INAUGURAL - DISSERTATION

zur
Erlangung der Doktorwürde
der
Gesamtfakultät für Mathematik, Ingenieur- und Naturwissenschaften
der
Ruprecht-Karls-Universität
Heidelberg

vorgelegt von
Chaiyod Kamthorncharoen, M. Sc.
aus Bangkok, Thailand

Tag der mündlichen Prüfung: ....

Two-level Restricted Additive Schwarz Preconditioned Exact Newton with Applications

Betreuer: Prof. Dr. Peter Bastian
Prof. Dr. Kurt Roth

ABSTRACT

We study on Restricted Additive Schwarz Preconditioned Exact Newton method (RASPEN), a nonlinear preconditioning of Newton's method for solving the nonlinear algebraic systems of equations which result from the discretisation of partial differential equations (PDEs). The preconditioned system is created by the help of additive Schwarz method to enable the parallel computation and is supposed to be more suitable for Newton's method. We also propose the coarse grid correction for RASPEN due to the fact that the one-level scheme has a scalability concern when doing a large-scale computation. Adding the second level would remedy this drawback. Our coarse space is based on the idea of Nicolaides coarse space with some extensions. It does not need an explicit coarse mesh and can be constructed in the purely algebraic manner. Furthermore, the setup of the coarse problem can be done in parallel. We apply RASPEN on various scenarios in order to investigate the flexibility of RASPEN and the effectiveness of the two-level approach.

ZUSAMMENFASSUNG

Wir untersuchen die Restricted Additive Schwarz Preconditioned Exact Newton-Methode (RASPEN), eine nichtlineare Vorkonditionierung der Newton-Methode zur Lösung nichtlinearer algebraischer Gleichungssysteme, die aus der Diskretisierung partieller Differentialgleichungen (PDEs) resultieren. Das vorkonditionierte System wird mit Hilfe der additiven Schwarz-Methode erstellt, um eine parallele Berechnung zu ermöglichen, und die Newton-Methode zu beschleunigen und zu robustifizieren. Wir schlagen auch eine Grobgitterkorrektur für RASPEN vor, da das einstufige Schema bei großen Berechnungen ein Problem mit der Skalierbarkeit hat. Die Hinzufügung der zweiten Ebene beheft diesen Nachteil. Unser Grobraum basiert auf der Idee des Nicolaides-Grobraums mit einigen Erweiterungen. Er benötigt kein explizites Grobnetz und kann auf rein algebraische Weise konstruiert werden. Außerdem kann der Aufbau des groben Problems parallel erfolgen. Wir wenden RASPEN auf verschiedene Szenarien an, um die Flexibilität von RASPEN und die Wirksamkeit des zweistufigen Ansatzes zu untersuchen.

ACKNOWLEDGMENTS

First of all, I would like to express my sincere thanks to my supervisors, Prof. Dr. Peter Bastian and Prof. Dr. Kurt Roth for their valuable advice and heartfelt encouragement throughout my doctoral study. The completion of this dissertation would not have been possible without all support from them.

I am thankful to all my colleagues at the Interdisciplinary Center for Scientific Computing (IWR) of Heidelberg University, especially in our Parallel Computing group for a wonderful working environment and supportive contribution during my PhD journey. Furthermore, I would like to thank the Thai community in Heidelberg for making my stay in Germany a great experience.

I am grateful to the Development and Promotion of Science and Technology Talents Project (DPST) for financial support since my undergraduate study. In addition, I would like to thank the Graduate Academy of Heidelberg University for the completion grant I received throughout the final arc of my doctoral study.

At last, I wholeheartedly express my deepest gratitude to my family, especially my better half, for her constant love, endless tolerance, and understanding. I have been blessed to have them beside me through my ups and downs.

Contents

1 Introduction

In this work we consider the solution of nonlinear algebraic systems of equations arising from the discretization of partial differential equations (PDEs). It is commonly accepted that Newton's method is the most popular tool for solving those systems. Newton's method has a beneficial property that it is converging quadratically but it is only locally convergent. Accordingly, the starting point must not be too far away from the true solution. So, it requires a globalization technique such as line-search or trust-region method to enhance the robustness. In each Newton iteration a linearized system needs to be solved. A Krylov iterative solver is typically chosen for solving that system. Its important benefit is the Jacobian-free computation. Consequently, the true Jacobian could be never stored which would reduce memory requirement. But a Krylov method sometimes takes an incredibly huge number of iterations, if the Jacobian is ill-conditioned. A domain decomposition technique such as additive Schwarz could take part here as a preconditioner for a linearized system. Therefore, it enables parallelization in the nonlinear solver. This strategy is so-called Newton-Krylov-Schwarz. However, the nonlinear system itself is not preconditioned anyway.

One can do another approach by applying the domain decomposition method in the context of "Nonlinear Preconditioning". Cai and Keyes [CK02] have introduced Additive Schwarz Preconditioned Inexact Newton (ASPIN) in the early 2000s. The idea is instead of solving the original nonlinear system, the new system is constructed with the help of the additive Schwarz method. Then, we solve the new system by Newton's method. The resulting system is called the "preconditioned nonlinear system". Nonlinear preconditioning could increase the robustness and may find the solution faster. Domain decomposition also plays an important role in which the nonlinear problem is subdivided into nonlinear subproblems on subdomains. Since Newton's method is a global solver, it is hindered if the problem is hard to solve in some parts of the domain. It would slow down or ruin the convergence of the solver. With ASPIN strategy, one can distribute the workload based on the difficulty of subproblems.

Restricted Additive Schwarz Preconditioned Exact Newton (RASPEN), an ASPIN variant, is developed by Dolean et. al. [Dol+16] with the idea of using the restriction (and prolongation) operators that satisfy the partition of unity property and using the exact Jacobian instead of inexact one. Note that both ASPIN and RASPEN are derived based on overlapping subdomains, but nonlinear preconditioning can work for the nonoverlapping case as well. Many researchers also work in that direction and propose methods like Balancing Domain Decomposition method (BDD) from Bordeu, Boucard, and Gosselet [BBG09], nonlinear Dual-Primal Finite

Element Tearing and Interconnecting (FETI-DP), and nonlinear Balancing Domain Decomposition by Constraints methods (BDDC) from Klawonn, Lanser, and Rheinbach [KLR14]. RASPEN also has its version for nonoverlapping called SRASPEN [Cha+21]. But we do not consider it in this study.

There is a concern about scalability because RASPEN also has to solve a global linear system in which the condition number heavily depends on the number of subdomains. That means the purely single level does not provide good performance when we do scaling. A coarse grid correction plays a role to remedy that issue. ASPIN and RASPEN propose their own approach for coarse grid correction to overcome the scaling issue [MC05; Dol+16]. Both of them apply the second level with the coarser mesh which we believe that on some kinds of problems, for example, the highly heterogeneous problem, rediscretization the coefficients from fine mesh to coarser mesh cannot represent a good approximation. We will provide a coarse problem based on Nicolaides coarse space which basically does not need to explicitly construct a coarse mesh and can be done algebraically.

1.1 Thesis Outline

This thesis is separated into several chapters structured in the following:

Chapter 2: Numerical Solution for PDEs

We provide numerical tools for solving nonlinear partial differential equations (PDEs). We start at a weak formulation. After that, we describe the discretization schemes used in this study, Finite Element (FE) and Discontinuous Galerkin (DG) to obtain the algebraic system of equations. Then, we talk about Newton's method which is exploited to solve the resulting system, and linear solvers for solving a linearized equation appearing in each Newton iteration. The last part is the domain decomposition which plays a key role in our main method, RASPEN.

Chapter 3: Restricted Additive Schwarz Preconditioned Exact Newton

We provide the information on RASPEN e.g., how the method is derived, relation to ASPIN, and relation to the linear case. Its two-level approaches are also discussed. We explain our two-level approach which is an extension based on Nicolaides coarse space. And the last section would give the description of the RASPEN algorithm.

Chapter 4: Implementation of RASPEN in DUNE

We introduce software used throughout this study, DUNE, a software framework for solving PDEs with grid-based methods. We explain the RASPEN implementation

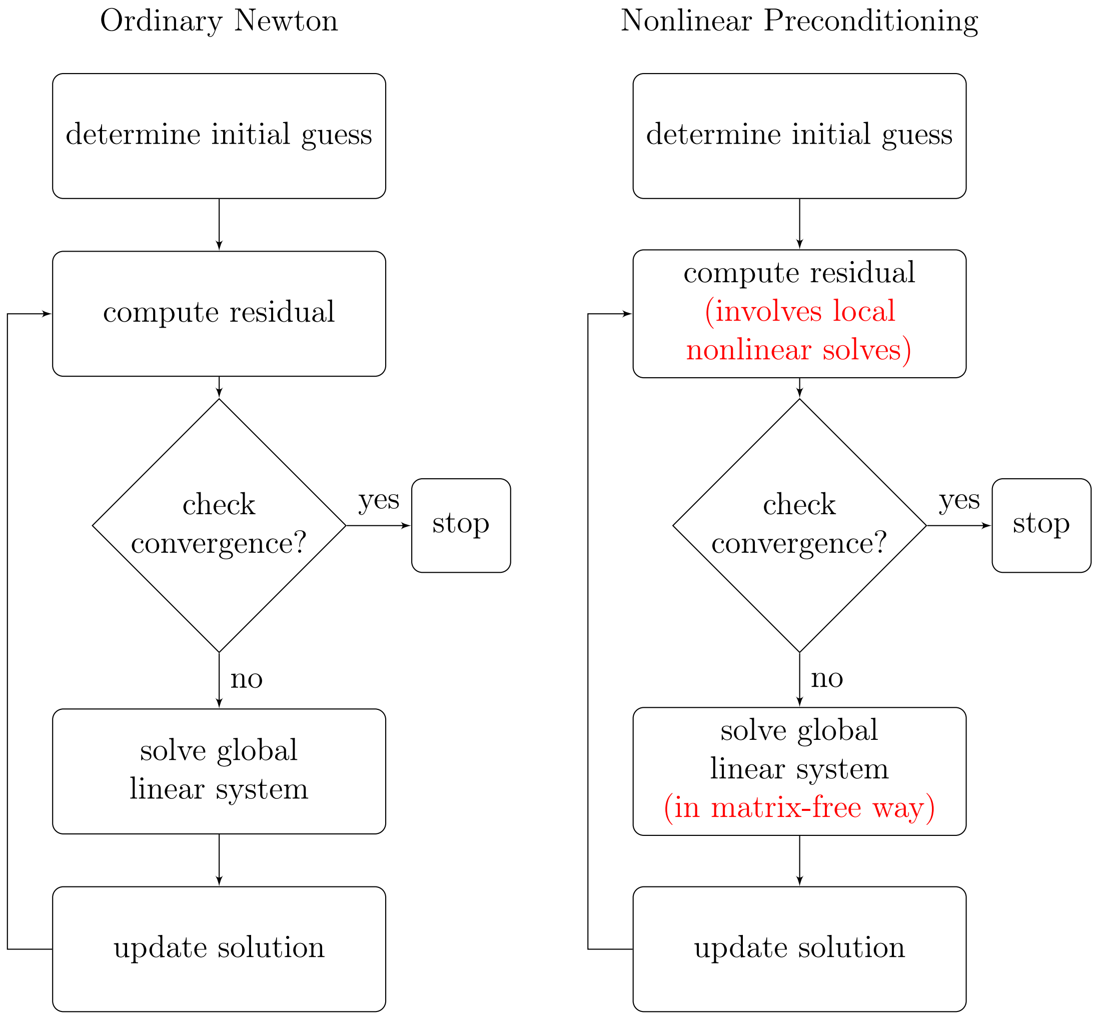


Figure 1.1: Diagrams of ordinary Newton and nonlinear preconditioning.

in DUNE style. And we also explain how to deal with parallel computation involved in many parts of our algorithm.

Chapter 5: Application of RASPEN to Model Problems

This chapter would provide the applications of RASPEN to several problems with discretization schemes. The numerical results of one-level and two-level RASPEN would be compared to Newton's method to show the convergence behavior of each method. The scalability would also be presented and compared between one-level and two-level versions.

Chapter 6: Conclusions

We finally summarize and discuss the outcomes of our study. Also, the future works will be mentioned.

1.2 Major Achievements

This section is dedicated to expressing what we achieve in this study

1) Flexible implementation of RASPEN in DUNE
   We provide a flexible implementation of RASPEN in DUNE. It is a software framework for solving PDEs developed by our research group. ASPIN and RASPEN have worked successfully on various nonlinear problems but mainly for Finite Difference (FD) and Finite Element (FE) discretization schemes [CKY01; CKM02; HC05; LKS13; Dol+16; SKN16]. We would show that the nonlinear preconditioning performs well with Discontinuous Galerkin (DG). RASPEN is tested on different discretizations like Finite Element (FE), Finite Volume (FV), and Discontinuous Galerkin (DG). It is also applied to many types of problems e.g., instationary problems, Richards' equation which has the nonlinearity on both spatial and temporal derivatives, and nonlinear systems of equations describing the reactive multiphase flow.

2) Detailed investigation of robustness aspect
   We present that RASPEN is more robust than Newton's method in several aspects. For stationary problems, RASPEN basically takes less number of Newton iterations to converge to the solution. Moreover, we apply RASPEN to the problem so that we can increase the (local) nonlinearity. In the end, we see that RASPEN can solve the higher nonlinearity problems compared to Newton's method.
   For instationary problems, RASPEN can handle a larger time step size which means it could take fewer steps to complete the simulation. We also provide an additional feature of RASPEN where we are allowed to divide the time step locally. It turns out that RASPEN could handle even a larger global time step size.

3) Formulation of almost fully algebraic coarse correction
   We introduce a coarse grid correction based on the extension of Nicolaides coarse space. The coarse space constructed from rediscretization of the fine grid does not represent a good coarse problem, especially in heterogeneous problems. We exploit the extended Nicolaides coarse space to construct the coarse problem in a way of Galerkin projection which can be done in an algebraic manner. In other words, we do not need an explicit coarse mesh to generate a coarse problem.

4) Parallel implementation of two-level scheme and evaluation of its scalability
   Although a coarse grid correction itself is a global problem, many parts of the

computation in our scheme can be done in parallel such as restriction and prolongation for the coarse problem, and coarse matrix assembly. Their details are explained in Chapter 4.

We apply the two-level schemes to evaluate the scalability. We do a weak scaling test running from 2×2 to 64×64 subdomains on the nonlinear Poisson problem. Using our coarse grid correction displays that the number of global linear iterations remains asymptotically constant which satisfies our expectation. For a strong scaling test, the schemes also provide a good scaling factor.

2 Numerical Solution of PDEs

In this chapter, we provide the recapitulation of a nonlinear system of equations arising from the discretization of the partial differential equations and how to solve that system. We start describing about problem formulation in *strong* and *weak* form with the example of (nonlinear) Poisson equation. After that, the Finite Element (FE) and Discontinuous Galerkin (DG) methods, the main discretization schemes in this work, are presented. The resulting nonlinear system of equations obtained after discretization of the weak formulation is typically solved by Newton's method. Hence, we talk about it thereafter and also the solvers for the linear system. The last section of this chapter would introduce the domain decomposition methods which play a key role in the next chapter.

2.1 Problem Formulation

We consider introducing the elliptic partial differential equation, for instance, the nonlinear Poisson equation which is commonly a starting point to explain

$$\begin{aligned} -\nabla \cdot (\varphi(u, \|\nabla u\|)\nabla u) &= f && \text{in } \Omega, && (2.1a)\\ u &= g && \text{on } \Gamma_D \subseteq \partial\Omega && (2.1b)\\ -\varphi(u, \|\nabla u\|)\nabla u \cdot \boldsymbol{\nu} &= j && \text{on } \Gamma_N = \partial\Omega \setminus \Gamma_D && (2.1c)\end{aligned}$$

where $\Omega \subset \mathbb{R}^n$ is a bounded domain, $f : \Omega \to \mathbb{R}$ denotes the source term, φ determines, in general, the diffusivity which is a (nonlinear) function, $\boldsymbol{\nu}$ is a unit outer normal vector to the domain Ω, and (2.1b) and (2.1c) represent the Dirichlet and Neumann boundary conditions respectively.

In the case of the pure Dirichlet problem, a *strong solution* is a function $u \in C^2(\Omega) \cap C^0(\bar{\Omega})$ which satisfies the condition (2.1a), and (2.1b). And in the case of the pure Neumann problem, a *strong solution* is a function $u \in C^2(\Omega) \cap C^1(\bar{\Omega})$ which satisfies the condition (2.1a), and (2.1c). But in general, a *strong solution* is difficult to achieve or impossible in some problems. The less restrictive conditions can be allowed in the *weak formulation* [Eva10; EG04] which gives you the *weak solution* instead. It resolves the limitations of the *strong* form because only the first derivative, in the weak sense is required. Moreover, it is an important basis of the Finite Element method which we would explain in the next section.

Let us start with multiplying the nonlinear Poisson equation (2.1a) by any *test* function $v \in C^1(\Omega) \cap C^0(\bar{\Omega})$ with $v = 0$ on Γ_D and then integrating by parts

$$\int_\Omega \varphi(u, \|\nabla u\|)\nabla u \cdot \nabla v \, dx + \int_{\Gamma_N} jv \, ds = \int_\Omega fv \, dx. \qquad (2.2)$$

By the property of space V that $v = 0$ on Γ_D, we see that the term $\int_{\Gamma_D} \nabla u \cdot \boldsymbol{\nu} v \, ds$ vanishes. We also assume that $f \in L^2(\Omega)$ and $g \in H^{\frac{1}{2}}(\partial\Omega)$. We seek a function $u \in V_g = \{u \in V | u = g \text{ on } \Gamma_D\}$ such that it satisfies (2.2) for the appropriate *test* functions $v \in V$.

With $V = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$, we can ensure the existence and uniqueness if some conditions are fulfilled. More precise statements and proofs can be found in [AL83]. We assume that φ is defined in such a way that there exists a unique solution for (2.2). One precise example of φ is $|\nabla u|^{p-2}$ where $p > 1$. There is a guaranteed (weak) solution $u \in W_0^{1,p}(\Omega)$ to the Dirichlet problem

$$\begin{aligned} -\nabla \cdot \left(|\nabla u|^{p-2} \, \nabla u \right) &= 1 && \text{in } \Omega, \\ u &= 0 && \text{on } \partial\Omega, \end{aligned}$$

see [Sak87] for details and reference therein. We would like to rewrite the problem into "residual form", that is

$$\text{Find } u \in U \text{ s.t. : } r(u,v) = \int_\Omega \varphi(u, \|\nabla u\|) \nabla u \cdot \nabla v - f v \, dx + \int_{\Gamma_N} j v \, ds = 0 \quad \forall v \in V \tag{2.3}$$

where we usually exploit this expression throughout this thesis.

2.2 Finite Element method

is basically the discretization idea to solve the weak formulation (2.3) numerically by substituting the function space V by the *finite dimensional* function space V_h (where h refers actually to mesh size) defined on finite element mesh $\mathcal{T}_h$ which subdivides the domain Ω into cells or elements. Cells can be an arbitrary shape but in this thesis, we generally choose a cube element (which is a quadrilateral in $2d$). Some useful materials for the introduction of Finite Element are provided in [EG04; Bra07].

Basis Representation and Algebraic problem

This subsection would describe the algebraic problem arising after inserting the basis representation. We assume that our trial and test spaces are spanned by global basis functions (we will describe how to construct them in the following subsections), i.e. $V_h = \text{span}\{\phi_1, \dots, \phi_n\}$. We can expand the solution $u_h = \sum_{j=1}^n (z)_j \phi_j$ as a linear combination of basis functions ϕ_j and $z \in \mathbb{R}^n$ is the coefficient vector. Hence, the residual form (2.3) becomes

$$\begin{aligned} \text{Find } u_h \in V_h \text{ s.t.:} \quad && r(u_h, v) &= 0 && \forall v \in V_h \\ \Leftrightarrow \quad && r\left(\sum_{j=1}^n (z)_j \phi_j, \phi_i \right) &= 0 && \forall i = 1, \dots, n \\ \Leftrightarrow \quad && R(z) &= 0, \end{aligned} \tag{2.4}$$

where $R : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear map resulting in a nonlinear algebraic equation and $R_i(z) = r_h\left(\sum_{j=1}^{n}(z)_j\phi_j, \phi_i\right)$. Newton's method typically solves this resulting nonlinear system of equations which would be described in Section 2.4.

The next question might be how to construct the finite dimensional function space V_h. We introduce the basic tools to construct the finite element mesh $\mathcal{T}_h$ and also its basis function here

1) set of *vertices* $\mathcal{X}_h = \{x_1, \ldots, x_N\}$ and *elements* $\mathcal{T}_h = \{T_1, \ldots, T_M\}$, that is, a nonoverlapping subdivision of the domain Ω into a union of element T

$$\bigcup_{T \in \mathcal{T}_h} T = \overline{\Omega}$$

where each element $T \in \mathcal{T}_h$ is closed, $\mathring{T}_i \cap \mathring{T}_j = \emptyset \quad \forall T_i, T_j \in \mathcal{T}_h$, and where each of the vertices of element T coincides with a vertex belonging in $\mathcal{X}_h$.
A face of an element T is denoted by F. It is a vertex, edge, or face in $1d, 2d$, and $3d$, respectively. A mesh is called *conforming* if each face $F \subset \partial T$ of element $T \in \mathcal{T}_h$ is either a face of another element T' or a subset of $\partial\Omega$

2) the index set of vertices $\mathcal{I}_h = \{1, \ldots, N\}$. We can also categorize it into the index set of interior and boundary vertices:

$$\mathcal{I}_h = \mathcal{I}_h^{int} \cup \mathcal{I}_h^{\partial\Omega}, \quad \mathcal{I}_h^{int} = \{i \in \mathcal{I}_h \,:\, x_i \in \Omega\}, \quad \mathcal{I}_h^{\partial\Omega} = \{i \in \mathcal{I}_h \,:\, x_i \in \partial\Omega\}.$$

3) a *local-to-global* index map

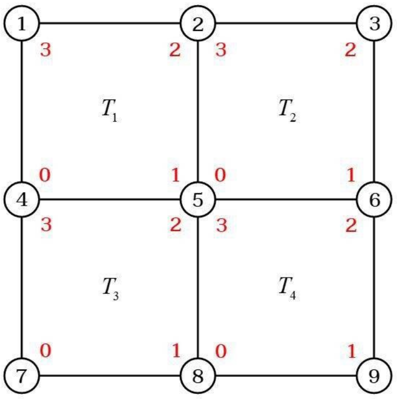
$$g_T : \{0, \ldots, n_T - 1\} \to \mathcal{I}_h$$

for every $T \in \mathcal{T}_h$ and n_T is a number of vertices of element T. It maps a local number of vertices of element T to a global vertex number.

4) last but not least, we introduce an *element transformation map*

$$\mu_T : \hat{T} \to T$$

which maps a point from corresponding *reference* element $\hat{T}$ to a point in *physical (real)* element T. It is assumed to be sufficiently differentiable with invertible Jacobian as well as consistent with the local numbering map g_T in such a way that $\mu_T(\hat{x}_i) = x_{g_T(i)}$.



$$g_{T_1}(0) = 4,\quad g_{T_1}(1) = 5,\quad g_{T_1}(2) = 2,\quad g_{T_1}(3) = 1$$
$$g_{T_2}(0) = 5,\quad g_{T_2}(1) = 6,\quad g_{T_2}(2) = 3,\quad g_{T_2}(3) = 2$$
$$g_{T_3}(0) = 7,\quad g_{T_3}(1) = 8,\quad g_{T_3}(2) = 5,\quad g_{T_3}(3) = 4$$
$$g_{T_4}(0) = 8,\quad g_{T_4}(1) = 9,\quad g_{T_4}(2) = 6,\quad g_{T_4}(3) = 5.$$

Figure 2.1: Example of *local-to-global* index map.

Reference Elements on Cube

As already mentioned in the previous lines, we have proposed an *element transformation map* from *reference* element to *physical* element. Therefore, in this subsection, we would describe what is the reference element, especially for cube form. A reference cube in d-dimension is defined by

$$\hat{\mathcal{Q}}^d := \{(x_1, \dots, x_d) \in \mathbb{R}^d : 0 \leq x_i \leq 1\}$$

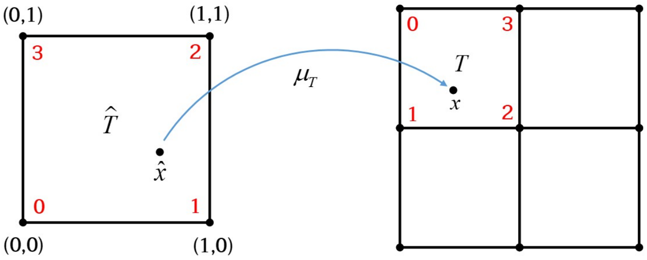


Figure 2.2: Illustrations of 2-d reference element on cube $\hat{T} = \hat{\mathcal{Q}}^2$ and element transformation map μ_T.

The benefit of the reference element is while we deal with the integration over elements or faces from the weak formulation, it is typically expensive to do it over each physical element. We can easily apply the simple map from the physical element to the reference element where we already had the basis function evaluation precomputed. Moreover, the formula for the Jacobian of transformation map (also its inverse) as well as the integration domain remains the same for all elements. Note that in the case of a parallelepiped in $\mathbb{R}^d, \mu_T$ is an affine linear map which means the Jacobian of transformation map μ_T at point $\hat{x}$, $J_{\mu_T}(\hat{x})$ is constant for all

$\hat{x} \in \hat{T}$. With the notation we purposed in this and previous section, we can define the conforming space of degree k in d dimension on mesh $\mathcal{T}_h$ by

$$V_h^{k,d}(\mathcal{T}_h) = \mathcal{Q}_h^{k,d}(\mathcal{T}_h) := \left\{ v \in C^0(\overline{\Omega}) \,:\, \forall T \in \mathcal{T}_h : v|_T = \mu_T \circ p_T \wedge p_T \in \mathbb{Q}_h^{k,d} \right\} \quad (2.5)$$

where $\mu_T : \hat{\mathcal{Q}} \to T$, and $\mathbb{Q}_h^{k,d} = \left\{ p \in C^\infty(\mathbb{R}^d) = \sum\limits_{0 \le \|\alpha\|_\infty \le k} c_\alpha x_1^{\alpha_1} \cdot \ldots \cdot x_d^{\alpha_d} \right\}$ represents the sets of polynomial whose maximal degree is k in d dimension and dim $\mathbb{Q}_h^{k,d} = n_{\hat{T}}^{k,d} = (k+1)^d$.

Lagrange Basis

Up to this point, we do not yet declare any basis function on our constructed finite element space But the key point of finite element is inserting the basis representation of the finite element solution. We start with constructing *Lagrange* basis function on reference (cube) element $\hat{T}$ by using the idea of *Lagrange* points and *Lagrange* polynomials as following definitions

$$L_{\hat{T}} = \left\{ \hat{x}_0^{\hat{T}}, \ldots, \hat{x}_{n_{\hat{T}}^{k,d}-1}^{\hat{T}} \right\} \qquad P_{\hat{T}} = \left\{ p_0^{\hat{T}}, \ldots, p_{n_{\hat{T}}^{k,d}-1}^{\hat{T}} \right\}$$

such that

$$p_i^{\hat{T}}(\hat{x}_j^{\hat{T}}) = \delta_{ij}.$$

Note that from now on, we can extend *the local-to-global* index map g_T in such a way that it maps from local numbering of Lagrange points (in physical element) to global numbering of Lagrange points (in physical element)

$$g_T : \{0, \ldots, n_{\hat{T}}^{k,d} - 1\} \to \mathcal{I}\left(V_h^{k,d}(\mathcal{T}_h)\right) = \left\{0, \ldots, \text{dim} V_h^{k,d}(\mathcal{T}_h) - 1\right\}.$$

Then, the inversion of the map is defined by

$$C(i) = \{(T, m) \in \mathcal{T}_h \times \mathbb{N} \,:\, g_T(m) = i\}$$

where it illustrates all possible elements T and its corresponding local index m that maps to the global vertex i. We eventually define the global Lagrange basis function by

$$\phi_i(x) = \begin{cases} p_m^{\hat{T}}(\mu_T^{-1}(x)) & x \in T \wedge (T, m) \in C(i) \\ 0 & \text{else} \end{cases} \quad , \quad i \in \mathcal{I}\left(V_h^{k,d}(\mathcal{T}_h)\right)$$

where each global basis function ϕ_i is corresponding to a global Lagrange point x_i.

We emphasize here that this definition is defined on continuous space so that, for instance, one can evaluate the basis function $\phi_i(x)$ from either (T, m) or $(T', m') \in C(i)$ if x is on the intersection of T and T' but there can also be discontinuous basis function space where we cannot do that. In that case, we need to be aware that on which element we are evaluating.

Finite Element Space

By this construction of global Lagrange basis function, we can incorporate the Dirichlet boundary condition on the corresponding global Lagrange points which lie on Dirichlet boundary Γ_D and evaluate Dirichlet function g or use it as a coefficient of a basis function. Let us define the index set of the Lagrange points on the Dirichlet boundary

$$\mathcal{I}^D\left(V_h^{k,d}(\mathcal{T}_h)\right) = \left\{i \in \mathcal{I}\left(V_h^{k,d}(\mathcal{T}_h)\right) \,:\, x_i \in \mathcal{X}_h^{k,d} \wedge x_i \in \Gamma_D\right\}$$

where $\mathcal{X}_h^{k,d}$ is set of global Lagrange points. We already defined earlier that the test space V would fulfill the zero Dirichlet boundary condition. Then, it becomes

$$V_{h,0}^{k,d}(\mathcal{T}_h) = \left\{v \in V_h^{k,d}(\mathcal{T}_h) \,:\, v(x_i) = 0 \quad \forall i \in \mathcal{I}^D\left(V_h^{k,d}(\mathcal{T}_h)\right)\right\}.$$

For the trial space, it gets more complicated to define. We start with

$$u_{h,g} = \sum_{i \in \mathcal{I}\left(V_h^{k,d}(\mathcal{T}_h)\right)} g(x_i)\phi_i \qquad \text{for all } i \in \mathcal{I}^D\left(V_h^{k,d}(\mathcal{T}_h)\right).$$

Then the trial space is

$$U_h^{k,d}(\mathcal{T}_h) = \left\{u \in V_h^{k,d}(\mathcal{T}_h) \,:\, u = u_{h,g} + w \wedge w \in V_{h,0}^{k,d}(\mathcal{T}_h)\right\}$$

which is actually not a function space because it is not closed under addition. But in practice, we can explicitly include the Dirichlet boundary on the degree of freedom in an algebraic problem. And also, our implementation in `Dune` can construct a subspace of finite element space in which we can leave out the basis functions that are corresponding to the Dirichlet boundary.

Element-wise Computations

This subsection would introduce the element-wise evaluations as well as some important notations we suppose to use for the rest of this thesis. We look into each term in $R(z)$ from (2.4) in order to see the element-wise computations. Let us denote some notations regarding the classification of element-wise computation in `Dune` sense.

1) The superscript V, B, and S would be assigned for Volume Integrals, Boundary Integrals, and Skeleton Integrals respectively.

2) The α-terms refer to the integrals which depend on both trial and test spaces. On the other hand, the λ-terms refer to the integrals which depends only on test space.

