of u_h and v (cf. [Riv08, Chapter 1]). These methods are not considered here.

A.3 DG Methods in the Unified Framework

This section derives the DG formulation defined by (1.6) and (1.7). We follow [ABCM02, Section 3.2]: first, rewrite the second-order system (1.1) as a first-order system by defining $\sigma := u'$:

$$\begin{aligned} \sigma(x) &= u'(x), \\ -\sigma'(x) &= f(x). \end{aligned}$$

Multiply both equations with test functions τ and v respectively, integrate over an element, and apply integration by parts:

$$\begin{aligned} \int_{x_{i-1}}^{x_i} \sigma\tau - u_i(x_i)\tau_i(x_i) + u_i(x_{i-1})\tau_i(x_{i-1}) &= -\int_{x_{i-1}}^{x_i} u\tau', \\ \int_{x_{i-1}}^{x_i} \sigma v' - \sigma_i(x_i)v_i(x_i) + \sigma_i(x_{i-1})v_i(x_{i-1}) &= \int_{x_{i-1}}^{x_i} fv. \end{aligned}$$

This is a weak formulation of (1.1) if $[x_{i-1}, x_i]$ is interpreted as any subinterval of Ω, rather than a mesh element. Next, let the test functions $u_h \in V$ and $\sigma_h \in V \times V$ denote an approximation for u and $\sigma = u'$ in the element interiors. Similarly, let the piecewise continuous functions $\hat{u} \in V$ and $\hat{\sigma} \in V \times V$ denote an approximation for u and $\sigma = u'$ at the element boundaries. These numerical fluxes are allowed to be discontinuous at the element boundaries. Next, consider the weak formulation for each mesh element, sum the results, and substitute the aforementioned approximations:

$$\begin{aligned} \int_\Omega \sigma_h\tau - \sum_{i=1}^{N} \Big(\hat{u}_i(x_i)\tau_i(x_i) - \hat{u}_i(x_{i-1})\tau_i(x_{i-1})\Big) &= -\int_\Omega u_h\tau', \\ \int_\Omega \sigma_h v' - \sum_{i=1}^{N} \Big(\hat{\sigma}_i(x_i)v_i(x_i) - \hat{\sigma}_i(x_{i-1})v_i(x_{i-1})\Big) &= \int_\Omega fv. \end{aligned}$$

This can be rewritten in terms of the jump operator:

$$\int_\Omega \sigma_h \tau - \sum_{i=0}^{N} [\hat{u}\tau]_i = -\int_\Omega u_h \tau'$$
$$\int_\Omega \sigma_h v' - \sum_{i=0}^{N} [\hat{\sigma} v]_i = \int_\Omega f v,$$

where, $\hat{u}$ and $\hat{\sigma}$ are numerical fluxes that approximate u and σ on the element boundaries. Depending on the choice of these numerical fluxes, different types of DG methods are obtained. Application of (A.1) to the first and (A.2) to the second equation gives:

$$\int_\Omega \sigma_h \tau - \sum_{i=0}^{N} [\hat{u}]_i \{\tau\}_i - \sum_{i=1}^{N-1} \{\hat{u}\}_i [\tau]_i = -\int_\Omega u_h \tau', \tag{A.4}$$

$$\int_\Omega \sigma_h v' - \sum_{i=1}^{N-1} [\hat{\sigma}]_i \{v\}_i - \sum_{i=0}^{N} \{\hat{\sigma}\}_i [v]_i = \int_a^b f v. \tag{A.5}$$

Using integration by parts and (A.1), the right hand side of (A.4) can be rewritten as:

$$-\int_\Omega u_h \tau' = \int_\Omega u_h' \tau - \sum_{i=0}^{N} [u_h]_i \{\tau\}_i - \sum_{i=1}^{N-1} \{u_h\}_i [\tau]_i .$$

Substitution of this expression into (A.4) yields:

$$\int_\Omega \sigma_h \tau - \sum_{i=0}^{N} [\hat{u} - u_h]_i \{\tau\}_i - \sum_{i=1}^{N-1} \{\hat{u} - u_h\}_i [\tau]_i = \int_\Omega u_h' \tau.$$

Substituting of $\tau = v'$ in this expression, and applying the result to the first term in (A.5), we obtain (1.6) and (1.7).

Appendix B

Computing the Coefficient Matrix for an IP scheme

In Chapter 1 and Chapter 2, DG methods for elliptic problems were considered. These methods require the solution of a linear system of the form (1.11), (1.12). This chapter discusses in detail how the corresponding coefficient matrix can be computed for IP methods. The one-dimensional case is considered in Section B.1. The two-dimensional case in Section B.2.

B.1 One-dimensional case

This section derives an explicit expression for the linear system of the form (1.11), (1.12) resulting from an IP method for the one-dimensional Poisson equation. To this end, consider the bilinear form (1.5) (see Section 1.1 for notational aspects):

$$B(u,v) \quad = \quad \int_\Omega u'v' + \sum_{i=0}^{N} \left(- \{u'\}_i [v]_i + \epsilon [u]_i \{v'\}_i + \frac{\eta_0}{h} [u]_i [v]_i \right).$$

In the interior, for $i = 1, ..., N-1$, we can use the definition of the jump and average operator (1.2) to write:

$$\begin{aligned}
- \{u'\}_i [v]_i &= -\frac{1}{2}(u_i'(x_i) + u_{i+1}'(x_i))(v_i(x_i) - v_{i+1}(x_i)) \\
&= -\frac{1}{2}u_i'(x_i)v_i(x_i) + \frac{1}{2}u_i'(x_i)v_{i+1}(x_i) - \frac{1}{2}u_{i+1}'(x_i)v_i(x_i) + \frac{1}{2}u_{i+1}'(x_i)v_{i+1}(x_i), \\
\epsilon [u]_i \{v'\}_i &= \frac{\epsilon}{2}(u_i - u_{i+1})(v_i' + v_{i+1}') \\
&= \frac{\epsilon}{2}u_i(x_i)v_i'(x_i) + \frac{\epsilon}{2}u_i(x_i)v_{i+1}'(x_i) - \frac{\epsilon}{2}u_{i+1}(x_i)v_i'(x_i) - \frac{\epsilon}{2}u_{i+1}(x_i)v_{i+1}'(x_i), \\
\frac{\eta_0}{h} [u]_i [v]_i &= \frac{\eta_0}{h}(u_i(x_i) - u_{i+1}(x_i))(v_i(x_i) - v_{i+1}(x_i)) \\
&= \frac{\eta_0}{h}u_i(x_i)v_i(x_i) - \frac{\eta_0}{h}u_i(x_i)v_{i+1}(x_i) - \frac{\eta_0}{h}u_{i+1}(x_i)v_i(x_i) + \frac{\eta_0}{h}u_{i+1}(x_i)v_{i+1}(x_i).
\end{aligned}$$

