

Composing Scalable Nonlinear Algebraic Solvers*

Peter R. Brune[†]
Matthew G. Knepley[‡]
Barry F. Smith[†]
Xuemin Tu[§]

Abstract. Most efficient linear solvers use composable algorithmic components, with the most common model being the combination of a Krylov accelerator and one or more preconditioners. A similar set of concepts may be used for nonlinear algebraic systems, where nonlinear composition of different nonlinear solvers may significantly improve the time to solution. We describe the basic concepts of nonlinear composition and preconditioning and present a number of solvers applicable to nonlinear partial differential equations. We have developed a software framework in order to easily explore the possible combinations of solvers. We show that the performance gains from using composed solvers can be substantial compared with gains from standard Newton–Krylov methods.

Key words. iterative solvers, nonlinear problems, parallel computing, preconditioning, software

AMS subject classifications. 65F08, 65Y05, 65Y20, 68W10

DOI. 10.1137/130936725

1. Introduction. Large-scale algebraic solvers for nonlinear partial differential equations (PDEs) are an essential component of modern simulations. Newton–Krylov methods [23] have well-earned dominance: They are generally robust and may be built from preexisting linear solvers and preconditioners, including fast multilevel preconditioners such as multigrid [5, 7, 9, 68, 73] and domain decomposition methods [56, 64, 65, 67]. Newton’s method starts from whole-system linearization. The linearization leads to a large sparse linear system where the matrix may be represented either explicitly by storing the nonzero coefficients or implicitly by various “matrix-free” approaches [11, 42]. However, Newton’s method has a number of drawbacks as a stand-alone solver. The repeated construction and solution of the linearization cause memory bandwidth and communication bottlenecks to come to the fore with regard to performance. Also possible is a lack of convergence robustness when the initial guess is far from the solution. Luckily, a large design space for nonlinear solvers exists to complement, improve, or replace Newton’s method. Only a small part of this space has yet been explored either experimentally or theoretically.

*Received by the editors September 12, 2013; accepted for publication (in revised form) January 7, 2015; published electronically November 5, 2015. This material was based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract DE-AC02-06CH11357.

<http://www.siam.org/journals/sirev/57-4/93672.html>

[†]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 (prbrune@mcs.anl.gov, bsmith@mcs.anl.gov).

[‡]Searle Chemistry Laboratory, Computations Institute, University of Chicago, Chicago, IL 60637 (knepley@ci.uchicago.edu).

[§]Department of Mathematics, University of Kansas, Lawrence, KS 66045 (xtu@math.ku.edu).

In this paper we consider a wide collection of solvers for nonlinear equations. In direct equivalence to the case of linear solvers, we use a small number of algorithmic building blocks to produce a vast array of solvers with different convergence and performance properties. Two related solver techniques, nonlinear composition and preconditioning, are used to construct these solvers. The contributions of this article are twofold: the introduction of a systematic approach to combining nonlinear solvers mathematically and within software, and the demonstration of the construction of efficient solvers for several problems of interest. Implementations of the solvers in this article are available in the PETSc library and may be brought to bear on relevant applications, whether simulated on a laptop or on a supercomputer.

2. Background. We concern ourselves with the solution of nonlinear equations of the form

$$(2.1) \quad \mathbf{F}(\mathbf{x}) = \mathbf{b}$$

for a general discretized nonlinear function $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and right-hand side (RHS) \mathbf{b} . We define the nonlinear residual as

$$(2.2) \quad \mathbf{r}(\mathbf{x}) = \mathbf{F}(\mathbf{x}) - \mathbf{b}.$$

We retain both notations as certain solvers (see section 5.3) require the RHS to be modified. We will use

$$(2.3) \quad \mathbf{J}(\mathbf{x}) = \frac{\partial \mathbf{F}(\mathbf{x})}{\partial \mathbf{x}}$$

to denote the Jacobian of $\mathbf{F}(\mathbf{x})$.

The linear system

$$\mathbf{Ax} = \mathbf{b}$$

with residual

$$\mathbf{r}(\mathbf{x}) = \mathbf{Ax} - \mathbf{b}$$

is an important special case from which we can derive valuable insight into solvers for the nonlinear problem. Stationary solvers for linear systems repeatedly apply a linear operator in order to progressively improve the solution. The application of a linear stationary solver by defect correction may be written as

$$(2.4) \quad \mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{P}^{-1}(\mathbf{Ax}_i - \mathbf{b}),$$

where \mathbf{P}^{-1} is a linear operator, called a preconditioner, whose action approximates, in some sense, the inverse of \mathbf{A} . The Jacobi, Gauss–Seidel, and multigrid iterations are all examples of linear stationary solvers.

Composition of linear preconditioners \mathbf{P}^{-1} and \mathbf{Q}^{-1} may proceed in two different ways, producing two new stationary solvers. The first is the additive combination

$$\mathbf{x}_{i+1} = \mathbf{x}_i - (\alpha_{\mathbf{P}}\mathbf{P}^{-1} + \alpha_{\mathbf{Q}}\mathbf{Q}^{-1})(\mathbf{Ax}_i - \mathbf{b}),$$

with weights $\alpha_{\mathbf{P}}$ and $\alpha_{\mathbf{Q}}$. The second is the multiplicative combination

$$\begin{aligned} \mathbf{x}_{i+1/2} &= \mathbf{x}_i - \mathbf{P}^{-1}(\mathbf{Ax}_i - \mathbf{b}), \\ \mathbf{x}_{i+1} &= \mathbf{x}_{i+1/2} - \mathbf{Q}^{-1}(\mathbf{Ax}_{i+1/2} - \mathbf{b}). \end{aligned}$$

Compositions consisting of \mathbf{P}^{-1} and \mathbf{Q}^{-1} are an effective acceleration strategy if \mathbf{P}^{-1} eliminates a portion of the error space and \mathbf{Q}^{-1} handles the rest. A now mature theory for these compositions was developed in the 1980s and 1990s in the context of domain decomposition methods [63, 67].

Linear left- and right-preconditioning, when used in conjunction with Krylov iterative methods [59], is standard practice for the parallel solution of linear systems of equations. We write the use of a linear Krylov method as $\mathbf{K}(\mathbf{A}, \mathbf{x}, \mathbf{b})$, where \mathbf{A} is the matrix, \mathbf{x} the initial solution, and \mathbf{b} the RHS.

Linear left-preconditioning recasts the problem as

$$\mathbf{P}^{-1}(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0,$$

while right-preconditioning takes two stages,

$$\begin{aligned}\mathbf{A}\mathbf{P}^{-1}\mathbf{y} &= \mathbf{b}, \\ \mathbf{P}^{-1}\mathbf{y} &= \mathbf{x},\end{aligned}$$

where one solves for the preconditioned solution \mathbf{y} and then transforms \mathbf{y} using \mathbf{P}^{-1} to the solution of the original problem.

3. Nonlinear Composed Solvers. We take the basic patterns from the previous section and apply them to the nonlinear case. We emphasize that unlike the linear case, the nonlinear case requires the approximate solution as well as the residual to be defined both in the outer solver and in the preconditioner. With this in mind, we will show how composition and preconditioning may be systematically transferred to the nonlinear case.

We use the notation $\mathbf{x}_{i+1} = \mathbf{M}(\mathbf{r}, \mathbf{x}_i)$ for the action of a nonlinear solver. In most cases, but not always, $\mathbf{r} = \mathbf{r}(\mathbf{x}_i)$, that is, \mathbf{r} is simply the most recent residual. It is also useful to consider the action of a solver that is dependent on the previous m approximate solutions and the previous m residuals as $\mathbf{x}_{i+1} = \mathbf{M}(\mathbf{r}, \mathbf{x}_{i-m+1}, \dots, \mathbf{x}_i, \mathbf{r}_{i-m+1}, \dots, \mathbf{r}_i)$. Methods that store and use information from previous iterations such as previous solutions, residuals, or step directions will have those listed in the per-iteration inputs as well.

Nonlinear composition consists of a sequence or series of two (or more) solution methods \mathbf{M} and \mathbf{N} , which both provide an approximate solution to (2.1). Nonlinear preconditioning, on the other hand, may be cast as a modification of the residual \mathbf{r} through application of an inner method \mathbf{N} . The modified residual is then provided to an outer solver \mathbf{M} , which solves the preconditioned system.

An additive composition may be written as

$$(3.1) \quad \mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_{\mathbf{M}}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i) - \mathbf{x}_i) + \alpha_{\mathbf{N}}(\mathbf{N}(\mathbf{r}, \mathbf{x}_i) - \mathbf{x}_i)$$

for weights $\alpha_{\mathbf{M}}$ and $\alpha_{\mathbf{N}}$. The multiplicative composition is

$$(3.2) \quad \mathbf{x}_{i+1} = \mathbf{M}(\mathbf{r}, \mathbf{N}(\mathbf{r}, \mathbf{x}_i)) = \mathbf{M}(\mathbf{r}(\mathbf{N}(\mathbf{r}(\mathbf{x}_i), \mathbf{x}_i)), \mathbf{N}(\mathbf{r}(\mathbf{x}_i), \mathbf{x}_i)),$$

which simply states: update the solution using the current solution and residual with the first solver and then update the solution again using the resulting new solution and new residual with the second solver. Nonlinear left-preconditioning may be directly recast from the linear stationary solver case:

$$(3.3) \quad \mathbf{P}^{-1}(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0,$$

which we can rewrite as a fixed-point problem

$$(3.4) \quad \mathbf{x} - \mathbf{P}^{-1}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{x}$$

with nonlinear analogue

$$(3.5) \quad \mathbf{x} = \mathbf{N}(\mathbf{r}, \mathbf{x}),$$

so the equivalent preconditioned nonlinear problem can be reached by subtracting \mathbf{x} from both sides, giving

$$(3.6) \quad \mathbf{x} - \mathbf{N}(\mathbf{r}, \mathbf{x}) = 0.$$

Thus, the left-preconditioned residual is given by

$$(3.7) \quad \mathbf{r}^l(\mathbf{x}) = \mathbf{x} - \mathbf{N}(\mathbf{r}, \mathbf{x}).$$

Under certain circumstances the recast system will have better conditioning and less severe nonlinearities than the original system. The literature provides several examples of nonlinear left-preconditioning: nonlinear successive overrelaxation (SOR) has been used in place of linear left-applied SOR [19], and additive Schwarz-preconditioned inexact Newton (ASPIN) [16] uses an overlapping nonlinear additive Schwarz method (NASM) to provide the left-preconditioned problem for a Newton's method solver. Walker and Ni's fixed-point preconditioned Anderson mixing [70] uses a similar methodology. Many of these methods or variants thereof are discussed and tested later in this paper.

Nonlinear right-preconditioning¹ involves a different recasting of the residual. This time, we treat the nonlinear solver \mathbf{N} as a nonlinear transformation of the problem to one with solution \mathbf{y} , as $\mathbf{x} = \mathbf{P}^{-1}\mathbf{y}$ is a linear transformation of \mathbf{x} . Accordingly, the nonlinear system in (2.1) can be rewritten as the solution of the system perturbed by the preconditioner

$$(3.8) \quad \mathbf{F}(\mathbf{N}(\mathbf{F}, \mathbf{y})) = \mathbf{b}$$

and the solution by outer solver \mathbf{M} as

$$(3.9) \quad \mathbf{y}_{i+1} = \mathbf{M}(\mathbf{r}(\mathbf{N}(\mathbf{F}, \cdot)), \mathbf{x}_i)$$

followed by

$$\mathbf{x}_{i+1} = \mathbf{N}(\mathbf{r}, \mathbf{y}_{i+1}).$$

Nonlinear right-preconditioning may be interpreted as \mathbf{M} putting preconditioner \mathbf{N} “within striking distance” of the solution. One can solve for \mathbf{y} in (3.8) directly, with an outer solver using residual $\mathbf{r}(\mathbf{N}(\mathbf{r}, \mathbf{x}))$. However, the combination of inner solve and function evaluation is significantly more expensive than computing $\mathbf{F}(\mathbf{x})$ and should be avoided. We will show that when $\mathbf{M}(\mathbf{r}, \mathbf{x})$ is a Newton–Krylov solver, (3.9) is equivalent to (3.2). Considering them to have similar mathematical and algorithmic properties is appropriate in general.