3) The set of faces (or codimension 1) of the elements is denoted by $\mathcal{F}_h$. It can be partitioned into $\mathcal{F}_h^i, \mathcal{F}_h^N$, and $\mathcal{F}_h^D$. $\mathcal{F}_h^i$ is called a set of *interior skeleton* which is the intersection of elements. $\mathcal{F}_h^N$ and $\mathcal{F}_h^D$ are formed by the intersections of elements and domain boundaries which $\mathcal{F}_h^N$ is called a set of *Neumann boundary intersections* and $\mathcal{F}_h^D$ is called a set of *Dirichlet boundary intersections.* Carefully looking at the face F, that is, the intersection of an element with another element or domain boundary $\partial\Omega$, we propose further notation regarding the associated element. In `Dune` language, we denote the terms *inside* element T_F^- and *outside* element T_F^+ with respect to the direction of the outer normal vector $\boldsymbol{\nu}_F$ that always points from inside to outside element and from inside element to outside of the domain. There would be more explanation in Section 2.3

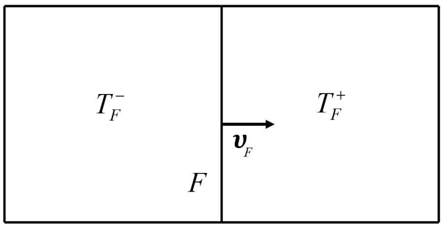


Figure 2.3: Illustrations of *inside* element T_F^-, *outside* element T_F^+, and outer normal vector $\boldsymbol{\nu}_F$.

Thereby, our residual form (2.3) can be a summation of these following labeled terms

$$r\left(u,v\right) = \sum_{T\in\mathcal{T}_h} \alpha_T^V(u,v) + \sum_{T\in\mathcal{T}_h} \lambda_T^V(v) + \sum_{F\in\mathcal{F}_h^N} \lambda_F^B(v) \tag{2.6}$$

where

$$\alpha_T^V(u,v) = \int_T \varphi(u, \|\nabla u\|)\, \nabla u \cdot \nabla v\, dx, \quad \lambda_T^V(v) = -\int_T fv\, dx, \quad \lambda_F^B(v) = \int_F jv\, ds.$$

We start with the α-**Volume** term: For any $(T,m)\in C(i)$ we get

$$\begin{aligned}\alpha_T^V(u_h,\phi_i) &= \int_T \varphi_h\, \nabla u_h \cdot \nabla \phi_i\, dx \\ &= \int_T \varphi_h \left(\sum_j (z)_j \left(\nabla\phi_j \cdot \nabla\phi_i\right) \right) dx \\ &= \int_{\hat{T}} \hat{\varphi}_h \left(\sum_n (z)_{g_T(n)} (J_{\mu_T}^{-T}(\hat{x}) \hat{\nabla} p_n^{\hat{T}}(\hat{x})) \cdot (J_{\mu_T}^{-T}(\hat{x}) \hat{\nabla} p_m^{\hat{T}}(\hat{x})) \right) \\ &\qquad |\mathrm{det} J_{\mu_T}(\hat{x})|\, d\hat{x}.\end{aligned}$$

Note that the integral can be computed by numerical integration in appropriate order and the gradient of functions can also be computed in reference element by this chain rule property:

$$\nabla w(\mu_T(\hat{x})) = J_{\mu_T}^{-T}(\hat{x})\hat{\nabla}\hat{w}(\hat{x}).$$

We also remark that

$$\varphi_h := \varphi(u_h, \|\nabla u_h\|) = \varphi\left(\sum_j (z)_j \phi_j, \left\|\sum_j (z)_j \nabla\phi_j\right\|\right),$$

and

$$\hat{\varphi_h} := \varphi\left(\sum_n (z)_{g_T(n)} p_n^{\hat{T}}(\hat{x}), \left\|\sum_n (z)_{g_T(n)} (J_{\mu_T}^{-T}(\hat{x})\hat{\nabla} p_n^{\hat{T}}(\hat{x}))\right\|\right)$$

Next, we go to λ-**Volume** term:
For any $(T, m) \in C(i)$ we have

$$\lambda_T^V(\phi_i) = -\int_T f\phi_i \, dx = -\int_{\hat{T}} f(\mu_T(\hat{x})) p_m^{\hat{T}}(\hat{x}) |\det J_{\mu_T}(\hat{x})| \, d\hat{x}.$$

Lastly, the λ-**Boundary** term:
For $F \in \mathcal{F}_h^N$ and $(T_F^-, m) \in C(i)$ we obtain

$$\lambda_F^B(\phi_i) = \int_F j\phi_i \, ds = \int_{\hat{F}} j(\mu_F(\hat{x})) p_m^{\hat{T}}(\eta_F(\hat{x})) \sqrt{|\det(J_{\mu_F}^T(\hat{x}) J_{\mu_F}(\hat{x}))|} \, ds$$

Here, we need to introduce two more mappings because the integration is over a face not an element as we did before.

> $\mu_F : \hat{F} \to F$ maps a point from reference (element of the) face $\hat{F}$ to a corresponding point in physical face F in global coordinate
>
> $\eta_F : \hat{F} \to \hat{T}$ maps a point from reference (element of the) face $\hat{F}$ to a corresponding point in reference element $\hat{T}$

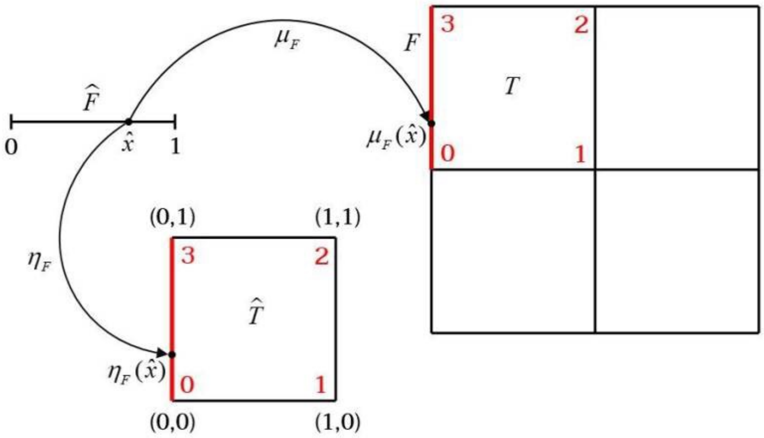


Figure 2.4: Illustrations of additional map μ_F and η_F.

With these element-wise computations we have shown above, we are eventually able to evaluate the algebraic residual $R(z)$. In order to solve the algebraic problem $R(z) = 0$ with Newton's method, the information of the Jacobian of R is required which we observe that

1) the entries of the Jacobian can be computed element by element

2) it depends only on α-terms

3) it can be achieved numerically by finite differences.

2.3 Discontinuous Galerkin method

We carry on with another discretization scheme with a slightly different idea. Discontinuous Galerkin method (DG) is also working based on discretizing the weak formulation as in finite element. The discontinuity is allowed at the element boundaries. Therefore, the basis functions could be constructed in such a way that their support is only in their element. Then, the finite element solution might have different values across the faces and edges. In addition, DG approach takes the Dirichlet boundary condition as additional terms in the weak formulation as so-called *penalty* terms following the idea of Nitsche's method [Nit71]. A variety of penalty terms would be described in this section. For more explanations, see [Riv08].

Let us start with the same model problem (2.1) but consider a slightly different test space $V = H^1(\Omega)$ without any restriction to zero at Dirichlet boundary Γ_D as before. Multiplying equation (2.1a) again by any function $v \in V$ and then integrating by parts, we obtain

$$\int_\Omega \varphi \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} \varphi \nabla u \cdot \boldsymbol{\nu} v \, ds = \int_\Omega f v \, dx. \tag{2.7}$$

Here, we see that the boundary integral $\int_{\partial\Omega} \varphi \,\nabla u \cdot \boldsymbol{\nu}\, v\, ds$ is not vanished. We purposely keep it so that the boundary conditions can be built in the weak sense. Note that we write $\varphi := \varphi(u, \|\nabla u\|)$ in short here.

Before we go any further, we would like to introduce the finite element space for DG first. Since we are working on the cube element as the same in the previous section, but relaxing the continuity constraint, the finite element space $V_h^{k,d}(\mathcal{T}_h)$ of degree k in d dimension on mesh $\mathcal{T}_h$ would be defined by

$$V_h^{k,d}(\mathcal{T}_h) = \mathcal{Q}_h^{k,d}(\mathcal{T}_h) := \left\{ v \in L^2(\Omega) \,:\, \forall T \in \mathcal{T}_h : v|_T = \mu_T \circ p_T \wedge p_T \in \mathbb{Q}_h^{k,d} \right\}. \tag{2.8}$$

The only difference is $v \in L^2(\Omega)$. Consequently, v is piecewise continuous on an element and could be discontinuous at the interfaces. Then, a unique value might not be defined on face F. Thus, we have to define the average and jump of v at the face F by

$$\{\!\{v\}\!\} = \frac{1}{2}\left(v|_{T_F^-} + v|_{T_F^+} \right), \qquad [\![v]\!] = v|_{T_F^-} - v|_{T_F^+}.$$

And we extend for the special case that the average and jump at the domain boundary $\partial\Omega$

$$\{\!\{v\}\!\} = [\![v]\!] = v$$

Next, we return to the weak formulation of the problem (2.1a) but this time we multiply by the test function $v \in V_h^{k,d}(\mathcal{T}_h)$ and then integrate over the whole domain Ω. Note that we can only do cell-wise integration by parts for the reason that v is piecewise continuous on an element. We obtain

$$\sum_{T \in \mathcal{T}_h} \int_T \varphi \,\nabla u \cdot \nabla v \, dx \;-\; \sum_{F \in \mathcal{F}_h} \int_F [\![\varphi \,\nabla u \cdot \boldsymbol{\nu}\, v]\!] \, ds \;=\; \sum_{T \in \mathcal{T}_h} \int_T f v \, dx. \tag{2.9}$$

We already denoted in the previous section that $\mathcal{F}_h = \mathcal{F}_h^i \cup \mathcal{F}_h^N \cup \mathcal{F}_h^D$. Hence, we have

$$\sum_{F \in \mathcal{F}_h} \int_F [\![\varphi \,\nabla u \cdot \boldsymbol{\nu}\, v]\!] \, ds \;=\; \sum_{F \in \mathcal{F}_h^i} \int_F [\![\varphi \,\nabla u \cdot \boldsymbol{\nu}\, v]\!] \, ds - \sum_{F \in \mathcal{F}_h^N} \int_F j v \, ds + \sum_{F \in \mathcal{F}_h^D} \int_F \varphi \,\nabla u \cdot \boldsymbol{\nu}\, v \, ds. \tag{2.10}$$

Using the property that $[\![\varphi \,\nabla u \cdot \boldsymbol{\nu} v]\!] = \{\!\{\varphi \,\nabla u \cdot \boldsymbol{\nu}\}\!\}[\![v]\!] + [\![\varphi \,\nabla u \cdot \boldsymbol{\nu}]\!]\{\!\{v\}\!\}$ and due to the fact that the exact solution u satisfies $[\![\varphi \,\nabla u \cdot \boldsymbol{\nu}]\!] = 0$ across the interfaces, the equation (2.10) becomes

$$\sum_{F \in \mathcal{F}_h} \int_F [\![\varphi \,\nabla u \cdot \boldsymbol{\nu}\, v]\!] \, ds \;=\; \sum_{F \in \mathcal{F}_h^i} \int_F \{\!\{\varphi \,\nabla u \cdot \boldsymbol{\nu}\}\!\}[\![v]\!] \, ds - \sum_{F \in \mathcal{F}_h^N} \int_F j v \, ds + \sum_{F \in \mathcal{F}_h^D} \int_F \varphi \,\nabla u \cdot \boldsymbol{\nu}\, v \, ds. \tag{2.11}$$

Substituting (2.11) into (2.9) and subtract $\sum\limits_{T\in\mathcal{T}_h}\int\limits_T f v\,dx$ from both sides. We get

$$\begin{aligned}\sum_{T\in\mathcal{T}_h}\int_T \varphi\,\nabla u\cdot\nabla v\,dx \;-\; \sum_{T\in\mathcal{T}_h}\int_T f v\,dx & \\ -\sum_{F\in\mathcal{F}_h^i}\int_F \{\!\{\varphi\,\nabla u\cdot\boldsymbol{\nu}\}\!\}[\![v]\!]\,ds & = 0. \\ +\sum_{F\in\mathcal{F}_h^N}\int_F jv\,ds \;-\; \sum_{F\in\mathcal{F}_h^D}\int_F \varphi\,\nabla u\cdot\boldsymbol{\nu}\,v\,ds & \end{aligned} \tag{2.12}$$

Next, in order to symmetrize $\sum\limits_{F\in\mathcal{F}_h^i}\int\limits_F \{\!\{\varphi\,\nabla u\cdot\boldsymbol{\nu}\}\!\}[\![v]\!]\,ds$ term we add the additional term $\epsilon\sum\limits_{F\in\mathcal{F}_h^i}\int\limits_F \{\!\{\varphi\,\nabla v\cdot\boldsymbol{\nu}\}\!\}[\![u]\!]\,ds$. Also, in order to symmetrize $\sum\limits_{F\in\mathcal{F}_h^D}\int\limits_F \varphi\,\nabla u\cdot\boldsymbol{\nu}\,v\,ds$ term we try to do the same way by adding $\epsilon\sum\limits_{F\in\mathcal{F}_h^D}\int\limits_F \varphi\,\nabla v\cdot\boldsymbol{\nu}\,u\,ds$ but it would make the system inconsistent. Then, we remedy it by subtracting by $\epsilon\sum\limits_{F\in\mathcal{F}_h^D}\int\limits_F \varphi\,\nabla v\cdot\boldsymbol{\nu}\,g\,ds$ because we know that $u-g=0$ on Dirichlet boundary Γ_D. So in the end, we have added the productive zero term theoretically. Then, we achieve

$$\begin{aligned}\sum_{T\in\mathcal{T}_h}\int_T \varphi\,\nabla u\cdot\nabla v\,dx \;-\; \sum_{T\in\mathcal{T}_h}\int_T f v\,dx & \\ -\sum_{F\in\mathcal{F}_h^i}\int_F \{\!\{\varphi\,\nabla u\cdot\boldsymbol{\nu}\}\!\}[\![v]\!]\,ds \;+\; \epsilon\sum_{F\in\mathcal{F}_h^i}\int_F \{\!\{\varphi\,\nabla v\cdot\boldsymbol{\nu}\}\!\}[\![u]\!]\,ds & = 0 \\ +\sum_{F\in\mathcal{F}_h^N}\int_F jv\,ds \;-\; \sum_{F\in\mathcal{F}_h^D}\int_F \varphi\,\nabla u\cdot\boldsymbol{\nu}\,v\,ds \;+\; \epsilon\sum_{F\in\mathcal{F}_h^D}\int_F \varphi\,(u-g)\,\nabla v\cdot\boldsymbol{\nu}\,ds & \end{aligned} \tag{2.13}$$

where ϵ plays a role in classifying the interior penalty methods. Here, we can see that we are able to enforce the Dirichlet boundary condition on our weak formulation. Last but not least, the penalty term comes up to penalize the interface jumps

$$\gamma\sum_{F\in\mathcal{F}_h^i\cup\mathcal{F}_h^D}\int_F [\![u]\!][\![v]\!]\,ds$$

where γ is a mesh-dependent penalty parameter that has to be sufficiently large enough in practice. We add the term $\gamma\sum\limits_{F\in\mathcal{F}_h^D}\int\limits_F g\,v\,ds$ in similar way for consistency reason. Eventually, our weak formulation for DG has become

$$
\begin{aligned}
&\sum_{T\in\mathcal{T}_h}\int_T \varphi\,\nabla u\cdot\nabla v\,dx \\
&-\sum_{F\in\mathcal{F}_h^i}\int_F\Big(\{\!\{\varphi\,\nabla u\cdot\boldsymbol{\nu}\}\!\}\llbracket v\rrbracket \;-\; \epsilon\,\{\!\{\varphi\,\nabla v\cdot\boldsymbol{\nu}\}\!\}\llbracket u\rrbracket \;-\; \gamma\llbracket u\rrbracket\llbracket v\rrbracket\Big)\,ds \\
&-\sum_{F\in\mathcal{F}_h^D}\int_F\Big(\varphi\,\nabla u\cdot\boldsymbol{\nu}\,v \;-\; \epsilon\,\varphi\,(u-g)\,\nabla v\cdot\boldsymbol{\nu} \;-\; \gamma\,uv\Big)\,ds \qquad\qquad = 0. \\
&-\sum_{T\in\mathcal{T}_h}\int_T f v\,dx \\
&+\sum_{F\in\mathcal{F}_h^N}\int_F jv\,ds-\sum_{F\in\mathcal{F}_h^D}\int_F\gamma\,gv\,ds
\end{aligned}
\tag{2.14}
$$

Or it can be rewritten in our acquainted residual form, that is

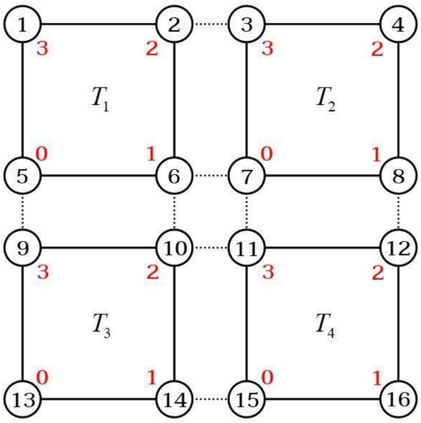
$$
\text{Find } u\in V_h^{k,d}(\mathcal{T}_h) \text{ s.t. :}\quad r(u,v)=0\quad\forall v\in V_h^{k,d}(\mathcal{T}_h) \tag{2.15}
$$

where $r(u,v)$ represents the LHS of equation (2.14). We intentionally rearrange terms like this in order that we can easily separate the residual form into our notations α-**Volume**, α-**Skeleton**, α-**Boundary**, λ-**Volume**, and λ-**Boundary**, respectively.

α-**Volume**, and λ-**Volume** terms were already shown in finite element section. For λ-**Boundary**, the integral term depending on the chosen interior penalty method on the Dirichlet Boundary is supplemented. α-**Skeleton** and α-**Boundary** appear for the first time here. Thus, the residual form can be expressed in the following way

$$
r\,(u,v)=\sum_{T\in\mathcal{T}_h}\alpha_T^V(u,v)+\sum_{F\in\mathcal{F}_h^i}\alpha_F^S(u,v)+\sum_{F\in\mathcal{F}_h^D}\alpha_F^B(u,v)+\sum_{T\in\mathcal{T}_h}\lambda_T^V(v)+\sum_{F\in\mathcal{F}_h^N\cup\mathcal{F}_h^D}\lambda_F^B(v) \tag{2.16}
$$

Due to the lack of continuity on mesh faces, this leads to the fact that DG has more degrees of freedom (unknowns) compared to the continuous case (see Figure 2.5). But this also leads to the advantage that we have more flexibility of basis functions. We can choose the basis functions which are locally defined on each element and have support only in their own element. The local Lagrange basis we have introduced in the previous section can also be reused.



$$\begin{array}{llll} g_{T_1}(0) = 5, & g_{T_1}(1) = 6, & g_{T_1}(2) = 2, & g_{T_1}(3) = 1 \\ g_{T_2}(0) = 7, & g_{T_2}(1) = 8, & g_{T_2}(2) = 4, & g_{T_2}(3) = 3 \\ g_{T_3}(0) = 13, & g_{T_3}(1) = 14, & g_{T_3}(2) = 10, & g_{T_3}(3) = 9 \\ g_{T_4}(0) = 15, & g_{T_4}(1) = 16, & g_{T_4}(2) = 12, & g_{T_4}(3) = 11. \end{array}$$

Figure 2.5: Example of *local-to-global* index map for discontinuous case.

ϵ denotes an interior penalty parameter which can separate several approaches for penalty methods as the following

- If $\epsilon = -1$ and γ is large enough, this method is called symmetric interior penalty Galerkin method (SIPG). Note that it is our default approach for the interior penalty method in this thesis.

- If $\epsilon = 0$ and γ is large enough, this method is called incomplete interior penalty Galerkin method (IIPG).

- If $\epsilon = +1$ and γ is a nonnegative number, this method is called nonsymmetric interior penalty Galerkin method (NIPG).

The following part would show the element-wise computation for this DG approach with symmetric interior penalty Galerkin method (SIPG). There is one term that will appear in this computation which is unit normal vector $\boldsymbol{\nu}$ which can be computed in the reference element by the following formula

$$\boldsymbol{\nu}(x) = J_{\mu_T}^{-T}(\hat{x})\,\hat{\boldsymbol{\nu}}(\hat{x}).$$

And we denote that our notations for the function v on the face F of the inside element T_F^- and the outside element T_F^+ are written as v^- and v^+ instead of $v|_{T_F^-}$ and $v|_{T_F^+}$. Note that the element-wise computation for α-**Volume** and λ-**Volume** are identical to ones in finite element.
We would like to start with λ-**Boundary**:

$$\sum_{F \in \mathcal{F}_h^N \cup \mathcal{F}_h^D} \lambda_F^B(v) = \sum_{F \in \mathcal{F}_h^N} \lambda_F^{B,N}(v) + \sum_{F \in \mathcal{F}_h^D} \lambda_F^{B,D}(v)$$

For $F \in \mathcal{F}_h^N$, we have already explained. Then, we would illustrate only $F \in \mathcal{F}_h^D$.

For $(T_F^-, m) \in C(i)$ we obtain

$$\begin{aligned}\lambda_F^{B,D}(\phi_i) &= -\int_F \gamma g \phi_i \, ds \\ &= -\int_{\hat F} \gamma \, g(\mu_F(\hat x)) \, p_m^{\hat T}(\eta_F(\hat x)) \sqrt{|\det(J_{\mu_F}^T(\hat x) J_{\mu_F}(\hat x))|} \, ds.\end{aligned}$$

Next, we proceed to α-**Boundary**:
For $F \in \mathcal{F}_h^D$ and $(T_F^-, m) \in C(i)$ we obtain

$$\begin{aligned}
\alpha_F^B(u_h, \phi_i) &= -\int_F \left((\varphi_h \, \nabla u_h \cdot \boldsymbol{\nu}) \phi_i \; + \; \varphi_h \, (u_h - g) \, (\nabla \phi_i \cdot \boldsymbol{\nu}) \; - \; \gamma u_h \phi_i \right) \, ds \\
&= -\int_F \Bigg(\varphi_h \, \sum_j (z)_j \, (\nabla \phi_j \cdot \boldsymbol{\nu}) \, \phi_i \; + \; \varphi_h \, \left(\left(\sum_j (z)_j \phi_j \right) - g \right) (\nabla \phi_i \cdot \boldsymbol{\nu}) \\
&\qquad - \gamma \left(\sum_j (z)_j \phi_j \right) \phi_i \Bigg) \; ds \\
&= -\int_{\hat F} \Bigg(\hat\varphi_h^- \, \left(\sum_n (z)_{g_{T_F^-}(n)} \left((J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat x)) \hat\nabla p_n^{\hat T}(\eta_F(\hat x))) \cdot (J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat x)) \hat{\boldsymbol{\nu}}(\eta_F(\hat x))) \right) \right) p_m^{\hat T}(\eta_F(\hat x)) \\
&\quad + \hat\varphi_h^- \, \left(\sum_n (z)_{g_{T_F^-}(n)} p_n^{\hat T}(\eta_F(\hat x)) - g(\mu_F(\hat x)) \right) \left((J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat x)) \hat\nabla p_m^{\hat T}(\eta_F(\hat x))) \cdot (J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat x)) \hat{\boldsymbol{\nu}}(\eta_F(\hat x))) \right) \\
&\quad - \gamma \, \left(\sum_n (z)_{g_{T_F^-}(n)} p_n^{\hat T}(\eta_F(\hat x)) \right) \, p_m^{\hat T}(\eta_F(\hat x)) \Bigg) \sqrt{|\det(J_{\mu_F}^T(\hat x) J_{\mu_F}(\hat x))|} \, ds.
\end{aligned}$$

Here, we remark that φ_h is already denoted in the finite element section. And we introduce

$$\hat\varphi_h^- := \varphi \left(\sum_n (z)_{g_{T_F^-}(n)} p_n^{\hat T}(\eta_F(\hat x)), \; \left\| \sum_n (z)_{g_{T_F^-}(n)} (J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat x)) \hat\nabla p_n^{\hat T}(\eta_F(\hat x))) \right\| \right)$$

and

$$\hat\varphi_h^+ := \varphi \left(\sum_n (z)_{g_{T_F^+}(n)} p_n^{\hat T}(\eta_F(\hat x)), \; \left\| \sum_n (z)_{g_{T_F^+}(n)} (J^{-T}_{\mu_{T_F^+}}(\eta_F(\hat x)) \hat\nabla p_n^{\hat T}(\eta_F(\hat x))) \right\| \right)$$

which would be employed in the next part.
We go forward to the last term, α-**Skeleton**:
For $(T_F^-, m) \in C(i)$ and $F \in \partial T_F^- \cap \mathcal{F}_h^i$. As the average and jump involve in this part, we note that in element T_F^-, we have chosen the basis functions ϕ which have support only on T_F^- so that $\{\!\{\phi\}\!\} = \frac{1}{2}\phi$ and $[\![\phi]\!] = \phi$.

$$\alpha_F^S(u_h, \phi_i) = -\int_F \left(\{\!\{\nabla u_h \cdot \boldsymbol{\nu}\}\!\} [\![\phi_i]\!] \; + \; \{\!\{\nabla \phi_i \cdot \boldsymbol{\nu}\}\!\} [\![u_h]\!] \; - \; \gamma [\![u_h]\!] [\![\phi_i]\!] \right) \, ds$$

We split it into three terms for more comprehensible realization.

$$
\begin{aligned}
&\int_F \{\!\{\varphi_h \, \nabla u_h \cdot \boldsymbol{\nu}\}\!\} [\![\phi_i]\!] \, ds \\
&= \int_F \frac{1}{2} \left(\varphi_h^- \sum_j (z^-)_j \, (\nabla \phi_j \cdot \boldsymbol{\nu}) + \varphi_h^+ \sum_j (z^+)_j \, (\nabla \phi_j \cdot \boldsymbol{\nu}) \right) \phi_i \, ds \\
&= \int_{\hat{F}} \frac{1}{2} \left(\hat{\varphi}_h^- \sum_n (z^-)_{g_{T_F^-}(n)} \left(J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat{x})) \hat{\nabla} p_n^{\hat{T}}(\eta_F(\hat{x})) \right) + \right. \\
&\qquad\qquad \left. \hat{\varphi}_h^+ \sum_l (z^+)_{g_{T_F^+}(l)} \left(J^{-T}_{\mu_{T_F^+}}(\eta_F(\hat{x})) \hat{\nabla} p_l^{\hat{T}}(\eta_F(\hat{x})) \right) \right) \\
&\qquad \cdot \left(J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat{x})) \hat{\boldsymbol{\nu}}(\eta_F(\hat{x})) \right) \left(p_m^{\hat{T}}(\eta_F(\hat{x})) \right) \sqrt{|\det(J^T_{\mu_F}(\hat{x}) J_{\mu_F}(\hat{x}))|} \, ds.
\end{aligned}
$$

$$
\begin{aligned}
&\int_F \{\!\{\varphi_h \, \nabla \phi_i \cdot \boldsymbol{\nu}\}\!\} [\![u_h]\!] \, ds \\
&= \int_F \frac{1}{2} (\nabla \phi_j \cdot \boldsymbol{\nu}) \left(\varphi_h^- \sum_j (z^-)_j \phi_j \; - \; \varphi_h^+ \sum_j (z^+)_j \phi_j \right) ds \\
&= \int_{\hat{F}} \frac{1}{2} \left(\left(J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat{x})) \hat{\nabla} p_m^{\hat{T}}(\eta_F(\hat{x})) \right) \cdot \left(J^{-T}_{\mu_{T_F^-}}(\eta_F(\hat{x})) \hat{\boldsymbol{\nu}}(\eta_F(\hat{x})) \right) \right) \\
&\qquad \left(\hat{\varphi}_h^- \sum_n (z^-)_{g_{T_F^-}(n)} p_n^{\hat{T}}(\eta_F(\hat{x})) \; - \; \hat{\varphi}_h^+ \sum_l (z^+)_{g_{T_F^+}(l)} p_l^{\hat{T}}(\eta_F(\hat{x})) \right) \sqrt{|\det(J^T_{\mu_F}(\hat{x}) J_{\mu_F}(\hat{x}))|} \, ds.
\end{aligned}
$$

$$
\begin{aligned}
&\int_F \gamma [\![u_h]\!] [\![\phi_i]\!] \, ds \\
&= \int_F \gamma \left(\sum_j (z^-)_j \phi_j \; - \; \sum_j (z^+)_j \phi_j \right) \phi_i \, ds \\
&= \int_{\hat{F}} \gamma \left(\sum_n (z^-)_{g_{T_F^-}(n)} p_n^{\hat{T}}(\eta_F(\hat{x})) - \sum_l (z^+)_{g_{T_F^+}(l)} p_l^{\hat{T}}(\eta_F(\hat{x})) \right) \left(p_m^{\hat{T}}(\eta_F(\hat{x})) \right) \\
&\qquad \sqrt{|\det(J^T_{\mu_F}(\hat{x}) J_{\mu_F}(\hat{x}))|} \, ds
\end{aligned}
$$

which complete all contributions for DG with SIPG method.

2.4 Newton's Method for Nonlinear System of Equations

As the discretization schemes from the previous section typically result in the (nonlinear) algebraic system of equations, the further work is to solve that system. Newton's method is a powerful and well-known technique to solve the arisen (nonlinear) system of equations. We assume the system is given in the form

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0, \\ f_2(x_1, \ldots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, \ldots, x_n) &= 0, \end{aligned} \tag{2.17}$$

where we can write in the compact form by defining the vector-valued function

$$F(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix} = \begin{bmatrix} f_1(x_1, \ldots, x_n) \\ f_2(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{bmatrix} \tag{2.18}$$

where $x = [x_1, x_2, \ldots, x_n]^T$. So, we obtain $F(x) = 0$.

The idea behind Newton's method is based on the linear approximation of F at the current solution $x^k \in \mathbb{R}^n$. Let us start with the initial approximation $x^0 \in \mathbb{R}^n$ (assumably close to the solution x)

$$F(x) \approx F(x^0) + DF(x^0)(x - x^0)$$

where $DF(x)$ is an $n \times n$ *Jacobian* matrix, defined as

$$DF(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \cdots & \frac{\partial f_n}{\partial x_n}(x) \end{bmatrix}. \tag{2.19}$$

Since we want to find x such that $F(x) = 0$ and assume that the Jacobian matrix DF is invertible, we have a better approximate solution x^1

$$x^1 = x^0 - \left[DF(x^0)\right]^{-1} F(x^0).$$

Continuing in this way, we obtain even better approximate solutions. Then, it leads to the following iteration

$$x^{k+1} = x^k - \left[DF(x^k)\right]^{-1} F(x^k).$$

This iteration is called **"Newton Iteration"**. It is expected to give a quadratic convergence rate if the initial solution x^0 is adequately close to the true solution. The

Algorithm 1 Newton's Method for Nonlinear Systems

```
for $k = 0, 1, \dots, convergence$ do
    evaluation $F(x^k)$ and $DF(x^k)$
    solve the linear system $DF(x^k)v^k = F(x^k)$
    update $x^{k+1} = x^k - v^k$
end for
```

burdensome of each Newton iteration arises from the computation of the inverse of the Jacobian matrix at every iteration. But, in practice, it can be done by solving a linear system, i.e., For each step k, find a vector v^k that satisfies $DF(x^k)v^k = F(x^k)$. Thus, the new approximate solution can be obtained by $x^{k+1} = x^k - v^k$.

In fact, the standalone Newton method is locally convergent. To enhance the robustness of Newton when we are not sure whether our initial guess is close to the solution, the globalization techniques e.g. line-search (adjust the step length) and trust-region method (ideally choose a good direction) are invented to overcome this drawback. Note that we do not use the trust-region method in this study but more details and recent advances in trust-region methods can be found in [Yua94; Yua15]. Here, we would give a brief description only of the line-search method.