Similarly, at the boundary, for $i = 0, N$, we can use the definition of the jump and average operator (1.3) to write:

$$\begin{aligned}
- \{u'\}_0 [v]_0 &= u_1'(x_0)v_1(x_0), & - \{u'\}_N [v]_N &= -u_N'(x_N)v_N(x_N), \\
\epsilon [u]_0 \{v'\}_0 &= -\epsilon u_1(x_0)v_1'(x_0), & \epsilon [u]_N \{v'\}_N &= \epsilon u_N(x_N)v_N'(x_N), \\
\frac{\eta_0}{h} [u]_0 [v]_0 &= \frac{\eta_0}{h}u_1(x_0)v_1(x_0), & \frac{\eta_0}{h} [u]_N [v]_N &= \frac{\eta_0}{h}u_N(x_N)v_N(x_N).
\end{aligned}$$

Using these expressions, the bilinear form can be rewritten in terms of volume integrals, contributions of interior edges, and contributions of boundary edges:

$$\begin{aligned} B(u,v) &= \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} u'v' \\ &\quad + \sum_{i=1}^{N-1} \left(D_i^-(u,v) + C_i^-(u,v) + C_i^+(u,v) + D_i^+(u,v) \right) \\ &\quad + D_0^+(u,v) + D_N^-(u,v), \end{aligned}$$

where the contributions from the interior edges read, for $i = 1, ..., N-1$:

$$\begin{aligned} D_i^-(u,v) &:= -\frac{1}{2} u_i'(x_i) v_i(x_i) + \frac{\epsilon}{2} u_i(x_i) v_i'(x_i) + \frac{\eta_0}{h} u_i(x_i) v_i(x_i), \\ C_i^-(u,v) &:= \frac{1}{2} u_i'(x_i) v_{i+1}(x_i) + \frac{\epsilon}{2} u_i(x_i) v_{i+1}'(x_i) - \frac{\eta_0}{h} u_i(x_i) v_{i+1}(x_i), \\ C_i^+(u,v) &:= -\frac{1}{2} u_{i+1}'(x_i) v_i(x_i) - \frac{\epsilon}{2} u_{i+1}(x_i) v_i'(x_i) - \frac{\eta_0}{h} u_{i+1}(x_i) v_i(x_i), \\ D_i^+(u,v) &:= \frac{1}{2} u_{i+1}'(x_i) v_{i+1}(x_i) - \frac{\epsilon}{2} u_{i+1}(x_i) v_{i+1}'(x_i) + \frac{\eta_0}{h} u_{i+1}(x_i) v_{i+1}(x_i), \end{aligned}$$

and the contributions from the edges located at the domain boundary read:

$$\begin{aligned} D_0^+(u,v) &:= u_1'(x_0) v_1(x_0) - \epsilon u_1(x_0) v_1'(x_0) + \frac{\eta_0}{h} u_1(x_0) v_1(x_0), \\ D_N^-(u,v) &:= -u_N'(x_N) v_N(x_N) + \epsilon u_N(x_N) v_N'(x_N) + \frac{\eta_0}{h} u_N(x_N) v_N(x_N). \end{aligned}$$

Taking the local support of the monomial basis functions into account, the diagonal blocks in the matrix A read, for all $i = 1, ..., N$:

$$(A_{ii})_{\ell k} = B(\phi_k^{(i)}, \phi_\ell^{(i)}) = \int_{x_{i-1}}^{x_i} (\phi_k^{(i)})' (\phi_\ell^{(i)})' + D_{i-1}^+(\phi_k^{(i)}, \phi_\ell^{(i)}) + D_i^-(\phi_k^{(i)}, \phi_\ell^{(i)})$$

and the off-diagonal blocks:

$$\begin{aligned} (A_{i+1,i})_{\ell k} &= B(\phi_k^{(i)}, \phi_\ell^{(i+1)}) = C_i^-(\phi_k^{(i)}, \phi_\ell^{(i+1)}), && \text{for all } i = 1, ..., N-1, \\ (A_{i-1,i})_{\ell k} &= B(\phi_k^{(i)}, \phi_\ell^{(i-1)}) = C_{i-1}^+(\phi_k^{(i)}, \phi_\ell^{(i-1)}), && \text{for all } i = 2, ..., N, \end{aligned}$$

All other off-diagonal blocks are zero, since they correspond to non-adjacent elements.

Using the definition of the monomial test functions (1.8), the individual terms in these expressions can be computed. For the volume integrals, we obtain:

$$\begin{aligned} \int_{x_{i-1}}^{x_i} (\phi_k^{(i)})' (\phi_\ell^{(i)})' &= \begin{cases} 0, & \text{for } k = 1, \text{ or } l = 1, \\ \left(\frac{2}{h}\right)^2 (k-1)(\ell-1) \int_{x_{i-1}}^{x_i} \left(\frac{x - x_{i-\frac{1}{2}}}{\frac{1}{2}h} \right)^{k+\ell-4} \mathrm{d}x, & \text{else} \end{cases} \\ &= \begin{cases} 0, & \text{for } k = 1, \text{ or } l = 1, \\ \frac{2}{h} \frac{(k-1)(\ell-1)}{k+\ell-3} (1 + (-1)^{k+\ell}), & \text{else.} \end{cases} \end{aligned}$$