¹Note that nonlinear right-preconditioning is a misnomer because it does not simplify to right linear preconditioning in the linear case; it actually results in the new linear system $A(I - P^{-1}A)y = (I - AP^{-1})b$. However, as with linear right-preconditioning, the inner solver is applied before the function (matrix-vector product in the linear case) evaluation, hence the name.

In the special case of Newton’s method, nonlinear right-preconditioning is referred to by Cai [15] and Cai and Li [17] as nonlinear elimination [45]. The idea behind nonlinear elimination is to use a local nonlinear solver to fully resolve difficult localized nonlinearities when they begin to cause difficulties for the global Newton’s method; see section 4.4. Grid sequencing [41, 66] and pseudotransient [39] continuation methods work by a similar principle, using precursor solves to put Newton’s method at an initial guess from which it has fast convergence. Alternating a global linear or nonlinear step with local nonlinear steps has been studied as the LATIN method [20, 43, 44]. Full approximation scheme (FAS) preconditioned nonlinear GMRES (NGMRES), discussed below, for recirculating flows [53] is another application of nonlinear right-preconditioning, where the FAS iteration is stabilized and accelerated by constructing a combination of several previous FAS iterates.

For solvers based on a search direction, left-preconditioning is much more natural. In fact, general line searches may be expressed as left-preconditioning by the nonlinear Richardson method. Left-preconditioning also has the property that for problems with poorly scaled residuals, the inner solver may provide a tenable search direction when one cannot be found based on the original residual. A major difficulty with the left-preconditioned option is that the function evaluation may be much more expensive. Line searches involving the direct computation of $\mathbf{x} - \mathbf{M}(\mathbf{r}, \mathbf{x})$ at points along the line may be overly expensive given that the computation of the residual now requires nonlinear solves. Line searches over $\mathbf{x} - \mathbf{M}(\mathbf{r}, \mathbf{x})$ may also miss stagnation of weak inner solvers and must be monitored. One may also base the line search on the unpreconditioned residual. The line search based on $\mathbf{x} - \mathbf{M}(\mathbf{r}, \mathbf{x})$ is often recommended [37] and is the “correct” one for general left-preconditioned nonlinear solvers.

Our basic notation for compositions and preconditioning is described in Table 3.1.

Table 3.1 *Nonlinear compositions and preconditioning given outer and inner solvers \mathbf{M} and \mathbf{N} .*

| Composition type | Symbol | Statement | Abbreviation |
|-------------------------------|--------------|--|-------------------------------------|
| Additive composite | + | $\mathbf{x} + \alpha_{\mathbf{M}} (\mathbf{M}(\mathbf{r}, \mathbf{x}) - \mathbf{x}) + \alpha_{\mathbf{N}} (\mathbf{N}(\mathbf{r}, \mathbf{x}) - \mathbf{x})$ | $\mathbf{M} + \mathbf{N}$ |
| Multiplicative composite | * | $\mathbf{M}(\mathbf{r}, \mathbf{N}(\mathbf{r}, \mathbf{x}))$ | $\mathbf{M} * \mathbf{N}$ |
| Left-preconditioning | $-L$ | $\mathbf{M}(\mathbf{x} - \mathbf{N}(\mathbf{r}, \mathbf{x}), \mathbf{x})$ | $\mathbf{M} -L \mathbf{N}$ |
| Right-preconditioning | $-R$ | $\mathbf{M}(\mathbf{r}(\mathbf{N}(\mathbf{r}, \mathbf{x})), \mathbf{x})$ | $\mathbf{M} -R \mathbf{N}$ |
| Inner linearization inversion | \backslash | $\mathbf{y} = \mathbf{J}(\mathbf{x})^{-1} \mathbf{r}(\mathbf{x}) = \mathbf{K}(\mathbf{J}(\mathbf{x}), \mathbf{y}_0, \mathbf{r}(\mathbf{x}))$ | $\text{NEWT} \backslash \mathbf{K}$ |

4. Solvers. We now introduce several algorithms, the details of their implementation and use, and an abstract notion of how they may be composed. We first describe outer solution methods and how composition is applied to them. We then move on to solvers used primarily as inner methods. The distinction is arbitrary but leads to the discussion of decomposition methods in section 5.

4.1. Line Searches. The most popular strategy for increasing robustness or providing globalization in the solution of nonlinear PDEs is the line search. Given a functional $f(\mathbf{x})$, a starting point \mathbf{x}_i , and a direction \mathbf{d} , we compute $\lambda \approx \arg \min_{\mu > 0} f(\mathbf{x}_i + \mu \mathbf{d})$. The functional $f(\cdot)$ may be $\|\mathbf{r}(\cdot)\|_2^2$ or a problem-specific objective function. Theoretical guarantees of convergence may be made for many line search procedures if \mathbf{d} is a descent direction. In practice, many solvers that do not converge when only full or simply damped steps are used converge well when combined with a line search. Different variants of line searches are appropriate for different solvers and we may organize them into two groups based on a single choice taken in the algorithm: whether or not the full step is likely to be sufficient (for example, with Newton’s method near

the solution). If it is likely to be sufficient, the algorithm should default to taking the full step in a performance-transparent way. If not, and some shortening or lengthening of the step is assumed to be required, the line search begins from the premise that it must determine this scaling factor.

In the numerical solution of nonlinear PDEs, we generally want to guarantee progress in the minimization of $f(\mathbf{x}) = \|\mathbf{r}\|_2^2$ at each stage. However, $\nabla\|\mathbf{r}\|_2^2$ is not \mathbf{r} , but instead $2\mathbf{J}^\top(\mathbf{x})\mathbf{r}(\mathbf{x})$. With Newton's method, the Jacobian has been computed and iteratively inverted before the line search and may be used in the Jacobian-vector product. Also note that the transpose product is not required for the line search, since the slope of $f(\mathbf{x})$ in the direction of step \mathbf{y} may be expressed as the scalar quantity $s = \mathbf{r}(\mathbf{x})^\top(\mathbf{J}(\mathbf{x})\mathbf{y})$. A cubic backtracking (BT) line search as described in Dennis and Schnabel [25] is used in conjunction with methods based on Newton's method in this work. The one modification necessary is that it is modified to act on the optimization problem arising from $\|\mathbf{r}(\mathbf{x})\|_2^2$. BT defaults to taking the full step if that step is sufficient with respect to the Wolfe conditions [74], and it does no more work unless necessary. The BT line search may stagnate entirely for ill-conditioned Jacobians [69]. For a general step BT is not appropriate for a few reasons. First, steps arising from methods other than Newton's method are more likely to be ill-scaled, and as such the assumption that the full step is appropriate is generally invalid. For this reason we only use BT in the case of Newton's method. Second, there are also cases where we lose many of the other assumptions that make BT appropriate. Most of the algorithms described here, for example, do not necessarily assemble the Jacobian and require many more iterations than does Newton's method. Jacobian assembly, just to perform the line search, would become extremely burdensome in these cases. This situation removes any possibility of using many of the technologies we could potentially import from optimization, including safeguards and guarantees of minimization. However, in many cases one can still assume that the problem has optimization-like qualities. Suppose that $\mathbf{r}(\mathbf{x})$ is the gradient of some (hidden) objective functional $f(\mathbf{x})$ instead of $\|\mathbf{r}\|_2^2$. Trivially, one may minimize the hidden $f(\mathbf{x} + \lambda\mathbf{y})$ by finding its critical points, which are roots of

$$\mathbf{y}^\top \mathbf{r}(\mathbf{x} + \lambda\mathbf{y}) = \frac{df(\mathbf{x} + \lambda\mathbf{y})}{d\lambda},$$

using a secant method. The resulting critical point (CP) line search is outlined in Algorithm 1.

ALGORITHM 1. CP Line Search.

```

1: procedure CP( $\mathbf{r}, \mathbf{y}, \lambda_0, n$ )
2:    $\lambda_{-1} = 0$ 
3:   for  $i = 0$  do  $n - 1$ 
4:      $\lambda_{i+1} = \lambda_i - \frac{\mathbf{y}^\top \mathbf{r}(\mathbf{x} + \lambda_i \mathbf{y})(\lambda_i - \lambda_{i-1})}{\mathbf{y}^\top \mathbf{r}(\mathbf{x} + \lambda_i \mathbf{y}) - \mathbf{y}^\top \mathbf{r}(\mathbf{x} + \lambda_{i-1} \mathbf{y})}$ 
5:   end for
6: return  $\lambda_n$ 

```

CP differs from BT in that it will not minimize $\|\mathbf{r}(\mathbf{x} + \lambda\mathbf{y})\|_2$ over λ . However, CP shows much more rapid convergence than does L2 (see Algorithm 2) for certain solvers and problems. Both line searches may be started from λ_0 as an arbitrary or problem-dependent damping parameter, and $\lambda_{-1} = 0$. In practice, one iteration is usually satisfactory. For highly nonlinear problems, however, overall convergence may be

accelerated by a more exact line search corresponding to a small number of iterations. CP has also been suggested for nonlinear conjugate gradient (NCG) methods [62].

If our assumption of optimization-like qualities in the problem becomes invalid, then we can do little besides attempt to find a minimum of $\|\mathbf{r}(\mathbf{x} + \lambda\mathbf{y})\|_2$. An iterative secant search for a minimum value of $\|\mathbf{r}(\mathbf{x} + \lambda\mathbf{y})\|_2$ is defined in Algorithm 2.

ALGORITHM 2. L2 Line Search.

```

1: procedure L2( $\mathbf{r}, \mathbf{y}, \lambda_0, n$ )
2:    $\lambda_{-1} = 0$ 
3:   for  $i = 0$  do  $n - 1$ 
4:      $\nabla_{\mathbf{y}}\|\mathbf{r}(\mathbf{x} + \lambda_i\mathbf{y})\|_2^2 = \frac{3\|\mathbf{r}(\mathbf{x} + \lambda_i\mathbf{y})\|_2^2 - 4\|\mathbf{r}(\mathbf{x} + \frac{1}{2}(\lambda_i + \lambda_{i-1})\mathbf{y})\|_2^2 + \|\mathbf{r}(\mathbf{x} + \lambda_{i-1}\mathbf{y})\|_2^2}{(\lambda_i - \lambda_{i-1})}$ 
5:      $\nabla_{\mathbf{y}}\|\mathbf{r}(\mathbf{x} + \lambda_{i-1}\mathbf{y})\|_2^2 = \frac{\|\mathbf{r}(\mathbf{x} + \lambda_i\mathbf{y})\|_2^2 - 4\|\mathbf{r}(\mathbf{x} + \frac{1}{2}(\lambda_i + \lambda_{i-1})\mathbf{y})\|_2^2 + 3\|\mathbf{r}(\mathbf{x} + \lambda_{i-1}\mathbf{y})\|_2^2}{(\lambda_i - \lambda_{i-1})}$ 
6:      $\lambda_{i+1} = \lambda_i - \frac{\nabla_{\mathbf{y}}\|\mathbf{r}(\mathbf{x} + \lambda_i\mathbf{y})\|_2^2(\lambda_i - \lambda_{i-1})}{\nabla_{\mathbf{y}}\|\mathbf{r}(\mathbf{x} + \lambda_i\mathbf{y})\|_2^2 - \nabla_{\mathbf{y}}\|\mathbf{r}(\mathbf{x} + \lambda_{i-1}\mathbf{y})\|_2^2}$ 
7:   end for
8: return  $\lambda_n$ 

```

When converged, L2 is equivalent to an optimal damping in the direction of the residual and will forestall divergence. $\nabla_{\mathbf{y}}$ is calculated by polynomial approximation, requiring two additional residual evaluations per application. In practice and in our numerical experiments the number of inner iterations n is 1.

4.2. Nonlinear Richardson (NRICH). The nonlinear analogue to the Richardson iteration is merely the simple application of a line search. NRICH takes a step in the negative residual direction and scales that step sufficiently to guarantee convergence. NRICH is known as steepest descent [32] in the optimization context, where \mathbf{r} is the gradient of a functional $f(\cdot)$ noted above. NRICH is outlined in Algorithm 3.

ALGORITHM 3. NRICH Iteration.

```

1: procedure NRICH( $\mathbf{r}, \mathbf{x}_i$ )
2:    $\mathbf{d} = -\mathbf{r}(\mathbf{x}_i)$ 
3:    $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda\mathbf{d}$   $\triangleright \lambda$  determined by line search
4: return  $\mathbf{x}_{i+1}$ 

```

NRICH is often slow to converge for general problems and stagnates quickly. However, different step directions than \mathbf{r} may be generated by nonlinear preconditioning and can improve convergence dramatically.