Line-Search Method

The method is to adjust the step length in an appropriate way (usually shorten). After getting the direction v^k to the next solution, we can restrict how far the current solution goes in that direction. The *update* step in Algorithm 1 can be written as

$$x^{k+1} = x^k - \alpha^k v^k \tag{2.20}$$

where α is sometimes called a damping parameter and (2.20) is called a damped Newton iteration. The parameter α^k can be chosen by the backtracking line search strategy. The technique we used in this study is from [HR89].

Algorithm 2 Line Search Method, Hackbusch and Reusken, 1989 [HR89]

```
Input $\tau$ (damping factor), $x$ (current solution), $v$ (Newton direction)
set $\alpha = 1, i = 0$
while $i \leq$ max_it do
    if $F(x - \alpha v) \leq (1 - \alpha/4)\, F(x)$ then
        break
    end if
    $\alpha = \tau\alpha$
    $i = i + 1$
end while
```

2.5 Methods for Solving Linear System

This section would describe methods for solving the linear system coming up between Newton iteration, see Algorithm 1. There are two classical approaches of solver so-called *direct* and *iterative* methods which have their own benefit. Assume that we want to solve a linear system

$$Ax = b$$

where A might represent the $n \times n$ *Jacobian* matrix from the previous section.

Direct solver theoretically provides an exact solution of a linear system after a finite number of steps. Many commonly known methods are LU, QR, or Cholesky factorizations. We are fortunate that matrix A which comes up from FEM is typically a sparse matrix. Hence, we can exploit a sparse direct solver which could reduce much complexity in terms of time and memory consumption. Nowadays, there are several softwares for the sparse direct solvers, for example, SuperLU [DGL97], UMFPACK [Dav04], and MUMPS [Ame+01; Ame+19] which are basically based on LU decomposition. However, the number of operations of the sparse direct solver is still an issue, especially in $3d$-problem. So, it might not be appropriate for a large linear system. But we can reduce it by looking into an alternative approach.

Another class is *iterative* solver which does not provide an exact but a *good enough* result. One starts with an initial guess x^0, it then gradually approaches the exact solution until a specified stopping criterion. It consumes significantly less memory compared to the direct solver. And the number of operations is much reduced. Therefore, it is more appropriate for the large system. In [Bar+94], they introduce two types of iterative methods so-called *stationary* and *instationary*. *Stationary* is older, simpler to implement, easier to understand the analysis, but usually less effective. The renowned ones are Jacobi, Gauss-Seidel, and SSOR method. *Instationary* is more recent and usually more effective. The oldest and best-known one is the Conjugate Gradient (CG) method which minimizes the energy norm of error in the *Krylov* subspace

$$\mathcal{K}_k = \operatorname{span}\{b, Ab, \dots, A^{k-1}b\}.$$

The idea of CG starts from the fact that solving a linear system $Ax = b$ is equivalent to finding a minimizer of the quadratic function

$$f(x) = \frac{1}{2}x^{\mathrm{T}}Ax - b^{\mathrm{T}}x$$

when A is symmetric positive definite (SPD). Then, in each iteration k, we seek an A-orthogonal search direction and find an approximated solution x^k in $\mathcal{K}_k$ which is (hopefully) closer to the exact solution. A Krylov solver as well as its variants could become more memory-efficient by taking advantage of the *matrix-free* strategy because it needs only the matrix-vector multiplication. The popular extensions for a nonsymmetric case are Generalized Minimal RESidual (GMRES) [Saa93; SS86] and BiConjugate Gradient Stabilized (BiCGStab) [Vor92] methods. More variants and details of iterative solvers can be found in [Saa03; DJN15; Bas20; GV13].

All of them can also be implemented for running in parallel. The next concern is the convergent issue. If matrix A is not well-conditioned, the iterative solver might not converge. The convergence rate of CG depends on a condition number

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}.$$

If it is close to 1 (eigenvalues are clustered), we can say that A is well-conditioned. So, the linear system is easy to solve by an iterative solver. We can also enhance this spectral property by *preconditioner*. It could help to transform the system and hopefully, the transformed system has a smaller condition number. The standard approach is to multiply the system with a (nonsingular) matrix B, then the transformed linear system looks like

$$BAx = Bb \tag{2.21}$$

where B is chosen such that BA has a better condition but the transformed system still has the same solution as the original system. There are several approaches to apply the preconditioners. One approach we are interested in and would describe in the next section is using the idea of domain decomposition as a preconditioner.

Algorithm 3 Preconditioned Conjugate Gradient Method (PCG)

```
**Input** initial guess $x^0$, tolerance parameter $\epsilon$, $(x, y)$ stands for dot product $x^T y$
$r^0 = b - Ax^0$
$z^0 = Br^0$
$p^0 = z^0$
$i = 0$
**while** $\|r^{(i)}\|/\|r^0\| > \epsilon$ **do**
    $q^{(i)} = Ap^{(i)}$
    $\alpha^{(i)} = (z^{(i)}, r^{(i)})/(p^{(i)}, q^{(i)})$
    $x^{(i+1)} = x^{(i)} + \alpha^{(i)} p^{(i)}$
    $r^{(i+1)} = r^{(i)} - \alpha^{(i)} q^{(i)}$
    $z^{(i+1)} = Br^{(i+1)}$
    $\beta^{(i)} = (r^{(i+1)}, z^{(i+1)})/(r^{(i)}, z^{(i)})$
    $p^{(i+1)} = z^{(i+1)} + \beta^{(i)} p^{(i)}$
    $i = i + 1$
**end while**
```

We note that in our main scheme we would employ both *direct* and *iterative* solvers. For more details, we would explain it in Chapter 3.

2.6 Domain Decomposition

The classical example of domain decomposition would be mostly referred to as a pioneer work from Schwarz in 1870 [Sch70] where he has proved the existence and uniqueness of the Poisson problem on a union of simple geometries. It was not initially supposed to be a numerical algorithm at that time. But after that, it inspires parallel computing and develops this research area a lot. The idea is that instead of doing computation on the whole considered domain Ω, we can do it on the subdomain Ω_k, which is a partition of Ω so that the local computation is pretty much cheaper. Then, we communicate the local solutions to glue them up from each subdomain Ω_k. Keep iterating this procedure until the approximate solution is enhanced enough to reach the stopping criterion. Notably, these subdomain problems are able to be solved in parallel. Furthermore, this Schwarz algorithm is extended to become an efficient preconditioner for iterative linear solvers, in particular, the Krylov methods such as Conjugate Gradient (CG) and GMRES. Note that in this section, we derive the formulation in an algebraic way.

Let us start with the PDE as the model problem. After that, we insert a basis representation so that we obtain the nonlinear algebraic equation or ultimately, the linear system $Ax = b$ where $A \in \mathbb{R}^{n\times n}$, and $x, b \in \mathbb{R}^n$. n represents the number of the corresponding nodes or degrees of freedom. Let us define the index set $\mathcal{I}$

$$\mathcal{I} := \{1, \dots, n\}, \qquad |\mathcal{I}| := n.$$

Next, we would like to define the partitioning of the index set $\hat{\mathcal{I}}_k \subset \mathcal{I}$

$$\bigcup_{k=1}^{p} \hat{\mathcal{I}}_k = \mathcal{I} \qquad\qquad \hat{\mathcal{I}}_k \cap \hat{\mathcal{I}}_l = \emptyset,\ k \neq l.$$

Apparently, $\hat{\mathcal{I}}_k \subset \mathcal{I}$. Before we go any further, let us define the graph $G(A) = (\mathcal{I}, E)$ where $E = \{(i,j) \in \mathcal{I} \times \mathcal{I} \mid a_{ij} \neq 0\}$ represents the set of edges. Then, we would like to introduce the extension operator $\mathcal{E}$

$$\mathcal{E}(\mathcal{I}') = \mathcal{I}' \cup \{\, j \mid (i,j) \in E \ \wedge \ i \in \mathcal{I}'\}\,.$$

It interprets that the index set could extend by a layer of nodes. By this operator, we could achieve the extended index set $\mathcal{I}_k := \mathcal{I}_k^{\delta},\ \delta \in \mathbb{N}$ where

$$\mathcal{I}_k^{i+1} = \mathcal{E}(\mathcal{I}_k^{i}), \text{ and } \quad \mathcal{I}_k^{0} = \hat{\mathcal{I}}_k$$

It is also clear to be seen that $\bigcup_{k=1}^{p} \mathcal{I}_k = \mathcal{I}$, and we denote $|\mathcal{I}_k| = n_k$. Next, we would like to introduce restriction operators $R_k : \mathbb{R}^n \to \mathbb{R}^{n_k}$, in such a way that a local-to-global map $g_k : \{1, \dots, n_k\} \to \mathcal{I}$ satisfies

$$(R_k x)_i = (x)_{g_k(i)}$$

where $i \in \{1, \dots, n_k\}$. And R_k can be expressed by $n_k \times n$ matrix. With these restriction operators, we can define submatrix

$$A_k = R_k A R_k^T.$$

Note that R_k^T is so-called prolongation (extension) operator. We can also denote restriction operators $\hat{R}_k : \mathbb{R}^n \to \mathbb{R}^{n_k}$ which fulfill the partition of unity (PU) property

$$\sum_{k=1}^{p} R_k^T \hat{R}_k = I = I^T = \sum_{k=1}^{p} \hat{R}_k^T R_k.$$

One can express $\hat{R}_k$ in terms of $\hat{R}_k = D_k R_k$ where D_k can be a particular choice of a diagonal matrix that fulfills the partition of unity [DJN15]. Thus, we can define the additive Schwarz (AS) and multiplicative Schwarz (MS) methods as Algorithms 4 and 5.

Algorithm 4 Additive Schwarz Method

```
for $i = 0, 1, \ldots$ do
    $x^{i+1} = x^i + \omega \left[\sum_k R_k^T \left(R_k A R_k^T\right)^{-1} R_k\right] (b - Ax^i)$
end for
```

Algorithm 5 Multiplicative Schwarz Method

```
for $i = 0, 1, \ldots$ do
    for $k = 1, \ldots, p$ do
        $x^{i+\frac{k}{p}} = x^{i+\frac{k-1}{p}} + \left[R_k^T \left(R_k A R_k^T\right)^{-1} R_k\right] \left(b - Ax^{i+\frac{k-1}{p}}\right)$
    end for
end for
```

One might notice that Algorithm 4 is equivalent to Richardson's iteration with the preconditioner $\left[\sum_k R_k^T \left(R_k A R_k^T\right)^{-1} R_k\right]$. Here, we call $B = \sum_k R_k^T \left(R_k A R_k^T\right)^{-1} R_k$ an *additive Schwarz preconditioner* for which it is proved that the condition number is

$$\kappa(BA) \leq c(1 + \frac{H}{\delta})H^{-2} \tag{2.22}$$

where $H \approx \text{diam}(\Omega)/p^{\frac{1}{d}}$, δ refers to the overlap size, and c is a constant independent of H, δ. Note that ω is a damping parameter concerning the convergence issue. But in practice, it can be omitted, if we employ the additive Schwarz as a precondioner in the CG method [Bas20]. One can clearly see that the additive Schwarz is quite a straightforward parallel computing since all subdomain problems can be solved simultaneously. But in the case of multiplicative Schwarz, it needs some trick. Let us say that we can appropriately distribute $\mathcal{I}_k$ where we can suppose that $J = \{1, \ldots, p\}$ can be defined by

$$J = \bigcup_{n=1}^{c} J_n \quad , J_i \cap J_j = \emptyset, \, i \neq j$$

satisfying that $R_i A R_j^T = 0$ for all $i, j \in J_n$ where c refers to *color*. It is sufficient to be 4 for structured mesh in two-dimension, and 8 in three-dimension. This technique is called the *coloring* technique [CW93]. It interprets that the subproblems with the same *color* are independent (more proofs can be found in [Bas20]). So, in the end, the multiplicative Schwarz method can be parallelized. In addition, we also propose the restricted additive Schwarz (RAS) and restricted multiplicative Schwarz (RMS) methods as the following Algorithms 6 and 7.

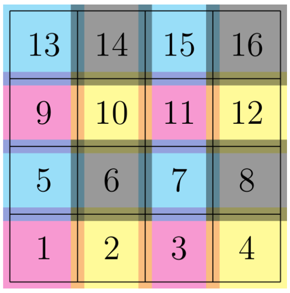


Figure 2.6: Coloring of a structured mesh in 2D.

Generally, we do not directly employ the Schwarz methods as a solver, but we preferably employ them as a *preconditioner*. Note that restricted version would actually provide a nonsymmetric system if it is used as a preconditioner even when A is symmetric. Due to that reason, it requires a nonsymmetric Krylov solver like GMRES or BiCGStab.

Algorithm 6 Restricted Additive Schwarz Method

```
for $i = 0, 1, \ldots$ do
    $x^{i+1} = x^i + \omega \left[ \sum_k \hat{R}_k^T \left( R_k A R_k^T \right)^{-1} R_k \right] (b - Ax^i)$
end for
```

Algorithm 7 Restricted Multiplicative Schwarz Method

```
for $i = 0, 1, \ldots$ do
    for $k = 1, \ldots, p$ do
        $x^{i+\frac{k}{p}} = x^{i+\frac{k-1}{p}} + \left[ \hat{R}_k^T \left( R_k A R_k^T \right)^{-1} R_k \right] \left( b - Ax^{i+\frac{k-1}{p}} \right)$
    end for
end for
```

Coarse Grid Correction

One might notice that the condition number in (2.22) depends heavily on H which is related to a number of subdomains p, that is, if we increase p (with keeping the global problem size), the convergence rate is diminished. The reason is told from the Fourier modes analysis that the high-frequency errors are reduced really well. It is as opposed to the low-frequency parts which are reduced slower. Accordingly, using purely *one-level* might be not good for scalability. Thus, the *two-level* preconditioner turns up to remedy this issue via a *Coarse Grid Correction*. The added second level is for solving a linear system in a coarser grid which is called a coarse problem. It is typically a global problem but a smaller size which is proportional to the number of subdomains. So, the extra cost might be negligible if the number of subdomains is not too large [DJN15]. We would like to extend the definition of the restriction operator from fine grid to coarse grid $R_0 : \mathbb{R}^n \to \mathbb{R}^{n_H}$ where n_H is a number of degrees of freedom for a coarse problem.

Let us explain more details on each entry of R_0. Assume that V_h is a finite dimensional subspace and $V_h = \text{span}\{\phi_1, \ldots, \phi_n\}$. Then in a coarse grid, we also assume that we have a finite dimensional subspace $V_H \in V_h$ where $V_H = \text{span}\{\psi_1, \ldots, \psi_{n_H}\}$. And the relation between two sets of basis functions can be written as

$$\psi_i = \sum_j (R_0)_{ij} \phi_j$$

where we can see that it is a change of basis functions from fine level to coarse level. Then, we define $A_0 = R_0 A R_0^T$, an $n_H \times n_H$ matrix, represents a coefficient matrix for a coarse problem. Therefore, the additive Schwarz and multiplicative Schwarz methods with a coarse grid correction are written in Algorithms 8 and 9. Note that the multiplicative version can be defined in several ways, but here, we propose the symmetrized version.

Algorithm 8 Additive Schwarz Method with Coarse Grid Correction

```
for $i = 0, 1, \ldots$ do
    $x^{i+1} = x^i + \omega \left[ \sum_{k=0}^{p} R_k^T \left( R_k A R_k^T \right)^{-1} R_k \right] (b - Ax^i)$
end for
```

Algorithm 9 Multiplicative Schwarz Method with Coarse Grid Correction

```
for $i = 0, 1, \ldots$ do
    for $k = 1, \ldots, p$ do
        $x^{i+\frac{k}{2p+1}} = x^{i+\frac{k-1}{2p+1}} + \left[ R_k^T \left( R_k A R_k^T \right)^{-1} R_k \right] \left( b - Ax^{i+\frac{k-1}{2p+1}} \right)$
    end for
    $x^{i+\frac{p+1}{2p+1}} = x^{i+\frac{p}{2p+1}} + \left[ R_0^T \left( R_0 A R_0^T \right)^{-1} R_0 \right] \left( b - Ax^{i+\frac{p}{2p+1}} \right)$
    for $k = p, \ldots, 1$ do
        $x^{i+\frac{2p+2-i}{2p+1}} = x^{i+\frac{2p+1-i}{2p+1}} + \left[ R_k^T \left( R_k A R_k^T \right)^{-1} R_k \right] \left( b - Ax^{i+\frac{2p+1-i}{2p+1}} \right)$
    end for
end for
```

In [TW05], they have proved that the condition number of the two-level additive and multiplicative Schwarz becomes

$$\kappa(BA) \leq c(1 + \frac{H}{\delta}). \tag{2.23}$$

However, there are several approaches to define a *coarse problem*. The interesting example is the Nicolaides coarse space [Nic87] because the coarse system can be fully constructed in an algebraic way as well as its extension. This means that we do not explicitly need the coarse mesh to construct the coarse system.

Nicolaides Coarse Space

The simple and cheap coarse space was introduced by Nicolaides in 1987 [Nic87] in the idea of deflation subspace. We define the restriction operator R_0 as a $p \times n$ matrix whose structure looks like

$$R_0^T = \left[R_1^T D_1 R_1 \boldsymbol{t} \ , \ \dots \ , \ R_p^T D_p R_p \boldsymbol{t} \right] \tag{2.24}$$

where $\boldsymbol{t}$ is a vector that corresponds to the errors that are not corrected well from the single level solve. The classical Nicolaides says $\boldsymbol{t}$ is a coefficient vector such that the corresponding finite element function is the constant function. With this definition, the coarse space is made of those column vectors which have local support in one subdomain and its neighbors (in case of overlapping subdomains). The size of unknowns in a coarse problem equals a number of subdomains which is relatively small. The coarse problem itself might not be solved in parallel but the construction of the coarse system can be done in parallel. Suppose that matrix A is known in each subdomain, $A_0 = R_0 A R_0^T$ can be computed locally. We consider $R_0 A$, whose each row comes from the partition of unity which is known in its subdomain multiplies with matrix A. Then, it can obviously be done in parallel. For $R_0 A R_0^T$, it needs some information of the partition of unity from its neighbor.

In theory, we must concern only about the floating subdomains which do not touch the global boundary where the Dirichlet boundary condition is imposed. But in practice, the degrees of freedom that are incorporated with Dirichlet nodes are already not involved in our system. Due to that reason, we do not need to consider whether the subdomain is a floating subdomain or not.

Extended Nicolaides Coarse Space

But in many cases, one vector per subdomain from the classical Nicolaides coarse space does not turn out to be a good enough method to represent the coarse space when there are jumps of the coefficients, see [Nat+11; Dol+12; Spi+14; XN14]. Basically, $\boldsymbol{t}$ is not necessary to represent only a constant function, it could be an arbitrary set of vectors $\{\boldsymbol{t}_i\}$ that are linearly independent. The extension of Nicolaides coarse space would be created in such a way that $\boldsymbol{t}_i$ is a coefficient vector such that the corresponding finite element function represents the Cartesian coordinate function. Let we denote $g_i : \Omega \to \mathbb{R},\ i = 0, \ldots, d$

$$g_i(x) = \begin{cases} 1 & , i = 0 \\ x_i & , i \neq 0 \end{cases}$$

where d depends on the problem or selected basis functions. We could see that the image of g_i is the corresponding Cartesian coordinate in i-th dimension when $i \neq 0$. Let $\phi = \{\phi_1, \ldots \phi_N\}$ be the basis functions of our considered finite element space. Therefore, $\boldsymbol{t}_0$ ... $\boldsymbol{t}_d$ are defined in a such way that

$$\sum_j (\boldsymbol{t}_i)_j \ \phi_j = g_i$$

Then it turns out that the number of unknowns in coarse problem is proportional to the number of subdomains which could still be relatively small. Therefore, we can express R_0 in the following form.

$$R_0^T = [R_1^T D_1 R_1 \boldsymbol{t}_0,\ \ldots,\ R_1^T D_1 R_1 \boldsymbol{t}_d,\ \ldots,\ R_p^T D_p R_p \boldsymbol{t}_0,\ \ldots,\ R_p^T D_p R_p \boldsymbol{t}_d] \qquad (2.25)$$

3 Restricted Additive Schwarz Preconditioned Exact Newton

As we have known from the previous chapter that Newton's method is only locally convergent, the initial guess has to be sufficiently close to the solution. Accordingly, a good starting point has to be carefully chosen. Otherwise, the Newton method might lose its strength. The robustness is consequently a major concern. *Nonlinear Preconditioning* came up and has been developed in recent decades to precondition the nonlinear system like the idea in the linear case so that we can potentially enhance the robustness of Newton's method. As in the linear case, the parallel Schwarz method is typically used as a preconditioner to speed up the convergence. The Additive Schwarz Preconditioned Inexact Newton (ASPIN) introduced in the early 2000s by [CK02] can be referred to as a basis of the nonlinear preconditioner. Next, the Restricted Additive Schwarz Preconditioned Exact Newton (RASPEN), an ASPIN variant, is proposed by [Dol+16] with its two-level version. To add the second level is also an important issue in the nonlinear case to bolster the scalability of the method when it has to deal with a huge number of subdomains. This chapter would provide the details of nonlinear preconditioners, especially RASPEN, including one- and two-level versions.

3.1 One-level RASPEN

Suppose we want to solve the nonlinear algebraic problem arising after the discretization of nonlinear partial differential equations by well-known discretization schemes such as Finite Element (FE) or Discontinuous Galerkin (DG)

$$F(x) = 0 \tag{3.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear function and $n = |\mathcal{I}|$ where $\mathcal{I} = \{1, \dots, n\}$ is an index set.

Let us recall that in the previous chapter, we have defined $\mathcal{I}_k \subseteq \mathcal{I}$ for $1 \leq k \leq p$ to be the extended index set, and $n_k = |\mathcal{I}_k|$. We have also defined the restriction operator $R_k : \mathbb{R}^n \to \mathbb{R}^{n_k}$. We also denote the second restrictions $\hat{R}_k : \mathbb{R}^n \to \mathbb{R}^{n_k}$ which satisfy a partition of unity (PU) property

$$\sum_{k=1}^{p} R_k^T \hat{R}_k = I = I^T = \sum_{k=1}^{p} \hat{R}_k^T R_k. \tag{3.2}$$

The main approach is to transform the original nonlinear system F to an equivalent system $\mathcal{F}$ which has the same solution. It can be done by domain decomposition method in additive Schwarz fashion, that is, the new system is obtained from the solutions of nonlinear subdomain problems.

Thus, we define the subdomain solution operator $G_k : \mathbb{R}^n \to \mathbb{R}^{n_k}$ for $1 \leq k \leq p$: For any $x \in \mathbb{R}^n$, and any $k \in \{1, \ldots, p\}$ it holds that

$$R_k F\left((I - R_k^T R_k)x + R_k^T G_k(x)\right) = 0. \tag{3.3}$$

With the characteristics of F emerging from the discretization of PDE by the usual tools like FE or DG, the equation (3.3) is typically a nonlinear subproblem on k^{th} subdomain with the Dirichlet boundary condition on the subdomain boundary. Using the aforementioned subdomain solution operators, the nonlinear fixed point iteration is then defined as

$$x^{l+1} = \sum_{k=1}^{p} \hat{R}_k^T G_k(x^l). \tag{3.4}$$

Then, we obtain the new nonlinear system

$$\mathcal{F}(x) := x - \sum_{k=1}^{p} \hat{R}_k^T G_k(x) = 0 \tag{3.5}$$

which holds at the fixed point of iteration (3.4). We refer to the new nonlinear system $\mathcal{F}(x) := 0$ as *preconditioned nonlinear system.*

Lemma 1. *If the fixed-point iteration* (3.4) *converges to a unique fixed point x^*, it coincides with the solution z of the original problem* (3.1)*.*

Proof. Assume that x^* is the unique fixed point of the fixed-point iteration (3.4). Assume that z is the unique solution of $F(z) = 0$, we then have for any $k \in \{1, \ldots, p\}$:

$$0 = F(z) = F\left((I - R_k^T R_k)z + R_k^T R_k z\right),$$

i.e. $G_k(z) = R_k z$. From this follows

$$\sum_{k=1}^{p} \hat{R}_k^T G_k(z) = \sum_{k=1}^{p} \hat{R}_k^T R_k z = \left(\sum_{k=1}^{p} \hat{R}_k^T R_k\right) z = z,$$

i.e. z is a fixed point of (3.4). Since the fixed point is unique we have $z = x^*$. □

Hence, we know that the preconditioned system has the same solution as the original system. Therefore, we instead solve the preconditioned system by Newton's method.

$$x^{l+1} = x^l - \lambda^l \left[\nabla \mathcal{F}(x^l)\right]^{-1} \mathcal{F}(x^l). \tag{3.6}$$

Then, each Newton iteration requires two major steps

- The local nonlinear problems (3.3) have to be solved in each subdomain. This can be done by using Newton's method in parallel. For the linear subdomain solves, we employ a sparse direct solver (one may use iterative solvers as well).
- The tangential system $\nabla\mathcal{F}(x^l)v^l = \mathcal{F}(x^l)$ is required to be solved. That involves the computation of the Jacobian $\nabla\mathcal{F}(x)$. It is costly to be set up and if we consider solving the tangential system by a Krylov method, one is able to take advantage of a matrix-free implementation. This will be shown below.

In the following, we show details of the computation of the Jacobian $\nabla\mathcal{F}(x)$. From the definition of $\mathcal{F}$ we obtain

$$\nabla\mathcal{F}(x^l) = \nabla\left(x^l - \sum_{k=1}^{p} \hat{R}_k^T G_k(x^l)\right) = I - \sum_{k=1}^{p} \hat{R}_k^T \nabla G_k(x^l), \tag{3.7}$$

i.e. the Jacobians of the subdomain solution operator are required. We set the abbreviation

$$x_k = x_k(x) = (I - R_k^T R_k)x + R_k^T G_k(x).$$

From (3.3), we obtain by the chain rule

$$\begin{aligned} & R_k \nabla F(x_k)\left[I - R_k^T R_k + R_k^T \nabla G_k(x)\right] = 0 \\ \Leftrightarrow\quad & R_k\nabla F(x_k) - R_k\nabla F(x_k)R_k^T R_k + R_k\nabla F(x_k)R_k^T\nabla G_k(x) = 0 \\ \Leftrightarrow\quad & R_k\nabla F(x_k)R_k^T\nabla G_k(x) = R_k\nabla F(x_k)R_k^T R_k - R_k\nabla F(x_k) \\ \Leftrightarrow\quad & \nabla G_k(x) = R_k - \left[R_k\nabla F(x_k)R_k^T\right]^{-1} R_k\nabla F(x_k). \end{aligned} \tag{3.8}$$

Then, inserting this result into (3.7) yields

$$\begin{aligned} \nabla\mathcal{F}(x^l) &= I - \sum_{k=1}^{p} \hat{R}_k^T \left\{R_k - \left[R_k\nabla F(x_k^l)R_k^T\right]^{-1} R_k\nabla F(x_k^l)\right\} \\ &= I - \sum_{k=1}^{p} \hat{R}_k^T R_k + \sum_{k=1}^{p} \hat{R}_k^T \left[R_k\nabla F(x_k^l)R_k^T\right]^{-1} R_k\nabla F(x_k^l) \\ &= \sum_{k=1}^{p} \hat{R}_k^T \left[R_k\nabla F(x_k^l)R_k^T\right]^{-1} R_k\nabla F(x_k^l), \end{aligned} \tag{3.9}$$

where we have set $x_k^l = (I - R_k^T R_k)x^l + R_k^T G_k(x^l)$. One might notice that there is the term $\left[R_k\nabla F(x_k^l)R_k^T\right]^{-1}$. This is generally a dense matrix even the local Jacobian $R_k\nabla F(x_k^l)R_k^T$ is sparse. And in practice, it is too costly to compute. So, $\nabla\mathcal{F}(x^l)$ should never be set up. But, we can see that the matrix-vector product $\nabla\mathcal{F}(x^l)v^l$ can be computed by solving local linear systems where the matrix $R_k\nabla F(x_k^l)R_k^T$ can be reused from the local subdomain solves.

Relation to Linear Case

For the special case when the model problem is linear, $F(x)$ can be expressed by $F(x) = Ax - b$. We consider the nonlinear subdomain problems

$$\begin{aligned} & R_k F\left((I - R_k^T R_k)x + R_k^T G_k(x)\right) = 0 \\ = \quad & R_k A\left((I - R_k^T R_k)x + R_k^T G_k(x)\right) - R_k b = 0 \\ \Leftrightarrow \quad & G_k(x) = R_k x + \left[R_k A R_k^T\right]^{-1} R_k(b - Ax). \end{aligned} \tag{3.10}$$

Substituing $G_k(x)$ into (3.5), we achieve

$$\begin{aligned} \mathcal{F}(x) &= x - \sum_{k=1}^{p} \hat{R}_k^T \left[R_k x + \left[R_k A R_k^T\right]^{-1} R_k(b - Ax)\right] \\ &= -\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k(b - Ax) \\ &= \hat{B}Ax - \hat{B}b \quad = \quad 0 \end{aligned} \tag{3.11}$$

where $\hat{B} = \sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k$. Let us remark that Newton's method will solve the preconditioned system $\mathcal{F}$ in one iteration because $\mathcal{F}$ is affine linear and solving the Jacobian system with GMRES is equivalent to the restricted additive Schwarz method in Algorithm 6.

Relation to One-level ASPIN

Since RASPEN is one of the variants of ASPIN [CK02], we would like to depict the differences between those two. We might look at their name and see that there are two different letters. The first one is R in RASPEN, standing for "restricted". In RASPEN, we define the restriction operators in such a way that they satisfies the partition of unity property

$$\sum_{k=1}^{p} R_k^T \hat{R}_k = I = I^T = \sum_{k=1}^{p} \hat{R}_k^T R_k.$$

But the restriction operators in ASPIN are defined by

$$R_k R_k^T = I_k \qquad \text{for } k = 1, \ldots, p$$

which are lacking of the partition of unity property. So, the preconditioned nonlinear system can be written by

$$\mathcal{F}(x) := x - \sum_{k=1}^{p} R_k^T G_k(x) = 0 \tag{3.12}$$

which has a convergence issue on the overlapped region and need to be resolved by the relaxation parameter [Dol+16].

The second different letter is E and I, standing for "exact" and "inexact" respectively. This means ASPIN uses the inexact Jacobian for the reason of reducing the computation cost of exact Jacobian [Dol+16] but this could slow down the convergence and in fact, as we already explained that each component is already computed somewhere before, we can reuse it. Therefore, the Jacobian of $\mathcal{F}_{\text{ASPIN}}$ yields

$$\nabla \mathcal{F}_{\text{ASPIN}}(x^l) = \left(\sum_{k=1}^{p} R_k^T \left[R_k \nabla F(x^l) R_k^T \right]^{-1} R_k \right) \nabla F(x^l) \tag{3.13}$$

where the global Jacobian $\nabla F(x^l)$ is required to be computed after subdomain solves. The drawback of ASPIN [Dol+16] is that it is over-corrected on the overlapped region because it is lacking of the partition of unity (PU) property. This would not cause any problem in the linear case because it could be compensated by the damping parameter.