For the edge contributions, it follows that:

$$
\begin{aligned}
D_i^-(\phi_k^{(i)},\phi_\ell^{(i)}) &= \begin{cases} \frac{1}{h}\big(-2(k-1)+2\epsilon(\ell-1)+\eta_0\big), & \text{for } i=N,\\ \frac{1}{h}\big(-(k-1)+\epsilon(\ell-1)+\eta_0\big), & \text{else,}\end{cases}\\
D_{i-1}^+(\phi_k^{(i)},\phi_\ell^{(i)}) &= \begin{cases} \frac{1}{h}\big(-2(k-1)+2\epsilon(\ell-1)+\eta_0\big)(-1)^{k+\ell-2}, & \text{for } i=1,\\ \frac{1}{h}\big(-(k-1)+\epsilon(\ell-1)+\eta_0\big)(-1)^{k+\ell-2}, & \text{else,}\end{cases}\\
C_i^-(\phi_k^{(i)},\phi_\ell^{(i+1)}) &= \frac{1}{h}\big((k-1)-\epsilon(\ell-1)-\eta_0\big)(-1)^{\ell-1}, \qquad \text{for } i=1,\dots N-1,\\
C_{i-1}^+(\phi_k^{(i)},\phi_\ell^{(i-1)}) &= \frac{1}{h}\big((k-1)-\epsilon(\ell-1)-\eta_0\big)(-1)^{k-1}, \qquad \text{for } i=2,\dots N.
\end{aligned}
$$

B.2 Two-dimensional case

This section derives an explicit expression for the linear system of the form (1.11), (1.12) resulting from an IP method for the two-dimensional Poisson equation. To this end, consider the bilinear form (2.6) (see Section 2.1 for notational aspects):

$$
B(u,v) \;=\; \int_\Omega \nabla u\cdot\nabla v + \sum_{e\in E}\int_e \Big(-\{\nabla u\}\cdot[v] + \epsilon\,[u]\cdot\{\nabla v\} + \frac{\eta_0}{|e|}\,[u]\cdot[v]\Big).
$$

For an interior edge $e=\partial K_i\cap\partial K_j\in E^\circ$, we can use the definition of the jump and average operator (2.2) to write:

$$
\begin{aligned}
-\{\nabla u\}\cdot[v] &= -\frac{1}{2}(\nabla u_i+\nabla u_j)\cdot(v_i\boldsymbol{n}_i+v_j\boldsymbol{n}_j)\\
&= -\frac{1}{2}\nabla u_i\cdot(v_i\boldsymbol{n}_i) - \frac{1}{2}\nabla u_i\cdot(v_j\boldsymbol{n}_j) - \frac{1}{2}\nabla u_j\cdot(v_i\boldsymbol{n}_i) - \frac{1}{2}\nabla u_j\cdot(v_j\boldsymbol{n}_j)\\
\epsilon\,[u]\cdot\{\nabla v\} &= \frac{\epsilon}{2}(u_i\boldsymbol{n}_i+u_j\boldsymbol{n}_j)\cdot(\nabla v_i+\nabla v_j)\\
&= \frac{\epsilon}{2}(u_i\boldsymbol{n}_i)\cdot\nabla v_i + \frac{\epsilon}{2}(u_i\boldsymbol{n}_i)\cdot\nabla v_j + \frac{\epsilon}{2}(u_j\boldsymbol{n}_j)\cdot\nabla v_i + \frac{\epsilon}{2}(u_j\boldsymbol{n}_j)\cdot\nabla v_j\\
\frac{\eta_0}{|e|}\,[u]\cdot[v] &= \frac{\eta_0}{|e|}(u_i\boldsymbol{n}_i+u_j\boldsymbol{n}_j)\cdot(v_i\boldsymbol{n}_i+v_j\boldsymbol{n}_j)\\
&= \frac{\eta_0}{|e|}u_iv_i - \frac{\eta_0}{|e|}u_iv_j - \frac{\eta_0}{|e|}u_jv_i + \frac{\eta_0}{|e|}u_jv_j
\end{aligned}
$$

Similarly, for a boundary edge $e=\partial K_i\cap\partial\Omega\in E^\partial$, we can use the definition of the jump and average operator (2.3) to write:

$$
\begin{aligned}
-\{\nabla u\}\cdot[v] &= -(\nabla u_i)\cdot(v_i\boldsymbol{n}_i)\\
\epsilon\,[u]\cdot\{\nabla v\} &= \epsilon(u_i\boldsymbol{n}_i)\cdot\nabla v_i\\
\frac{\eta_0}{|e|}\,[u]\cdot[v] &= \frac{\eta_0}{|e|}(u_i\boldsymbol{n}_i)\cdot(v_i\boldsymbol{n}_i) = \frac{\eta_0}{|e|}u_iv_i
\end{aligned}
$$

Using these expressions, the bilinear form can be rewritten in terms of volume integrals, contributions of interior edges, and contributions of boundary edges:

$$
\begin{aligned}
B(u,v) \;=\; &\sum_{i=1}^N \int_{K_i}\nabla u\cdot\nabla v\\
&+ \sum_{e=\partial K_i\cap\partial K_j\in E^\circ}\big(D_e^{ii}(u,v)+C_e^{ij}(u,v)+C_e^{ji}(u,v)+D_e^{jj}(u,v)\big)\\
&+ \sum_{e=\partial K_i\cap\partial\Omega\in E^\partial} D_e^{ii}(u,v),
\end{aligned}
$$

where the contributions from the interior edge $e = \partial K_i \cap \partial K_j \in E^\circ$ read:

$$D_e^{ii}(u,v) := \int_e \left(-\frac{1}{2}\nabla u_i \cdot (v_i \boldsymbol{n}_i) + \frac{\epsilon}{2}(u_i \boldsymbol{n}_i)\cdot \nabla v_i + \frac{\eta_0}{|e|} u_i v_i \right),$$
$$C_e^{ij}(u,v) := \int_e \left(-\frac{1}{2}\nabla u_i \cdot (v_j \boldsymbol{n}_j) + \frac{\epsilon}{2}(u_i \boldsymbol{n}_i)\cdot \nabla v_j - \frac{\eta_0}{|e|} u_i v_j \right),$$

and the contributions from the boundary edge $e = \partial K_i \cap \partial K_j \in E^\partial$ are defined as:

$$D_e^{ii}(u,v) := \int_e \left(-\nabla u_i \cdot (v_i \boldsymbol{n}_i) + \epsilon (u_i \boldsymbol{n}_i)\cdot \nabla v_i + \frac{\eta_0}{|e|} u_i v_i \right).$$