ALGORITHM 4. NRICH Iteration: Left-Preconditioned by $\mathbf{M}(\cdot)$.

```

1: procedure NRICH( $\mathbf{x} - \mathbf{M}(\mathbf{r}, \mathbf{x}), \mathbf{x}_i$ )
2:    $\mathbf{d} = \mathbf{M}(\mathbf{r}, \mathbf{x}_i) - \mathbf{x}_i$ 
3:    $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda\mathbf{d}$   $\triangleright \lambda$  determined by line search
4: return  $\mathbf{x}_{i+1}$ 

```

As shown in Algorithm 4, we replace the original residual equation $\mathbf{r}(\mathbf{x})$ with $\mathbf{x} - \mathbf{M}(\mathbf{r}, \mathbf{x})$ and apply NRICH to the new problem. There are two choices for \mathbf{r} in the line search. The first is based on minimizing the original residual, as in the unpreconditioned case; the second minimizes the norm of the preconditioned residual

instead. Minimizing the unpreconditioned residual with a preconditioned step is more likely to stagnate, as there is no guarantee that the preconditioned step is a descent direction with respect to the gradient of $\|\mathbf{r}(\mathbf{x})\|_2^2$.

4.3. Anderson Mixing (ANDERSON). ANDERSON [2] constructs a new approximate solution as a combination of several previous approximate solutions and a new trial.

ALGORITHM 5. ANDERSON.

- 1: **procedure** ANDERSON($\mathbf{r}, \mathbf{x}_i \cdots \mathbf{x}_{i-m+1}$)
 - 2: $\mathbf{x}_i^M = \mathbf{x}_i + \lambda \mathbf{r}(\mathbf{x}_i)$
 - 3: **minimize** $\left\| \mathbf{r} \left(\left(1 - \sum_{k=i-m}^{i-1} \alpha_k \right) \mathbf{x}_i^M + \sum_{k=i-m}^{i-1} \alpha_k \mathbf{x}_k^M \right) \right\|_2$ over $\{\alpha_{i-m} \cdots \alpha_i\}$
 - 4: $\mathbf{x}_{i+1} = \left(1 - \sum_{k=i-m}^i \alpha_k \right) \mathbf{x}_i^M + \sum_{k=i-m}^{i-1} \alpha_k \mathbf{x}_k^M$
 - 5: **return** \mathbf{x}_{i+1}
-

In practice, the nonlinear minimization problem in Algorithm 5 is simplified. The α_i are computed by considering the linearization

$$(4.1) \quad \mathbf{r} \left(\left(1 - \sum_{k=i-m}^{i-1} \alpha_k \right) \mathbf{x}_i^M + \sum_{k=i-m}^{i-1} \alpha_k \mathbf{x}_k^M \right) \approx \left(1 - \sum_{k=i-m}^{i-1} \alpha_k \right) \mathbf{r}(\mathbf{x}_i^M) + \sum_{k=i-m}^{i-1} \alpha_k \mathbf{r}(\mathbf{x}_k^M)$$

and solving the related linear least-squares problem

$$(4.2) \quad [\mathbf{r}(\mathbf{x}_j) - \mathbf{r}(\mathbf{x}^M)]^\top [\mathbf{r}(\mathbf{x}_i) - \mathbf{r}(\mathbf{x}^M)] \alpha_j = \mathbf{r}(\mathbf{x}_i)^\top [\mathbf{r}(\mathbf{x}_i) - \mathbf{r}(\mathbf{x}^M)].$$

ANDERSON solves (4.2) by dense factorization. The work presented here uses the SVD-based least-squares solve from LAPACK in order to allow for potential singularity of the system arising from stagnation to be detected outside of the subsystem solve. A number of related methods fall under the broad category of nonlinear series accelerator methods. These include ANDERSON as stated above. NGMRES [72] is a variant that includes conditions to avoid stagnation and direct inversion in the iterative subspace (DIIS) [55], which formulates the minimization problem in an alternative fashion. Both right- and left-preconditioning can be applied to ANDERSON. With right-preconditioning, the computation of $\mathbf{x}_i^M = \mathbf{x}_{i-1} + \lambda \mathbf{d}$ is replaced with $\mathbf{x}^M = \mathbf{M}(\mathbf{r}, \mathbf{x}_{i-1})$ and $\mathbf{r}(\mathbf{x}_i)$ with $\mathbf{r}(\mathbf{x}^M)$. This incurs an extra function evaluation, although this is usually included in the nonlinear preconditioner application. Left-preconditioning can be applied by replacing \mathbf{r} with \mathbf{r}^l . Right-preconditioned NGMRES has been applied for recirculating flows [53] using FAS and in stabilizing lagged Newton's methods [18, 60]. Simple preconditioning of NGMRES has also been proposed in the optimization context [22]. Preconditioned ANDERSON has been leveraged as an outer accelerator for the Picard iteration [47, 71].

We can use the same formulation expressed in (4.2) to determine α_M and α_N (or any number of weights) in (3.1), using the solutions and final residuals from a series of inner nonlinear solvers instead of the sequence of previous solutions. This sort of residual-minimizing technique is generally applicable in additive compositions of solvers. All instances of additive composition in the experiments presented here use this formulation.

4.4. Newton–Krylov Methods (NEWT\K). In NEWT\K methods, the search direction is determined by inexact iterative inversion of the Jacobian applied to the residual by using a preconditioned Krylov method.

ALGORITHM 6. NEWT\K.

- 1: **procedure** NEWT\K(\mathbf{r}, \mathbf{x}_i)
 - 2: $\mathbf{d} = \mathbf{J}(\mathbf{x}_i)^{-1}\mathbf{r}(\mathbf{x}_i)$ \triangleright approximate inversion by Krylov method
 - 3: $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda\mathbf{d}$ \triangleright λ determined by line search
 - 4: **return** \mathbf{x}_{i+1}
-

NEWT\K cannot be guaranteed to converge far away from the solution, and it is routinely enhanced by a line search. In our formalism, NEWT\K with a line search can be expressed as NRICH left-preconditioned by NEWT\K coming from Algorithm 4. However, we will consider line search globalized NEWT\K as the standard and will omit the outer NRICH when using it, leading to Algorithm 6. Other globalizations, such as trust-region methods, are also often used in the context of optimization but will not be covered here.

NEWT\K is the general-purpose workhorse of a majority of simulations requiring solution of nonlinear equations. Numerous implementations and variants exist, both in the literature and as software [12, 27]. The general organization of the components of NEWT\K is shown in Figure 4.1. Note that the vast majority of potential customizations occurs at the level of the linear preconditioner and that access to fast solvers is limited to that step.

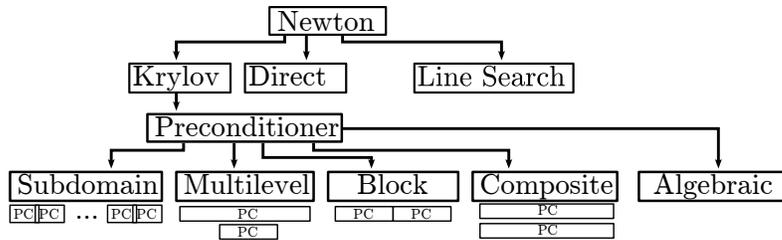


Fig. 4.1 Organization of a Newton–Krylov solver with potential points of customization. Although one can alter the type, parameters, or tolerances of Newton’s method, the Krylov solver, or the line search, we see that the majority of the potential customization and composition occurs at the level of the preconditioner. Possibilities include the use of algebraic preconditioners, such as direct solvers or sparse factorizations, or domain decomposition, multigrid, or composite solvers, which all have subcomponent solvers and preconditioners. These divisions may either be by subdomain, as with the additive Schwarz methods, or by physics field components, as with block preconditioners (for example, one block for pressures and one for velocities).

Nonlinear right-preconditioning of Newton’s method can take two forms. We describe our choice and comment on the alternative. At first glance, right-preconditioning of Newton’s method requires construction or application of the right-preconditioned Jacobian

$$(4.3) \quad \frac{\partial \mathbf{F}(\mathbf{M}(\mathbf{r}, \mathbf{y}))}{\partial \mathbf{y}} = \mathbf{J}(\mathbf{M}(\mathbf{r}, \mathbf{y})) \frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{y})}{\partial \mathbf{y}},$$

and we note that the computation of $\frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{y}_i)}{\partial \mathbf{y}_i}$ is generally impractical. Consider the application of nonlinear right-preconditioning on an iteration of Newton's method, which would proceed in two steps:

$$\begin{aligned} \mathbf{y}_{i+1} &= \mathbf{x}_i - \lambda \left(\frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{x}_i)}{\partial \mathbf{x}_i} \right)^{-1} \mathbf{J}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i))^{-1} \mathbf{r}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i)), \\ \mathbf{x}_{i+1} &= \mathbf{M}(\mathbf{r}, \mathbf{y}_{i+1}). \end{aligned}$$

We may nest these two steps as

$$(4.4) \quad \mathbf{x}_{i+1} = \mathbf{M} \left(\mathbf{r}, \mathbf{x}_i - \lambda \left(\frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{x}_i)}{\partial \mathbf{x}_i} \right)^{-1} \mathbf{J}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i))^{-1} \mathbf{r}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i)) \right),$$

and by Taylor expansion we see that

$$\mathbf{M}(\mathbf{r}, \mathbf{x} - \lambda \mathbf{d}) = \mathbf{M}(\mathbf{r}, \mathbf{x}) - \lambda \left(\frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{x})}{\partial \mathbf{x}} \right) \mathbf{d} + \dots,$$

giving the simplification

$$\begin{aligned} x_{i+1} &\approx \mathbf{M}(\mathbf{r}, \mathbf{x}_i) - \lambda \left(\frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{x}_i)}{\partial \mathbf{x}_i} \right) \left(\frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{x}_i)}{\partial \mathbf{x}_i} \right)^{-1} \mathbf{J}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i))^{-1} \mathbf{r}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i)) \\ &= \mathbf{M}(\mathbf{r}, \mathbf{x}_i) - \lambda \mathbf{J}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i))^{-1} \mathbf{r}(\mathbf{M}(\mathbf{r}, \mathbf{x}_i)). \end{aligned}$$

In this form, $\text{NEWT}\backslash\text{K}(\mathbf{F}(\mathbf{M}(\mathbf{r}, \mathbf{x})), \mathbf{x})$ is easily implemented, as shown in Algorithm 7.

ALGORITHM 7. Right-Preconditioned $\text{NEWT}\backslash\text{K}$.

- 1: **procedure** $\text{NK}(\mathbf{r}(\mathbf{M}(\mathbf{r}, \mathbf{x})), \mathbf{x}_i)$
 - 2: $\mathbf{x}_{i+\frac{1}{2}} = \mathbf{M}(\mathbf{r}, \mathbf{x}_i)$
 - 3: $\mathbf{d} = \mathbf{J}(\mathbf{x}_{i+\frac{1}{2}})^{-1} \mathbf{r}(\mathbf{x}_{i+\frac{1}{2}})$
 - 4: $\mathbf{x}_{i+1} = \mathbf{x}_{i+\frac{1}{2}} + \lambda \mathbf{d}$ $\triangleright \lambda$ determined by line search
 - 5: **return** \mathbf{x}_{i+1}
-

Note that $\text{NEWT}\backslash\text{K} -_R \mathbf{M}$ is merely $\mathbf{M} * \text{NEWT}\backslash\text{K}$ in this form: one solver runs after the other. The connection between composition and preconditioning in Newton's method provides an inkling of the advantages afforded to $\text{NEWT}\backslash\text{K}$ as an inner composite solver combined with other methods. An alternative approach applies an approximation to (4.3) directly [4]. The approximation is

$$\begin{aligned} \mathbf{F}(\mathbf{M}(\mathbf{r}, \mathbf{y}_i)) &= \mathbf{J}(\mathbf{M}(\mathbf{r}, \mathbf{y}_i)) \left(\frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{y}_i)}{\partial \mathbf{y}_i} \right) (\mathbf{y}_{i+1} - \mathbf{y}_i) \\ &\approx \mathbf{J}(\mathbf{M}(\mathbf{r}, \mathbf{y}_i)) (\mathbf{M}(\mathbf{r}, \mathbf{y}_i + [\mathbf{y}_{i+1} - \mathbf{y}_i]) - \mathbf{x}_i), \end{aligned}$$

which is solved for $[\mathbf{y}_{i+1} - \mathbf{y}_i]$. Right-preconditioning by the approximation requires an application of the nonlinear preconditioner for every inner Krylov iterate and limits our choices of Krylov solver to those tolerant of nonlinearity, such as flexible GMRES [58].