3.2 Two-level RASPEN

We learned from the previous chapter that the single level solver is not appropriate for a huge number of subdomains because it greatly increases the condition number. So the convergence is much slower. Therefore, in this section, we consider adding a coarse grid correction to RASPEN to sort out the disadvantage of the one-level approach. In [Dol+16], their approach is using multiplicative coarse space based on the Full Approximation Scheme (FAS). While ASPIN [CK02; MC05] uses a simple additive coarse space that requires (precomputed) coarse space solution. Recently, multiplicative approaches without relying on FAS are provided in [HL20]. The coarse correction can be applied before or after the local correction, or both before and after the local correction. We do it in a slightly different way, i.e., we also apply a coarse grid correction in a multiplicative way with the different approach for a coarse grid setup. We will go through these methods in the following subsections

3.2.1 Additive Approach

This subsection is recalled from [HL20]. We first start with a simple additive approach. Let us assume that we have a generic coarse space, then we can define a corresponding restriction operator that maps from a fine space to a coarse space $R_0 : \mathbb{R}^n \to \mathbb{R}^{n_H}$ where $n_H = |\mathcal{I}_H|$ where $\mathcal{I}_H = \{1, \ldots, n_H\}$ is an index set for a coarse space. The coarse nonlinear problem is defined as

$$R_0 F\left(x - R_0^T C_0(x)\right) = 0 \tag{3.14}$$

where $C_0 : \mathbb{R}^n \to \mathbb{R}^{n_H}$ is the coarse correction operator. The new nonlinear system with the additive coarse grid correction reads

$$\mathcal{F}_{\text{Add}}(x) := x - \sum_{k=1}^{p} \hat{R}_k^T G_k(x) + R_0^T C_0(x) = 0. \tag{3.15}$$

We can simply rewrite (3.15) in terms of corrections $C_k(x)$ when we define $C_k(x) := R_k x - G_k(x)$. Therefore, the nonlinear system (3.15) becomes

$$\mathcal{F}_{\text{Add}}(x) := \sum_{k=1}^{p} \hat{R}_k^T C_k(x) + R_0^T C_0(x) = 0. \tag{3.16}$$

Again, we solve the system by Newton's method. The linearization leads to

$$x^{l+1} = x^l - \lambda^l \left[\nabla \mathcal{F}_{\text{Add}}(x^l)\right]^{-1} \mathcal{F}_{\text{Add}}(x^l) \tag{3.17}$$

where the evaluation of $\nabla \mathcal{F}_{\text{Add}}(x)$ requires the Jacobian of the coarse grid correction operator which can be derived as

$$\begin{aligned} & R_0 \nabla F(x_0) \left[I - R_0^T \nabla C_0(x)\right] = 0 \\ \Leftrightarrow\quad & R_0 \nabla F(x_0) - R_0 \nabla F(x_0) R_0^T \nabla C_0(x) = 0 \\ \Leftrightarrow\quad & R_0 \nabla F(x_0) R_0^T \nabla C_0(x) = R_0 \nabla F(x_0) \\ \Leftrightarrow\quad & \nabla C_0(x) = \left[R_0 \nabla F(x_0) R_0^T\right]^{-1} R_0 \nabla F(x_0). \end{aligned} \tag{3.18}$$

where we set $x_0 = x_0(x) = x - R_0^T C_0(x)$. With the results from (3.8) and (3.18), $\nabla \mathcal{F}_{\text{Add}}(x)$ can be expressed by

$$\begin{aligned} \nabla \mathcal{F}_{\text{Add}}(x) &= I - \sum_{k=1}^{p} \hat{R}_k^T \left\{ R_k - \left[R_k \nabla F(x_k) R_k^T\right]^{-1} R_k \nabla F(x_k) \right\} \\ &\quad + R_0^T \left\{ \left[R_0 \nabla F(x_0) R_0^T\right]^{-1} R_0 \nabla F(x_0) \right\} \\ &= I - \sum_{k=1}^{p} \hat{R}_k^T R_k + \sum_{k=1}^{p} \hat{R}_k^T \left[R_k \nabla F(x_k) R_k^T\right]^{-1} R_k \nabla F(x_k) \\ &\quad + R_k^T \left[R_k \nabla F(x_0) R_0^T\right]^{-1} R_0 \nabla F(x_0) \\ &= \sum_{k=1}^{p} \hat{R}_k^T \left[R_k \nabla F(x_k) R_k^T\right]^{-1} R_k \nabla F(x_k) \\ &\quad + R_0^T \left[R_0 \nabla F(x_0) R_0^T\right]^{-1} R_0 \nabla F(x_0). \end{aligned} \tag{3.19}$$

The additive approach seems to be over-correction which is a drawback of one-level ASPIN as well. It looks like the error components are corrected twice on both fine grid and coarse grid. So in practice, the authors apply the globalization strategy to resolve the convergence behavior. Therefore, we believe that it is better to do it multiplicatively.

Relation to Linear Case

Again, we would show that for the linear case, it becomes the additive Schwarz preconditioner. We can recall $G_k(x)$,

$$G_k(x) = R_k x + \left[R_k A R_k^T\right]^{-1} R_k(b - Ax)$$

Then, we consider the nonlinear coarse problem

$$\begin{aligned} & R_0 F\left(x - R_0^T C_0(x)\right) = 0 \\ = \quad & R_0 A\left(x - R_0^T C_0(x)\right) - R_0 b = 0 \\ \Leftrightarrow \quad & C_0(x) = -\left[R_0 A R_0^T\right]^{-1} R_0(b - Ax). \end{aligned} \tag{3.20}$$

Substituing $G_k(x)$ and $C_0(x)$ into (3.15), we obtain

$$\begin{aligned} \mathcal{F}_{\text{Add}}(x) = \; & -\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k(b - Ax) \\ & - R_0^T \left[R_0 A R_0^T\right]^{-1} R_0(b - Ax) \\ = \; & \hat{B}_{\text{Add}} A x - \hat{B}_{\text{Add}} b \quad = \quad 0 \end{aligned} \tag{3.21}$$

where $\hat{B}_{\text{Add}} = \left[\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k + R_0^T \left[R_0 A R_0^T\right]^{-1} R_0\right]$. Solving the Jacobian system with GMRES is also equivalent to the restricted additive Schwarz with coarse grid correction which consequently provides a lower number of linear iterations and does not depend on the number of subdomains.

Two-level ASPIN

Two-level ASPIN from [CK02; MC05] which applies the coarse correction in an additive way is briefly explained in the following part. Let us define the nonlinear coarse problem

$$F_0(x_0) = 0 \tag{3.22}$$

with the unique solution x_0^*, and the nonlinear function $F_0 : \mathbb{R}^{n_H} \to \mathbb{R}^{n_H}$ can be obtained by the coarser discretization or Galerkin approach. And we denote $\tilde{R}_0$ which plays a similar role as R_0 but in the residual space. Then, the coarse correction $C_0^A(x)$ is defined by

$$F_0\left(C_0^A(x) + x_0^*\right) = -\tilde{R}_0 F(x) \tag{3.23}$$

which we can see that in order to solve for a coarse correction, the coarse solution x_0^* is required and should be precomputed by solving (3.22). The preconditioned system looks similar to $\mathcal{F}_{\text{Add}}$.

3.2.2 Multiplicative Approach

Next, we consider the alternative approaches in a multiplicative way to add a coarse space. It can be added either before or after, or both before and after nonlinear subdomain solves. This subsection is also recalled from [HL20].

Added before subdomain solves

The new preconditioned nonlinear system with a coarse correction becomes

$$\mathcal{F}_{\mathrm{B}}(x) := x - \sum_{k=1}^{p} \hat{R}_k^T G_k(x - R_0^T C_0(x)) = 0. \tag{3.24}$$

Solving (3.24) with Newton's method, the linearization yields

$$x^{l+1} = x^l - \lambda^l \left[\nabla \mathcal{F}_{\mathrm{B}}(x^l)\right]^{-1} \mathcal{F}_{\mathrm{B}}(x^l) \tag{3.25}$$

where the Jacobian $\nabla\mathcal{F}_{\mathrm{B}}(x)$ requires the results from (3.8) and (3.18). We obtain

$$\begin{aligned}\nabla\mathcal{F}_{\mathrm{B}}(x) &= I - \sum_{k=1}^{p} \hat{R}_k^T \nabla \left[G_k(x - R_0^T C_0(x))\right] \\ &= I - \sum_{k=1}^{p} \hat{R}_k^T \left[\nabla G_k(x - R_0^T C_0(x))(I - R_0^T \nabla C_0(x))\right] \\ &= I - \left[I - \sum_{k=1}^{p} \hat{R}_k^T \left[R_k \nabla F(x_{kb}) R_k^T\right]^{-1} R_k \nabla F(x_{kb}) - R_0^T \nabla C_0(x) \right. \\ &\qquad \left. + \sum_{k=1}^{p} \hat{R}_k^T \left[R_k \nabla F(x_{kb}) R_k^T\right]^{-1} R_k \nabla F(x_{kb}) R_0^T \nabla C_0(x) \right] \\ &= I - (I - A)(I - B)\end{aligned} \tag{3.26}$$

where $A = \sum_{k=1}^{p} \hat{R}_k^T \left[R_k \nabla F(x_{kb}) R_k^T\right]^{-1} R_k \nabla F(x_k)$, $B = R_0^T \left[R_0 \nabla F(x_0) R_0^T\right]^{-1} R_0 \nabla F(x_0)$, and $x_{kb} = (I - R_k^T R_k)x + R_k^T G_k(x) - R_0^T C_0(x)$. $\mathcal{F}_{\mathrm{B}}(x)$ indicates that this method adds the coarse correction **B**efore the nonlinear subdomain solves.

Added after subdomain solves

Another approach is to apply a coarse correction after nonlinear subdomain solves. The new preconditioned nonlinear system is thus defined by

$$\mathcal{F}_{\mathrm{A}}(x) := x - \sum_{k=1}^{p} \hat{R}_k^T G_k(x) + R_0^T C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x) \right) = 0. \tag{3.27}$$

The Newton iteration is given by

$$x^{l+1} = x^l - \lambda^l \left[\nabla \mathcal{F}_{\mathrm{A}}(x^l)\right]^{-1} \mathcal{F}_{\mathrm{A}}(x^l). \tag{3.28}$$

The Jacobian $\nabla \mathcal{F}_{\text{A}}(x)$ is derived as

$$\begin{aligned}
\nabla \mathcal{F}_{\text{A}}(x) &= I - \sum_{k=1}^{p} \hat{R}_k^T \nabla G_k(x) + R_0^T \nabla C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x) \right) \sum_{k=1}^{p} \hat{R}_k^T \nabla G_k(x) \\
&= I - \left[I - R_0^T \nabla C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x) \right) \right] \left[\sum_{k=1}^{p} \hat{R}_k^T \nabla G_k(x) \right] \\
&= I - \left[I - R_0^T \nabla C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x) \right) \right] \left[I - \sum_{k=1}^{p} \hat{R}_k^T \left[R_k \nabla F(x_k) R_k^T \right]^{-1} R_k \nabla F(x_k) \right] \\
&= I - (I - A)(I - B)
\end{aligned} \tag{3.29}$$

where $A = R_0^T \left[R_0 \nabla F(x_{00}) R_0^T \right]^{-1} R_0 \nabla F(x_{00})$, $B = \sum_{k=1}^{p} \hat{R}_k^T \left[R_k \nabla F(x_k) R_k^T \right]^{-1} R_k \nabla F(x_k)$, and $x_{00} = \sum_{k=1}^{p} \hat{R}_k^T G_k(x) - R_0^T C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x) \right)$. $\mathcal{F}_{\text{A}}(x)$ indicates that this method adds the coarse correction **A**fter the nonlinear subdomain solves.

Added before and after subdomain solves

The symmetric variant is obtained by applying coarse corrections before and after nonlinear subdomain solves. The system is written by

$$\mathcal{F}_{\text{BA}}(x) := x - \sum_{k=1}^{p} \hat{R}_k^T G_k(x - R_0^T C_0(x)) + R_0^T C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x - R_0^T C_0(x)) \right) = 0. \tag{3.30}$$

Solving the above system by Newton's method, the Newton iteration reads

$$x^{l+1} = x^l - \lambda^l \left[\nabla \mathcal{F}_{\text{BA}}(x^l) \right]^{-1} \mathcal{F}_{\text{BA}}(x^l). \tag{3.31}$$

We can derive the Jacobian $\nabla \mathcal{F}_{\text{BA}}(x)$ as

$$\begin{aligned}
\nabla \mathcal{F}_{\text{BA}}(x) &= I - \left[\sum_{k=1}^{p} \hat{R}_k^T \nabla G_k(x - R_0^T C_0(x)) \right] \left[(I - R_0^T \nabla C_0(x)) \right] \\
&\quad + \left[R_0^T \nabla C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x - R_0^T C_0(x)) \right) \right] \left[\sum_{k=1}^{p} \hat{R}_k^T \nabla G_k(x - R_0^T C_0(x)) \right] \left[(I - R_0^T \nabla C_0(x)) \right] \\
&= I - \left[I - R_0^T \nabla C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x - R_0^T C_0(x)) \right) \right] \left[\sum_{k=1}^{p} \hat{R}_k^T \nabla G_k(x - R_0^T C_0(x)) \right] \left[(I - R_0^T \nabla C_0(x)) \right] \\
&= I - (I - A)(I - B)(I - C)
\end{aligned} \tag{3.32}$$

where $A = R_0^T \left[R_0 \nabla F(x_{000}) R_0^T \right]^{-1} R_0 \nabla F(x_{000})$, $B = \sum_{k=1}^{p} \hat{R}_k^T \left[R_k \nabla F(x_{kb}) R_k^T \right]^{-1} R_k \nabla F(x_k)$, $C = R_0^T \left[R_0 \nabla F(x_0) R_0^T \right]^{-1} R_0 \nabla F(x_0)$, $x_{000} = \sum_{k=1}^{p} \hat{R}_k^T G_k(x - R_0^T C_0(x))$

$-R_0^T C_0 \left(\sum_{k=1}^p \hat{R}_k^T G_k(x - R_0^T C_0(x))\right)$, and $x_{kb} = (I - R_k^T R_k)x + R_k^T G_k(x) - R_0^T C_0(x)$. $\mathcal{F}_{\text{BA}}(x)$ indicates that this method adds the coarse corrections **B**efore and **A**fter the nonlinear subdomain solves.

From our perspective, we do not expect that the results of adding the coarse solve before or after or both before and after are so different. It might work really well for symmetrization in the linear case. But we do not think it is helpful in the nonlinear case and do not worth trying. We expect that their performances are not that different. Therefore, we decide to mainly consider adding a coarse solve after solving nonlinear subdomain problems as our main approach. We expect that solving the subdomain problems first would provide a good starting point for solving the coarse system.

Relation to Linear Case

We would also like to show that the multiplicative approach here is equivalent to the multiplicative Schwarz as a preconditioner if the system is linear. Note that this is the case for which we add the coarse correction after solving the subdomain problems. $G_k(x)$ can be recalled from the additive approach

$$G_k(x) = R_k x + \left[R_k A R_k^T\right]^{-1} R_k(b - Ax).$$

Next, we consider the nonlinear coarse problem (3.14) but the starting point becomes $\tilde{x} = \sum_{k=1}^p \hat{R}_k^T G_k(x)$

$$\begin{aligned} & R_0 F\left(\tilde{x} - R_0^T C_0(\tilde{x})\right) = 0 \\ = \quad & R_0 A\left(\tilde{x} - R_0^T C_0(\tilde{x})\right) - R_0 b = 0 \\ \Leftrightarrow \quad & C_0(\tilde{x}) = -\left[R_0 A R_0^T\right]^{-1} R_0(b - A\tilde{x}). \end{aligned} \tag{3.33}$$

Substituing $G_k(x)$ and $C_0(\tilde{x})$ into (3.27), we get

$$\begin{aligned} \mathcal{F}_{\text{A}}(x) = \; & -\sum_{k=1}^p \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k(b - Ax) \\ & - R_0^T \left[R_0 A R_0^T\right]^{-1} R_0(b - A\tilde{x}) \quad = \quad 0 \end{aligned} \tag{3.34}$$

For an explicit form, we rearrange (3.34) into

$$\begin{aligned}
\mathcal{F}_{\text{A}}(x) =\ & -\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k(b - Ax) \\
& - R_0^T \left[R_0 A R_0^T\right]^{-1} R_0(b - A\sum_{k=1}^{p} \hat{R}_k^T G_k(x)) \\
=\ & -\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k(b - Ax) \\
& - R_0^T \left[R_0 A R_0^T\right]^{-1} R_0(b - Ax - A\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k(b - Ax)) \\
=\ & -\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k(b - Ax) \\
& - R_0^T \left[R_0 A R_0^T\right]^{-1} R_0 \left(\left(I - A\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k\right)(b - Ax)\right) \\
=\ & \hat{B}_{\text{A}} A x - \hat{B}_{\text{A}} b \quad = \quad 0
\end{aligned} \tag{3.35}$$

where $\hat{B}_{\text{A}} = \left[\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k + R_0^T \left[R_0 A R_0^T\right]^{-1} R_0 \left(I - A\sum_{k=1}^{p} \hat{R}_k^T \left[R_k A R_k^T\right]^{-1} R_k\right)\right]$. The Jacobian system arising in Newton iteration is solved by GMRES and it is again equivalent to restricted multiplicative Schwarz with coarse grid correction.

FAS-based Multiplicative Approach

We have mentioned that two-level RASPEN proposed in [Dol+16] using the Full Approximation Scheme (FAS) for the coarse correction. We would also like to briefly explain it here. Let us recall that $\tilde{R}_0$ is denoted in the same way as two-level ASPIN. Then, we can define the FAS correction $C_0(x)$ by

$$F_0\left(C_0(x) + R_0 x\right) = F_0(R_0 x) - \tilde{R}_0 F(x). \tag{3.36}$$

One might notice that we do not need to know the coarse solution like in two-level ASPIN. The authors choose to solve the FAS correction $C_0(x)$ before the subdomain problems. So, the preconditioned system becomes similar to $\mathcal{F}_{\text{B}}$.

The full approximation scheme needs the rediscretization for the coarse level. The question might appear when there is a variability of coefficients, for example, the diffusion coefficient in an elliptic problem which could be highly fluctuated among cells in a fine grid. How to redefine this function in terms of a coarse mesh is questionable. In [Bas99], the author mentioned this concern and proposed the remedy which is in a purely algebraic way.

3.2.3 Our Approach

Due to the concern of using the coarser mesh by rediscretization, we would like to construct the second level of RASPEN in an algebraic way or by the Galerkin product. We exploit the Nicolaides coarse space and its extension proposed in Section 2.6 because it can be fully constructed in an algebraic fashion. As we already decided in the previous section, we consider adding the coarse correction after the subdomain solves in a multiplicative way. Recalling the simply multiplicative approach (3.27)

$$\mathcal{F}_{\text{A}}(x) := x - \sum_{k=1}^{p} \hat{R}_k^T G_k(x) + R_0^T C_0 \left(\sum_{k=1}^{p} \hat{R}_k^T G_k(x) \right) = 0$$

or we can write it in terms of correction

$$\mathcal{F}_{\text{A}}(x) := \sum_{k=1}^{p} \hat{R}_k^T C_k(x) + R_0^T C_0 \left(x - \sum_{k=1}^{p} \hat{R}_k^T C_k(x) \right) = 0 \tag{3.37}$$

with R_0 from Nicolaides scheme, we defined in (2.24) and (2.25). The coarse computation does not require a coarse solution like in ASPIN and is not based on the Full Approximation Scheme (FAS) like in RASPEN. We can see that the Jacobian of $\mathcal{F}_{\text{A}}$ needs the information of $\left[R_0 \nabla F(x_0) R_0^T\right]$ which can be done by simply matrix-matrix multiplication (more details will be explained in chapter 4).

Our approach solves the local nonlinear subdomain problems first. Then we glue up the local solutions and use it as the initial guess for the (global) coarse nonlinear problem (3.14). And we solve the coarse nonlinear problem by Newton's method with the direct solver, since the coarse problem size is small compared to the original problem. One might notice that $R_0 \nabla F(x_0) R_0^T$ is required in nonlinear coarse problem solve and can be reused in the global linear part in the same way as in the one-level method.

For the global tangential system $\nabla\mathcal{F}(x^l)v^l = \mathcal{F}(x^l)$, we need the local linear solves in the same way as in the one-level scheme as well. The additional task from the second level comes after solving the first level completely. We recall the Jacobian $\nabla\mathcal{F}_{\text{A}}(x)$ and their abbreviations from (3.29)

$$\begin{aligned} \nabla\mathcal{F}(x^l)v^l &= \left[I - (I-A)(I-B)\right] v^l = \left[B + A(I-B)\right] v^l \\ &= Bv^l + A(v^l - Bv^l) \end{aligned} \tag{3.38}$$

which Bv^l is what we have done in the one-level. What we need to do more is subtract it by v^l before solving the linear system for the coarse problem. Algorithm 10 displays the evaluation of $\nabla\mathcal{F}(x^l)v^l$ when solving the global linear system by an iterative solver.

Algorithm 10 Computing $\nabla\mathcal{F}(x)v$

```
init $v$
compute $y_k = R_k \nabla F(x_k) v$
solve local linear problem $\left[R_k \nabla F(x_k) R_k^T\right] z_k = y_k$    // can be done in parallel
communicate $z_k \to z$
compute $y_0 = R_0 \nabla F(x_0)(v - z)$
solve local linear problem $\left[R_0 \nabla F(x_0) R_0^T\right] z_0 = y_0$    // can be done in rank 0
prolongate $R_0^T z_0$    // can be done in parallel
compute $z = z + R_0^T z_0$
```

3.3 Algorithm Description

This section would present RASPEN in an algorithmic perspective. Note that all approaches (e.g. one-, two-level or additive, multiplicative coarse correction) share the same structure. The only difference is the order of computation. In this section, we mainly describe two-level RASPEN with a coarse grid correction added after local nonlinear solves. Let $\mathcal{F}$ be a preconditioned nonlinear system obtained from the approach we consider. The RASPEN algorithm is presented in Algorithm 11.

Algorithm 11 RASPEN

```
init $l = 0, x^0$
while $l <$max_outer_iter and $\|F(x^l)\|/\|F(x^0)\| > tol_{\text{outer}}$ do
    solve nonlinear subdomain problems $R_k F\left((I - R_k^T R_k)x^l + R_k^T \hat{x}_k^l\right) = 0$
    set $\tilde{x}^l = \sum_{k=1}^{p} \tilde{R}_k^T \hat{x}_k^l$
    solve nonlinear coarse problem $R_0 F\left(\tilde{x}^l - R_0^T \hat{c}_0^l\right) = 0$
    evaluation $\mathcal{F}(x^l) = x^l - \tilde{x}^l + R_0^T \hat{c}_0^l$
    solve the global tangential system $\nabla\mathcal{F}(x^l) v^l = \mathcal{F}(x^l)$
    update $x^{l+1} = x^l - \lambda^l v^l, \quad l = l + 1$
end while
```

One might exploit $\|\mathcal{F}(x)\|$ as a stopping criterion instead of $\|F(x)\|$. For a better comparison between other methods, we select $\|F(x)\|$. We generally set $tol_{\text{outer}} = 10^{-6}$. We look more precisely into the evaluation of $\mathcal{F}(x)$. It requires the solution of the nonlinear subdomain problems $F_k(\hat{x}_k; x) = R_k F\left((I - R_k^T R_k)x + R_k^T \hat{x}_k\right) = 0$ for a given x which can be solved by Newton's method in parallel; see Algorithm 12. The tangential system that appeared in this algorithm is typically solved by a sparse direct solver and λ is a step length from a line-search algorithm. Note that Algorithm 12 can also apply to the nonlinear coarse problem in a similar way.

After we achieve the subdomain solutions $\hat{x}_k$, we glue them together (with a partition of unity) and use this updated solution as the initial guess for the nonlinear coarse problem. This problem is also typically solved by Newton's method with a direct solver and can be done only in one processor since the size is not too big compared to the fine-grid problem. Then, we obtain the coarse correction and prolongate it to the fine grid. Finally, we have all ingredients for computing $\mathcal{F}(x)$

Algorithm 12 Solving $F_k(\hat{x}_k;x) = R_kF\left((I - R_k^TR_k)x + R_k^T\hat{x}_k\right) = 0$ for a given x

```
init $i = 0, \hat{x}_k^0$
while $i <$max_inner_iter and $\|F_k(\hat{x}_k^i;x)\|/\|F_k(\hat{x}_k^0;x)\| > tol_{\text{inner}}$ do
    compute $\nabla F_k(\hat{x}_k^i;x) = R_k\nabla F\left((I - R_k^TR_k)x + R_k^T\hat{x}_k\right)R_k^T$
    solve $[\nabla F_k(\hat{x}_k^i;x)]\, s_k^i = F_k(\hat{x}_k^i;x)$
    update $\hat{x}_k^{i+1} = \hat{x}_k^i - \lambda_k^i s_k^i, \quad i = i+1$
end while
```

The global tangential system is considered as the global linear problem and is literally solved by an iterative solver like the GMRES method where we can take advantage of a matrix-free implementation - the global Jacobian in the global tangential system is not explicitly built and stored. That means it could save a lot of memory. Moreover, computing the residual in GMRES or any iterative Krylov solvers requires the evaluation of $\nabla\mathcal{F}(x^l)v^l$ which in fact, demands the local linear solves as explained in Algorithm 10. The local Jacobian obtained from solving nonlinear subdomain problems could be reused for linear solves arising within this evaluation. This happens as well for the coarse Jacobian. One might notice that the linear system itself is not preconditioned but our method already applied the preconditioner in the nonlinear concept. Let us remark that, in practice, the main difference between two-level additive and multiplicative approaches is the ordering of the computation of the coarse correction. We simply summarize it below

- $\mathcal{F}_{\text{Add}}(x)$, nonlinear subdomain solutions and a coarse correction can be computed simultaneously
- $\mathcal{F}_{\text{B}}(x)$, a coarse correction is computed first, then update the solution before solving nonlinear subproblems
- $\mathcal{F}_{\text{A}}(x)$, nonlinear subproblems are computed first, then update the solution before solving for a coarse correction
- $\mathcal{F}_{\text{BA}}(x)$, a coarse correction is computed first, then update the solution before solving nonlinear subproblems, then the coarse correction is computed again and update the solution.

As we already described that an additive approach might be overly corrected, and the globalization strategy should play a crucial role here. The symmetrized version $\mathcal{F}_{\text{BA}}$ looks not worth trying for a nonlinear case, so we omit it. The one we decide to mainly consider is to apply the coarse correction after the subdomain solves $\mathcal{F}_{\text{A}}$ because it would provide the better initial guess for the nonlinear coarse system. The implementation details will be described in the next chapter.

4 Implementation of RASPEN in DUNE

This chapter would talk about our main software used in this thesis. DUNE, the Distributed and Unified Numerics Environment, is a free software framework mainly written in C++ for solving partial differential equations with a grid-based method. The discretization schemes such as Finite Element (FE), Finite Volumes (FV), or Discontinuous Galerkin (DG) are suitably implemented in DUNE along with several mesh refinement techniques, efficient solvers, especially in parallel.

The first section would give the reader an overview of DUNE. Then, in the following sections, we look into the specific module of DUNE, `dune-pdelab`, a powerful toolbox for solving PDEs. After that, it is the part that would explain the implementation details of our method, RASPEN and its two-level version. The last part would explain the parallel computations which occur in our algorithm.

4.1 Overview of DUNE

This section dedicates to describing the overview of DUNE, and what the skeleton workflow looks like. To get started, we introduce that DUNE is a compilation of libraries, working for some specific functionalities such as grid manager, linear algebra, and discretizations and well-written in their own place so-called "module". The advantage of modularization is that users can choose and use only the modules that depend on their projects. This framework also profits for considerably easier reading and understanding of what each module explicitly functions. DUNE consists of modules that are categorized as "core modules" because they are used in almost every project.

- `dune-common`: the origin of everything in DUN(E)iverse, depended by all other modules and contains the basic infrastructures for all Dune modules.

- `dune-geometry`: contains the information of reference element such as mapping from reference element to real element and also the quadrature rules.

- `dune-grid`: contains the grid interface, enabling the user to access the iterator over vertices, intersections, or elements. It also includes some basic tools for the grid implementation like `YaspGrid`.

- `dune-istl`: is abbreviated from iterative solver template library. It provides a zoo of iterative solvers for linear systems which can exploit a lot of blocking and sparsity patterns. These are implemented in both sequential and parallel version which are convenient for this study.

- dune-localfunctions: implements the warehouse of basis functions of finite element spaces defined on the reference elements.

Each module has its own dependency and it is a must requirement for building each module to work. One might see the difficulty to track the individual dependencies. But with a script dunecontrol provided in dune-common, it resolves this problem. The buildsystem can track down the dependencies and build the modules in the correct order. Next, we would like to present dune-pdelab which is a powerful and appropriate module for solving PDEs. The introduction of DUNE and module dune-pdelab is introduced in DUNE/PDELab Course [Dun21]. It is typically an annual week-long workshop, usually held in March, brought up by Parallel Computing Group, IWR, Heidelberg University. We implement our one- and two-level RASPEN, and all numerical experiments in the same style as dune-pdelab. In the following section, we deliver the overview of dune-pdelab.

4.2 Overview of DUNE-PDELab

dune-pdelab is a powerful and flexible module for solving PDEs. It depends on two core modules, dune-localfunctions and dune-istl described above. It is built on top of dune-functions which allows to access to the information of bases of finite element spaces in terms of global aspect. dune-pdelab can handle a variety of discretization methods like Finite Element (FE) and Discontinuous Galerkin (DG) with a lot of available basis functions. Furthermore, it provides several methods to apply the constraints to the degrees of freedom depending on boundary conditions. It also features the data communication and distribution which plays a key role in the application of the restriction operators satisfying with the partition of unity property. It can also deal with the zoo of linear solvers and preconditioners which are available for both sequential and parallel versions. In addition, it can manipulate the local grid adaptivity. A lot of local operators where the element-wise computations are implemented for some interesting and well-known problems like convection-diffusion, Navier-Stokes, or Maxwell's equations are already provided but implementing a new one is not a big deal.

dune-pdelab is written as such a high-level abstraction. That would be more friendly and suitable for new users because it is not necessary to implement the whole thing; some functions are already provided for you elsewhere. In order to implement the assembler of the algebraic problem, the user only needs to define the objects which are well separated and structured. Then, we eventually create the so-called GridOperator object which would represent the global assembler of the problem. We would give the overview and function of some important classes.

The first interesting class implemented in dune-pdelab we would like to introduce is GridFunctionSpace (GFS). Its idea is to define the discrete function space from the given type of user-selected basis and let the user access the grid information. It also includes the information of constraints on (sub)domain boundaries.

Note that GridFunctionSpace for trial and test spaces are not necessary to be identical. So, one can define it separately if using a method like the Petrov-Galerkin method [EG04]. The next one is called *local operator* containing all element-wise computation corresponding to the considered problem. The parameter class for the problem is needed as an argument for the local operator because some functions like a source term (RHS) or diffusion coefficients are evaluated here. Then, we are able to construct the GridOperator object. GFS and local operator are required arguments for GridOperator. The matrix assembler machinery can be called as a method inside an object of type GridOperator as well as the evaluation of residual.

In the end, we solve the algebraic system by choosing an appropriate solver as well as its preconditioner which is already provided in dune-pdelab. The nonlinear solver such as Newton's method is also already implemented in dune-pdelab. In practice, we try to replace Newton class by the implementation of our method RASPEN

4.3 RASPEN Implementation

This section would explain the implementation of RASPEN in dune style. Before we go further into the explanation, we would like to denote some words that clarify a specific part of the algorithm.

- *outer Newton* - represents when we apply Newton method to the preconditioned nonlinear system $\mathcal{F}(x) = 0$.
- *inner Newton* - represents when we apply Newton method to solve the local nonlinear subproblems $R_k F\left((I - R_k^T R_k)x + R_k^T G_k(x)\right) = 0$. In this process, we usually choose a direct solver as a linear solver.
- *global linear solve* - represents when we apply a linear solver to the tangential system of the global nonlinear problem $\mathcal{F}(x)$,

  $$\nabla\mathcal{F}(x)v = \mathcal{F}(x)$$

  which we normally choose GMRES because the residual $(\mathcal{F}(x) - \mathcal{F}(x)v)$ is required and $\nabla\mathcal{F}(x)v$ can be computed by the local linear solves. So, $\nabla\mathcal{F}$ is never explicitly built or stored.