Taking the local support of the monomial basis functions into account, the diagonal blocks in the matrix A read, for all $i = 1, ..., N$:

$$(A_{ii})_{\ell k} = B(\phi_k^{(i)}, \phi_\ell^{(i)}) = \int_{K_i} \nabla \phi_k^{(i)} \cdot \nabla \phi_\ell^{(i)} + \sum_{e \in E,\, e \cap \partial K_i \neq \varnothing} D_e^{ii}(\phi_k^{(i)}, \phi_\ell^{(i)}),$$

and the off-diagonal blocks, for all $i, j = 1, ..., N$ with $e = \partial K_i \cap \partial K_j \in E^\circ$:

$$(A_{ji})_{\ell k} = B(\phi_k^{(i)}, \phi_\ell^{(j)}) = C_e^{ij}(\phi_k^{(i)}, \phi_\ell^{(j)}).$$

All other off-diagonal blocks are zero, since they correspond to non-adjacent elements.

Using the definition of the monomial test functions (2.7), the individual terms in these expressions can be computed. To this end, define:

$$p_k^{(i)}(x) := \left(\frac{x - x_{i-\frac{1}{2}}}{\frac{1}{2}h_x} \right)^k = \begin{cases} 1, & \text{for } x = x_i, \\ (-1)^k, & \text{for } x = x_{i-1}, \end{cases}$$
$$q_k^{(i)}(y) := \left(\frac{y - y_{i-\frac{1}{2}}}{\frac{1}{2}h_y} \right)^k = \begin{cases} 1, & \text{for } y = y_i, \\ (-1)^k, & \text{for } y = y_{i-1}, \end{cases}$$
$$P_k := \begin{cases} \int_{x_{i-1}}^{x_i} p_k^{(i)}(x)\,\mathrm{d}x = \left(\frac{h_x}{2}\frac{1}{k+1}(1-(-1)^{k+1}) \right), & \text{for } k \geq 0, \\ 0, & \text{else} \end{cases}$$
$$Q_k := \begin{cases} \int_{y_{i-1}}^{y_i} q_k^{(i)}(x)\,\mathrm{d}x = \left(\frac{h_y}{2}\frac{1}{k+1}(1-(-1)^{k+1}) \right), & \text{for } k \geq 0, \\ 0, & \text{else.} \end{cases}$$

Using this notation, the gradient of a test function reads:

$$\nabla \phi_k^{(i)}(x,y) = \begin{bmatrix} \frac{2k_x}{h_x} p_{k_x-1}^{(i)}(x) q_{k_y}^{(i)}(y) \\ \frac{2k_y}{h_y} p_{k_x}^{(i)}(x) q_{k_y-1}^{(i)}(y) \end{bmatrix}$$

Compute the volume integrals:

$$\begin{aligned} \int_{K_i} \nabla \phi_k^{(i)} \cdot \nabla \phi_\ell^{(i)} &= \int_{x_{i-1}}^{x_i} \int_{y_{i-1}}^{y_i} \left(\frac{2k_x}{h_x} p_{k_x-1}^{(i)}(x) q_{k_y}^{(i)}(y) \frac{2\ell_x}{h_x} p_{\ell_x-1}^{(i)}(x) q_{\ell_y}^{(i)}(y) \right. \\ &\quad \left. \frac{2k_y}{h_y} p_{k_x}^{(i)}(x) q_{k_y-1}^{(i)}(y) \frac{2\ell_y}{h_y} p_{\ell_x}^{(i)}(x) q_{\ell_y-1}^{(i)}(y) \right) \mathrm{d}y\,\mathrm{d}x \\ &= \left(\frac{2}{h_x} \right)^2 k_x l_x P_{k_x+\ell_x-2} Q_{k_y+\ell_y} + \left(\frac{2}{h_y} \right)^2 k_y \ell_y P_{k_x+\ell_x} Q_{k_y+\ell_y-2} \end{aligned}$$

Next, observe that

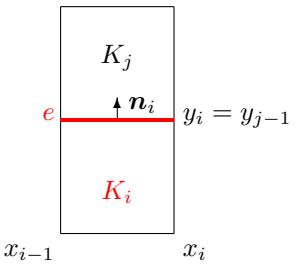
$$\phi_k^{(i)}(x,y)\phi_\ell^{(j)}(x,y) = p_{k_x}^{(i)}(x) q_{k_y}^{(j)}(y) p_{\ell_x}^{(i)}(x) q_{\ell_y}^{(j)}(y)$$

and

$$\nabla\phi_k^{(i)}(x,y)\cdot\left(\phi_\ell^{(j)}(x,y)\begin{bmatrix}n_x\\n_y\end{bmatrix}\right)=$$
$$\frac{2k_xn_x}{h_x}p_{k_x-1}^{(i)}(x)q_{k_y}^{(i)}(y)p_{\ell_x}^{(j)}(x)q_{\ell_y}^{(j)}(y)+\frac{2k_yn_y}{h_y}p_{k_x}^{(i)}(x)q_{k_y-1}^{(i)}(y)p_{\ell_x}^{(j)}(x)q_{\ell_y}^{(j)}(y)$$

Using these expressions, the edge contributions can be computed.