 ALGORITHM 8. Left-Preconditioned NEWT\K.

- 1: **procedure** NEWT\K($\mathbf{x} - \mathbf{M}(\mathbf{r}, \mathbf{x}), \mathbf{x}_i$)
 - 2: $\mathbf{d} = \frac{\partial(\mathbf{x}_i - \mathbf{M}(\mathbf{r}, \mathbf{x}_i))}{\partial \mathbf{x}_i}^{-1} (\mathbf{x}_i - \mathbf{M}(\mathbf{r}, \mathbf{x}_i))$ \triangleright approximate inversion by Krylov method
 - 3: $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda \mathbf{d}$ $\triangleright \lambda$ determined by line search
 - 4: **return** \mathbf{x}_{i+1}
-

For nonlinear left-preconditioning, shown in Algorithm 8, we replace the computation $\mathbf{r}(\mathbf{x}_i)$ with $\mathbf{x}_i - \mathbf{M}(\mathbf{r}, \mathbf{x}_i)$ and take the Jacobian of the function as an approximation of

$$(4.5) \quad \frac{\partial(\mathbf{x}_i - \mathbf{M}(\mathbf{r}, \mathbf{x}_i))}{\partial \mathbf{x}_i} = \mathbf{I} - \frac{\partial \mathbf{M}(\mathbf{r}, \mathbf{x}_i)}{\partial \mathbf{x}_i},$$

where now the Jacobian is a linearization of $\mathbf{M}(\mathbf{r}, \mathbf{x})$ and impractical to compute in most cases. In the particular case where the preconditioner is the NASM (see section 5.2), this is known as ASPIN. In the case of NASM preconditioning, one has local block Jacobians, so the approximation of the preconditioned Jacobian as

$$(4.6) \quad \frac{\partial(\mathbf{x} - \mathbf{M}(\mathbf{r}, \mathbf{x}))}{\partial \mathbf{x}} = \frac{\partial(\mathbf{x} - (\mathbf{x} - \sum_b \mathbf{J}^b(\mathbf{x}^b)^{-1} \mathbf{r}^b(\mathbf{x}^b)))}{\partial \mathbf{x}} \approx \sum_b \mathbf{J}^b(\mathbf{x}^{b*})^{-1} \mathbf{J}(\mathbf{x})$$

is used instead. The iteration requires only one inner nonlinear iteration and a small number of block solves per outer nonlinear iteration. By contrast, direct differencing would require one inner iteration at each inner linear iteration for the purpose of repeated approximate Jacobian application. Note the similarity between ASPIN and the alternative form of right-preconditioning in terms of required components, with ASPIN being more convenient in the end.

4.5. Quasi-Newton. The class of methods for nonlinear systems of equations known as quasi-Newton (QN) methods [24] uses previous changes in residual and solution or other low-rank expansions [40] to form an approximation of the inverse Jacobian by a series of low-rank updates. The approximate Jacobian inverse is then used to compute the search direction. If the update is done directly, it requires storing and updating the dense matrix $\mathbf{K} \approx \mathbf{J}^{-1}$, which is impractical for large systems. One may overcome this limitation by using limited-memory variants of the method [46, 51, 14, 49], which apply the approximate inverse Jacobian by a series of low-rank updates. The general form of QN methods is similar to that of NEWT\K and is shown in Algorithm 9.

 ALGORITHM 9. QN Update.

- 1: **procedure** QN($\mathbf{r}, \mathbf{x}_i, \mathbf{s}_{i-1} \dots \mathbf{s}_{i-m}, \mathbf{y}_{i-1} \dots \mathbf{y}_{i-m}$)
 - 2: $\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda \mathbf{K}_i(\mathbf{K}_0, \mathbf{s}_{i-1} \dots \mathbf{s}_{i-m}, \mathbf{y}_{i-1} \dots \mathbf{y}_{i-m}) \mathbf{r}(\mathbf{x}_i)$ $\triangleright \lambda$ determined by line search
 - 3: $\mathbf{s}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$
 - 4: $\mathbf{y}_i = \mathbf{r}(\mathbf{x}_{i+1}) - \mathbf{r}(\mathbf{x}_i)$
 - 5: **return** \mathbf{x}_{i+1}
-

A popular variant of the method, L-BFGS, can be applied efficiently by a two-loop recursion [51]:

ALGORITHM 10. Two-Loop Recursion for L-BFGS.

```

1: procedure  $\mathbf{K}_i(\mathbf{K}_0, \mathbf{s}_{i-1} \dots \mathbf{s}_{i-m}, \mathbf{y}_{i-1} \dots \mathbf{y}_{i-m}, \mathbf{r}(\mathbf{x}_i))$ 
2:    $\mathbf{d}_1 = \mathbf{r}(\mathbf{x}_i)$ 
3:   for  $k = i - 1$  do  $i - m$ 
4:      $\alpha_k = \frac{\mathbf{s}_k^\top \mathbf{d}_1}{\mathbf{y}_k^\top \mathbf{s}_k}$ 
5:      $\mathbf{d}_1 = \mathbf{d}_1 - \alpha_k \mathbf{y}_k$ 
6:   end for
7:    $\mathbf{d}_2 = \mathbf{K}_0 \mathbf{d}_1$ 
8:   for  $k = i - m$  do  $i - 1$ 
9:      $\beta_k = \frac{\mathbf{y}_k^\top \mathbf{d}_2}{\mathbf{y}_k^\top \mathbf{s}_k}$ 
10:     $\mathbf{d}_2 = \mathbf{d}_2 + (\alpha_k - \beta_k) \mathbf{s}_k$ 
11:  end for
12: return  $\mathbf{d}_2$ 

```

Note that the update to the Jacobian in Algorithm 10 is symmetric and shares many of the same limitations as NCG (see section 4.6). It may be perplexing, from an optimization perspective, to see L-BFGS and NCG applied to nonlinear PDEs as opposed to optimization problems. However, their exclusion from our discussion would be as glaring as leaving conjugate gradient methods out of a discussion of Krylov solvers for linear PDEs. In this paper we adhere to the limitations of NCG and QN and use them only for problems with symmetric Jacobians. In the case where L-BFGS is inapplicable, Broyden's "good" and "bad" methods [13] have efficient limited memory constructions [14, 26, 29] and are recommended instead [52]. The overall performance of all these methods, however, may rival that of Newton's method [38] in many cases. QN methods also have deep connections with ANDERSON [28]. ANDERSON and the Broyden methods belong to a general class of Broyden-like methods [29].

The initial approximate inverse Jacobian \mathbf{K}_0 is often taken as the weighted [61] identity. However, one also can take $\mathbf{K}_0 = \mathbf{J}_0^{-1}$ for lagged Jacobian \mathbf{J}_0 . Such schemes greatly increase the robustness of lagged Newton's methods when solving nonlinear PDEs [10]. Left-preconditioning for QN is trivial, and either the preconditioned or unpreconditioned residual may be used with the line search.

4.6. Nonlinear Conjugate Gradients. NCG methods [31] are a simple extension of the linear conjugate gradient method but with the optimal step length in the conjugate direction determined by a line search rather than the exact formula that works in the linear case. NCG is outlined in Algorithm 11. NCG requires the storage of one additional vector for the conjugate direction \mathbf{c}_i but has significantly faster convergence than does NRICH for many problems. Several choices exist for constructing the parameter β_i [21, 30, 31, 35]. We choose the Polak–Ribière–Polyak [54] variant. The application of nonlinear left-preconditioning is straightforward. Right-preconditioning is not conveniently applicable as is the case with QN in section 4.5 and linear conjugate gradient.

NCG has limited applicability because it suffers from the same issues as its linear cousin for problems with nonsymmetric Jacobian. A common practice when solving nonlinear PDEs with NCG is to rephrase the problem in terms of the normal equations, which involves finding the root of $\mathbf{r}_N(\mathbf{x}) = \mathbf{J}^\top \mathbf{r}(\mathbf{x})$. Since the Jacobian must be

ALGORITHM 11. NCG.

```

1: procedure NCG( $\mathbf{r}, \mathbf{x}_i, \mathbf{c}_{i-1}, \mathbf{r}_{i-1}$ )
2:    $\mathbf{r}_i = \mathbf{r}(\mathbf{x}_i)$ 
3:    $\beta_i = \frac{\mathbf{r}_i^\top (\mathbf{r}_i - \mathbf{r}_{i-1})}{\mathbf{r}_{i-1}^\top \mathbf{r}_{i-1}}$ 
4:    $\mathbf{c}_i = -\mathbf{r}(\mathbf{x}_i) + \beta_i \mathbf{c}_{i-1}$ 
5:    $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda \mathbf{c}_i$  ▷  $\lambda$  determined by line search
6: return  $\mathbf{x}_{i+1}$ 

```

assembled and multiplied at every iteration, however, the normal equation solver is not a reasonable algorithmic choice, even with drastic lagging of Jacobian assembly. The normal equations also have a much worse condition number than does the original PDE. In our experiments, we use the conjugacy-ensuring CP line search.

5. Decomposition Solvers. An extremely important class of methods for linear problems is based on domain or hierarchical decomposition, and so we consider nonlinear variants of domain decomposition and multilevel algorithms. We introduce three different nonlinear solver algorithms based on point-block solves, local subdomain solves, and coarse-grid solves.

These methods require a trade-off between the amount of computation dedicated to solving local or coarse problems and the communication for global assembly and convergence monitoring. Undersolving the subproblems wastes the communication overhead of the outer iteration, and oversolving exacerbates issues of load imbalance and leads rapidly to diminishing returns with respect to convergence. The solvers in this section exhibit a variety of features related to these trade-offs.

The decomposition solvers do not guarantee convergence. While they might converge, the obvious extension is to use them in conjunction with the global solvers as nonlinear preconditioners or accelerators. As the decomposition solvers expose more possibilities for parallelism or acceleration, their effective use in the nonlinear context should provide similar benefits to the analogous solvers in the linear context. A major disadvantage of these solvers is that each of them requires additional information about the local problems, a decomposition, or the hierarchy of discretizations of the domain.

5.1. Gauss–Seidel–Newton. Effective methods may be constructed by exact solves on subproblems. Suppose that the problem easily decomposes into n_b small block subproblems. If Newton’s method is applied multiplicatively by blocks, the resulting algorithm is known as Gauss–Seidel–Newton (GSN) and is shown in Algorithm 12. Similar methods are commonly used as nonlinear smoothers [33, 34]. To construct GSN, we define the individual block Jacobians $\mathbf{J}^b(\mathbf{x}^b)$ and residuals $\mathbf{r}^b(\mathbf{x}^b)$; \mathbf{R}^b , which restricts to a block; and \mathbf{P}^b , which injects the solution to that block back into the overall solution. The point-block solver runs until the norm of the block residual is less than ϵ^b or m_b steps have been taken. GSN solves small subproblems with a fair amount of computational work per degree of freedom and thus has high arithmetic intensity. It requires typically greater than two times the work per degree of freedom of NRICH for scalar problems, and even more for vector problems where the local Newton solve is over several local degrees of freedom.

When used as a solver, GSN requires one reduction per iteration. However, stationary solvers, in both the linear and nonlinear cases, do not converge robustly for large problems. Accordingly, GSN would typically be used as a preconditioner, mak-

ALGORITHM 12. GSN.

```

1: procedure GSN( $\mathbf{r}, \mathbf{x}_i$ )
2:    $\mathbf{x}_{i+1} = \mathbf{x}_i$ 
3:   for  $b = 1$  do  $n_b$ 
4:      $\mathbf{x}_{i,0}^b = \mathbf{R}^b \mathbf{x}_{i+1}$ 
5:     while  $\|\mathbf{r}^b\|_2 > \epsilon^b$  and  $j < m_b$  do
6:        $j = j + 1$ 
7:        $\mathbf{x}_{i,j}^b = \mathbf{x}_{i,j-1}^b - \mathbf{J}^b(\mathbf{x}_{i,j-1}^b)^{-1} \mathbf{r}_{i,j-1}^b$   $\triangleright$  direct inversion of block Jacobian
8:        $\mathbf{r}_{i,j}^b = \mathbf{r}^b(\mathbf{x}_{i,j}^b)$ 
9:     end while
10:     $\mathbf{x}_{i+1} = \mathbf{x}_{i+1} - \mathbf{P}^b(\mathbf{x}_{i,0}^b - \mathbf{x}_{i,j}^b)$ 
11:  end for
12: return  $\mathbf{x}_{i+1}$ 

```

ing monitoring of global convergence within the GSN iterations unnecessary. Thus, in practice there is no synchronization; GSN is most easily implemented with multiplicative update on serial subproblems and additively in parallel.