Note that RASPEN class has the same signature as Newton so that the user can replace it easily. Let us start the implementation with the constructor we use to create the RASPEN object

```
RASPEN(GridOperator& go, SolutionVector& x, LinearSolver& ls)
```

go is an object of type GridOperator containing the kinds of stuff we have aforementioned, x is considered as an initial guess of the problem and a solution vector in the end, and ls is a linear solver we choose to apply in *inner Newton*. Note that the following pseudocodes are written from the perspective of one processor. Each processor has only the information of its own grid (including overlap region). Then, to exchange some information with its neighbors can be done by a method in `go`. The main application can be called by `apply()` method. So we start looking into the implementation details. The first important thing in RASPEN is the restriction operator satisfying the partition of unity property. Since the data is already distributed to subdomains, we do not need to explicitly have the restriction operator. We remind that $\hat{R}_k$ can be expressed as $\hat{R}_k = D_k R_k$ (in Section 2.6). So D_k plays a key role to fulfill the partition of unity property. In practice, we do not need the matrices since we do also not explicitly have R_k. Instead, we can create a vector of the same type as `SolutionVector` to represent the diagonal entries of D_k. We remark that there are several options to define the partition of unity vector p_k but in this study, we impose it by the following

$$(p_k)_i = \frac{1}{\xi_i} \quad \text{where } \xi_i \text{ is a number of subdomains to which } (x_k)_i \text{ belongs.}$$

This signifies that it finds the average value on the overlapped region so that it requires a communication to get the information of its neighbors. The pseudocode of this part is shown here

```
// make partition of unity
SolutionVector p=1;
do communicate and sum up the value of p on the overlapped dofs
do find the inverse value of p on the overlapped dofs
do get information of constrained and subdomain boundary dofs
do restrict those dofs to 0
```

Note that the constrained and subdomain boundary dofs can be given by `go`. Next, we would display what happens in the main loop in high-level and later we describe some parts in lower level.

```
// outer Newton
while(not converged)
 do inner Newton
 do communication to glue up local solutions
 do global linear solve
 do update the global solution
 do check the convergence criteria
end while
```

We go further to explain the *inner Newton* part where the local nonlinear subproblems are solved (in parallel) and the algorithm is already described in Algorithm 12. Here we could explain the lower level in the pseudocode paradigm.

```
// inner Newton
Matrix A
SolutionVector z
SolutionVector r
while(not converged)
```

```
 do go.Jacobian(x,A)     // update Jacobian
 do ls.apply(A,z,r)      // do a linear solve
 do update solution and defect(r)
 do check the convergence
end while
```

We can see that this part is solely done locally in each processor. `go.Jacobian(x,A)` is called for updating the local matrix `A`; `z` is a solution of the linearized equation; and `r` is a residual vector considered as the RHS term which can be computed by calling `go.residual(x,r)`. After each processor finishes the job, we need a communication to glue up the local solution along with the partition of unity in order to evaluate $\mathcal{F}(x)$ which is the RHS of the tangential system.

The next part would be the global linear solve. Even though the name is *global* but in practice, it still requires some local communications. Here we employ an iterative solver, usually GMRES, to solve the linear system. In each linear iteration, it requires the evaluation of $\nabla\mathcal{F}(x)v$ which we have already explained in Algorithm 10 that it involves the local linear solves. We would explain it in the lower level in Section 4.5.

```
// global linear solve
RestartedGMResSolver (LinearOperator& op,
                      ScalarProduct& sp,
                      Preconditioner& prec,
                      double reduction,
                      int restart,
                      int maxit,
                      int verbose)
```

The above block of code shows the constructor of `RestartedGMResSolver` class implemented in `dune-istl`. `LinearOperator` represents how to deal with the matrix-vector multiplication which in our case, it has to handle the local subdomain solves. `ScalarProduct` has to be chosen in an overlapped version since we solve the global tangential system. `Preconditioner` is not chosen here because our scheme has already preconditioned the nonlinear system. Then, we can call method `apply()` to solve the problem. The returned result is applied to update the global solution in outer Newton with the line-search method in order to achieve more stability.

4.4 Two-level RASPEN Implementation

The difference between one-level and two-level is that we add a coarse grid correction. As we have mentioned in the previous section, we exploit the extended Nicolaides to generate our coarse problem. The main algorithm is basically done in the same way as in the one-level scheme. But it is supplemented with *inner coarse Newton*. Therefore, the main loop of two-level RASPEN would look like

```
// outer Newton for two-level
while(not converged)
 do inner Newton
 do communication to glue up local solutions
 do inner coarse Newton
 do global linear solve
 do update the global solution
 do check the convergence criteria
end while
```

Note that the order to solve *inner Newton* or *inner coarse Newton* depends on the scheme we have introduced in Section 3.2. Here we basically exert the multiplicative version that adds the coarse correction after the local nonlinear solves. Hence, we do *inner Newton* before *inner coarse Newton.*

The restriction operator from fine grid to coarse grid R_0 introduced in Section 2.6 has to be defined. Again here, we create a vector of type `SolutionVector` to represent the partition of unity. But we have to multiply it by $\boldsymbol{t}$ which depends on the number of dimensions and variables. The following block would show how to define R_0 for one variable case.

```
// make partition of unity R0
std::vector<SolutionVector> R0(dim+1)
std::vector<SolutionVector> t (dim+1)
do assign values to each element of t
for (int j=0; j<dim+1; j++)
 R0[j] = p*t[j]            // element-wise product
end for
```

We assume that the first entry of vector $\boldsymbol{t}$ is an coefficient vector representing the constant function corresponding to the finite element function. The following entries are the coefficient vectors representing the Cartesian coordinate function corresponding to the finite element function. After that, we multiply element-wise by the partition of unity we have defined in the one-level scheme. We eventually obtain the restriction operator R_0 (for the considered processor). Note that each element in `R0` is a column vector in R_0^T. Then, we are able to do the *inner coarse Newton* to solve the nonlinear coarse problem which is a supplement of two-level RASPEN.

```
// inner coarse Newton
Matrix Ac
Vector zc
Vector rc
while(not converged)
 do go.Jacobian(x,A)       // update fine-grid Jacobian
 do coarse matrix assemble
 do ls.apply(Ac,zc,rc)     // do a coarse linear solve
 do prolongate zc
 do update solution and defect(rc)
 do check the convergence
end while
```

We write the coarse matrix assemble here in high-level but it is not done by `go` as in the local matrix case. `zc` and `rc` are vectors for the coarse problem. `rc` is computed by restricting the residual vector from fine grid to coarse grid. `zc` is a coarse solution obtained from coarse linear solve. It is needed to be prolongated back to the fine grid before updating. We would talk about the restriction and prolongation of the coarse problem in the lower level in Section 4.5. Actually, the coarse system is global, but we can assemble the coarse matrix by the local computations with some communications which we would also describe in Section 4.5. Note that solving the coarse problem by `ls.apply()` can be done in only one processor.

The further part is *global linear solve* which looks basically the same in the one-level approach but we require the linear coarse solve after linear subdomain solves. This has literally explained in Section 3.2.3 that we need to solve the local subdomain solves then subtract the result by the initial guess to obtain the initial guess for the linear coarse system. The coarse matrix can be reused from *inner coarse Newton* as the same for the fine-grid matrix.

4.5 Parallel Computation

There are parts of our method that can be done in parallel even when we deal with the coarse grid problem which is actually a global problem. Also in the global tangential system where we might have to compute the matrix-vector multiplication, we could set it up in parallel. This section would go through our algorithm and explain how we apply the parallel computation to the specific parts.

Restriction and Prolongation for Coarse Grid Problem

In *inner coarse Newton*, problem has to be set up in the coarse grid where the RHS is $R_0 F(x)$. We know that $F(x)$ can be computed by `go.residual(x,r)`. Then, we want to restrict `r` to the coarse grid residual `rc`. Note that we have done it locally. So, each processor contains a part of the full vector `rc`. Therefore, we need to distribute each part to one processor to do the coarse global solve (we usually use rank 0 in practice)

```
// restriction from fine to coarse grid

Vector rc(NumberofProcessors*NumberofVariables*(dim+1))
// rc is defined on rank 0

Vector local_rc(NumberofVariables*(dim+1))

for (int j=0; j<NumberofVariables*(dim+1); j++)
 local_rc[j] = R0[j].dot(r)
end for

do distribute local_rc to rank 0 and store in rc
```

On rank 0, we define `rc` vector which is of size (number of subdomains)*(number of variables)*(dim+1) to store the RHS for the coarse problem. The restriction looks quite a straightforward procedure. The next thing we would like to explain is the prolongation. After the coarse problem is solved, the obtained solution zc has to be prolongated back to the fine grid in order to update the global solution. It can be done by the following block of code

```
// prolongation from coarse to fine grid
SolutionVector d
do broadcast zc to local_zc

for (int j=0; j<d.size(); j++)
 for (int i=0; i<local_zc.size(); i++)
  d[j] += R0[j]*local_zc[i]
 end for
end for

do communicate d on overlapped dofs
```

zc is broadcasted to every processor. Here we know that which part of zc comes from which processor. We prolongate each part by multiplying with R_0^T on each processor which can be done by the above pseudocode and in the end, we require a communication on overlapped dofs.

Coarse Matrix Assembly

This subsection would show how to compute $A_0 = R_0 \nabla F(x) R_0^T$ in parallel. With the extended Nicolaides coarse space, we could assemble the coarse matrix A_0 in parallel by this following pseudocode.

```
// coarse matrix assembly
int rank_size = NumberofProcessors
int component = NumberofVariables*(dim+1)
SolutionVector R0_                  // R0 dummy

for (int j=0; j<rank_size; j++)     // iterate over each processors
 for (int J=0; J<component; J++)    // iterate over each components
  R0_=0

  if (j is self) R0_=R0[J]
  do communicate R0_

  if (j is neighbor or self) do A*R0_
  do communicate A*R0_   // be aware of over calculation on overlap!

  for (int I=0; I<component; I++)
   if (j is neighbor or self)
    AcRow[I][j*component+J] = R0[I].dot(A*R0_)
    // AcRow stores row of coarse matrix
  end if

 end for
end for
```

```
do distribute each AcRow to rank 0 to complete Ac
```

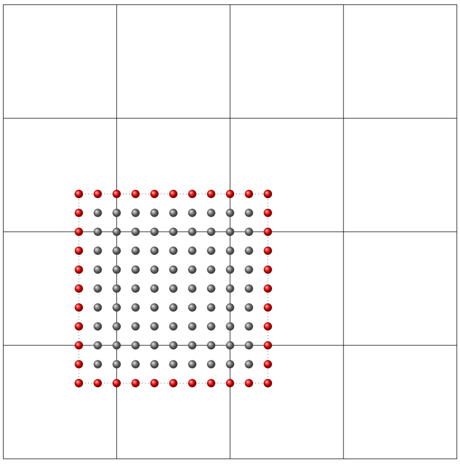


Figure 4.1: A subdomain with overlap = 1. Gray points are the dofs belonging to the considered subdomain. Red points are the dofs involved in coarse matrix assembly for this subdomain and belong only to its neighbors.

Note that we have R_0 defined locally in each processor and the local matrix `A` generated from `go` in each processor. Since each column vector of R_0^T is considered to have a support on its subdomain and some parts of its neighbors (overlapped region), one might notice that the computation between two column vectors from not connected subdomains is not necessary to do. Moreover, only a little information of the neighbors is needed because it requires only the dofs that is adjacent to the considered domain. It is from the fact that the rows of local matrix `A` which correspond to the border dofs contain the coefficients which correspond to the dofs that is adjacent to it. So, as shown in Figure 4.1, only the red points are involved in the computation for the considered domain. Every processor does its job and store the information of some rows of the coarse matrix A_0. In the end, they are distributed to rank 0 to complete A_0

Evaluation of $\nabla\mathcal{F}(x)v$

This matrix-vector multiplication is basically required in every linear iteration when we apply a Krylov iterative method like Conjugate Gradient (CG) or GMRES. The class that implements the evaluation of the matrix-vector multiplication is of type `LinearOperator` as we have mentioned. It is required as an argument for the GMRES solver. But by Algorithm 10, we can see that $\nabla\mathcal{F}(x)v$ is not the usual matrix-vector multiplication. It contains the local linear solves instead.

```
// evaluation of matrix-vector multiplication
Matrix A
SolutionVector v
SolutionVector y
SolutionVector z

do y = A*v
do set constrained dofs of y to 0
do ls.apply(A,z,y)
do communicate z with partition of unity

if (coarse grid)
 Matrix AC
 Matrix Ac
 SolutionVector d
 Vector yc
 Vector zc

 do v = v-z
 do y = AC*v
 do set constrained dofs of y to 0
 do restrict y to coarse grid and store in yc
 do ls.apply(Ac,zc,yc)
 do broadcast zc
 do prolongate zc to fine grid and store in d
 do communicate d on overlapped dofs
 do y = z+d

else

 do y = z

end if
```

Note that `A*v` interprets $R_k \nabla F(x) v$ even $R_k \nabla F(x)$ is not exactly the local matrix $A_k = R_k \nabla F(x) R_k^T$. $R_k \nabla F(x)$ differs from $R_k \nabla F(x) R_k^T$ by some additional columns corresponding to the Dirichlet boundary dofs [Dol+16] but the local matrix generated from `dune-pdelab` machinery is already incorporated those degrees of freedom. Therefore, we could do it in our implementation. `Matrix A` can be reused from *inner Newton* and `Matrix Ac` can be reused from *inner coarse Newton* as well. We emphasize that `Matrix AC` is not a coarse matrix but it is a fine-grid matrix updated within *inner coarse Newton* which can also be reused here. Note again here that, the above block of code explains the multiplicative version with the coarse correction added after the subdomain solves.

5 Numerical Expriments

5.1 Overview

This section dedicates to giving an overview of the numerical experiments chapter. It would provide the applications of RASPEN on several problems, with different discretization schemes (FE or DG), from stationary to instationary, from simple scalar-value problems to complex systems of PDEs, in order to inspect the flexibility and performance of RASPEN.

We would like to start with the nonlinear Poisson problem which is a scalar-value problem where we know the analytical solution. This part would show the validity of RASPEN and its two-level variants so that it can provide the accurate outcome.

Then, we step further to investigate the robustness of RASPEN in p-Laplace problems where p is the parameter indicating the difficulty of the problem; The higher value of p, the harder task for the solver. We aim to see that RASPEN can handle the higher value of p than the ordinary Newton method.

The problems so far are stationary problems. Therefore, we would advance to the instationary problems where the temporal derivative is involved. We look into the diffusive wave approximation in order to show that RASPEN is applicable to the transient problem. The next problem is Richards' equation describing the flow in the porous medium where there is also a nonlinearity term on the temporal derivative. Then, we move ahead to the complex systems of PDEs describing the leaching and carbonation of concrete.

5.2 Nonlinear Poisson Equation

The first example we consider is the nonlinear Poisson equation as follows:

$$
\begin{aligned}
-\nabla \cdot \big((1+u)^2\, \nabla u\big) &= 0 && \text{in } \Omega = [0,1]^2, && (5.1a)\\
u &= x && \text{on } \Gamma_D = \{(0,y) \cup (1,y) \subset \partial\Omega)\}, && (5.1b)\\
-\nabla u \cdot \nu &= 0 && \text{on } \Gamma_N = \partial\Omega \setminus \Gamma_D && (5.1c)
\end{aligned}
$$

where we know the exact solution: $u(x,y) = \big(\,(2^3-1)\,x+1\big)^{\frac{1}{3}} - 1$ (shown in Figure 5.1).

We apply RASPEN to the algebraic equations resulted from discretization schemes like Finite Element (FE) and Discontinuous Galerkin (DG).

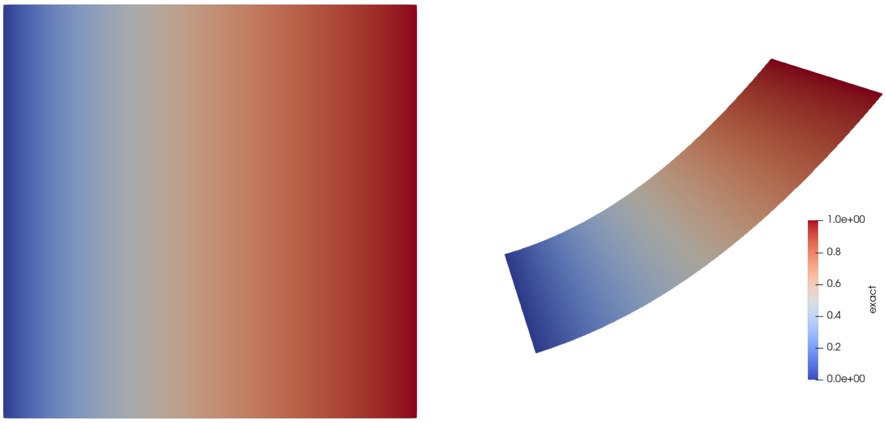


Figure 5.1: Illustration of the exact solution of the problem (5.1).

Numerical Results

We mention again that all numerical results in this study provided by DUNE software. The square domain $[0, 1]^2$ is discretized into a rectangular grid by YaspGrid. The Q_1 finite element is employed for both finite element and discontinuous Galerkin methods. We choose the initial condition as $u(x, y) = x$. We investigate the scalability of RASPEN and Newton on a large number of processors by fixing the number of degrees of freedom per subdomain. We start from 2×2 subdomains and increase it until 64×64 subdomains. For finite element, we fix the number of cells per subdomain to 64×64 and 128×128. And for discontinuous Galerkin, we choose 16×16 and 32×32. The overlapped region is given in such a way that it keeps the same ratio between subdomain diameter (H) and overlap size δ.

Table 5.1 and 5.2 show the comparison in terms of the number of iterations between RASPEN (and its variants) and Newton. Three columns of RASPEN represent the results from RASPEN with the multiplicative extended Nicolaides coarse space (Multi-RASPEN), RASPEN with the additive extended Nicolaides coarse space (Add-RASPEN), and RASPEN without coarse space (1-level-RASPEN), respectively. And two columns of Newton stand for Newton with GMRES as a linear solver preconditioned with the restricted Additive Schwarz method + extended Nicolaides coarse space (2-level-Newton), and with the restricted Additive Schwarz method but no coarse space (1-level-Newton), respectively. The *outer itr.* means the outer Newton iteration when the error reduction is below the criterion 10^{-6}. It is clearly see that RASPEN and its variants can converge with less number of outer iteration than Newton and the error reduction is illustrated in Figure 5.2 and 5.3. The *local nonlinear + coarse nonlinear itr.* depicts the average number of *inner Newton* iteration applied to the local nonlinear problems in each subdomain and the number of *inner coarse Newton* applied to solve the coarse nonlinear problem.

64×64 cells/subd.	RASPEN			Newton	
	outer itr.	local nonlinear + coarse nonlinear itr.	global linear itr.	outer itr.	global linear itr.
2×2 subd	3 (3) [3]	3 + 4 (3 + 4) [3]	10 (20) [18]	4 [4]	19 [18]
		2 + 3 (2.5 + 2) [3]	15 (22) [20]		20 [20]
		3 + 1 (2 + 2) [2]	13 (22) [20]		20 [20]
					20 [20]
4×4 subd	3 (3) [3]	3 + 4 (3 + 4) [3]	20 (28) [50]	4 [4]	24 [58]
		2.25 + 2 (2.25 + 2) [2.25]	22 (33) [56]		25 [60]
		1.25 + 2 (1.38 + 2) [1.75]	22 (34) [56]		26 [60]
					26 [60]
8×8 subd	3 (3) [3]	2.81 + 4 (2.81 + 4) [2.81]	19 (27) [145]	4 [4]	24 [172]
		2 + 2 (2 + 2) [2.03]	24 (35) [173]		24 [153]
		1 + 1 (1 + 1) [1.28]	24 (35) [168]		26 [159]
					26 [165]
16×16 subd	3 (3) [3]	2 + 4 (2 + 4) [2]	17 (25) [419]	4 [4]	22 [431]
		2 + 2 (2 + 2) [1.94]	24 (35) [448]		23 [433]
		1 + 1 (1 + 1) [1.06]	24 (35) [385]		24 [462]
					24 [426]
32×32 subd	3 (3) [3]	2 + 4 (2 + 4) [2]	15 (22) [1590]	4 [4]	19 [1301]
		1.94 + 2 (1.97 + 2) [1.85]	24 (34) [1154]		21 [1475]
		0.94 + 1 (0.98 + 1) [1.03]	22 (33) [1101]		20 [1117]
					21 [882]
64×64 subd	3 (3) [3]	2 + 4 (2 + 4) [2]	13 (19) [3108]	4 [4]	17 [3425]
		1.83 + 2 (1.83 + 2) [1.67]	23 (33) [3836]		18 [2849]
		0.80 + 1 (0.85 + 1) [1]	19 (29) [3069]		16 [4195]
					18 [2959]

Table 5.1: Number of iterations of RASPEN and Newton for finite element with fixed 64×64 cells per subdomain.

128×128	RASPEN			Newton	
cells/subd.	outer itr.	local nonlinear + coarse nonlinear itr.	global linear itr.	outer itr.	global linear itr.
2×2 subd	3 (3) [3]	3 + 4 (3 + 4) [3]	16 (21) [18]	4 [4]	19 [20]
		2 + 3 (2.5 + 3) [3]	16 (22) [20]		20 [19]
		3 + 2 (2 + 2) [2]	17 (22) [21]		20 [20]
					20 [20]
4×4 subd	3 (3) [3]	3 + 4 (3 + 4) [3]	20 (30) [52]	4 [4]	27 [58]
		2.25 + 3 (2.25 + 2) [2.25]	22 (36) [56]		27 [60]
		1.5 + 2 (1.38 + 2) [1.63]	23 (36) [56]		27 [60]
					28 [60]
8×8 subd	3 (3) [3]	2.63 + 4 (2.63 + 4) [2.63]	20 (30) [149]	4 [4]	26 [171]
		2 + 3 (2 + 2) [2.03]	24 (38) [174]		26 [154]
		1 + 2 (1 + 2) [1.25]	24 (39) [161]		28 [159]
					29 [165]
16×16 subd	3 (3) [3]	2 + 4 (2 + 4) [2]	19 (27) [430]	4 [4]	24 [448]
		2 + 2 (2 + 2) [1.94]	24 (38) [461]		25 [425]
		1 + 1 (1 + 1) [1.06]	25 (38) [464]		27 [463]
					29 [439]
32×32 subd	3 (3) [3]	2 + 4 (2 + 4) [2]	16 (24) [1488]	4 [4]	21 [1422]
		1.97 + 2 (1.97 + 2) [1.82]	24 (38) [1489]		23 [1210]
		1 + 1 (1 + 1) [1.03]	25 (37) [1317]		23 [1074]
					23 [987]
64×64 subd	3 (3) [4]	2 + 4 (2 + 4) [2]	14 (21) [2804]	4 [4]	18 [2928]
		1.86 + 2 (1.86 + 2) [1.6]	24 (37) [2560]		21 [2813]
		0.87 + 1 (0.9 + 1) [1]	23 (37) [3209]		18 [3479]
		[1]	[1739]		21 [3296]

Table 5.2: Number of iterations of RASPEN and Newton for finite element with fixed 128×128 cells per subdomain.

These numbers, of course, do not appear in the ordinary Newton method. The *inner Newton* and *inner coarse Newton* are solved by a sparse direct solver with the error reduction criterion is set to 10^{-9}. The *global linear itr.* is the iteration of GMRES applied to solve the tangential system which we set the reduction criterion to 10^{-9}. The coarse grid correction exhibits its strength here as we can see that the number of linear iterations remains nearly constant in two-level variants. And it does not depend on the number of subdomains as in the one-level scheme.

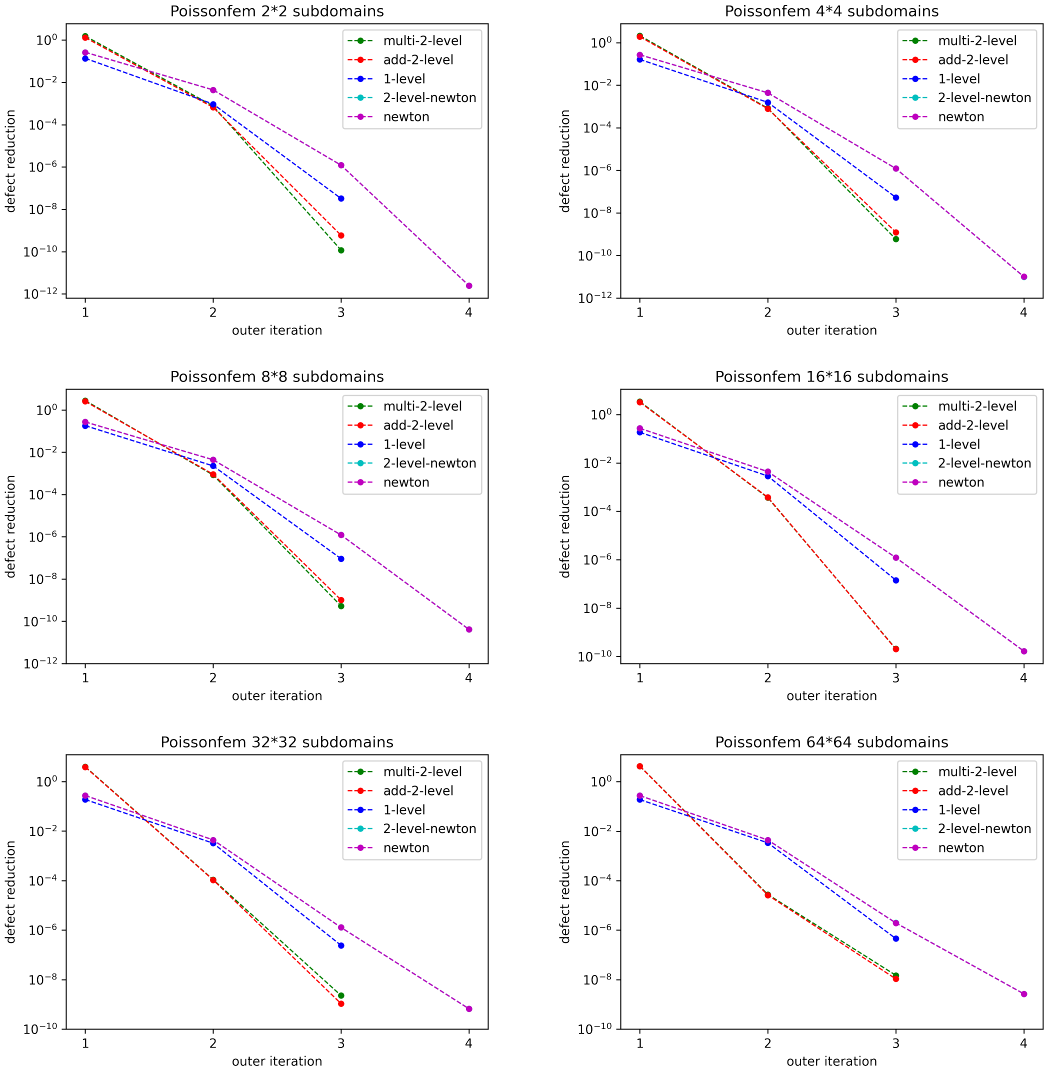


Figure 5.2: Illustrations of error reduction of finite element with fixed 64×64 cells per subdomain.

The error reduction of two-level RASPEN does not look good at the first iteration but after that, it reduces rapidly so that it converges faster than other variants in terms of the number of outer iterations. Note that, the ordinary Newton with or without coarse grid is applied to solve the same (global) nonlinear problem; therefore, the error reduction of two-level-Newton and one-level-Newton look almost the same. Tables 5.3 and 5.4 and Figures 5.4 and 5.5 represent the results from discontinuous Galerkin in the same aspects. They display the flexibility of RASPEN that it is available to apply for the different discretisation schemes and still provide good performance.

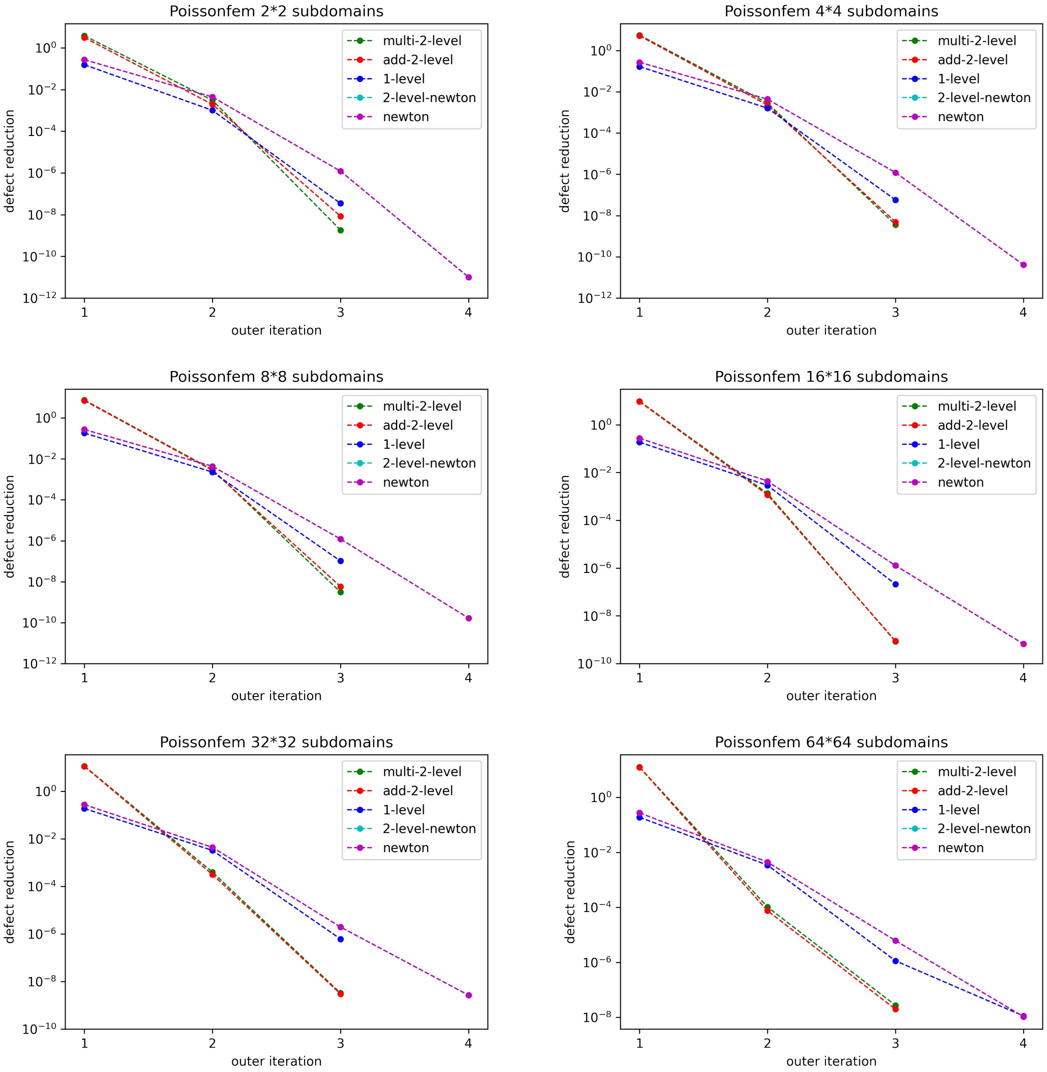


Figure 5.3: Illustrations of error reduction of finite element with fixed 128×128 cells per subdomain.