For a horizontal edge $e\in E$ defined by the line segment between (x_{i-1},y_i) and (x_i,y_i) with outward normal $\boldsymbol{n}_i=\begin{bmatrix}0\\1\end{bmatrix}$



we obtain:

$$\begin{aligned}D_e^{ii}(\phi_k^{(i)},\phi_\ell^{(i)}) &= -\frac{1}{2}b\left(\frac{2k_y}{h_y}P_{k_x+\ell_x}q_{k_y+\ell_y-1}^{(i)}(y_i)\right)\\ &\quad+\frac{\epsilon}{2}b\left(\frac{2\ell_y}{h_y}P_{k_x+\ell_x}q_{k_y+\ell_y-1}^{(i)}(y_i)\right)\\ &\quad+\frac{\eta_0}{h_x}\left(P_{k_x+\ell_x}q_{k_y+\ell_y}^{(i)}(y_i)\right)\\ &= \left(-b\frac{k_y}{h_y}+\epsilon b\frac{\ell_y}{h_y}+\frac{\eta_0}{h_x}\right)P_{k_x+\ell_x},\end{aligned}$$

where $b=2$ if the edge $e\in E^\partial$ is located at the boundary, and $b=1$ if $e\in E^\circ$ is in the interior. If $e=\partial K_i\cap\partial K_j$ is an *interior* edge , we can use that $y_i=y_{j-1}$ and that $\boldsymbol{n}_j=-\boldsymbol{n}_i=\begin{bmatrix}0\\-1\end{bmatrix}$ to obtain:

$$\begin{aligned}C_e^{ij}(\phi_k^{(i)},\phi_\ell^{(j)}) &\overset{y_i=y_{j-1}}{=} -\frac{1}{2}\left(-\frac{2k_y}{h_y}P_{k_x+\ell_x}q_{k_y-1}^{(i)}(y_i)q_{\ell_y}^{(j)}(y_{j-1})\right)\\ &\quad+\frac{\epsilon}{2}\left(\frac{2\ell_y}{h_y}P_{k_x+\ell_x}q_{k_y}^{(i)}(y_i)q_{\ell_y-1}^{(j)}(y_{j-1})\right)\\ &\quad-\frac{\eta_0}{h_x}\left(P_{k_x+\ell_x}q_{k_y}^{(i)}(y_i)q_{\ell_y}^{(j)}(y_{j-1})\right)\\ &= \left(\frac{k_y}{h_y}-\epsilon\frac{\ell_y}{h_y}-\frac{\eta_0}{h_x}\right)P_{k_x+\ell_x}(-1)^{\ell_y}.\end{aligned}$$

Similarly, for a horizontal interior edge $e\in E$ between (x_{i-1},y_{i-1}) and (x_i,y_{i-1}) with outward normal $\boldsymbol{n}_i=\begin{bmatrix}0\\-1\end{bmatrix}$, we get:

$$\begin{aligned}D_e^{ii}(\phi_k^{(i)},\phi_\ell^{(i)}) &= -\frac{1}{2}b\left(-\frac{2k_y}{h_y}P_{k_x+\ell_x}q_{k_y+\ell_y-1}^{(i)}(y_{i-1})\right)\\ &\quad+\frac{\epsilon}{2}b\left(-\frac{2\ell_y}{h_y}P_{k_x+\ell_x}q_{k_y+\ell_y-1}^{(i)}(y_{i-1})\right)\\ &\quad+\frac{\eta_0}{h_x}\left(P_{k_x+\ell_x}q_{k_y+\ell_y}^{(i)}(y_{i-1})\right)\\ &= \left(-b\frac{k_y}{h_y}+\epsilon b\frac{\ell_y}{h_y}+\frac{\eta_0}{h_x}\right)P_{k_x+\ell_x}(-1)^{k_y+\ell_y},\end{aligned}$$

and, if e is in the interior:

$$\begin{aligned} C_e^{ij}(\phi_k^{(i)}, \phi_\ell^{(j)}) \quad &\overset{y_{i-1}=y_j}{=} \quad -\frac{1}{2}\left(\frac{2k_y}{h_y} P_{k_x+\ell_x} q_{k_y-1}^{(i)}(y_{i-1}) q_{\ell_y}^{(j)}(y_j)\right) \\ &\quad +\frac{\epsilon}{2}\left(-\frac{2\ell_y}{h_y} P_{k_x+\ell_x} q_{k_y}^{(i)}(y_{i-1}) q_{\ell_y-1}^{(j)}(y_j)\right) \\ &\quad -\frac{\eta_0}{h_x}\left(P_{k_x+\ell_x} q_{k_y}^{(i)}(y_{i-1}) q_{\ell_y}^{(j)}(y_j)\right) \\ &= \quad \left(\frac{k_y}{h_y} - \epsilon\frac{\ell_y}{h_y} - \frac{\eta_0}{h_x}\right) P_{k_x+\ell_x}(-1)^{k_y}. \end{aligned}$$

For a vertical interior edge $e \in E$ between (x_i, y_{i-1}) and (x_i, y_i) with outward normal $\boldsymbol{n}_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, we get:

$$\begin{aligned} D_e^{ii}(\phi_k^{(i)}, \phi_\ell^{(i)}) \quad &= \quad -\frac{1}{2} b\left(\frac{2k_x}{h_x} p_{k_y+\ell_y-1}^{(i)}(x_i) Q_{k_y+\ell_y}\right) \\ &\quad +\frac{\epsilon}{2} b\left(\frac{2\ell_x}{h_x} p_{k_y+\ell_y-1}^{(i)}(x_i) Q_{k_y+\ell_y}\right) \\ &\quad +\frac{\eta_0}{h_y}\left(p_{k_y+\ell_y}^{(i)}(x_i) Q_{k_y+\ell_y}\right) \\ &= \quad \left(-b\frac{k_x}{h_x} + \epsilon b\frac{\ell_x}{h_x} + \frac{\eta_0}{h_y}\right) Q_{k_y+\ell_y}, \end{aligned}$$

and, if e is in the interior:

$$\begin{aligned} C_e^{ij}(\phi_k^{(i)}, \phi_\ell^{(j)}) \quad &\overset{x_i=x_{j-1}}{=} \quad -\frac{1}{2}\left(-\frac{2k_x}{h_x} p_{k_x-1}^{(i)}(x_i) p_{\ell_x}^{(j)}(x_{j-1}) Q_{k_y+\ell_y}\right) \\ &\quad +\frac{\epsilon}{2}\left(\frac{2\ell_x}{h_x} p_{k_x}^{(i)}(x_i) p_{\ell_x-1}^{(j)}(x_{j-1}) Q_{k_y+\ell_y}\right) \\ &\quad -\frac{\eta_0}{h_y}\left(p_{k_x}^{(i)}(x_i) p_{\ell_x}^{(j)}(x_{j-1}) Q_{k_y+\ell_y}\right) \\ &= \quad \left(\frac{k_x}{h_x} - \epsilon\frac{\ell_x}{h_x} - \frac{\eta_0}{h_y}\right)(-1)^{\ell_x} Q_{k_y+\ell_y}. \end{aligned}$$