5.2. Nonlinear Additive Schwarz Method. GSN can be an efficient underlying kernel for sequential nonlinear solvers. For parallel computing, however, an additive method with overlapping subproblems has desirable properties with respect to communication and arithmetic intensity. The NASMs allow for medium-sized subproblems to be solved with a general method, and the corrections from that method to be summed into a global search direction. Here we limit ourselves to decomposition by subdomain rather than splitting the problem into fields. While eliminating fields might be tempting, the construction of effective methods of this sort is significantly more involved than in the linear case, and many interesting problems do not have such a decomposition readily available. Using subdomain problems \mathbf{F}^B , restrictions \mathbf{R}^B , injections \mathbf{P}^B , and solvers \mathbf{M}^B , Algorithm 13 outlines the NASM solver with n_B subdomains.

ALGORITHM 13. NASM.

```

1: procedure NASM( $\mathbf{r}, \mathbf{x}_i$ )
2:   for  $B = 1$  do  $n_B$ 
3:      $\mathbf{x}_0^B = \mathbf{R}^B \mathbf{x}_i$ 
4:      $\mathbf{x}^B = \mathbf{M}^B(\mathbf{r}^B, \mathbf{x}_0^B)$ 
5:      $\mathbf{y}^B = \mathbf{x}_0^B - \mathbf{x}^B$ 
6:   end for
7:    $\mathbf{x}_{i+1} = \mathbf{x}_i - \sum_{B=1}^{n_B} \mathbf{P}^B \mathbf{y}^B$ 
8: return  $\mathbf{x}_{i+1}$ 

```

Two choices exist for the injections \mathbf{P}^B in the overlapping regions. We will use NASM to denote the variant that uses overlapping injection corresponding to the whole subproblem step. The second variant, restricted additive Schwarz (RAS),

injects the step in a nonoverlapping fashion. The subdomain solvers are typically Newton–Krylov, but the other methods described in this paper are also applicable.

5.3. Full Approximation Scheme. FAS [6] accelerates convergence by advancing the nonlinear solution on a series of coarse discretizations of the problem. As with standard linear multigrid, the cycle may be constructed either additively or multiplicatively, with multiplicative being more effective in terms of per-iteration convergence. Additive approaches provide the usual advantage of allowing the coarse-grid corrections to be computed in parallel. We do not consider additive FAS in this paper.

Given the smoother $\mathbf{M}_s(\mathbf{r}, \mathbf{x})$ at each level, as well as restriction (\mathbf{R}), prolongation (\mathbf{P}), and injection ($\hat{\mathbf{R}}$) operators, and the coarse nonlinear function \mathbf{F}^H , the FAS V-cycle takes the form shown in Algorithm 14.

ALGORITHM 14. FAS.

```

1: procedure FAS( $\mathbf{r}, \mathbf{x}_i$ )
2:    $\mathbf{x}_s = \mathbf{M}_s(\mathbf{r}, \mathbf{x}_i)$ 
3:    $\mathbf{x}_i^H = \hat{\mathbf{R}}\mathbf{x}_s$ 
4:    $\mathbf{b}^H = \mathbf{R}[\mathbf{b} - \mathbf{F}(\mathbf{x}_s)] + \mathbf{F}^H(\mathbf{x}_i^H)$ 
5:    $\mathbf{x}_c = \mathbf{x}_s + \mathbf{P}[\text{FAS}(\mathbf{F}^H - \mathbf{b}^H, \mathbf{x}_i^H) - \mathbf{x}_i^H]$ 
6:    $\mathbf{x}_{i+1} = \mathbf{M}_s(\mathbf{r}, \mathbf{x}_c)$ 
7: return  $\mathbf{x}_{i+1}$ 

```

The difference between FAS and linear multigrid is the construction of the coarse RHS \mathbf{b}^H . In FAS it is guaranteed that if an exact solution \mathbf{x}^* to the fine problem is found,

$$(5.1) \quad \text{FAS}(\mathbf{r}^H - \mathbf{b}^H, \hat{\mathbf{R}}\mathbf{x}^*) - \hat{\mathbf{R}}\mathbf{x}^* = 0.$$

The correction is not necessary in the case of linear multigrid. However, FAS is mathematically identical to standard multigrid when applied to a linear problem.

The stellar algorithmic performance of linear multigrid methods is well documented, but the arithmetic intensity may be low. FAS presents an interesting alternative to NEWT\K – MG, since it may be configured with high arithmetic intensity operations at all levels. The smoothers are themselves nonlinear solution methods. FAS-type methods are applicable to optimization problems as well as nonlinear PDEs [8], with optimization methods used as smoothers [50].

5.4. Summary. We have now introduced the mathematical construction of nonlinear composed solvers. We have also described two classes of nonlinear solvers, based on solving either the global problem or some partition of it. The next step is to describe a set of test problems with different limitations with respect to which methods will work for them, construct a series of instructive example solvers using composition of the previously defined solvers, and test them. We have made an effort to create flexible and robust software for composed nonlinear solvers. The organization of the software is in the spirit of PETSc and includes several interchangeable component solvers, including NEWT\K; iterative solvers such as ANDERSON and QN that may contain an inner preconditioner; decomposition solvers such as NASM and FAS, built out of subdomain solvers; and metasolvers implementing compositions. A diagram enumerating the composed solver framework analogous to that of the preconditioned NEWT\K case is shown in Figure 5.1.

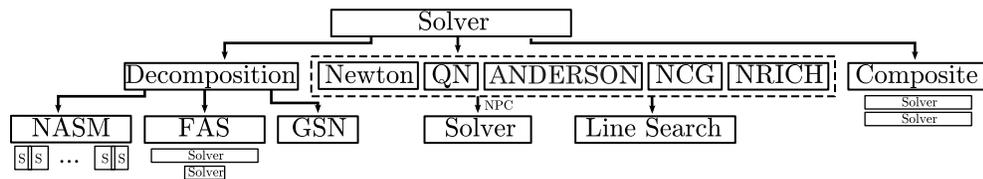


Fig. 5.1 Organization of the components of a composed nonlinear solver. We discard the difference between solvers and preconditioners and see that nesting and potential for recursive customization live at every level of the tree. Possibilities include iterative solvers (including Newton's method) with nonlinear preconditioning, composite solvers consisting of several subsolvers, or decomposition solvers consisting of subdomain or coarse nonlinear solvers.

6. Experiments. We will demonstrate the efficacy of nonlinear composition and preconditioning by experiments with a suite of nonlinear PDEs that show interesting behavior in regimes with difficult nonlinearities. These problems are nonlinear elasticity, the lid-driven cavity with buoyancy, and the p -Laplacian.

One goal of this paper is to be instructive. We first try to solve the problem efficiently with the standard solvers, then choose a subset of the above solvers for each problem and show how to gain advantage by using composition methods. Limitations of discretization or problem regime are noted, and we discuss how the solvers may be used under these limitations. We also explain how readers may run the examples in this paper and experiment with the solvers both on the test examples shown here and on their own problems. We feel we have no “skin in the game” and are not trying to show that some approaches are better or worse than others.

6.1. Methods. Our set of test algorithms is defined by the preconditioning of an iterative solver with a decomposition solver, or the composition of two solvers with different advantages. In the case of the decomposition solvers, we choose one or two configurations per test problem in order to avoid the combinatorial explosion of potential methods that already is apparent from the two forms of composition combined with the multitude of methods.

Our primary measure of performance is time to solution, which is both problem and equipment dependent. Since it depends strongly on the relative cost of function evaluation, Jacobian assembly, matrix multiplication, and subproblem or coarse solve, we also record the numbers of nonlinear iterations, linear iterations, linear preconditioner applications, function evaluations, Jacobian evaluations, and nonlinear preconditioner applications. These measures allow us to characterize efficiency disparities in terms of major units of computational work.

We make a good faith effort to tune each method for performance while keeping subsolver parameters invariant. Occasionally, we err on the side of oversolving for the inner solvers, to make the apples-to-apples comparison more consistent between closely related methods. The results here should be taken as a rough guide to improving solver performance and robustness. The test machine is a 64-core AMD Opteron 6274-based [1] machine with 128 GB of memory. The problems are run with one MPI process per core. Plots of convergence are generated with Matplotlib [36], and plots of solutions are generated with Mayavi [57].

6.2. Nonlinear Elasticity. A Galerkin formulation for nonlinear elasticity may be stated as

$$(6.1) \quad \int_{\Omega} F \cdot S : \nabla v \, d\Omega + \int_{\Omega} b \cdot v \, d\Omega = 0$$

for all test functions $v \in \mathcal{V}$; $F = \nabla \mathbf{u} + I$ is the deformation gradient. We use the Saint Venant–Kirchhoff model of hyperelasticity with second Piola–Kirchhoff stress tensor $S = \lambda \text{tr}(E)I + 2\mu E$ for Lagrangian Green strain $E = F^\top F - I$ and Lamé parameters λ and μ , which may be derived from a given Young’s modulus and Poisson ratio. We solve for displacements $\mathbf{u} \in \mathcal{V}$, given a constant load vector b imposed on the structure.

The domain Ω is a 60° cylindrical arch of inner radius 100 m. Homogeneous Dirichlet boundary conditions are imposed on the outer edge of the ends of the arch. The goal is to “snap through” the arch, causing it to sag under the load rather than merely compressing it. To cause the sag, we set $b = -\mathbf{e}_y$ and set the Lamé constants consistent with a Young’s modulus of 100 and a Poisson ratio of 0.2. The nonlinearity is highly activated during the process of snapping through and may be tuned to present a great deal of difficulty to traditional nonlinear solvers. The problem is discretized by using hexahedral \mathbf{Q}_1 finite elements on a logically structured grid, deformed to form the arch. The grid is $401 \times 9 \times 9$, and therefore the problem has 97,443 degrees of freedom. The problem is a three-dimensional extension of a problem put forth by Wriggers [75]. The unstressed and converged solutions are shown in Figure 6.1.

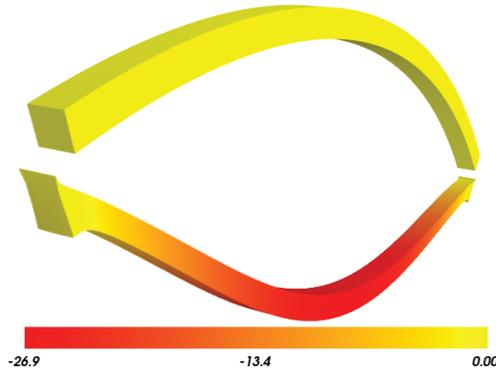


Fig. 6.1 *Unstressed and stressed configurations for the elasticity test problem. Coloration indicates vertical displacement in meters.*

For three-dimensional hexahedral FEM discretizations, GSN and related algorithms are inefficient because for each degree of freedom each element in the support of the degree of freedom must be visited, resulting in eight visits to an interior element per sweep. Instead, we focus on algorithms requiring only a single visit to each cell, restricting us to function and Jacobian evaluations. Such approaches are also available in the general unstructured FEM case, and these experiments may guide users in regimes where no decomposition solvers are available.