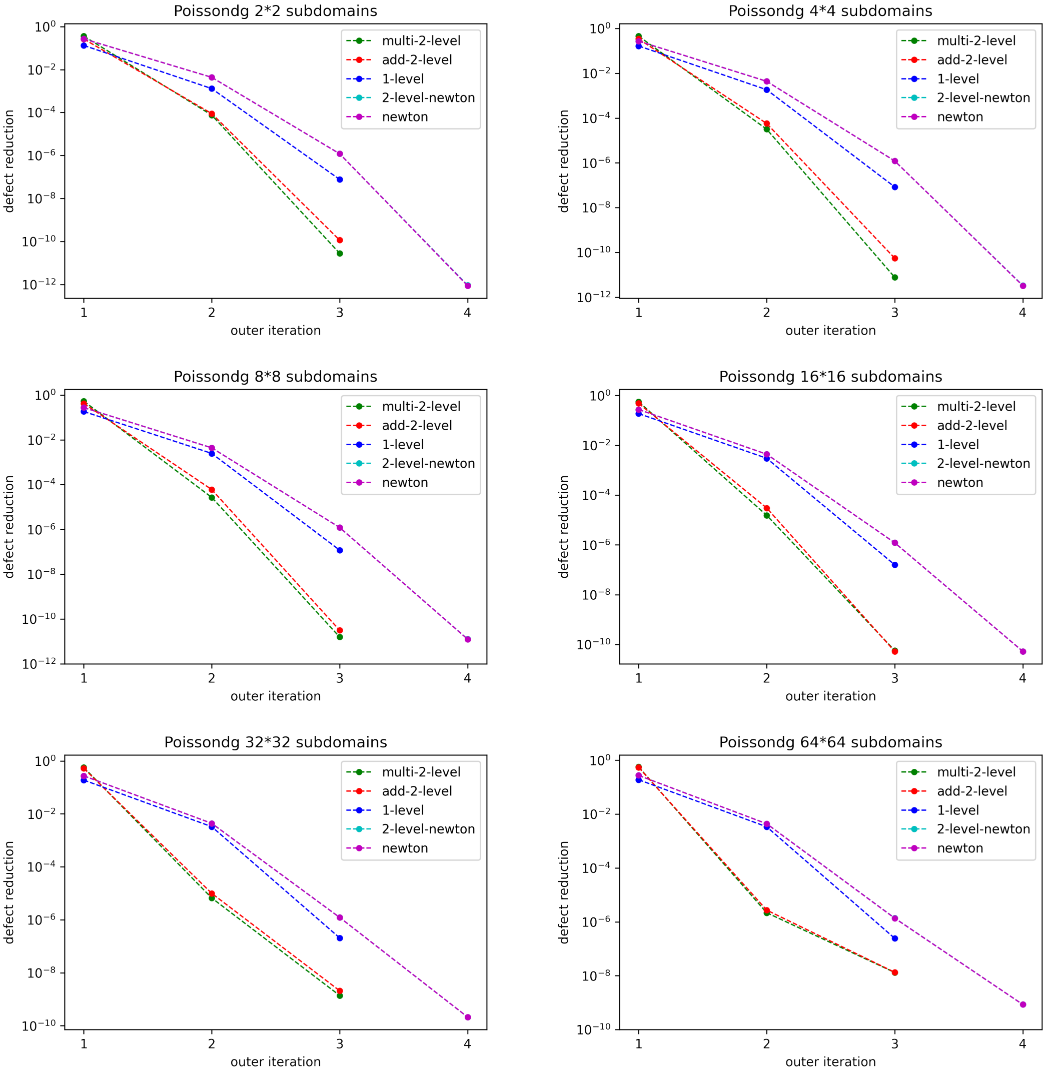
16 × 16	RASPEN			Newton	
cells/subd.	outer itr.	local nonlinear + coarse nonlinear itr.	global linear itr.	outer itr.	global linear itr.
2 × 2 subd	3 (3) [3]	3 + 4 (3 + 4) [3] 3 + 2 (3 + 2) [3] 1 + 1 (1 + 2) [2]	43 (55) [78] 45 (56) [87] 42 (58) [88]	4 [4]	55 [75] 53 [82] 56 [81] 58 [77]
4 × 4 subd	3 (3) [3]	3 + 4 (3 + 4) [3] 2.25 + 2 (2.25 + 2) [2.25] 1 + 2 (1 + 2) [1.75]	52 (64) [205] 58 (71) [273] 57 (75) [221]	4 [4]	65 [263] 65 [277] 66 [229] 70 [189]
8 × 8 subd	3 (3) [3]	2.72 + 4 (2.72 + 4) [2.72] 2.13 + 2 (2.13 + 2) [2.13] 1 + 1 (1 + 1) [1.38]	47 (57) [521] 61 (78) [555] 61 (79) [598]	4 [4]	57 [626] 60 [607] 62 [670] 65 [741]
16 × 16 subd	3 (3) [3]	2 + 4 (2 + 4) [2] 1.94 + 2 (1.95 + 2) [1.94] 1 + 1 (1 + 1) [1.06]	41 (50) [2242] 59 (79) [2600] 57 (77) [1714]	4 [4]	50 [1827] 54 [2703] 52 [1528] 57 [1484]
32 × 32 subd	3 (3) [3]	2 + 4 (2 + 4) [2] 1.85 + 2 (1.85 + 2) [1.88] 0.92 + 1 (0.89 + 1) [1.03]	34 (42) [8087] 58 (77) [5018] 52 (72) [5238]	4 [4]	42 [10087] 47 [8708] 45 [5187] 52 [4513]
64 × 64 subd	3 (3) [3]	2 + 4 (2 + 4) [2] 1.72 + 2 (1.72 + 2) [1.71] 0.61 + 1 (0.61 + 1) [1]	27 (37) [33603] 55 (76) [23458] 45 (64) [27876]	4 [4]	36 [37069] 40 [37225] 38 [18593] 50 [28468]

Table 5.3: Number of iterations of RASPEN and Newton for DG with fixed 16×16 cells per subdomain.

32 × 32	RASPEN			Newton	
cells/subd.	outer itr.	local nonlinear + coarse nonlinear itr.	global linear itr.	outer itr.	global linear itr.
2 × 2 subd	3 (3) [3]	3 + 4 (3 + 4) [3]	21 (26) [25]	4 [4]	25 [25]
		2 + 3 (2.5 + 2) [3]	22 (29) [27]		26 [27]
		2 + 2 (2 + 2) [2]	24 (30) [27]		26 [27]
					26 [27]
4 × 4 subd	3 (3) [3]	3 + 4 (3 + 4) [3]	27 (38) [81]	4 [4]	32 [85]
		2.25 + 2 (2.25 + 2) [2.25]	30 (44) [90]		34 [86]
		1.25 + 2 (1.38 + 2) [1.75]	31 (46) [90]		35 [84]
					35 [86]
8 × 8 subd	3 (3) [3]	2.72 + 4 (2.72 + 4) [2.72]	24 (37) [200]	4 [4]	32 [206]
		2 + 2 (2.03 + 2) [2.03]	33 (45) [225]		32 [242]
		1 + 1 (1 + 1) [1.36]	34 (46) [224]		35 [236]
					36 [204]
16 × 16 subd	3 (3) [3]	2 + 4 (2 + 4) [2]	23 (33) [565]	4 [4]	28 [597]
		2 + 2 (2 + 2) [1.94]	33 (46) [582]		30 [655]
		1 + 1 (1 + 1) [1.06]	33 (47) [765]		32 [620]
					32 [739]
32 × 32 subd	3 (3) [3]	2 + 4 (2 + 4) [2]	20 (28) [1694]	4 [4]	24 [1461]
		1.94 + 2 (1.94 + 2) [1.85]	33 (45) [2585]		27 [1795]
		0.93 + 1 (0.93 + 1) [1.03]	29 (43) [1502]		26 [1694]
					27 [1419]
64 × 64 subd	3 (3) [3]	2 + 4 (2 + 4) [2]	17 (24) [5032]	4 [4]	21 [4721]
		1.81 + 2 (1.81 + 2) [1.69]	32 (44) [3834]		23 [5083]
		0.81 + 1 (0.84 + 1) [1]	35 (40) [4588]		20 [6343]
					24 [6338]

Table 5.4: Number of iterations of RASPEN and Newton for DG with fixed 32×32 cells per subdomain.

For the time measurement, we inspect a strong scaling by fixing the size of the global problem and then increasing the number of processors to reduce the workload per processor. Note that, we examine only on finite element method. We do an experiment on the machine, AMD EPYC 7713 which is a 64-core processor with a base frequency of 2 GHz [AMD21]. Therefore, we run the test on $2 \times 2, 4 \times 4$, and 8×8 subdomains. We look into the global problem size of 512×512, 1024×1024, and 2048×2048 cells per subdomain and measure its scalability. Tables 5.5 - 5.7 illustrate the CPU time and its scalability. We can see that Newton with the coarse grid correction for the linear solver (2-level-Newton) is the fastest in terms of CPU time, but two-level-RASPEN variants provide great scalability.



Poissondg 2*2 subdomains

Poissondg 4*4 subdomains

Poissondg 8*8 subdomains

Poissondg 16*16 subdomains

Poissondg 32*32 subdomains

Poissondg 64*64 subdomains

Figure 5.4: Illustrations of error reduction of DG with fixed 16×16 cells per subdomain.

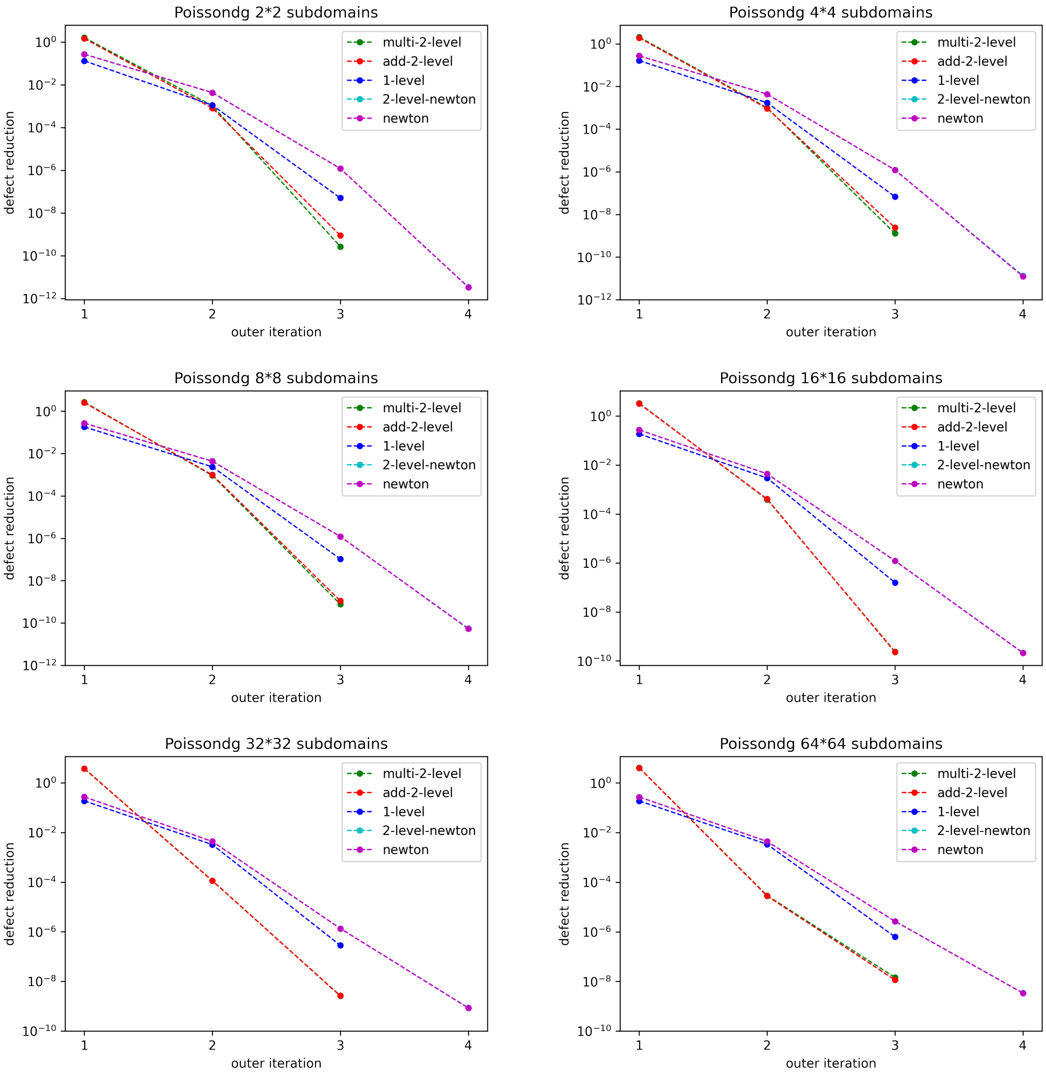


Figure 5.5: Illustrations of error reduction of DG with fixed 32×32 cells per subdomain.

512×512 cells		2×2 subd.	4×4 subd.	8×8 subd.
Multi-RASPEN	CPU time (s)	76.29	13.25	4.13
	scaling	-	5.76	36.94
Add-RASPEN	CPU time (s)	96.61	16.91	5.19
	scaling	-	5.71	18.62
1-level-RASPEN	CPU time (s)	122.76	41.6	21.04
	scaling	-	2.95	5.84
2-level-Newton	CPU time (s)	21.86	4.02	1.88
	scaling	-	5.44	11.63
1-level-Newton	CPU time (s)	27.29	9.35	4.71
	scaling	-	2.92	5.79

Table 5.5: Strong scaling experiment on global problem size 512×512 cells.

1024×1024 cells		2×2 subd.	4×4 subd.	8×8 subd.
Multi-RASPEN	CPU time (s)	683.2	99.25	19.64
	scaling	-	6.88	34.79
Add-RASPEN	CPU time (s)	819.2	124.6	24.08
	scaling	-	6.57	34.02
1-level-RASPEN	CPU time (s)	1284	386.8	155.4
	scaling	-	3.32	8.26
2-level-Newton	CPU time (s)	132.6	24.6	8.03
	scaling	-	5.39	16.51
1-level-Newton	CPU time (s)	211.6	89.75	56.68
	scaling	-	2.36	3.73

Table 5.6: Strong scaling experiment on global problem size 1024×1024 cells.

2048×2048 cells		2×2 subd.	4×4 subd.	8×8 subd.
Multi-RASPEN	CPU time (s)	7784.7	801.1	147.9
	scaling	-	9.72	52.64
Add-RASPEN	CPU time (s)	11111	1037.4	180.3
	scaling	-	10.71	61.63
1-level-RASPEN	CPU time (s)	14268	4262.6	1379.7
	scaling	-	3.35	10.34
2-level-Newton	CPU time (s)	939.7	155.28	51.26
	scaling	-	6.05	18.33
1-level-Newton	CPU time (s)	1466.3	626.94	496.84
	scaling	-	2.34	2.95

Table 5.7: Strong scaling experiment on global problem size 2048×2048 cells.

5.3 P-Laplace Equation

This section provides the numerical experiments to investigate the robustness of RASPEN compared to the ordinary Newton method. The p-Laplace equation is introduced as the following:

$$-\Delta_p u = 1 \qquad \text{in } \Omega = [0,1]^2, \tag{5.2a}$$

$$u = 0 \qquad \text{on } \partial\Omega \tag{5.2b}$$

where the p-Laplace operator $\Delta_p := \nabla \cdot (|\nabla u|^{p-2} \nabla u)$ is defined for $p \geq 2$. One might notice that if $p = 2$, the problem becomes the linear Poisson equation. p could be set unequally for different parts of the domain in order to impose the local nonlinearity which could trouble the convergence issue of Newton method. We consider the problem in two different scenarios as follows:

5.3.1 First Scenario

We set up p in the following way as proposed in [KLR14]. Let us assume that domain Ω is composed of nonoverlapped subdomains Ω_i

$$\Omega = \bigcup_{i=1}^{p} \Omega_i.$$

Then we introduce a small frame $\Omega_{i,I}$ embedded in each subdomain Ω_i and define the surrounded area $\Omega_{i,\eta}$ by

$$\Omega_{i,\eta} := \{x \in \Omega_i \ ; \ \mathrm{dist}(x, \partial\Omega_i) < \eta\}.$$

Furthermore, We denote the whole embedded area by $\Omega_I := \bigcup_{i=1}^{N} \Omega_{i,I}$ and the outer area by $\Omega_\eta := \bigcup_{i=1}^{N} \Omega_{i,\eta}$. In the first example, we always fix $p = 2$ on Ω_η

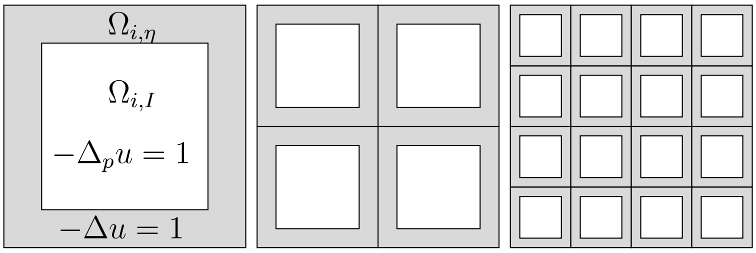


Figure 5.6: Left: Subdomain Ω_i, with the embedded frame $\Omega_{i,I}$ surrounded by the outer part $\Omega_{i.\eta}$.
Middle: Example of a computational domain for 4 subdomains.
Right: Example of a computational domain for 16 subdomains.

and slightly increase p on Ω_I so that the local nonlinearity also increases until the methods is nearly unable to solve. We hopefully expect that RASPEN could solve for higher number of p compared to the ordinary Newton method. Since the homogeneous Dirichlet boundary conditions are imposed, using the initial guess from linear interpolation as in the previous example would give $u^{(0)} = 0$ which is not a good guess because the Jacobian matrix is then singular. So, we impose the initial condition by this function

$$u^{(0)}(x, y) = x(1 - x)y(1 - y)$$

which also satisfies the Dirichlet boundary conditions. And we choose $\eta = \frac{1}{8}\times$ length of subdomain.

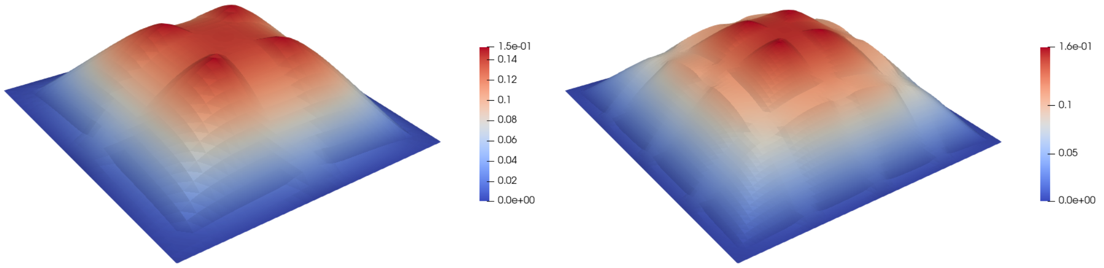
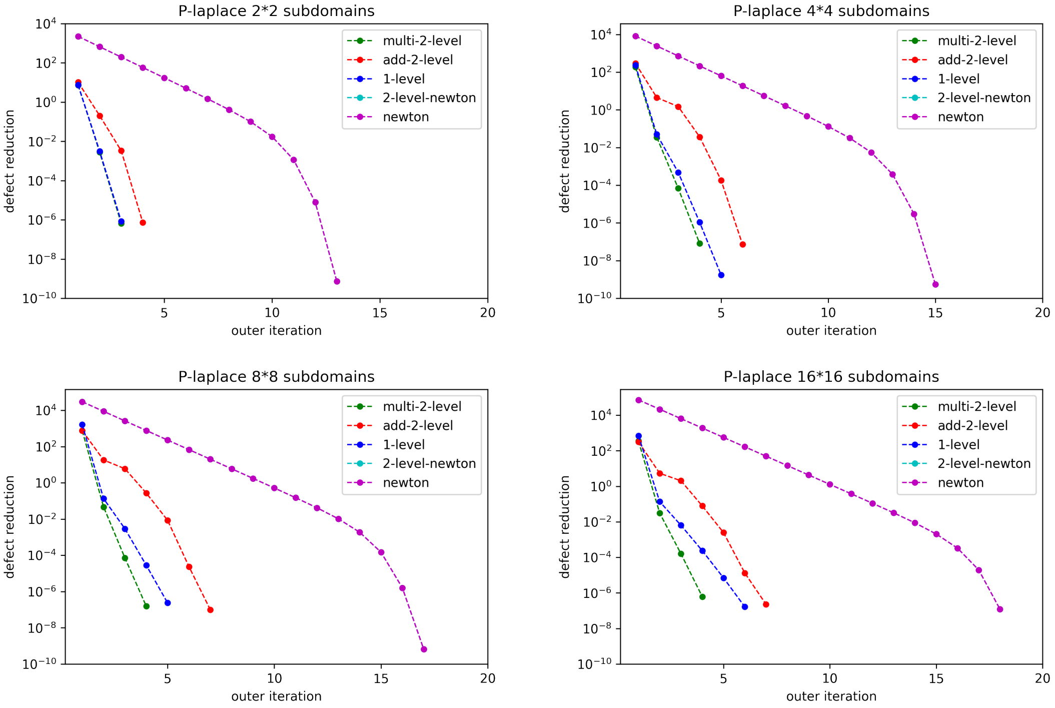
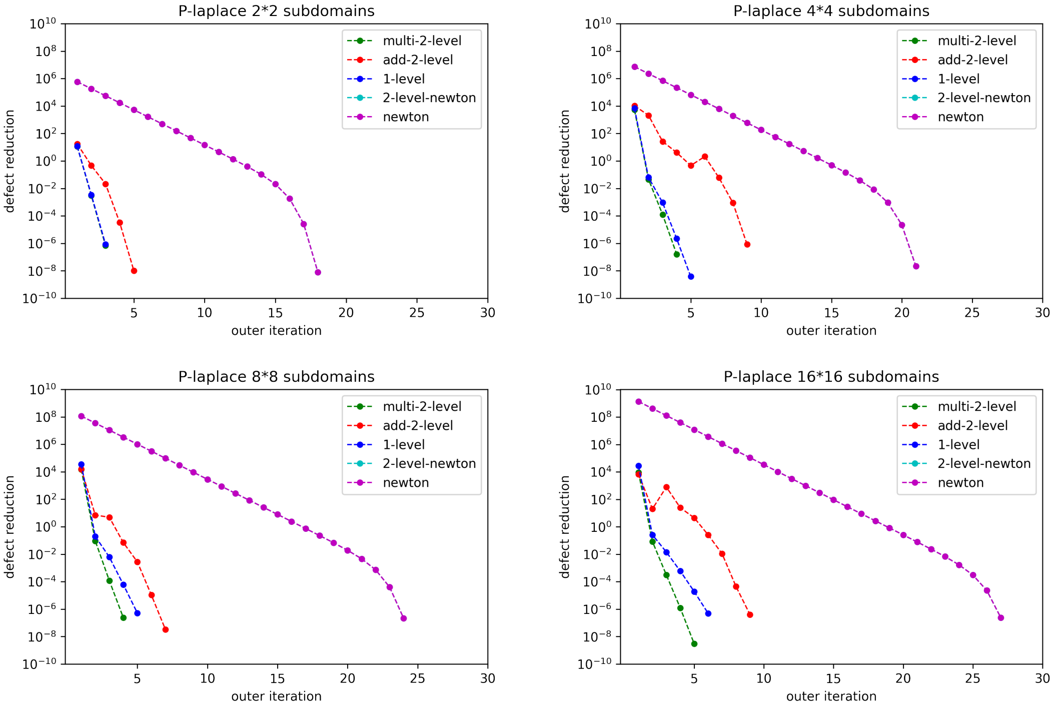


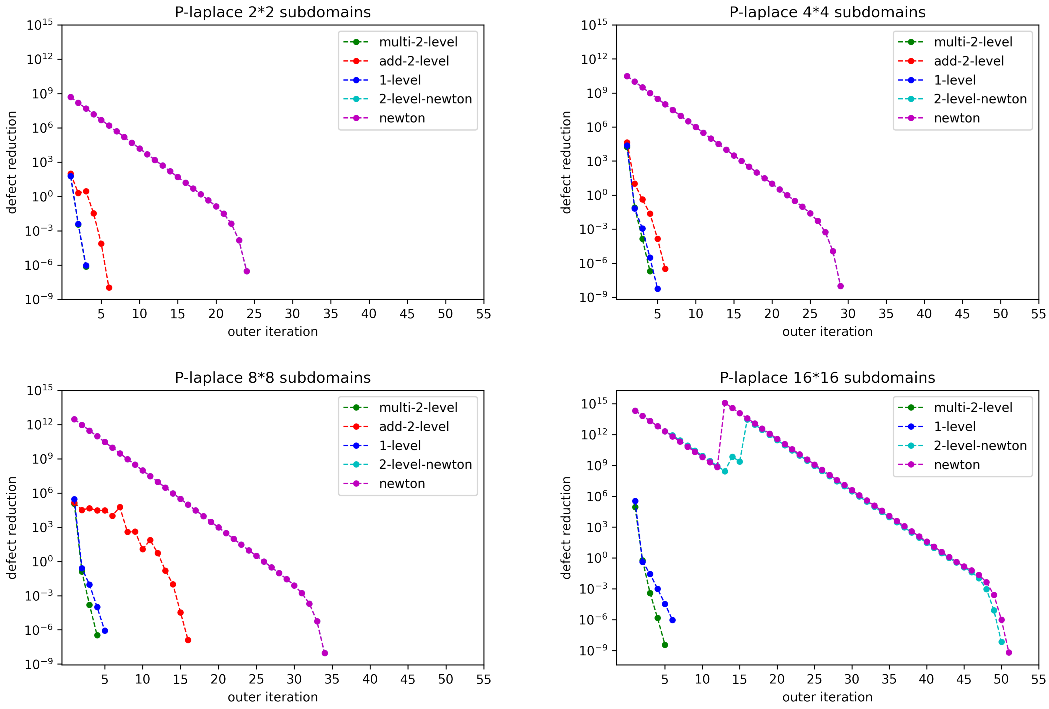
Figure 5.7: Left: The solution of the first scenario where the domain Ω is divided into 4 subdomains.
Right: The solution of the first scenario where the domain Ω is divided into 16 subdomains.


P-laplace 2*2 subdomains
P-laplace 4*4 subdomains
P-laplace 8*8 subdomains
P-laplace 16*16 subdomains
defect reduction
outer iteration
multi-2-level
add-2-level
1-level
2-level-newton
newton
10^{4}
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
5
10
15
20

(a) $p = 4$


P-laplace 2*2 subdomains
P-laplace 4*4 subdomains
P-laplace 8*8 subdomains
P-laplace 16*16 subdomains
defect reduction
outer iteration
multi-2-level
add-2-level
1-level
2-level-newton
newton
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
5
10
15
20
25
30

(b) $p = 4.5$



(c) $p = 5$

Figure 5.8: Illustrations of error reduction of the first scenario of p-Laplace problem with $p = 4,\ 4.5,\ 5$.

Numerical Results

Again here, the computational square domain $[0, 1]^2$ is discretized by YaspGrid and Q_1 finite element is employed. We fix the number of cells per subdomain to 128×128 and increase the value of p from 4 to 5 where some solver fails or struggles to converge. We run the experiments on 2×2 to 16×16 subdomains. The visualized results of 4 and 16 subdomain cases are shown in Figure 5.7. The error reduction of this scenario is illustrated in Figure 5.8. One can see that increasing p makes the problem harder to solve by the number of outer iterations. Moreover, the ordinary Newton with or without coarse grid takes many more outer iterations compared to RASPEN. RASPEN can still keep the number of outer iterations, even though the problem gets more difficult. RASPEN with multiplicative linear Nicolaides coarse space (Multi-RASPEN) outperforms on the higher value of p and the higher number of subdomains. Moreover, this experiment exhibits that RASPEN could be even better at the computational time. Table 5.8 presents the time used to finish the first scenario from the different schemes with $p = 4$, 4.5, and 5 on 8×8 subdomains. We can clearly see that multiplicative two-level RASPEN provides the best results.

Methods	CPU time (s)		
	p=4	p=4.5	p=5
Multi-RASPEN	7.267	8.142	8.981
Additive-RASPEN	17.762	18.028	47.549
1-level-RASPEN	22.038	23.553	26.049
2-level-Newton	8.730	12.154	18.233
1-level-Newton	10.673	14.304	20.301

Table 5.8: Computational time of the first scenario of the p-Laplace problem with p = 4, 4.5, 5 on 8 × 8 subdomains.

5.3.2 Second Scenario

We consider another type of local nonlinearity which is not embedded in the subdomain but connected with other subdomains as in Figure 5.9. We can see the channel lying horizontally through the subdomain and connecting with neighbors to become a long horizontal channel. We denote Ω_C to be the union of the channels and the rest of the domain is Ω_R. And we propose the coefficient functions $\alpha : \Omega \to \mathbb{R}$ by

$$\alpha(x) = \begin{cases} \tilde{\alpha} & \text{if } x \in \Omega_C, \\ 1 & \text{elsewhere.} \end{cases}$$

Then, we consider the problem

$$-\alpha \Delta_p u = 1 \qquad \text{in } \Omega = [0,1]^2, \tag{5.3a}$$

$$u = 0 \qquad \text{on } \partial\Omega \tag{5.3b}$$

where $p = 4$ holds for the entire domain Ω and $\tilde{\alpha} = \{10^3, 10^6\}$ which causes the high condition numbers to the systems [Lan15].

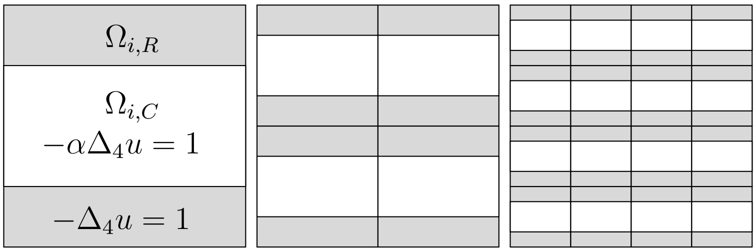


Figure 5.9: Left: Subdomain Ω_i, with the channel $\Omega_{i,C}$ lying horizontally.
Middle: Example of a computational domain in the second scenario for 4 subdomains.
Right: Example of a computational domain in the second scenario for 16 subdomains.

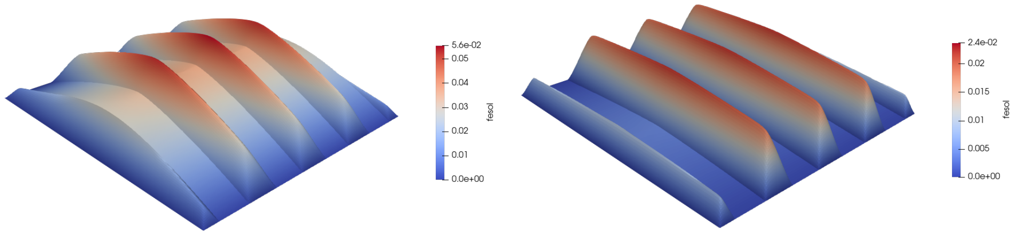


Figure 5.10: The solution of the second scenario where the domain Ω is divided into 16 subdomains with a coefficient $\tilde{\alpha} = 10^3$ (left) and $\tilde{\alpha} = 10^6$ (right).