For a vertical interior edge $e \in E^\circ$ between (x_{i-1}, y_{i-1}) and (x_{i-1}, y_i) with outward normal $\boldsymbol{n}_i = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$, we get:

$$\begin{aligned} D_e^{ii}(\phi_k^{(i)}, \phi_\ell^{(i)}) \quad &= \quad -\frac{1}{2} b\left(-\frac{2k_x}{h_x} p_{k_x+\ell_x-1}^{(i)}(x_{i-1}) Q_{k_y+\ell_y}\right) \\ &\quad +\frac{\epsilon}{2} b\left(-\frac{2\ell_x}{h_x} p_{k_x+\ell_x-1}^{(i)}(x_{i-1}) Q_{k_y+\ell_y}\right) \\ &\quad +\frac{\eta_0}{h_x}\left(p_{k_x+\ell_x}^{(i)}(x_{i-1}) Q_{k_y+\ell_y}\right) \\ &= \quad \left(-b\frac{k_x}{h_x} + \epsilon b\frac{\ell_x}{h_x} + \frac{\eta_0}{h_y}\right)(-1)^{k_x+\ell_x} Q_{k_y+\ell_y}. \end{aligned}$$

and, if e is in the interior:

$$\begin{aligned} C_e^{ij}(\phi_k^{(i)}, \phi_\ell^{(j)}) \quad &\overset{x_{i-1}=x_j}{=} \quad -\frac{1}{2}\left(-\frac{2k_x}{h_x} p^{(i)}_{k_x-1}(x_{i-1}) p^{(j)}_{\ell_x}(x_j) Q_{k_y+\ell_y}\right) \\ &\qquad +\frac{\epsilon}{2}\left(\frac{2\ell_x}{h_x} p^{(i)}_{k_x}(x_{i-1}) p^{(j)}_{\ell_x-1}(x_j) Q_{k_y+\ell_y}\right) \\ &\qquad -\frac{\eta_0}{h_y}\left(p^{(i)}_{k_x}(x_{i-1}) p^{(j)}_{\ell_x}(x_j) Q_{k_y+\ell_y}\right) \\ &= \quad \left(\frac{k_x}{h_x} - \epsilon\frac{\ell_x}{h_x} - \frac{\eta_0}{h_y}\right)(-1)^{k_x} Q_{k_y+\ell_y}. \end{aligned}$$

Appendix C

BILU in Detail

This section considers the BILU preconditioner described in Section 5.2 on a more detailed level, by specifying what happens at each recursion level.

Consider a coefficient matrix A that has the following recursive block tridiagonal structure with K 'levels': it is an $n_1 \times n_1$ block tridiagonal matrix,

$$A := \begin{bmatrix} A_1 & C_1 & & \\ B_2 & A_2 & \ddots & \\ & \ddots & \ddots & C_{n_1-1} \\ & & B_{n_1} & A_{n_1} \end{bmatrix},$$

in which each diagonal block A_j itself is an $n_2 \times n_2$ block tridiagonal matrix, in which each diagonal block in turn is an $n_3 \times n_3$ block tridiagonal matrix, and so on, until the diagonal blocks are scalars ($n_K := 1$). Every off-diagonal block under consideration is assumed to be a (regular) diagonal matrix.

To construct the BILU decompositions, introduce the following notation to identify the different recursion levels:

$$A = D_1^{\{\}},$$

and

$$D_{j_k}^{\{j_1,\ldots,j_{k-1}\}} = \begin{bmatrix} A_1^{\{j_1,\ldots,j_k\}} & C_1^{\{j_1,\ldots,j_k\}} & & \\ B_2^{\{j_1,\ldots,j_k\}} & A_2^{\{j_1,\ldots,j_k\}} & \ddots & \\ & \ddots & \ddots & C_{n_k-1}^{\{j_1,\ldots,j_k\}} \\ & & B_{n_k}^{\{j_1,\ldots,j_k\}} & A_{n_k}^{\{j_1,\ldots,j_k\}} \end{bmatrix},$$

$$T_{j_k}^{\{j_1,\ldots,j_{k-1}\}} = \begin{bmatrix} E_1^{\{j_1,\ldots,j_k\}} & G_1^{\{j_1,\ldots,j_k\}} & & \\ F_2^{\{j_1,\ldots,j_k\}} & E_2^{\{j_1,\ldots,j_k\}} & \ddots & \\ & \ddots & \ddots & G_{n_k-1}^{\{j_1,\ldots,j_k\}} \\ & & F_{n_k}^{\{j_1,\ldots,j_k\}} & E_{n_k}^{\{j_1,\ldots,j_k\}} \end{bmatrix},$$

for all $k = 1, \ldots, K$, and for all $j_k = 1, \ldots, n_{k-1}$ (setting $n_0 := 1$).

The BILU decomposition of the matrix $D_{j_k}^{\{j_1,\ldots,j_{k-1}\}}$, and of $A = D_1^{\{\}}$ in particular, can now

be written as:

$$D_{j_k}^{\{j_1,\ldots,j_{k-1}\}} \approx \begin{bmatrix} I & & & \\ L_2^{\{j_1,\ldots,j_k\}} & I & & \\ & \ddots & \ddots & \\ & & L_{n_k}^{\{j_1,\ldots,j_k\}} & I \end{bmatrix} \begin{bmatrix} D_1^{\{j_1,\ldots,j_k\}} & & & \\ & D_2^{\{j_1,\ldots,j_k\}} & & \\ & & \ddots & \\ & & & D_{n_k}^{\{j_1,\ldots,j_k\}} \end{bmatrix} \begin{bmatrix} I & U_1^{\{j_1,\ldots,j_k\}} & & \\ & I & \ddots & \\ & & \ddots & U_{n_k-1}^{\{j_1,\ldots,j_k\}} \\ & & & I \end{bmatrix},$$

where $D_1^{\{j_1,\ldots,j_k\}} = A_1^{\{j_1,\ldots,j_k\}}$, and, for all $\ell = 2, ..., n_k$,

$$\begin{aligned} L_\ell^{\{j_1,\ldots,j_k\}} &= B_\ell^{\{j_1,\ldots,j_k\}} T_{\ell-1}^{\{j_1,\ldots,j_k\}}, \\ U_{\ell-1}^{\{j_1,\ldots,j_k\}} &= T_{\ell-1}^{\{j_1,\ldots,j_k\}} C_{\ell-1}^{\{j_1,\ldots,j_k\}}, \\ D_\ell^{\{j_1,\ldots,j_k\}} &= A_\ell^{\{j_1,\ldots,j_k\}} - B_\ell^{\{j_1,\ldots,j_k\}} T_{\ell-1}^{\{j_1,\ldots,j_k\}} C_{\ell-1}^{\{j_1,\ldots,j_k\}}. \end{aligned}$$