For the experiments, we emphasize the role that nonlinear composition and series acceleration may play in the acceleration of nonlinear solvers. The problem is amenable to NCG, ANDERSON, and NEWT\K with an algebraic multigrid preconditioner. We test a number of combinations of these solvers. Even though we have a logically structured grid, we approach the problem as if no reasonable grid hierarchy or domain decomposition were available. The solver combinations we use are listed in Table 6.1. In all the following tables, “Solver” denotes outer solver, “LPC” denotes the linear PC when applicable, “NPC” denotes the nonlinear PC when applicable, “Side” denotes the type of preconditioning, “Smooth” denotes the level smoothers

Table 6.1 *Series of solvers for the nonlinear elasticity test problem.*

| Name | Solver | LPC | NPC | Side | Smooth | LS |
|------------------------|------------|-----|-----------|------|--------|-------|
| NCG | NCG | | | | | CP |
| NEWT\K-MG | NEWT\K | MG | - | - | SOR | BT |
| NCG-L (NEWT\K-MG) | NCG | - | NEWT\K-MG | L | SOR | CP |
| ANDERSON-R (NEWT\K-MG) | ANDERSON | - | NEWT\K-MG | R | SOR | CP |
| NCG(10)+(NEWT\K-MG) | NCG,NEWT\K | MG | - | - | SOR | CP/BT |
| NCG(10) * (NEWT\K-MG) | NCG,NEWT\K | MG | - | - | SOR | CP/BT |

used in the multilevel method (MG/FAS), and “LS” denotes line search. For the NEWT\K methods, we precondition the inner GMRES solve with a smoothed aggregation algebraic multigrid method provided by the GAMG package in PETSc. The relative tolerance for the GMRES solve is 10^{-3} . For all instances of the NCG solver, the CP line search initial guess for λ is the final value from the previous nonlinear iteration. In the case of ANDERSON-R (NEWT\K-MG) the inner line search is L2. ANDERSON stores up to 30 previous solutions and residuals for all experiments. The outer line search for NCG-L (NEWT\K-MG) is a second-order secant approximation rather than first-order as depicted in Algorithm 1. For the composite examples, 10 iterations of NCG are used as one of the subsolvers, denoted NCG(10). The additive composition’s weights are determined by the ANDERSON minimization mechanism as described in section 4.3.

The example, SNES ex16.c, can be run directly by using a default PETSc installation. The command line used for these experiments is

```
./ex16 -da_grid_x 401 -da_grid_y 9 -da_grid_z 9 -height 3 -width 3
-rad 100 -young 100 -poisson 0.2 -loading -1 -ploading 0
```

The solvers shown in Table 6.2 converge slowly. NEWT\K-MG takes 27 nonlinear iterations and 1,618 multigrid V-cycles to reach convergence. NCG requires 8,991 function evaluations and, since it is unpreconditioned, scales unacceptably with problem size with respect to number of iterations. Note in Figure 6.2 that while NCG takes an initial jump and then decreases at a constant rate, NEWT\K-MG is near stagnation until the end, when it suddenly drops into the basin of attraction.

Results for composition are listed in Table 6.3. Additive and multiplicative composite combinations provide roughly similar speedups for the problem, with more total outer iterations in the additive case. As shown in Figure 6.3, neither the additive nor the multiplicative methods stagnate. After the same initial jump and convergence at the pace of NCG, the basin of attraction is reached and the entire iteration converges quadratically, as should be expected with NEWT\K.

Left-preconditioning of NCG and ANDERSON with NEWT\K-MG provides even greater benefits, as shown in Figure 6.4 and Table 6.4. The number of iterations is nearly halved from the unpreconditioned case, and the number of linear precondi-

Table 6.2 *(NEWT\K-MG) and NCG results.*

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|-------------|-------|-------|-------|------|-----|------|-----|
| (NEWT\K-MG) | 23.43 | 27 | 1556 | 91 | 27 | 1618 | - |
| NCG | 53.05 | 4495 | 0 | 8991 | - | - | - |

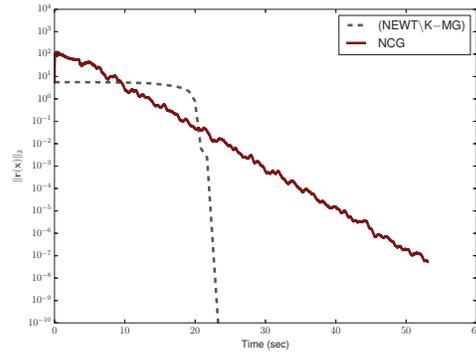


Fig. 6.2 $(NEWT\backslash K - MG)$ and NCG convergence.

Table 6.3 $NCG(10) + (NEWT\backslash K - MG)$ and $NCG(10) * (NEWT\backslash K - MG)$ results.

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|-------------------------------------|-------|-------|-------|------|-----|-----|-----|
| $NCG(10) + (NEWT\backslash K - MG)$ | 14.92 | 9 | 459 | 218 | 9 | 479 | – |
| $NCG(10) * (NEWT\backslash K - MG)$ | 16.34 | 11 | 458 | 251 | 11 | 477 | – |

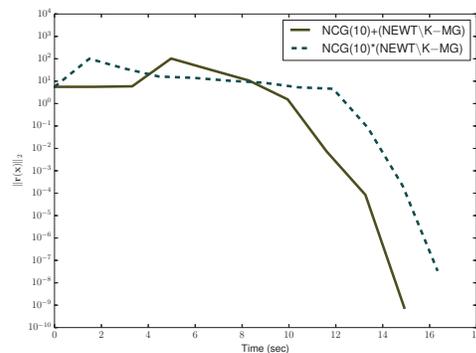


Fig. 6.3 $NCG(10) + (NEWT\backslash K - MG)$ and $NCG(10) * (NEWT\backslash K - MG)$ convergence.

tioner applications is only slightly increased compared with the composition cases. In Figure 6.4, we see that the Newton's method-preconditioned nonlinear Krylov solvers never dive as with $NEWT\backslash K$ but instead maintain rapid, mostly constant convergence. This constant convergence is a significantly more efficient path to solution than the $NEWT\backslash K - MG$ solver alone.

6.3. Driven Cavity. The driven cavity formulation used for the next set of experiments can be stated as

$$(6.2) \quad -\Delta \mathbf{u} - \nabla \times \Omega = 0,$$

$$(6.3) \quad -\Delta \Omega + \nabla \cdot (\mathbf{u}\Omega) - Gr \nabla_x T = 0,$$

$$(6.4) \quad -\Delta T + Pr \nabla \cdot \mathbf{u} = 0.$$

The fluid motion is driven in part by a moving lid and in part by buoyancy. A Grashof

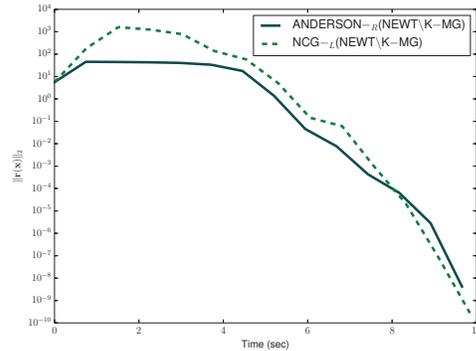


Fig. 6.4 $ANDERSON-R(NEWT\K-MG)$ and $NCG-L(NEWT\K-MG)$ convergence.

Table 6.4 $ANDERSON-R(NEWT\K-MG)$ and $NCG-L(NEWT\K-MG)$ results.

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|-------------------------|------|-------|-------|------|-----|-----|-----|
| $ANDERSON-R(NEWT\K-MG)$ | 9.65 | 13 | 523 | 53 | 13 | 548 | 13 |
| $NCG-L(NEWT\K-MG)$ | 9.84 | 13 | 529 | 53 | 13 | 554 | 13 |

number $Gr = 2e4$, Prandtl number $Pr = 1$, and lid velocity of 100 are used in these experiments. No-slip, rigid-wall Dirichlet conditions are imposed for \mathbf{u} . Dirichlet conditions are used for Ω , based on the definition of vorticity, $\Omega = \nabla \times \mathbf{u}$, where along each constant coordinate boundary the tangential derivative is zero. Dirichlet conditions are used for T on the left and right walls, and insulating homogeneous Neumann conditions are used on the top and bottom walls. A finite-difference approximation with the usual five-point stencil is used to discretize the boundary value problem in order to obtain a nonlinear system of equations. Upwinding is used for the divergence (convective) terms and central differencing for the gradient (source) terms.

In these experiments, we emphasize the use of ANDERSON, MG or FAS, and GSN. The solver combinations are listed in Table 6.5. The multilevel methods use a simple series of structured grids, with the smallest being 5×5 and the largest 257×257 . With four fields per grid point, we have an overall system size of 264,196 unknowns. In $NEWT\K-MG$, the Jacobian is rediscritized on each level. GSN is used as the smoother for FAS and solves the four-component block problem corresponding to (6.2) per grid point in a simple sweep through the processor local part of the domain.

The example, SNES ex19.c, can be run directly by using a default PETSc installation. The command line used for these experiments is

```
./ex19 -da_refine 6 -da_grid_x 5 -da_grid_y 5 -grashof 2e4 -lidvelocity 100 -prandtl 1.0
```

and the converged solution using these options is shown in Figure 6.5.

All linearized problems in the $NEWT\K$ iteration are solved using GMRES with geometric multigrid preconditioning to a relative tolerance of 10^{-8} . There are five levels, with Chebyshev-SOR smoothers on each level. For FAS, the smoother is GSN(5). The coarse-level smoother is five iterations of $NEWT\K-LU$, chosen for robustness. In the cases of $ANDERSON-R(NEWT\K-MG)$ and $FAS+(NEWT\K-MG)$, the $NEWT\K-MG$ step is damped by one half. Note that acting alone, this damping

Table 6.5 *Solvers for the lid driven cavity problem.*

| Name | Solver | LPC | NPC | Side | MG/FASSmooth. | LS |
|------------------------------------|------------|-----|--------|------|---------------|----|
| (NEWT\K-MG) | NEWT\K | MG | - | - | SOR | BT |
| ANDERSON- _R (NEWT\K-MG) | ANDERSON | MG | NEWT\K | R | SOR | - |
| FAS | FAS | - | - | - | GSN | - |
| NRICH- _L FAS | NRICH | - | FAS | L | GSN | L2 |
| ANDERSON- _R FAS | ANDERSON | - | FAS | R | GSN | - |
| FAS * (NEWT\K-MG) | FAS/NEWT\K | MG | - | - | SOR/GSN | BT |
| FAS+(NEWT\K-MG) | FAS/NEWT\K | MG | - | - | SOR/GSN | BT |

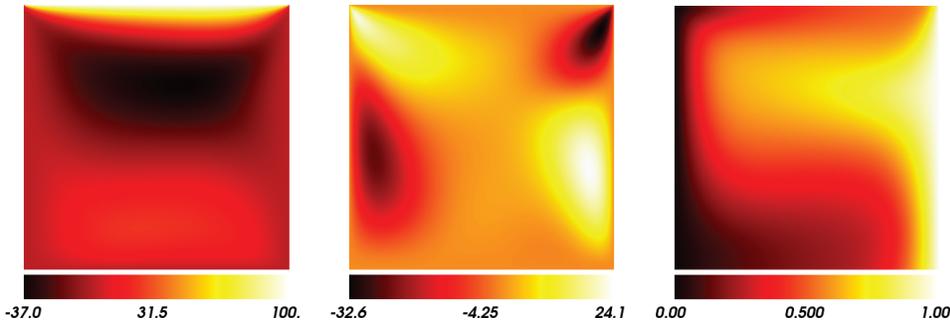


Fig. 6.5 u_x , u_y , and temperature profiles for the lid driven cavity solution.

Table 6.6 $ANDERSON -_R (NEWT\K - MG)$ and $(NEWT\K - MG)$ results.

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|------------------------------|------|-------|-------|------|-----|-----|-----|
| $ANDERSON -_R (NEWT\K - MG)$ | 7.48 | 10 | 220 | 21 | 50 | 231 | 10 |
| $(NEWT\K - MG)$ | 9.83 | 17 | 352 | 34 | 85 | 370 | - |

would cut the rate of convergence to linear with a rate constant $\frac{1}{2}$. The ideal step size is recovered by the minimization procedure. NEWT\K - MG is undamped in all other tests. In these experiments we emphasize the total number of V-cycles, linear and nonlinear, since they dominate the runtime and contain the lion’s share of the communication and floating-point operations.

In Figures 6.6 and 6.7 and Table 6.7 we show the convergence of FAS with and without the acceleration of NRICH and ANDERSON. NRICH -_L FAS takes 50 V-cycles, at the expense of three more fine-level function evaluations per iteration. ANDERSON -_R FAS reduces the number of V-cycles to 24 at the expense In Table 6.6, we see that Newton’s method converges in 17 iterations, with 370 V-cycles. Using ANDERSON instead of a line search provides some benefit, as ANDERSON - $(NEWT\K - MG)$ takes 231 V-cycles and 10 iterations. of more communication. Composed nonlinear methods are shown in Table 6.8. Multiplicative composition consisting of FAS and NEWT\K-MG reduces the total number of V-cycles to 130. Additive composition using the least-squares minimization is less effective, taking 298 V-cycles. Note in Figure 6.8 that both the additive and multiplicative solvers show that combining FAS and NEWT\K-MG may speed solution, with multiplicative combination being significantly more effective.