Numerical Results

The computational domain and Q_1 finite element are employed as the same in the first scenario. The channel is of width $\frac{1}{2}\times$ length of subdomain lying in the middle of each subdomain. The visualized results of 16 subdomain cases are shown in Figure 5.10. We run the experiments on 2×2 to 16×16 subdomains. For $\tilde{\alpha} = 10^3$, the one-level approach is not a consistent solver for this scenario. It does not converge on 2×2 and 8×8 subdomain cases. But we can see that the multiplicative second level still outperforms any other approaches. It happens also for $\tilde{\alpha} = 10^6$. While other methods become inconsistent; the one-level approach does not converge on 2×2 subdomain case, the additive second level does not converge in 16×16 subdomain case, and they are struggling to solve when the number of subdomains increases, the multiplicative second level still looks consistent in terms of convergence behavior as shown in Figure 5.11.

5.4 Diffusive Wave Approximation

Two previous sections talk about the stationary problems. This section would show that RASPEN can also apply to the instationary problems. We consider a model problem so-called diffusive wave approximation [Kin86] describing the overland flows written in the form

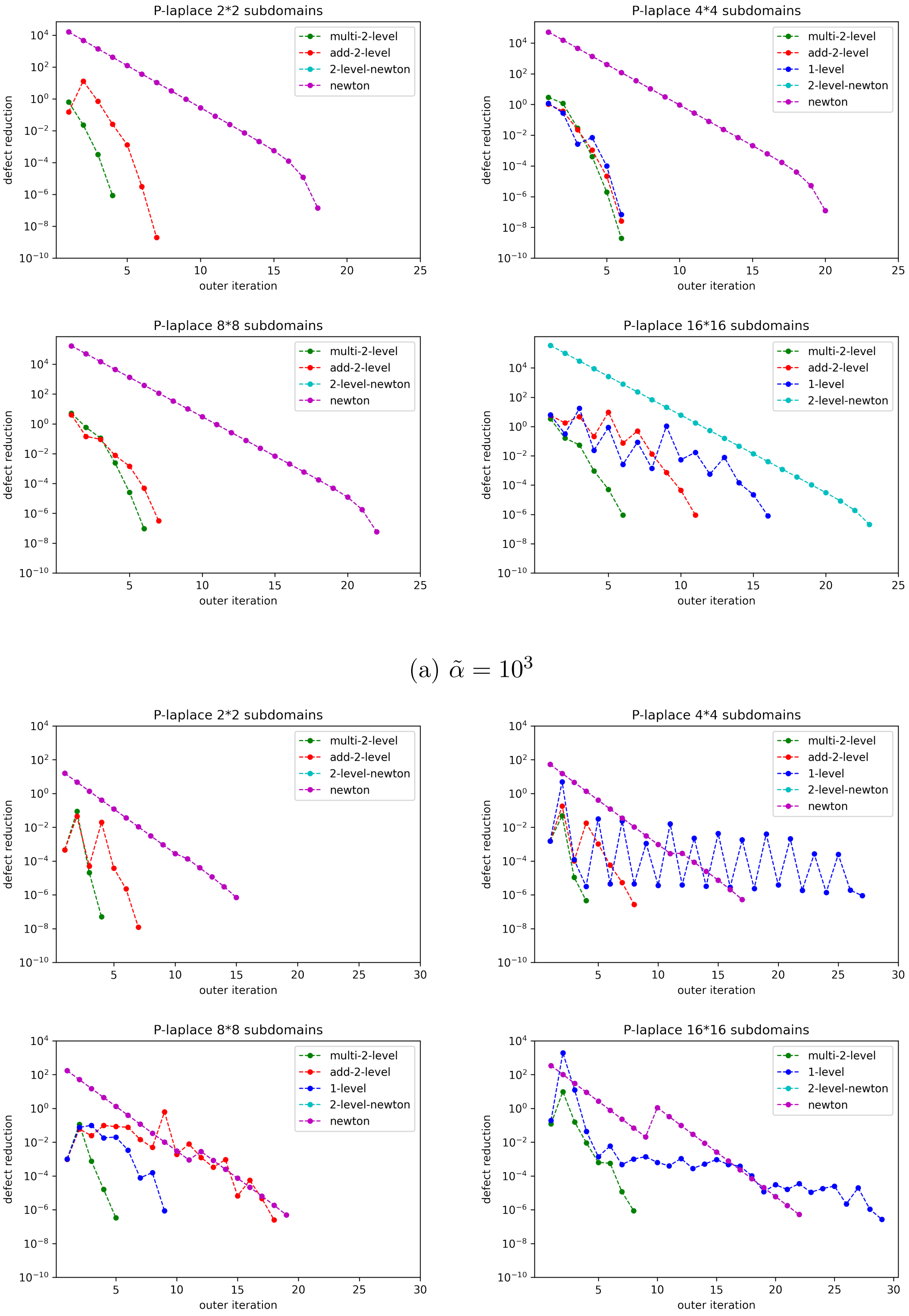
$$\partial_t u - \nabla \cdot (D(u, \nabla u)\nabla u) = f \qquad \text{in } \Omega \times \Sigma = [0,1]^2 \times [0, \infty], \tag{5.4a}$$

$$u = u_0 \qquad \text{in } \Omega \times \{t = 0\} \tag{5.4b}$$

where u stands for the water height, and D is the diffusion coefficient depending on u and ∇u given by

$$D(u, \nabla u) = \frac{(u - z)^{\alpha_M}}{C_M |\nabla u|^{1-\gamma_M}}$$

where z represents the bathymetry, α_M and γ_M are the parameters from Manning's formula, C_M is renowned for the friction coefficient, and f is a source/sink function which would be precipitation such as rainfall in this study. We can see that



(a) $\tilde{\alpha} = 10^3$

(b) $\tilde{\alpha} = 10^6$

Figure 5.11: Illustrations of error reduction of the second scenario of p-Laplace problem with $\tilde{\alpha} = \{10^3, 10^6\}$.

equations (5.4) have an additional term from the previous sections, that is, the temporal derivative term. We discretize the time interval into subintervals and find the solution for each substep. That would be considered as we solve the series of stationary problems. And we can apply RASPEN to solve these. We start the weak formulation by multiplying a test function and integrating in space and it results that

$$\frac{d}{dt}\int_{\Omega} uv\,dx + \int_{\Omega} D(u,\nabla u)\nabla u \cdot \nabla v + \int_{\partial\Omega} jv\,ds = 0 \qquad \begin{array}{l}\forall v \in V(t),\\ t \in \Sigma,\end{array} \tag{5.5}$$

where $V(t) = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D(t)\}$. We denote the temporal residual form by

$$m^{\mathrm{L2}}(u,v) = \int_{\Omega} uv\,dx$$

and the spatial residual form is actually the residual form of the nonlinear Poisson problem. Therefore, we can express the residual form by

$$\frac{d}{dt} m^{\mathrm{L2}}(u,v) + r^{\mathrm{NLP}}(u,v) = 0 \quad \forall v \in V(t), t \in \Sigma.$$

We can choose the finite-dimensional space $V_h^{k,d}(\mathcal{T}_h,t)$ which is introduced in Chapter 2 but might be depend on time here. Then, we apply the method of lines to discretize the temporal space. We subdivide the time interval from t^0 to $t^0{+}T$ into subintervals as follows:

$$\overline{\Sigma} = \{t^0\} \cup (t^0,t^1] \cup \ldots \cup (t^{N-1},t^N]$$

with $t^N = t^0 + T$, $t^{j-1} < t^j$ for $1 \leq j \leq N$ and we denote the time step by $\Delta t^j = t^{j+1} - t^j$. Then, we apply the one-step-θ, a simple ODE integrator. So, the problem becomes

$$\begin{aligned}\text{Find } u_h^{j+1} \in U_h(t^{j+1}) \text{ s.t.: } &\frac{1}{\Delta t^j}(m_h^{\mathrm{L2}}(u_h^{j+1},v;t^{j+1}) - m_h^{\mathrm{L2}}(u_h^j,v;t^j)) + \\ &\theta r_h^{\mathrm{NLP}}(u_h^{j+1},v;t^{j+1}) + (1-\theta) r_h^{\mathrm{NLP}}(u_h^j,v;t^j) = 0 \quad \forall v \in V_h(t^{j+1}).\end{aligned} \tag{5.6}$$

where $u_h(t) \in U_h(t) = u_{h,g}(t) + V_h(t)$ and the function $u_{h,g}(t)$ satisfied the boundary conditions. The parameter of one-step-θ method could be 0 which implies the explicit Euler method, 1/2 which implies the Crank-Nicolson method, and 1 for the implicit Euler method. In this study, we mainly focus on using the implicit Euler method. Furthermore, we reorder the terms and the problem then reads:

$$\text{Find } u_h^{j+1} \in U_h(t^{j+1}) \text{ s.t.:} \quad r_h^j(u_h^{j+1},v) + s_h^j(v) = 0 \quad \forall v \in V_h(t^{j+1}).$$

where

$$\begin{aligned} r_h^j(u,v) &= \quad m_h^{\mathrm{L2}}(u,v;t^{j+1}) + \Delta t^j r_h^{\mathrm{NLP}}(u,v;t^{j+1}),\\ s_h^j(v) &= -m_h^{\mathrm{L2}}(u_h^j,v;t^j).\end{aligned}$$

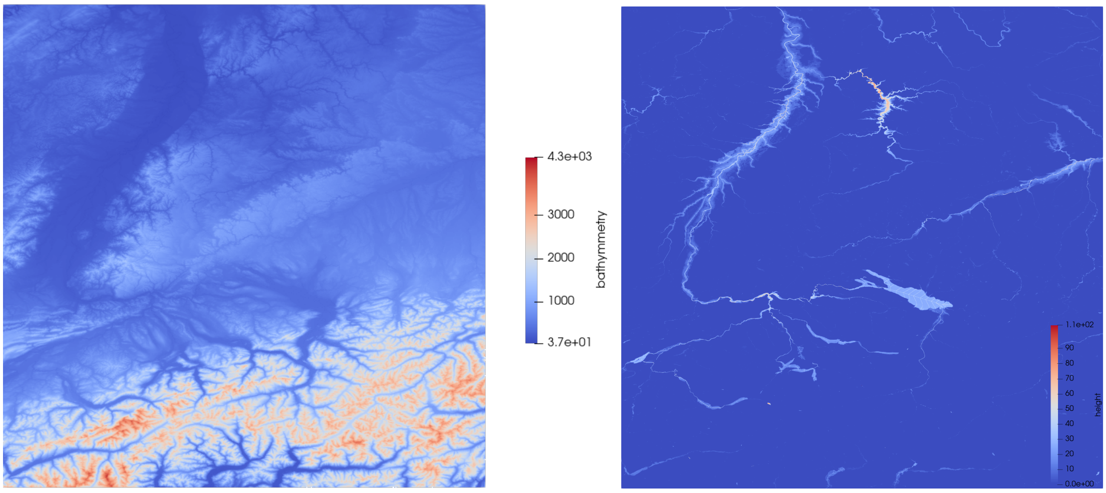


Figure 5.12: Left: Bathymetry information of Baden-Württemberg, Germany. Right: Water height profile after 300 days.

Numerical Results

We apply this model to simulate the shallow water flow in Baden-Württemberg, Germany region. We employ the bathymetry information collected on a scale of 90 × 90 square meters. This model is chosen with parameters $C_M = 1, \alpha_M = 1$, and $\gamma_M = 0.5$. We discretize it by the Finite Volume method which is actually the discontinuous Galerkin method with polynomial degree 0 (piecewise constant) as basis functions. The computational domain is subdivided into 4800×4800 cells. We provide the constant precipitation rate as the rain falls all the time with the rate 2 $l/\text{day} \cdot m^2$. And we set the initial height of water to 0.001 m.

We initiate the time step size to 18,000 s and after some successful steps, we increase the time step size by the factor of $\sqrt{2}$, but if the method fails to solve, we reduce the time step size by the factor of $\sqrt{2}$ as well. We keep maintaining the reduced time step size until some successful steps. We then enable to increase the time step back again. The time step can eventually be up to 86,400 s (1 day) which is our maximum size.

One additional feature could be incorporated here. The local nonlinear problems sometimes cannot be solved with the large time step size. But it does not happen on every subproblem. From the advantage of RASPEN that we solve the subdomain problems locally, we could reduce the time step size only for those subproblems which are failed to solve instead of reducing the global time step size when the method once fails like in the ordinary Newton. Assume that ${}_{j}^{j+1}F_k(\hat{x}_k^{j+1}; x^j)$ is the nonlinear subdomain problems that we want to solve for the time step t^j to t^{j+1}. One can choose n, a number of substeps, to allow the local time split scheme as the following algorithm

Algorithm 13 Solving ${}_{j}^{j+1}F_k(\hat{x}_k^{j+1}; x^j) = 0$ for a given x^j with n substeps

```
init $i = 0, \hat{x}_k^j$
for $i = 1, \ldots, n$ do
    solve ${}_{j+\frac{(i-1)}{n}}^{j+\frac{i}{n}}F_k(\hat{x}_k^{j+\frac{i}{n}}; x^{j+\frac{(i-1)}{n}}) = 0$        // by Algorithm 12
    update $x^{j+\frac{i}{n}}$        $, i = i + 1$
end for
solve ${}_{j}^{j+1}F_k(\hat{x}_k^{j+1}; x^{j+1}) = 0$        // solve the last stage
```

The strategy is that we split the problem into substeps. We then solve the substep problems until it reaches the final stage which is the same as the original problem. The last substep problem would provide a solution at the final stage but it might not be a precise solution for the original system. We should recompute the full step system again with the solution from the last substep as an initial guess before going further to solve the coarse nonlinear problem or global linear system. It is essentially applicable by the nature of RASPEN that we are requested to seek local solutions. With this strategy, RASPEN would have more robustness in terms of time step size.

Note that we do the experiment only on two-level RASPEN with multiplicative linear Nicolaides coarse space (two-level RASPEN) and Newton with GMRES preconditioned with restricted additive Schwarz (two-level Newton). Figure 5.13 illustrates the time step size used in the calculation of the 90-day simulation of shallow water flow in Baden-Württemberg region with the time step size can be increased/decreased by the strategy we mentioned in the previous paragraphs. In practice, if we allow dividing the substeps until RASPEN converges, it would waste a lot of time on the failed time step. Restarting with a smaller time step size might be a better option. Hence, we should set the threshold for maximum substeps allowed for the locally split scheme which we choose 4 and 8 substeps in this experiment but note that these numbers might depend on problems. The top one shows the results of two-level RASPEN with 8-substeps allowed for local problems. The middle one shows the results of two-level RASPEN with 4-substep allowed for local problems. And the bottom one shows the result of two-level Newton. If we allow only 4 substeps on local problems, RASPEN does not perform better than two-level Newton. And if we increase the threshold to 8 substeps, RASPEN can finish the computation in less number of outer iterations than Newton. At the early stage of computation, RASPEN with 8-substeps can comfortably reach the maximum time step size but fall a bit to the stable time step size at 64,094 s while Newton stays around at 36,000-50,192 s. In the end, RASPEN takes 124 time steps for 8-substep and 151 time steps for 4-substep while Newton takes 139 time steps to complete the simulation. In terms of *outer iteration* RASPEN still outperforms like in the stationary problems. It takes an overall 606 (4.88 for each time step), and 612 (4.05 for each time step) outer iterations for 8-substep and 4-substep, respectively. Mean-

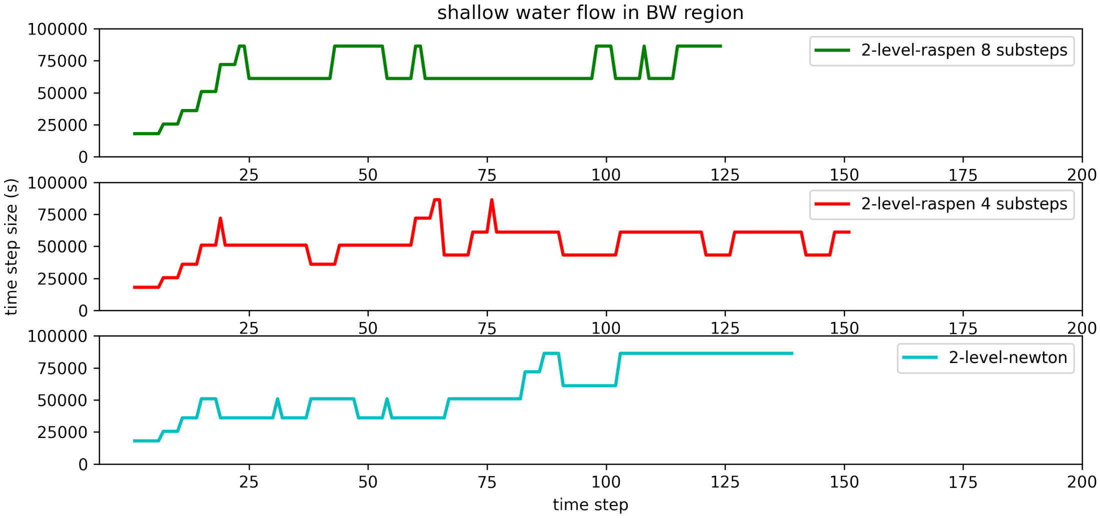


Figure 5.13: Time step size for each outer iteration compared between 2-level RASPEN and 2-level Newton.

while, Newton takes an overall 2,931 (21.09 for each time step) outer iterations. But RASPEN takes longer overall time. It takes 1.0893×10^5 and $8.8105 \times 10^4 s$ for 8-substep and 4-substep, respectively while Newton takes $3.3621 \times 10^4 s$.

5.5 Richards' Equation

Next, we look into a more challenging instationary problem that has nonlinearities on both temporal and spatial derivatives, Richards' equation. It describes the flow in an unsaturated porous medium. Note that the parametrization models are chosen following from [Kle16]. In this study, we focus on the head-based formulation of Richards' equation expressed as

$$\partial_t \theta(\phi) - \nabla \cdot (K\kappa(\phi)(\nabla\phi - \mathbf{g})) = 0 \tag{5.7}$$

where θ is a water content, ϕ is a hydraulic head, K is the conductivity for saturated condition, κ is the relative conductivity and $\mathbf{g}$ is a vector representing the gravitational direction, respectively. Let we define the saturation function Θ which is written as a function of the water content by

$$\Theta = \frac{\theta(\phi) - \theta_r}{\theta_s - \theta_r}. \tag{5.8}$$

where θ_r is the residual water content after complete drainage, and θ_s is the water content under the fully saturated conditions. In this study, the parametrization

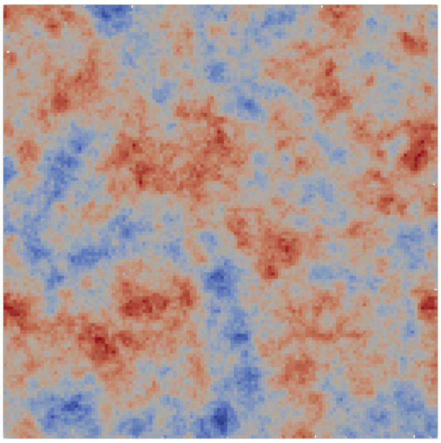


Figure 5.14: An example of conductivity field K generated from dune-randomfield.

model of the saturation Θ is chosen as the van Genuchten model [Gen80] expressed by

$$\Theta(\phi) = [1 + [\alpha|\phi|]^n]^{\frac{1-n}{n}} \tag{5.9}$$

where α, and n are the parameters from the van Genuchten model. Then, the water content relation can be expressed as

$$\theta(\phi) = \theta_r + [1 + [\alpha|\phi|]^n]^{\frac{1-n}{n}} \cdot [\theta_s - \theta_r] \tag{5.10}$$

Next, we exploit the parametrization models combining from Mualem [Mua76] and van Genuchten [Gen80] to determine the relative conductivity κ.

$$\kappa(\phi) = \Theta^\tau \cdot \left[1 - \left[1 - \Theta^{\frac{n}{n-1}}\right]^{\frac{1-n}{n}}\right]^2 \tag{5.11}$$

where τ is the parameter from the Mualem model. We can see that we could replace the saturation by the relation in (5.9). Therefore, it results that

$$\kappa(\phi) = [1 + [\alpha|\phi|]^n]^{\tau \cdot \frac{1-n}{n}} \cdot \left[1 - [\alpha|\phi|]^{n-1} [1 + [\alpha|\phi|]^n]^{\frac{1-n}{n}}\right]^2 \tag{5.12}$$

The conductivity field K is provided by `dune-randomfield`, a DUNE module working on the generator of Gaussian random fields based on the circulant embedding [Kle19].

We consider a square domain $\Omega := [0, 2] \times [0, 2]$ as a computational domain. The equation is discretized by Discontinuous Galerkin with rectangular Q_1 finite elements and we employ the implicit Euler time stepping scheme. The hydraulic head ϕ is initialized by $\phi^{(0)}(x, y) = -y$. The Dirichlet boundary conditions are applied to the top and bottom boundaries with the values are set to be -2 at the top

parameter	description	value
θ_r	residual water content	0.05
θ_s	saturated water content	0.35
α	van Genuchten parameter	3
n	van Genuchten parameter	2
τ	Mualem parameter	0.5
$\mathbf{g}$	gravitational direction	$[0, -1]^T$

Table 5.9: Parameters of Richards' equation used in the numerical experiments.

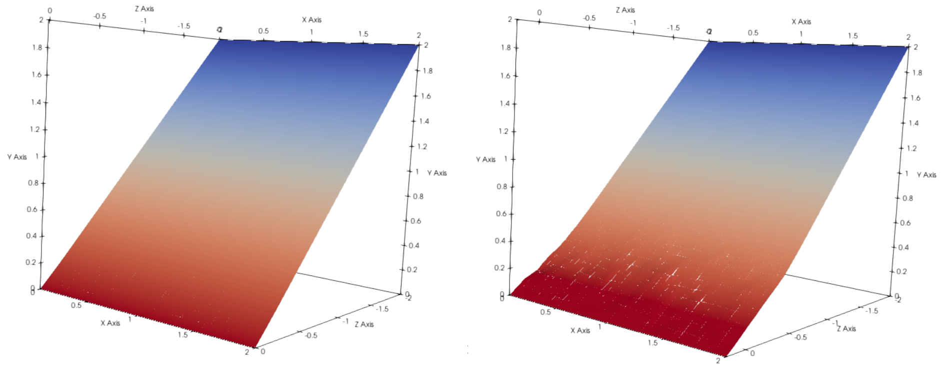


Figure 5.15: The initial head for this experiment (Left) and after slightly increasing in order to let the flow happens (Right).

and 0 at the bottom. The left and right boundaries are equipped with the homogeneous Neumann boundary conditions. With these settings, ϕ is initially imposed to be hydrostatic equilibrium since the gradient of the pressure head cancels out the gravitational vector. Therefore, we slightly increase the Dirichlet boundary value at the bottom from 0 to 0.2 in the first $20s$ so that we can see the flow going upward through the medium. We set the time step size to be $1s$ and run the experiments until $100s$.

Numerical Results

Note that we only consider applying two-level RASPEN with multiplicative linear Nicolaides coarse space (two-level RASPEN) and Newton with GMRES preconditioned with restricted additive Schwarz (two-level Newton) and we turn off the adaptive time step scheme used in the previous experiment. We run the experiments on 256 × 256 cells divided into 64 subdomains with an overlap size is two cells. With our random field generator and the setup parameters presented in Table 5.9, both RASPEN and Newton, in general, cannot solve for every generated random field. We generate a hundred conductivity fields and then apply RASPEN and Newton methods to solve Richards' equation with the conductivity parameter

	seed									
	1	2	3	4	5	6	7	8	9	10
RASPEN	237	221	230	241	243	239	215	223	234	246
Newton	293	283	296	293	306	285	296	290	299	294

	seed									
	11	12	13	14	15	16	17	18	19	20
RASPEN	243	218	235	238	219	233	226	229	226	227
Newton	308	303	293	-	289	296	293	296	291	298

	seed									
	21	22	23	24	25	26	27	28	29	30
RASPEN	235	250	223	232	232	230	235	251	238	238
Newton	289	-	278	299	-	298	295	308	296	289

	seed									
	31	32	33	34	35	36	37	38	39	40
RASPEN	249	247	241	225	235	235	-	232	238	224
Newton	298	308	301	297	295	296	-	289	293	291

	seed									
	41	42	43	44	45	46	47	48	49	50
RASPEN	239	230	258	248	237	233	238	236	236	240
Newton	300	-	297	299	299	294	302	291	295	305

	seed									
	51	52	53	54	55	56	57	58	59	60
RASPEN	234	238	239	241	249	232	245	226	226	239
Newton	289	295	289	304	302	296	301	293	280	293

	seed									
	61	62	63	64	65	66	67	68	69	70
RASPEN	-	232	246	243	226	230	246	235	220	233
Newton	-	292	296	299	301	281	297	295	-	301

	seed									
	71	72	73	74	75	76	77	78	79	80
RASPEN	228	246	235	235	-	229	231	238	244	234
Newton	280	302	297	-	-	299	300	309	297	286

	seed									
	81	82	83	84	85	86	87	88	89	90
RASPEN	240	228	238	238	234	217	231	222	225	-
Newton	312	297	290	302	294	286	288	-	282	-

	seed									
	91	92	93	94	95	96	97	98	99	100
RASPEN	240	239	221	245	238	241	221	239	246	237
Newton	297	303	287	301	293	291	282	294	293	297

Table 5.10: Numbers of outer iterations of RASPEN and Newton on Richards' problem with 100 random conductivity fields.
(-) stands for the solver cannot solve that problem.

K corresponded with those generated fields. The results turn out that RASPEN is able to solve 96 problems. Meanwhile, Newton is able to solve 89 problems. In addition, we observe that there is no case that Newton can solve but RASPEN cannot. That would rather imply that RASPEN has more robustness than Newton. In terms of the number the outer iteration, RASPEN takes 234.88 iterations on average for total outer iterations after 100 time steps while Newton takes 295.13 iterations on average.

5.6 Leaching and Carbonation of Concrete

In this part, we apply RASPEN to the system of PDEs. The problem is adopted and the explanation is excerpted from [TB]. It considers the leaching and carbonation of the concrete. Leaching is the dissolution of minerals forming the concrete. It would increase the porosity of the medium. Carbonation is the reaction of calcium oxide with carbonic acid forming the calcite (calcium carbonate)

$$CaO + H_2CO_3 \rightarrow CaCO_3 + H_2O$$

which would reduce the porosity of the medium. The whole system would consider the air flow in a porous medium, transport of calcium in water and carbon dioxide in air, and chemical reactions - leaching and carbonation - which affect the porosity. Hence, the system has primarily five unknowns: water pressure p_w, air saturation S_a, molar concentration of calcium dissolved in water C_a, concentration of carbon dioxide in air $\bar{c}$, and the porosity ϕ. Then, the system consists of five equations as the followings:

$$\partial_t(\phi S_w \rho_w) + \nabla \cdot q_w = F_w \tag{5.13}$$

$$\partial_t(\phi S_a \rho_a) + \nabla \cdot q_a = F_a \tag{5.14}$$

$$\partial_t(\phi S_w C_a) + \nabla \cdot (\frac{q_w}{\rho_w} C_a - D_w \nabla C_a) = F_{C_a} \tag{5.15}$$

$$\partial_t(\phi S_a \rho_{\bar{c}}) + \nabla \cdot (\frac{q_a}{\rho_a} \rho_{\bar{c}} - D_a \rho_a \nabla \bar{c}) = \bar{F}_{\bar{c}} \tag{5.16}$$

$$\partial_t \phi = R_l \partial_t L_\phi - R_c V_{CaCO_3} \tag{5.17}$$

where ρ_* are densities, F_* are nonlinear source terms, q_* are fluxes, D_* are the diffusion tensors, R_* are reaction rates, L_* are also reaction rates describing the amount of released substances caused by leaching, and V_{CaCO_3} is a constant referred to molar volume of calcite. We go further to explain what each equation represents. The first two equations (5.13) and (5.14) describe the mass balance equations for the water and air phase which q_* represents the flux from Darcy's law

$$q_* = \frac{-K_*(\phi) k_*(S_*)}{\mu_*} \rho_* (\nabla p_* - \rho_* \boldsymbol{g})$$

where μ_* is the viscosity and $\boldsymbol{g}$ is the gravity vector. $K_*(\phi)$ is the intrinsic permeability derived from Kozeny-Carman which is a function of initial porosity ϕ_0 and current porosity ϕ

$$K_*(\phi) = \bar{K}_* \left(\frac{\phi}{\phi_0}\right)^3 \left(\frac{1-\phi_0}{1-\phi}\right)^2$$

$\bar{K}_w$ and $\bar{K}_a$ are constants standing for the initial water and air intrinsic permeability where we set it by $\bar{K}_a = 50\bar{F}_w$. k_* is the relative permeability from van Genuchten and Mualem model [Gen80; Mua76]

$$k_w = S_w^{\frac{1}{2}} \left(1 - \left(1 - S_w^{\frac{1}{m}}\right)^m\right)^2$$

$$k_a = S_a^{\frac{1}{3}} \left(1 - (1-S_a)^{\frac{1}{m}}\right)^{2m}.$$

The water density ρ_w and water viscosity μ_w are set to be constant, but the air density ρ_a, depending on the pressure and the carbon dioxide concentration. The ideal gas law is exploited to evaluate the air density

$$\rho_a = M_a \frac{p_a}{RT}, \qquad M_a = \bar{c}M_{\bar{c}} + (1-\bar{c})M_{\bar{a}}$$

where R is the ideal gas constant, T is temperature, M_a is the molar mass of air including carbon dioxide, $M_{\bar{c}}$ and $M_{\bar{a}}$ are the molar masses of carbon dioxide and air without carbon dioxide. And the air viscosity μ_a is exerted by Wilke's equation

$$\mu_a = \frac{(1-\bar{c})\mu_{\bar{a}}}{(1-\bar{c}) + \bar{c}\varphi_{ac}} + \frac{\bar{c}\mu_{\bar{c}}}{\bar{c} + (1-\bar{c})\varphi_{ca}},$$

$$\varphi_{ac} = \frac{\left(1 + \left(\frac{\mu_{\bar{a}}}{\mu_{\bar{c}}}\right)^{\frac{1}{2}} \left(\frac{M_{\bar{c}}}{M_{\bar{a}}}\right)^{\frac{1}{4}}\right)^2}{\frac{4}{\sqrt{2}}\left(1 + \frac{M_{\bar{a}}}{M_{\bar{c}}}\right)^{\frac{1}{2}}} \quad , \quad \varphi_{ca} = \frac{\left(1 + \left(\frac{\mu_{\bar{c}}}{\mu_{\bar{a}}}\right)^{\frac{1}{2}} \left(\frac{M_{\bar{a}}}{M_{\bar{c}}}\right)^{\frac{1}{4}}\right)^2}{\frac{4}{\sqrt{2}}\left(1 + \frac{M_{\bar{c}}}{M_{\bar{a}}}\right)^{\frac{1}{2}}}$$

where $\mu_{\bar{c}}$ and $\mu_{\bar{a}}$ are the viscosities of gaseous carbon dioxide and air without carbon dioxide, respectively. Then, the source terms F_w and F_a are expressed by

$$\begin{aligned} F_w &= R_l M_w \partial_t L_w \\ F_a &= \qquad\qquad -R_c M_{\bar{c}} \end{aligned}$$

where

$$R_l = \frac{1}{1 + (5S_a)^4}, \qquad R_c = k_{gl}\phi S_w C_a p_a \bar{c}, \qquad k_{gl} = 0.492 S_w^{\frac{3}{2}} \left(1 - S_w^{\frac{3}{2}}\right)$$

We may notice that the unknown p_w is not explicitly written in (5.13). We need two more conditions here. The first is the saturations of water and air are sum up to 1;

$S_w + S_a = 1$. The second is the relation between the saturation and the capillary pressure p_c from the van Genuchten water retention curve

$$p_a = p_c + p_w = \frac{1}{\alpha}\left(S_w^{\frac{1-n}{n}} - 1\right)^{\frac{1}{n}} + p_w$$

where $m = 1 - \frac{1}{n}$.