Using these BILU decompositions, the matrices $D_1^{\{j_1,\ldots,j_k\}}, ..., D_{n_k}^{\{j_1,\ldots,j_k\}}$ and the approximation $T_{j_k}^{\{j_1,\ldots,j_{k-1}\}}$ for the inverse of $D_{j_k}^{\{j_1,\ldots,j_{k-1}\}}$ are computed simultaneously in the following manner (cf. Algorithm 9):

Algorithm 11
Computes the BILU factors $D_1^{\{j_1,\ldots,j_k\}}, ..., D_{n_k}^{\{j_1,\ldots,j_k\}}$, as well as an approximation $T_{j_k}^{\{j_1,\ldots,j_{k-1}\}}$ for the inverse of the input $D_{j_k}^{\{j_1,\ldots,j_{k-1}\}}$.

1) **if** $D_{j_k}^{\{j_1,\ldots,j_{k-1}\}}$ is a scalar ($k = K$), compute $T_{j_k}^{\{j_1,\ldots,j_{k-1}\}} = 1/D_{j_k}^{\{j_1,\ldots,j_{k-1}\}}$ directly, **else**:
2) $D_1^{\{j_1,\ldots,j_k\}} := A_1^{\{j_1,\ldots,j_k\}}$
3) **for** $\ell = 2, ..., n_k$ **do**
4) compute $T_{\ell-1}^{\{j_1,\ldots,j_k\}}$ that approximates the inverse of $D_{\ell-1}^{\{j_1,\ldots,j_k\}}$ by calling this algorithm (recursion)
5) $L_\ell^{\{j_1,\ldots,j_k\}} := B_\ell^{\{j_1,\ldots,j_k\}} T_{\ell-1}^{\{j_1,\ldots,j_k\}}$
6) $U_{\ell-1}^{\{j_1,\ldots,j_k\}} := T_{\ell-1}^{\{j_1,\ldots,j_k\}} C_{\ell-1}^{\{j_1,\ldots,j_k\}}$
7) $D_\ell^{\{j_1,\ldots,j_k\}} := A_\ell^{\{j_1,\ldots,j_k\}} - B_\ell^{\{j_1,\ldots,j_k\}} T_{\ell-1}^{\{j_1,\ldots,j_k\}} C_{\ell-1}^{\{j_1,\ldots,j_k\}}$
8) **end**
9) $E_{n_k}^{\{j_1,\ldots,j_k\}} := T_{n_k}^{\{j_1,\ldots,j_k\}}$
10) **for** $\ell = n_k - 1, ..., 1$ **do**
11) $E_\ell^{\{j_1,\ldots,j_k\}} := T_\ell^{\{j_1,\ldots,j_k\}} + U_\ell^{\{j_1,\ldots,j_k\}} E_{\ell+1}^{\{j_1,\ldots,j_k\}} L_{\ell+1}^{\{j_1,\ldots,j_k\}}$
12) **end**
13) **for** $\ell = 2, ..., n_k$ **do**
14) $F_\ell^{\{j_1,\ldots,j_k\}} := -T_\ell^{\{j_1,\ldots,j_k\}} L_\ell^{\{j_1,\ldots,j_k\}}$
15) $G_{\ell-1}^{\{j_1,\ldots,j_k\}} := -U_{\ell-1}^{\{j_1,\ldots,j_k\}} T_\ell^{\{j_1,\ldots,j_k\}}$
16) **end**

17) observe that $T_{j_k}^{\{j_1,...,j_{k-1}\}}$ is fully determined by the matrices E F and G above and ensure that $T_{j_k}^{\{j_1,...,j_{k-1}\}}$ is an $n_k \times n_k$ block tridiagonal matrix by setting elements outside the required nonzero pattern equal to zero.

Once the BILU decompositions are obtained at all levels, the result z of applying the preconditioner to r can be obtained similar to Algorithm 7, except that whenever the inverse of a block tridiagonal matrix is required, the algorithm is called recursively (cf. Algorithm 10):

Algorithm 12
Computes $z \approx (D_{j_k}^{\{j_1,...,j_{k-1}\}})^{-1} r$, assuming $D_1^{\{j_1,...,j_{\widehat{k}}\}}, ..., D_{n_{\widehat{k}}}^{\{j_1,...,j_{\widehat{k}}\}}$ are precomputed for $\widehat{k} = k, ..., K$.

1) **if** $D_{j_k}^{\{j_1,...,j_{k-1}\}}$ is a scalar ($k = K$), compute $z = 1/D_{j_k}^{\{j_1,...,j_{k-1}\}} r$ directly, **else**:
2) $s_1 := r_1$
3) **for** $\ell = 2, 3, ..., n_k$ **do**
4) $\quad s_\ell := r_\ell - B_\ell^{\{j_1,...,j_k\}} (D_{\ell-1}^{\{j_1,...,j_k\}})^{-1} s_{\ell-1}$ (by applying this algorithm recursively)
5) **end**
6) $z_{n_k} := (D_{n_k}^{\{j_1,...,j_k\}})^{-1} s_{n_k}$ (by applying this algorithm recursively)
7) **for** $\ell = n_k - 1, n_k - 2, ..., 1$ **do**
8) $\quad z_\ell := (D_\ell^{\{j_1,...,j_k\}})^{-1} (s_\ell - C_\ell z_{\ell+1})$ (by applying this algorithm recursively)
9) **end**

The BILU preconditioner applies this algorithm for $A = D_1^{\{\}}$.