6.4. Tuning the Solvers to Obtain Convergence. We now show how the composed and preconditioned solves may be tuned for more difficult nonlinearities when

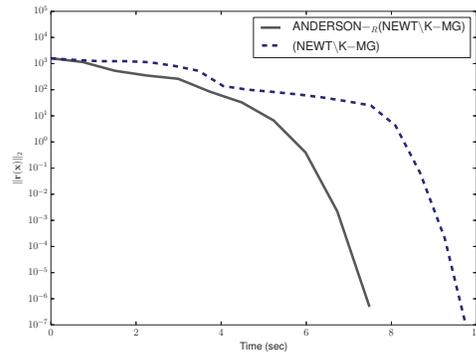


Fig. 6.6 $ANDERSON-R$ ($NEWT\backslash K-MG$) and ($NEWT\backslash K-MG$) convergence.

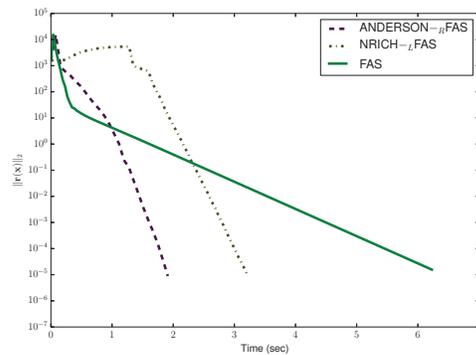


Fig. 6.7 $ANDERSON-R$ FAS, $NRICH-L$ FAS, and FAS convergence.

Table 6.7 $ANDERSON-R$ FAS, $NRICH-L$ FAS, and FAS results.

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|------------------|------|-------|-------|------|-----|-----|-----|
| $ANDERSON-R$ FAS | 1.91 | 24 | 0 | 447 | 83 | 166 | 24 |
| $NRICH-L$ FAS | 3.20 | 50 | 0 | 1180 | 192 | 384 | 50 |
| FAS | 6.23 | 162 | 0 | 2382 | 377 | 754 | — |

the basic methods fail to converge using the same model problem. For Grashof number $Gr < 10^4$ and Prandtl number $Pr = 1.0$, Newton's method converges well:

```

lid velocity = 100, prandtl # = 1, grashof # = 10000
0 SNES Function norm 715.271
1 SNES Function norm 623.41
2 SNES Function norm 510.225
.
.
6 SNES Function norm 0.269179
7 SNES Function norm 0.00110921
8 SNES Function norm 1.12763e-09
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 8
Number of SNES iterations = 8

```

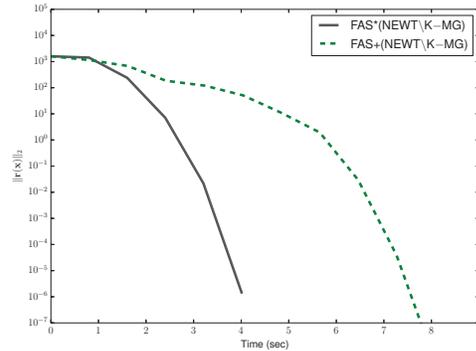


Fig. 6.8 $FAS^*(NEWT\backslash K - MG)$ and $FAS+(NEWT\backslash K - MG)$ convergence.

Table 6.8 $FAS^*(NEWT\backslash K - MG)$ and $FAS+(NEWT\backslash K - MG)$ results.

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|--------------------------------|------|-------|-------|------|-----|-----|-----|
| $FAS^*(NEWT\backslash K - MG)$ | 4.01 | 5 | 80 | 103 | 45 | 125 | – |
| $FAS+(NEWT\backslash K - MG)$ | 8.07 | 10 | 197 | 232 | 90 | 288 | – |

For higher Grashof number, Newton's method stagnates:

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -pc_type lu -snes_monitor_short
-snes_converged_reason
lid velocity = 100, prandtl # = 1, grashof # = 50000
0 SNES Function norm 1228.95
1 SNES Function norm 1132.29
.
.
29 SNES Function norm 580.937
30 SNES Function norm 580.899
```

It also fails with the standard continuation strategy from coarser meshes (not shown). We next try nonlinear multigrid, in the hope that multiple updates from a coarse solution are sufficient, using Gauss–Seidel as the nonlinear smoother:

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short -snes_converged_reason
-snes_type fas -fas_levels_snes_type ngs -fas_levels_snes_max_it 6 -snes_max_it 25
lid velocity = 100, prandtl # = 1, grashof # = 50000
0 SNES Function norm 1228.95
1 SNES Function norm 574.793
2 SNES Function norm 513.02
3 SNES Function norm 216.721
4 SNES Function norm 85.949
5 SNES Function norm 108.24
6 SNES Function norm 207.469
.
.
22 SNES Function norm 131.866
23 SNES Function norm 114.817
24 SNES Function norm 71.4699
25 SNES Function norm 63.5413
```

However, the residual norm just jumps around and the method does not converge.

We then accelerate the method with ANDERSON and obtain convergence:

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short -snes_converged_reason
-snes_type anderson -npc_snes_max_it 1 -npc_snes_type fas -npc_fas_levels_snes_type ngs
-npc_fas_levels_snes_max_it 6
lid velocity = 100, prandtl # = 1, grashof # = 50000
0 SNES Function norm 1228.95
1 SNES Function norm 574.793
2 SNES Function norm 345.592
3 SNES Function norm 155.476
4 SNES Function norm 70.2302
5 SNES Function norm 40.3618
6 SNES Function norm 29.3065
7 SNES Function norm 14.2497
8 SNES Function norm 4.80462
9 SNES Function norm 4.15985
10 SNES Function norm 2.13428
11 SNES Function norm 1.57717
12 SNES Function norm 0.60919
13 SNES Function norm 0.150496
14 SNES Function norm 0.0355709
15 SNES Function norm 0.00705481
16 SNES Function norm 0.00164509
17 SNES Function norm 0.000464835
18 SNES Function norm 6.02035e-05
19 SNES Function norm 1.11713e-05
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 19
Number of SNES iterations = 19
```

We can restore the convergence to a few iterates by increasing the power of the nonlinear smoothers in FAS. We replace GSN by six iterations of Newton using the default linear solver of GMRES plus ILU(0). Note that as a solver alone this does not converge, but it performs very well as a smoother for FAS:

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short -snes_converged_reason
-snes_type anderson -npc_snes_max_it 1 -npc_snes_type fas -npc_fas_levels_snes_type newtonls
-npc_fas_levels_snes_max_it 6 -npc_fas_levels_snes_linesearch_type basic
-npc_fas_levels_snes_max_linear_solve_fail 30 -npc_fas_levels_ksp_max_it 20
lid velocity = 100, prandtl # = 1, grashof # = 50000
0 SNES Function norm 1228.95
1 SNES Function norm 0.187669
2 SNES Function norm 0.0319743
3 SNES Function norm 0.00386815
4 SNES Function norm 2.24093e-05
5 SNES Function norm 5.38246e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 5
Number of SNES iterations = 5
```

Thus, we have demonstrated how one may experimentally add composed solvers to move from complete lack of convergence to convergence with a small number of iterations.

6.5. p -Laplacian. The regularized p -Laplacian formulation used for these experiments is

$$-\nabla \cdot \left(\left(\epsilon^2 + \frac{1}{2} |\nabla u|^2 \right)^{(p-2)/2} \nabla u \right) = c,$$

where $\epsilon = 10^{-5}$ is the regularization parameter and p the exponent of the Laplacian. When $p = 2$, the p -Laplacian reduces to the Poisson equation. We consider the case where $p = 5$ and $c = 0.1$. The domain is $[-1, 1] \times [-1, 1]$ and the initial guess is $u_0(x, y) = xy(1 - x^2)(1 - y^2)$. The grid used is 385×385 , leading to a total of 148,225 unknowns in the system. The initial and converged solutions are shown in Figure 6.9. The example, SNES ex15.c, can be run directly using a default PETSc installation.

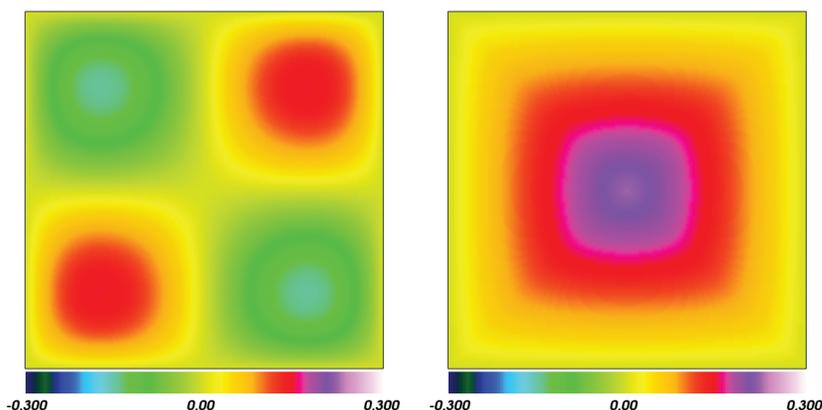


Fig. 6.9 Initial and converged solutions to the $p = 5$ p -Laplacian.

The command line used for these experiments is

```
./ex15 -da_refine 7 -da_overlap 6 -p 5.0 -lambda 0.0 -source 0.1
```

Table 6.9 Solvers for the p -Laplacian problem. “SubPC” denotes the linear solver at the block level.

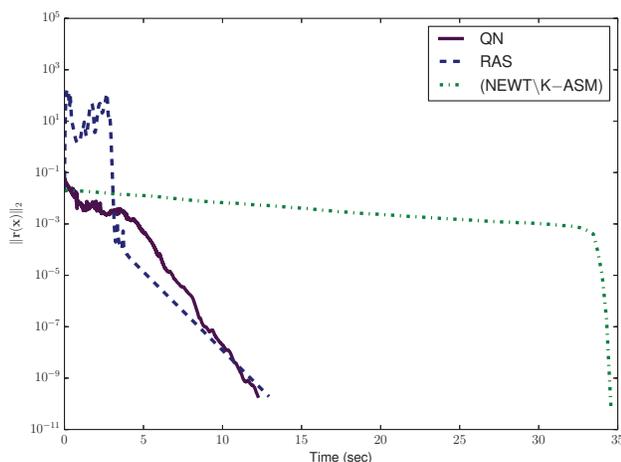
| Name | Solver | LPC | NPC | Side | SubPC | LS |
|----------------------------|------------|-----|------|------|-------|----|
| NEWT\K – ASM | NEWT\K | ASM | – | – | LU | BT |
| QN | QN | – | – | – | – | CP |
| RAS | RAS | – | – | – | LU | CP |
| RAS + (NEWT\K – ASM) | RAS/NEWT\K | ASM | – | – | LU | BT |
| RAS * (NEWT\K – ASM) | RAS/NEWT\K | ASM | – | – | LU | BT |
| ASPIN | NEWT\K | – | NASM | L | LU | BT |
| NRICH – _L (RAS) | NRICH | – | RAS | L | LU | CP |
| QN – _L (RAS) | QN | – | RAS | L | LU | CP |

We concentrate on additive Schwarz and QN methods, as listed in Table 6.9. We will show that combination has distinct advantages. This problem has difficult local nonlinearity that may impede global solvers, so local solvers should be an efficient remedy. We also consider a composition of Newton’s method with RAS and nonlinear preconditioning of Newton’s method in the form of ASPIN. NASM, RAS, and the linear additive Schwarz method (ASM) all have single subdomains on each processor that overlap each other by six grid points. We use the L-BFGS variant of QN, as explained in section 4.5. Note that the function evaluation is inexpensive for this problem and methods that use many evaluations perform well. The advantage of such methods is exacerbated by the large number of iterations required by Newton’s method. All the NEWT\K solvers use an inner GMRES iteration with a relative tolerance of 10^{-5} . GMRES for ASPIN is set to have an inner tolerance of 10^{-3} .