The next two equations (5.15) and (5.16) are the mass balance equations for calcium in water and carbon dioxide in air. The diffusion tensors D_w is actually the sum of the dispersion D_{ws} and the diffusion D_{wf} and is expressed by $D_w = S_w^{-2} D_{ws} + D_{wf}\boldsymbol{I}$. For D_{ws}, the Scheidegger tensor is applied and its coefficients are

$$(D_{ws})_{ij} = \left[a_T \delta_{ij} \ + \ (a_L - a_T)\,\frac{(v_w)_i (v_w)_j}{|v_w|^2}\right] |v_w|$$

for $i \geq 1, j \leq$ dimension, δ_{ij} is the Kronecker delta, velocity $v_w = q_w/\rho_w$, a_L and a_T are the dispersion coefficients on the longitudinal and traversal direction. We set $a_L = 1$ and $a_T = 1/8$ here. And the diffusion coefficient D_{wf} is defined by

$$D_{wf} = R_c\, 2.3 \times 10^{-13} e^{9.95\phi}.$$

For D_a, we only consider the effect of diffusion and is expressed by

$$D_a = 3 \times 10^{-10}\ \phi^{\frac{4}{3}}\ S_a^{\frac{10}{3}}$$

One might notice again that the unknown $\bar{c}$, the carbon dioxide concentration in air, is not explicitly seen in the temporal derivative of (5.16). We can simply transform the equation by dividing it by $M_{\bar{c}}/RT$. We then apply the Dalton's law: $p_{\bar{c}} = \bar{c} p_a$. Therefore, the equation (5.16) becomes

$$\partial_t(\phi S_a p_a \bar{c}) + \nabla \cdot (\frac{RT}{M_a} q_a \bar{c} - D_a \frac{M_a}{M_{\bar{c}}} p_a \nabla \bar{c}) \ = F_{\bar{c}}. \qquad (5.18)$$

The source terms F_{Ca} and $F_{\bar{c}}$ are expressed by

$$\begin{aligned} F_w &= R_l \partial_t L_{Ca} - \ R_c \\ F_a &= \qquad\qquad - \ R_c RT. \end{aligned}$$

And the last equation (5.17) describes the change of the porosity from the effect of leaching and carbonation.

Numerical Results

The computational domain is rectangular $(0, 1) \times (0, 0.5)$ cm. The top, left, and bottom boundaries are imposed with the homogeneous Neumann conditions. On the right boundary, we impose the homogeneous Neumann boundary only for calcium. The other variables are assigned by these following Dirichlet values, $S_a =$

$2.28, p_a = 0, \bar{c} = 0.5$. For the initial conditions, we set $S_a = 0.151, p_a =$ atmospheric pressure $= 101,325$ Pa, $\bar{c} = 0, \phi = 0.094$, and $Ca = 22$. The constants used in this experiment are provided in Table 5.11. The mesh is rectangular and axiparallel but not equidistant in x-direction. The closer to the right boundary, the finer the mesh (in x-axis). The image of the mesh is shown in Figure 5.16.

The system is discretized in space by the vertex-centered finite volume scheme and is discretized in time by the implicit Euler scheme. We employ the iterative operator splitting to solve the system. It creates two subsystems. One contains two variables: water pressure p_w and air saturation S_a. This part is called *flow part*. Another one contains the rest of variables: calcium Ca, carbon dioxide $\bar{c}$, and porosity ϕ. This part is called *chemical part*.

We set the maximum iterative step to 10. If it does not converges in ten steps, we restart with the smaller time step size, or if the results provide the unphysical values, we restart as well. We set the stopping criterion that the defects of both parts are lower than 10^{-10} or the defect reductions are below 10^{-4}. Note that we measure the defect with different approaches for each part. The Euclidean norm is applied to the flow part defect. But, the maximum norm is applied to the chemical part. These are by the means of the system's behavior. More details on the problem setup can be read in [TB]. They also introduce the *projected* line search to restrict the concentration of calcium and CO_2 to the nonnegative values which should help the convergence issue.

Since the chemical part is reacted pretty locally by its behavior, we believe that the second level would not be ideally helpful to this system so we do not have to add the second level to this part. Therefore, we apply the two-level scheme only on the flow part and the one-level scheme on the chemical part. Another thing to mention is that we turn off the local time split scheme for RASPEN in this experiment.

We run the experiments on 512×128 cells discretized in the manner of Figure 5.16 for 128 subdomains and simulate the reaction until 175 days ($1.512 \times 10^7 s$). We start with the initial time step size of 20 s. There is no maximum time step size

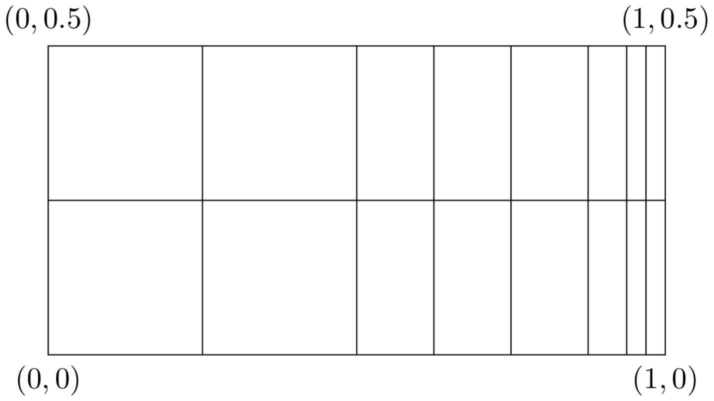


Figure 5.16: Mesh used in this numerical experiment.

parameter	description	value
$\bar{K}_w$	water permeability	1.0038×10^{-20}
$\bar{K}_a$	air permeability	5.019×10^{-19}
n	van Genuchten-Mualem parameter	1.65
α	van Genuchten water retention curve	1.89×10^{-2}
$\boldsymbol{g}$	gravity vector	$[0, -9.81]^T$
ρ_w	water density	998.205
p_{a_0}	atmosheric pressure	101,325
μ_w	viscosity of water	1.002×10^{-3}
$\mu_{\bar{a}}$	viscosity of air	1.8369×10^{-5}
$\mu_{\bar{c}}$	viscosity of CO_2	1.48×10^{-5}
R	ideal gas constant	8.3144598
T	temperature	293.15
ϕ_0	initial porosity	0.094
$M_{\bar{a}}$	molar mass of air without CO_2	28.954×10^{-3}
$M_{\bar{c}}$	molar mass of CO_2	44.01×10^{-3}
M_{CaO}	molar mass of CaO	56.0774×10^{-3}
V_{CaCO_3}	molar volume of calcite	36.92×10^{-6}

Table 5.11: Parameters of leaching and carbonation of concrete used in numerical experiments.

restrained in this experiment. And we keep increasing the time step size for each time step by the factor of 1.25 if the method can solve the system. On the other hand, if the method fails to solve, we decrease the time step size by the factor of 0.75 until it converges. This does not practically work for the real run because there will be so many steps to be failed when the method reaches its critical time step size. But we do it this way because we do not know that in advance and we would like to know how large the time step size each method can handle.

In Figure 5.17 we display the time step size exploited along the simulation in the logarithmic scale for RASPEN and Newton. The results carry out that RASPEN can take a larger time step size compared to Newton. The maximum time step size taken in RASPEN is 439,652 s while in Newton is 180,711 s. Therefore, RASPEN eventually takes only 107 time steps for operator splitting whereas Newton takes 222 time steps to complete the simulation.

We look into the computational time and we find out that RASPEN takes slightly more time than Newton even running fewer time steps. On the grounds that once Newton fails, it usually happens on solving subsystem problems, then it restarts with a reduced time step. But RASPEN typically solves subsystem problems pretty well so that it fails on exceeding the operator splitting iterations which means it would take more time to reach the failure. We observe another point that RASPEN takes less time on solving the flow subsystem than Newton takes but vice versa on the chemical subsystem. Then, one might couple RASPEN-Newton to solve

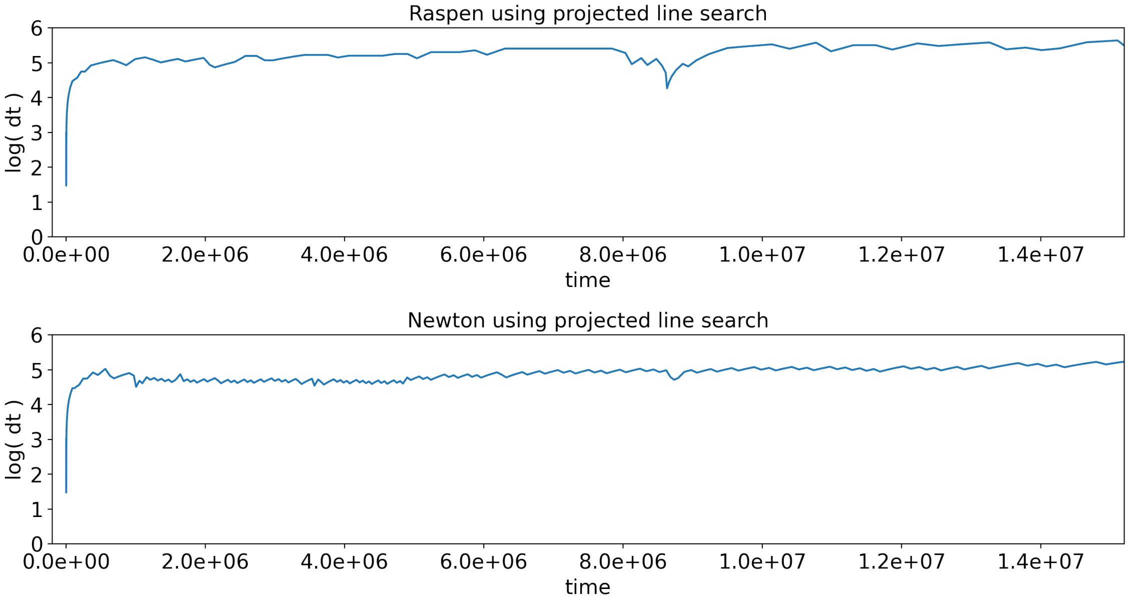


Figure 5.17: Time step size plotted in logarithmic scale along the simulation time.

the system by using RASPEN on the flow part and Newton on the chemical part.

Table 5.12 shows the results from 32 subdomain case with 256 × 64 cells when we apply the combinations of solvers to each subsystem problem and investigate the computational time and behavior of the solvers. Note that the computational time includes the failed steps from both solvers and the operator splitting part. We set the maximum iteration for operator splitting to 10 and do not set the maximum time step size like in the previous experiment. It can be seen that the combination of RASPEN on the flow part and Newton on the chemical part is the best one since it takes less computational time than any other combinations and can handle a larger time step size. And in the last columns show that the failure occurs less often compared to other combinations.

solver		computational time			op. split		failure	
flow	chem	flow (s)	chem (s)	total (s)	itr.	max (s)	solver	op. sp.
RASPEN	RASPEN	3.39e2	8.97e2	1.23e3	95	7.42e5	47	34
RASPEN	Newton	2.52e2	6.25e2	8.77e2	92	8.23e5	37	36
Newton	RASPEN	5.58e2	6.60e2	1.21e3	148	2.62e5	141	32
Newton	Newton	5.59e2	6.64e2	1.22e3	170	2.81e5	179	26

Table 5.12: Results of the combinations of solvers.

6 Conclusion

6.1 Summary

In this thesis, we have studied the Restricted Additive Schwarz Preconditioned Exact Newton (RASPEN), a nonlinear preconditioning. We have described RASPEN method and have discussed its two-level extension which is based on the Nicolaides coarse space. Then the overview of our main software used in this study and the implementation details of RASPEN were explained including the parallel computations of coarse problem setup. The flexibility of RASPEN has been shown by applying it to various applications. It has worked successfully well with different discretization schemes such as Finite Element, Discontinuous Galerkin, Cell-centered Finite Volume, or Vertex-centered Finite Volume.

We have also applied RASPEN to many types of problems. We started with the scalar-value problem, then stepped further to instationary problems. And lastly, we have tried on the complex system of PDEs. RASPEN has shown its competence in solving these problems. The robustness of RASPEN has been displayed by the p-laplace problem where we were able to increase the nonlinearity of the problem. We saw that RASPEN has managed to tackle higher nonlinearity problems compared to the ordinary Newton. We also presented that our two-level approach can be plugged in on both RASPEN and Newton. In Newton, we can apply the restricted additive Schwarz method plus our extended Nicolaides coarse space as a preconditioner of GMRES. The results have shown that RASPEN with our coarse space as well as Newton with our coarse space has overcome the scalability concern on the single level scheme. The number of *global linear* iterations when GMRES was applied to solve the tangential system does not depend on the number of subdomains anymore. Therefore, the computation time was greatly reduced for the large-scale run. On the flow problem in a porous medium where we generated random conductivity fields, RASPEN has presented that it has more robustness due to the fact that it can handle more random fields compared to Newton. The last experiment was on reactive two-phase flow in the porous medium which was prescribed by the complex system of PDEs. RASPEN has also performed well from the results that it can take the larger time step sizes compared to Newton.

One thing clearly to be seen is that RASPEN has converged to the solution faster in terms of *outer* iteration compared to Newton. This has been shown in every case we have experimented with that RASPEN has taken much less *outer* iterations than Newton. Therefore, we could say that RASPEN is a good *nonlinear preconditioner*. But it has also a downside, that is, in each *outer* iteration it needs a fairly additional amount of operations. That makes RASPEN usually runs longer than Newton to complete the simulations. Anyhow, we have shown some cases where RASPEN shines on both the number of iterations and computational time aspects like in the p-Laplace problem or in the *flow* subsystem of the reactive two-phase flow.

Furthermore, by the nature of RASPEN that it seeks for the subdomain solutions, we have proposed an additional feature that for instationary problems we could reduce the time step size locally on some subdomain problems which are more difficult to solve. It is helpful because if Newton failed to solve, it has to reduce the global time step size but RASPEN can reduce the time step size and solve the substep problems locally until it reached the same final stage. It eventually allowed us to take the larger global time step sizes as shown in the diffusive wave approximation example.

6.2 Future Works

We exploit `MPI` to distribute the jobs to computers. It assigns the jobs at the beginning of the computation. When some subdomain problems have higher nonlinearity, it is harder to solve and would take more time than the others. And when one turns on the oversubscription, the situation might get worse if they cluster the complicated problems together on the same computer. In general, we would not know in advance when and where the problems have more difficulties, otherwise, we can easily balance the workload. But in some cases, the problematic regions could change over time. Then assigning the jobs to computers beforehand is not a good strategy. It would be great if we could redistribute the jobs when we find out that one takes more time than the others. It is the idea of *dynamic loadbalancing.* And with oversubscription, we can set up the cluster in such a way that each computer should take the same amount of time to finish the jobs. This strategy should minimize the idle time of our parallel computer. Hence, it can run at full performance. We could also use

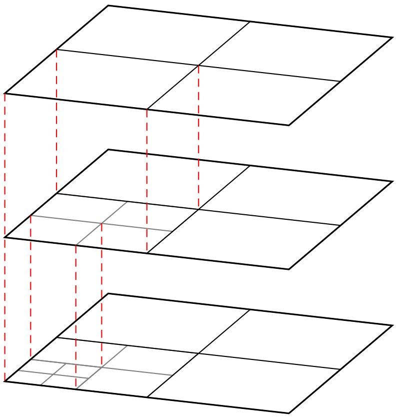


Figure 6.1: Recursive application of RASPEN

the information from the previous time step to decide how we should cluster the problems in the current time step. But it should be noted that this is not an easy issue on `MPI`.

From our observation, solving the local nonlinear problems in RASPEN using the ordinary Newton method is a major part when RASPEN fails. But we have shown that RASPEN is quite more robust. One might possibly apply RASPEN to those subdomain problems that Newton cannot solve. It means that we can apply RASPEN recursively. It would end up with the recursive application of RASPEN. For example, we start to decompose the considered domain to a number of subdomains so that we can apply RASPEN. We seek each subdomain solution in each iteration and in order to find each solution, one can consider decomposing the subdomain once more to a number of sub-subdomains. We then apply RASPEN to those sub-subdomains. It is not necessary to do it on all subdomains. One can select only the subdomains which might have more nonlinearity than any other subdomains. But again the issue is that we might not know that in advance. The *dynamic loadbalancing* might resolve this issue too.

Appendix

List of Figures

List of Tables

List of Algorithms

Bibliography

[ABF00] D. Arnold, D. Boffi, and R. Falk. "Approximation by quadrilateral finite elements". In: *Mathematics of Computation* 71 (June 2000). DOI: 10.1090/S0025-5718-02-01439-4.

[AL83] H. W. Alt and S. Luckhaus. "Quasilinear elliptic-parabolic differential equations". In: *Mathematische Zeitschrift* 183 (1983), pp. 311–341.

[AMD21] AMD, Inc. *AMD EPYC™ 7713*. 2021. URL: https://www.amd.com/en/products/cpu/amd-epyc-7713 (visited on 12/09/2021).

[Ame+01] P. Amestoy et al. "A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling". In: *SIAM Journal on Matrix Analysis and Applications* 23.1 (2001), pp. 15–41.

[Ame+19] P. Amestoy et al. "Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures". In: *ACM Transactions on Mathematical Software* 45 (1 2019), 2:1–2:26.

[Arm66] L. Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives". In: *Pacific Journal of Mathematics* 16 (Jan. 1966), pp. 1–3.

[Arn+02] D. N. Arnold et al. "Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems". In: *SIAM Journal on Numerical Analysis* 39.5 (2002), pp. 1749–1779. DOI: 10.1137/S0036142901384162. URL: https://doi.org/10.1137/S0036142901384162.

[Bar+94] R. Barrett et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics, 1994. DOI: 10.1137/1.9781611971538. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611971538. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611971538.

[Bas+08a] P. Bastian et al. "A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework". In: *Computing* 82.2 (July 2008), pp. 103–119.

[Bas+08b] P. Bastian et al. "A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE". In: *Computing* 82.2 (July 2008), pp. 121–138.

[Bas17] P. Bastian. *Lecture Notes on Scientific Computing with Partial Differential Equations*. 2017. URL: https://conan.iwr.uni-heidelberg.de/teaching/finiteelements_ws2017.

[Bas20] P. Bastian. *Lecture Notes on Parallel Solvers for Finite Elements*. 2020. URL: https://conan.iwr.uni-heidelberg.de/teaching/parsolve_ws2020.

[Bas99] P. Bastian. “Numerical Computation of multiphase flow in porous media”. Habilitationsschrift. 1999.

[BB07] M. Blatt and P. Bastian. “The Iterative Solver Template Library”. In: *Applied Parallel Computing – State of the Art in Scientific Computing*. Ed. by B. Kagström et al. Berlin/Heidelberg: Springer, 2007, pp. 666–675.

[BBG09] F. Bordeu, P.-A. Boucard, and P. Gosselet. “Balancing Domain Decomposition with Nonlinear Relocalization: Parallel Implementation for Laminates”. In: *First international conference on parallel, distributed and grid computing for engineering*. Ed. by B. H. V. Topping and P. Iványi. Stirlingshire, UK: Civil-Comp Press, Apr. 2009.

[BHM10] P. Bastian, F. Heimann, and S. Marnach. “Generic implementation of finite element methods in the Distributed and Unified Numerics Environment (DUNE)”. In: *Kybernetika* 46 (2010), pp. 294–315.

[Bra07] D. Braess. *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*. 3rd ed. Cambridge University Press, 2007. DOI: 10.1017/CBO9780511618635.

[Cai+94] X.-C. Cai et al. “Newton-Krylov-Schwarz Methods in CFD”. In: *Numerical methods for the Navier-Stokes equations: Proceedings of the International Workshop Held at Heidelberg, October 25–28, 1993*. Ed. by F.-K. Hebeker, R. Rannacher, and G. Wittum. Wiesbaden: Vieweg+Teubner Verlag, 1994, pp. 17–30. ISBN: 978-3-663-14007-8.

[Cai+98] X.-C. Cai et al. “Parallel Newton–Krylov–Schwarz Algorithms for the Transonic Full Potential Equation”. In: *SIAM Journal on Scientific Computing* 19.1 (1998), pp. 246–265.

[Cha+21] F. Chaouqui et al. *Linear and nonlinear substructured Restricted Additive Schwarz iterations and preconditioning*. 2021. arXiv: 2103.16999 [math.NA].

[CK02] X.-C. Cai and D. Keyes. “Nonlinearly Preconditioned Inexact Newton Algorithms”. In: *SIAM Journal on Scientific Computing* 24.1 (2002), pp. 183–200.

[CKM02] X.-C. Cai, D. E. Keyes, and L. Marcinkowski. “Non-linear additive Schwarz preconditioners and application in computational fluid dynamics”. In: *International Journal for Numerical Methods in Fluids* 40.12 (2002), pp. 1463–1470.

[CKY01] X.-C. Cai, D. E. Keyes, and D. P. Young. “A Nonlinear Additive Schwarz Preconditioned Inexact Newton Method for Shocked Duct Flow”. In: *Proceedings of the 13th International Conference on Domain Decomposition Methods*. 2001, pp. 343–350.

[CS99] X.-C. Cai and M. Sarkis. “A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems”. In: *SIAM Journal on Scientific Computing* 21.2 (1999), pp. 792–797. DOI: 10.1137/S106482759732678X. eprint: https://doi.org/10.1137/S106482759732678X. URL: https://doi.org/10.1137/S106482759732678X.

[CW93] X.-C. Cai and O. B. Widlund. “Multiplicative Schwarz Algorithms for Some Nonsymmetric and Indefinite Problems”. In: *SIAM Journal on Numerical Analysis* 30.4 (1993), pp. 936–952. DOI: 10.1137/0730049.

[Dav04] T. A. Davis. “Algorithm 832: UMFPACK V4.3—an Unsymmetric-Pattern Multifrontal Method”. In: *ACM Trans. Math. Softw.* 30.2 (June 2004), pp. 196–199. ISSN: 0098-3500. DOI: 10.1145/992200.992206.

[DGL97] J. W. Demmel, J. Gilbert, and X. S. Li. *SuperLU Users” Guide*. Tech. rep. USA, 1997.

[DJN15] V. Dolean, P. Jolivet, and F. Nataf. “An Introduction to Domain Decomposition Methods: algorithms, theory and parallel implementation”. Master. Lecture. France, Jan. 2015. URL: https://hal.archives-ouvertes.fr/cel-01100932.

[Dol+12] V. Dolean et al. “Analysis of a Two-level Schwarz Method with Coarse Spaces Based on Local Dirichlet-to-Neumann Maps”. In: *Computational Methods in Applied Mathematics* 12.4 (2012), pp. 391–414. DOI: doi:10.2478/cmam-2012-0027. URL: https://doi.org/10.2478/cmam-2012-0027.

[Dol+16] V. Dolean et al. “Nonlinear Preconditioning: How to Use a Nonlinear Schwarz Method to Precondition Newton’s Method”. In: *SIAM Journal on Scientific Computing* 38.6 (2016), A3357–A3380.

[Dun21] Dune Course Team. *Dune/PDELab Course*. 2021. URL: https://dune-pdelab-course.readthedocs.io/en/latest/ (visited on 04/21/2021).

[EG04] A. Ern and J.-L. Guermond. *Theory and Practice of Finite Elements*. Appl. Math. Sci. 159, Springer-Verlag, New York, 2004.

[Eva10] L. C. Evans. *Partial differential equations*. Providence, R.I.: American Mathematical Society, 2010.

[Gen80] M. T. van Genuchten. “A Closed-form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils”. In: *Soil Science Society of America* 44.5 (1980), pp. 892–898.

[GT06] L. Giraud and R. Tuminaro. “Algebraic domain decomposition preconditioners”. In: *Mesh partitioning techniques and domain decomposition methods* (2006), pp. 187–216.

[GV13] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013. ISBN: 9781421407944. URL: https://books.google.de/books?id=X5YfsuCWpxMC.

[HC05] F.-N. Hwang and X.-C. Cai. “A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier–Stokes equations”. In: *Journal of Computational Physics* 204.2 (2005), pp. 666–691. ISSN: 0021-9991.

[HL20] A. Heinlein and M. Lanser. “Additive and Hybrid Nonlinear Two-Level Schwarz Methods and Energy Minimizing Coarse Spaces for Unstructured Grids”. In: *SIAM Journal on Scientific Computing* 42.4 (2020), A2461–A2488. DOI: 10.1137/19M1276972. URL: https://doi.org/10.1137/19M1276972.

[HR89] W. Hackbusch and A. Reusken. “Analysis of a damped nonlinear multilevel method”. In: *Numerische Mathematik* 55 (Jan. 1989), pp. 225–246. DOI: 10.1007/BF01406516.

[HS52] M. R. Hestenes and E. Stiefel. “Methods of conjugate gradients for solving linear systems”. In: *Journal of research of the National Bureau of Standards* 49 (1952), pp. 409–435.

[Joh13] V. John. *Numerical Methods for Partial Differential Equations*. URL: https://www.wias-berlin.de/people/john/LEHRE/NUM_PDE_FUB/num_pde_fub.pdf. Last visited on 2021/04/21. 2013.

[Kin86] W. Kinzelbach. *Groundwater modelling : an introduction with sample programs in BASIC*. eng. Developments in water science ; 25. Amsterdam ; Elsevier, 1986. ISBN: 9780080870168.

[Kle16] O. Klein. “Preconditioned and Randomized Methods for Efficient Bayesian Inversion of Large Data Sets and their Application to Flow and Transport in Porous Media”. PhD thesis. Jan. 2016.

[Kle19] O. Klein. *dune-randomfield*. https://gitlab.dune-project.org/oklein/dune-randomfield. 2019.

[KLR14] A. Klawonn, M. Lanser, and O. Rheinbach. “Nonlinear FETI-DP and BDDC Methods”. In: *SIAM Journal on Scientific Computing* 36.2 (2014), A737–A765.

[Lan15] M. Lanser. “Nonlinear FETI-DP and BDDC Methods”. PhD thesis. Sept. 2015.

[LKS13] L. Liu, D. Keyes, and S. Sun. “Fully Implicit Two-phase Reservoir Simulation With the Additive Schwarz Preconditioned Inexact Newton Method”. In: Sept. 2013. ISBN: 9781613992685. DOI: 10.2118/166062-MS.

[MC05] L. Marcinkowski and X.-C. Cai. “Parallel Performance of Some Two-Level ASPIN Algorithms”. In: *Domain Decomposition Methods in Science and Engineering*. Ed. by T. J. Barth et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 639–646. ISBN: 978-3-540-26825-3.

[Mua76] Y. Mualem. “A new model for predicting the hydraulic conductivity of unsaturated porous media”. In: *Water Resources Research* 12.3 (1976), pp. 513–522.

[Nat+11] F. Nataf et al. “A Coarse Space Construction Based on Local Dirichlet to Neumann Maps”. In: *SIAM J. Scientific Computing* 33 (Feb. 2011), pp. 1623–1642.

[Nic87] R. A. Nicolaides. “Deflation of Conjugate Gradients with Applications to Boundary Value Problems”. In: *SIAM Journal on Numerical Analysis* 24.2 (1987), pp. 355–365. ISSN: 00361429. URL: http://www.jstor.org/stable/2157562.

[Nit71] J. Nitsche. “Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind”. In: *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 36 (1971), pp. 9–15.

[Ric31] L. A. Richards. “Capillary Conduction of Liquids Through Porous Mediums”. In: *Physics* 1.5 (1931), pp. 318–333.

[Riv08] B. Riviere. *Discontinuous Galerkin Methods For Solving Elliptic And Parabolic Equations: Theory and Implementation*. Vol. 35. Jan. 2008. DOI: 10.1137/1.9780898717440.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Second. Society for Industrial and Applied Mathematics, 2003. DOI: 10.1137/1.9780898718003. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898718003. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898718003.

[Saa93] Y. Saad. “A Flexible Inner-Outer Preconditioned GMRES Algorithm”. In: *SIAM Journal on Scientific Computing* 14.2 (1993), pp. 461–469. DOI: 10.1137/0914028. eprint: https://doi.org/10.1137/0914028. URL: https://doi.org/10.1137/0914028.

[Sak87] S. Sakaguchi. “Concavity properties of solutions to some degenerate quasilinear elliptic Dirichlet problems”. In: *Annali Della Scuola Normale Superiore Di Pisa-classe Di Scienze* 14 (1987), pp. 403–421.

[San20] O. Sander. *DUNE — The Distributed and Unified Numerics Environment*. Springer International Publishing, 2020. DOI: 10.1007/978-3-030-59702-3. URL: https://www.springer.com/gp/book/9783030597016.

[Sch70] H. A. Schwarz. *Über einen Grenzübergang durch alternirendes Verfahren*. Zürcher u. Furrer, 1870.

[SKN16] J. O. Skogestad, E. Keilegavlen, and J. Nordbotten. "Two-Scale Preconditioning for Two-Phase Nonlinear Flows in Porous Media". In: *Transport in Porous Media* 114 (Sept. 2016). DOI: 10.1007/s11242-015-0587-5.

[Sor82a] D. Sorensen. "Newton's Method with a Model Trust Region Modification". In: *SIAM Journal on Numerical Analysis* 19.2 (1982), pp. 409–426.

[Sor82b] D. C. Sorensen. "Newton's Method with a Model Trust Region Modification". In: *SIAM Journal on Numerical Analysis* 19.2 (1982), pp. 409–426. DOI: 10.1137/0719026.

[Spi+14] N. Spillane et al. "Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps". In: (Jan. 2014).

[SS86] Y. Saad and M. H. Schultz. "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 856–869. DOI: 10.1137/0907058. URL: https://doi.org/10.1137/0907058.

[Tan+09] J. Tang et al. "Comparison of Two-Level Preconditioners Derived from Deflation, Domain Decomposition and Multigrid Methods". In: *Journal of Scientific Computing, 39 (3), 2009* 39 (June 2009). DOI: 10.1007/s10915-009-9272-6.

[TB] M. Tóth and P. Bastian. "Projected Line Search in Newton and Raspen Solvers Applied to Leaching and Carbonation of Concrete". Preprint.

[TW05] A. Toselli and O. Widlund. *Domain Decomposition Methods – Algorithms and Theory*. Vol. 34. Jan. 2005. ISBN: 978-3-540-20696-5. DOI: 10.1007/b137868.

[Vor92] H. A. van der Vorst. "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 13.2 (1992), pp. 631–644. DOI: 10.1137/0913035. URL: https://doi.org/10.1137/0913035.

[WD89] O. Widlund and M. Dryja. *Towards a unified theory of domain decomposition algorithms for elliptic problems*. English (US). Technical Report 486, Ultracomputer Note 167. Department of Computer Science, Courant Institute, Dec. 1989.

[XN14] H. Xiang and F. Nataf. “Two-level algebraic domain decomposition preconditioners using Jacobi–Schwarz smoother and adaptive coarse grid corrections”. In: *Journal of Computational and Applied Mathematics* 261 (2014), pp. 1–13. ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2013.10.027. URL: https://www.sciencedirect.com/science/article/pii/S0377042713005712.

[Yua15] Y.-x. Yuan. “Recent advances in trust region algorithms”. In: *Mathematical Programming* 151 (June 2015). DOI: 10.1007/s10107-015-0893-2.

[Yua94] Y.-x. Yuan. “Trust region algorithms for nonlinear programming”. In: (Jan. 1994). DOI: 10.1090/conm/163/01559.