Bibliography

[AA07] P. F. Antonietti and B. Ayuso. Schwarz domain decomposition preconditioners for discontinuous Galerkin approximations of elliptic problems: non-overlapping case. *M2AN Math. Model. Numer. Anal.*, 41(1):21–54, 2007.

[AA08] P. F. Antonietti and B. Ayuso. Multiplicative Schwarz methods for discontinuous Galerkin approximations of elliptic problems. *M2AN Math. Model. Numer. Anal.*, 42(3):443–469, 2008.

[AA09] P. F. Antonietti and B. Ayuso. Two-level Schwarz preconditioners for super penalty discontinuous Galerkin methods. *Commun. Comput. Phys.*, 5(2-4):398–412, 2009.

[ABCM02] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, 39(5):1749–1779 (electronic), 2001/02.

[AC83] J. R. Appleyard and I. M. Cheshire. Nested factorisation. paper SPE 12264 presented at the SPE 1983 Reservoir Simulation Symposium, San Francisco, CA., 1983.

[AdDZ09] B. Ayuso de Dios and L. Zikatanov. Uniformly convergent iterative methods for discontinuous Galerkin discretizations. *J. Sci. Comput.*, 40(1-3):4–36, 2009.

[Arn82] D. N. Arnold. An interior penalty finite element method with discontinuous elements. *SIAM J. Numer. Anal.*, 19(4):742–760, 1982.

[BGRT09] F. Bassi, A. Ghidoni, S. Rebay, and P. Tesini. High-order accurate p-multigrid discontinuous Galerkin solution of the Euler equations. *Internat. J. Numer. Methods Fluids*, 60(8):847–865, 2009.

[BPX91] J. H. Bramble, J. E. Pasciak, and J. Xu. The analysis of multigrid algorithms with nonnested spaces or noninherited quadratic forms. *Math. Comp.*, 56(193):1–34, 1991.

[Bra93] J. H. Bramble. *Multigrid methods*, volume 294 of *Pitman Research Notes in Mathematics Series*. Longman Scientific & Technical, Harlow, 1993.

[BS06] S. C. Brenner and L.-Y. Sung. Multigrid algorithms for C^0 interior penalty methods. *SIAM J. Numer. Anal.*, 44(1):199–223 (electronic), 2006.

[BZ05] S. C. Brenner and J. Zhao. Convergence of multigrid algorithms for interior penalty methods. *Appl. Numer. Anal. Comput. Math.*, 2(1):3–18, 2005.

[FK01] X. Feng and O. A. Karakashian. Two-level additive Schwarz methods for a discontinuous Galerkin approximation of second order elliptic problems. *SIAM J. Numer. Anal.*, 39(4):1343–1365 (electronic), 2001.

[FK05] X. Feng and O. A. Karakashian. Two-level non-overlapping Schwarz preconditioners for a discontinuous Galerkin approximation of the biharmonic equation. *J. Sci. Comput.*, 22/23:289–314, 2005.

[FOLd05] K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. darmofal. p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 207(1):92–113, 2005.

[GK03] J. Gopalakrishnan and G. Kanschat. A multilevel discontinuous Galerkin method. *Numer. Math.*, 95(3):527–550, 2003.

[HCGR06] K. Hillewaert, N. Chevaugeon, P. Geuzaine, and J.-F. Remacle. Hierarchic multigrid iteration strategy for the discontinuous Galerkin solution of the steady Euler equations. *Internat. J. Numer. Methods Fluids*, 51(9-10):1157–1176, 2006.

[HJ91] R. A. Horn and C. R. Johnson. *Topics in matrix analysis*. Cambridge University Press, Cambridge, 1991.

[LT03] C. Lasser and A. Toselli. An overlapping domain decomposition preconditioner for a class of discontinuous Galerkin approximations of advection-diffusion problems. *Math. Comp.*, 72(243):1215–1238 (electronic), 2003.

[Mei83] J. A. Meijerink. Iterative mthods for the solution of linear equations based on incomplete block factorization of the matrix. paper SPE 12262 presented at the SPE 1983 Reservoir Simulation Symposium, San Francisco, CA., 1983.

[MvdV77] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31(137):148–162, 1977.

[NM06] C. R. Nastase and D. J. Mavriplis. High-order discontinuous Galerkin methods using an hp-multigrid approach. *J. Comput. Phys.*, 213(1):330–357, 2006.

[PP08] P.-O. Persson and J. Peraire. Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations. *SIAM J. Sci. Comput.*, 30(6):2709–2733, 2008.

[Riv08] B. Rivière. *Discontinuous Galerkin methods for solving elliptic and parabolic equations*, volume 35 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2008. Theory and implementation.

[RP87] E. M. Rønquist and A. T. Patera. Spectral element multigrid. I. Formulation and numerical results. *J. Sci. Comput.*, 2(4):389–406, 1987.

[Saa00] Y. Saad. Iterative methods for sparse linear systems. This is a revised version of the book published in 1996 by PWS Publishing, Boston. It can be downloaded from http://www-users.cs.umn.edu/ saad/books.html, 2000.

[Slo08] K. van 't Slot. Robust linear solvers for the adjoint equations in reservoir simulation. Master's thesis, Delft University of Technology, 2008.

[SS07] M. Sarkis and D. B. Szyld. Optimal left and right additive Schwarz preconditioning for minimal residual methods with Euclidean and energy norms. *Comput. Methods Appl. Mech. Engrg.*, 196(8):1612–1621, 2007.

[Tan08] J.M. Tang. *Twwo-Level Preconditioned Conjugate Gradient Methods with Applications to Bubbly Flow Problems*. PhD thesis, Delft University of Technology, 2008.

[TW05] A. Toselli and O. Widlund. *Domain decomposition methods—algorithms and theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2005.

[Xu92] Jinchao Xu. Iterative methods by space decomposition and subspace correction. *SIAM Rev.*, 34(4):581–613, 1992.

[XZ02] Jinchao Xu and Ludmil Zikatanov. The method of alternating projections and the method of subspace corrections in Hilbert space. *J. Amer. Math. Soc.*, 15(3):573–597 (electronic), 2002.