Table 6.10 contains results for the uncomposed solvers. The default solver, NEWT\K – ASM, takes a large number of outer Newton iterations and inner GMRES – ASM iterations. In addition, the line search is consistently activated, causing an average of more than three function evaluations per iteration. Unpreconditioned QN is able to converge to the solution efficiently and will be hard to beat. QN stores up to 10 previous solutions and residuals. RAS set to do one local Newton iteration

Table 6.10 *QN, RAS, and (NEWT\K - ASM) results.*

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|----------------|-------|-------|-------|------|-----|------|-----|
| QN | 12.24 | 2960 | 0 | 5921 | – | – | – |
| RAS | 12.94 | 352 | 0 | 1090 | 352 | 352 | – |
| (NEWT\K - ASM) | 34.57 | 124 | 3447 | 423 | 124 | 3574 | – |

**Fig. 6.10** *QN, RAS, and (NEWT\K - ASM) convergence.***Table 6.11** *RAS * (NEWT\K - ASM) and RAS + (NEWT\K - ASM) results.*

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|----------------------|-------|-------|-------|------|-----|------|-----|
| RAS * (NEWT\K - ASM) | 9.69 | 24 | 750 | 142 | 48 | 811 | – |
| RAS + (NEWT\K - ASM) | 12.78 | 33 | 951 | 232 | 66 | 1023 | – |

per subdomain per outer iteration proves to be efficient on its own after some initial problems, as shown in Figure 6.10.

In Table 6.11 and Figure 6.11, we show how Newton's method may be dramatically improved by composition with RAS and NASM. The additive composition reduces the number of outer iterations substantially. The multiplicative composition is even more effective, reducing the number of outer iterations by a factor of 5 and decreasing runtime by a factor of around 4.

The results for left-preconditioned methods are shown in Table 6.12 and Figure 6.12. ASPIN is competitive, taking 13 iterations and having performance characteristics similar to those of the multiplicative composition solver listed in Table 6.11. The subdomain solvers for ASPIN are set to converge to a relative tolerance of 10^{-3} or 20 inner iterations. Underresolving the local problems provides an inadequate search direction and causes ASPIN to stagnate. The linear GMRES iteration is also converged to 10^{-3} .

The most impressive improvements can be achieved by using a RAS as a nonlinear preconditioner. Simple NRICH acceleration does not provide much benefit compared with raw RAS with respect to outer iterations, and the preconditioner applications in the line search cause significant overhead. However, QN using inexact RAS as the left-

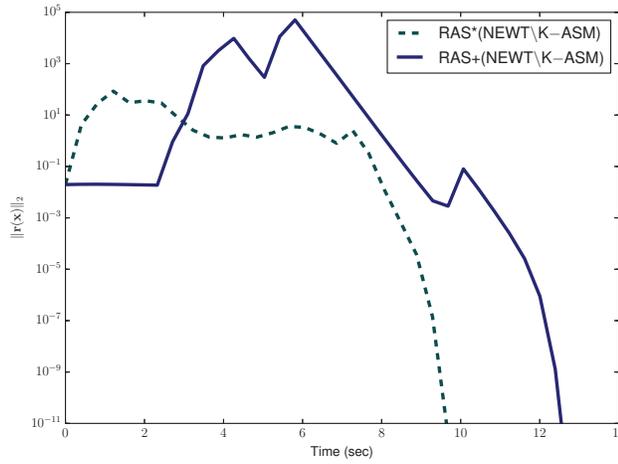


Fig. 6.11 $RAS^*(NEWT\backslash K-ASM)$ and $RAS+(NEWT\backslash K-ASM)$ convergence.

Table 6.12 $QN-LRAS$, $ASPIN$, and $NRICH-LRAS$ results.

| Solver | T | N. It | L. It | Func | Jac | PC | NPC |
|--------------|-------|-------|-------|------|-----|-----|-----|
| $QN-LRAS$ | 7.02 | 92 | 0 | 410 | 185 | 185 | 185 |
| $ASPIN$ | 9.30 | 13 | 332 | 307 | 179 | 837 | 19 |
| $NRICH-LRAS$ | 32.10 | 308 | 0 | 1889 | 924 | 924 | 924 |

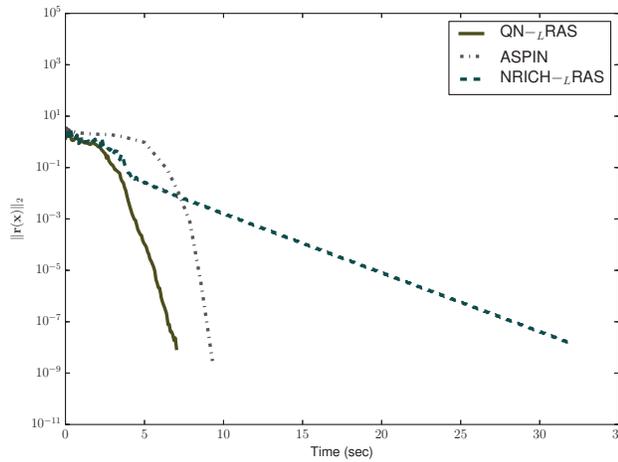


Fig. 6.12 $QN-LRAS$, $ASPIN$, and $NRICH-LRAS$ convergence.

preconditioned residual proves to be the most efficient solver for this problem, taking 185 Newton iterations per subdomain. Both NRICH and QN stagnate if the original residual is used in the line search instead of the preconditioned one. Subdomain QN methods [48] have been proposed before, but were built using block approximate

Jacobian inverses instead of working on the preconditioned system like ASPIN. As implemented here, QN $-_L$ RAS and ASPIN both construct left-preconditioned residuals and approximate preconditioned Jacobian constructions; both end up being very efficient.

7. Conclusion. The combination of solvers using nonlinear composition, when applied carefully, may greatly improve the convergence properties of nonlinear solvers. Hierarchical and multilevel inner solvers allow for high arithmetic intensity and low communication algorithms, making nonlinear composition a good option for extreme-scale nonlinear solvers.

Our experimentation, in this article and elsewhere, has shown that what works best when using nonlinear composition varies from problem to problem. Nonlinear composition introduces a slew of additional solver parameters at multiple levels of the hierarchy that may be tuned for optimal performance and robustness. A particular problem may be amenable to a simple solver, such as NCG, or to a combination of multilevel solvers, such as FAS * (NEWT\K - MG). The implementation in PETSc [3] allows for considerable flexibility in user choice of solver compositions.

Admittedly, we have been fairly conservative in the scope of our solver combinations. Our almost-complete restriction to combinations of a globalized method and a decomposition method is somewhat artificial in the nonlinear case. Without this restriction, however, the combinatorial explosion of potential methods would have quickly made the scope of this article untenable. Users may experiment, for their particular problem, with combinations of some number of iterations of arbitrary combinations of nonlinear solvers, with nesting much deeper than we explore here.

The use of nonlinear preconditioning allows for solvers that are more robust to difficult nonlinear problems. While effective linear preconditioners applied to the Jacobian inversion problem may speed the convergence of the inner solves, lack of convergence of the outer Newton's method may doom the solve. With nonlinear preconditioning, the inner and outer treatment of the nonlinear problem allows for very rapid solution. Nonlinear preconditioning and composition solvers allow for both efficiency and robustness gains, and the widespread adoption of these techniques would reap major benefits for computational science.

Acknowledgment. We thank Jed Brown for many meaningful discussions, suggestions, and sample code.

REFERENCES

- [1] ADVANCED MICRO DEVICES, *AMD Opteron 6200 Series Quick Reference Guide*, 2012.
- [2] D. G. ANDERSON, *Iterative procedures for nonlinear integral equations*, J. Assoc. Comput. Mach., 12 (1965), pp. 547–560.
- [3] S. BALAY, S. ABHYANKAR, M. F. ADAMS, J. BROWN, P. BRUNE, K. BUSCHELMAN, V. EIJKHOUT, W. D. GROPP, D. KAUSHIK, M. G. KNEPLEY, L. C. MCINNES, K. RUPP, B. F. SMITH, AND H. ZHANG, *PETSc Users Manual*, Tech. Rep. ANL-95/11, Revision 3.5, Argonne National Laboratory, 2014.
- [4] P. BIRKEN AND A. JAMESON, *On nonlinear preconditioners in Newton-Krylov methods for unsteady flows*, Internat. J. Numer. Methods Fluids, 62 (2010), pp. 565–573.
- [5] J. H. BRAMBLE, *Multigrid Methods*, Longman Scientific and Technical, Essex, UK, 1993.
- [6] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.
- [7] A. BRANDT, *Multigrid Techniques: 1984 Guide with Applications for Fluid Dynamics*, Tech. Rep. GMD-Studien Nr. 85, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, Germany, 1984.

- [8] A. BRANDT AND D. RON, *Multigrid solvers and multilevel optimization strategies*, in *Multilevel Optimization and VLSICAD*, Kluwer Academic, Norwell, MA, 2003, pp. 1–69.
- [9] W. L. BRIGGS, V. E. HENSON, AND S. F. MCCORMICK, *A Multigrid Tutorial*, 2nd ed., SIAM, Philadelphia, 2000.
- [10] J. BROWN AND P. BRUNE, *Low-rank quasi-Newton updates for robust Jacobian lagging in Newton-type methods*, in *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, ANS, La Grange Park, IL, 2013, pp. 2554–2565.
- [11] P. N. BROWN AND A. C. HINDMARSH, *Matrix-free methods for stiff systems of ODEs*, *SIAM J. Numer. Anal.*, 23 (1986), pp. 610–638.
- [12] P. N. BROWN AND Y. SAAD, *Convergence theory of nonlinear Newton–Krylov algorithms*, *SIAM J. Optim.*, 4 (1994), pp. 297–330.
- [13] C. G. BROYDEN, *A class of methods for solving nonlinear simultaneous equations*, *Math. Comp.*, 19 (1965), pp. 577–593.
- [14] R. BYRD, J. NOCEDAL, AND R. SCHNABEL, *Representations of quasi-Newton matrices and their use in limited memory methods*, *Math. Programming*, 63 (1994), pp. 129–156.
- [15] X.-C. CAI, *Nonlinear overlapping domain decomposition methods*, in *Domain Decomposition Methods in Science and Engineering XVIII*, *Lecture Notes in Comput. Sci.* 70, Springer, Berlin, 2009, pp. 217–224.
- [16] X.-C. CAI AND D. E. KEYES, *Nonlinearly preconditioned inexact Newton algorithms*, *SIAM J. Sci. Comput.*, 24 (2002), pp. 183–200.
- [17] X.-C. CAI AND X. LI, *Inexact Newton methods with restricted additive Schwarz based nonlinear elimination for problems with high local nonlinearity*, *SIAM J. Sci. Comput.*, 33 (2011), pp. 746–762.
- [18] N. N. CARLSON AND K. MILLER, *Design and application of a gradient-weighted moving finite element code I: In one dimension*, *SIAM J. Sci. Comput.*, 19 (1998), pp. 728–765.
- [19] T. F. CHAN AND K. R. JACKSON, *Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms*, *SIAM J. Sci. Statist. Comput.*, 5 (1984), pp. 533–542.
- [20] P. CRESTA, O. ALLIX, C. REY, AND S. GUINARD, *Nonlinear localization strategies for domain decomposition methods: Application to post-buckling analyses*, *Comput. Methods Appl. Mech. Engrg.*, 196 (2007), pp. 1436–1446.
- [21] Y. H. DAI AND Y. YUAN, *A nonlinear conjugate gradient method with a strong global convergence property*, *SIAM J. Optim.*, 10 (1999), pp. 177–182.
- [22] H. DE STERCK, *Steepest descent preconditioning for nonlinear GMRES optimization*, *Numer. Linear Algebra Appl.*, 20 (2013), pp. 453–471.
- [23] R. S. DEMBO, S. C. EISENSTAT, AND T. STEIHAUG, *Inexact Newton methods*, *SIAM J. Numer. Anal.*, 19 (1982), pp. 400–408.
- [24] J. E. DENNIS, JR. AND J. J. MORÉ, *Quasi-Newton methods, motivation and theory*, *SIAM Rev.*, 19 (1977), pp. 46–89.
- [25] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.
- [26] P. DEUFLHARD, R. FREUND, AND A. WALTER, *Fast secant methods for the iterative solution of large nonsymmetric linear systems*, *IMPACT Comput. Sci. Engrg.*, 2 (1990), pp. 244–276.
- [27] S. C. EISENSTAT AND H. F. WALKER, *Globally convergent inexact Newton methods*, *SIAM J. Optim.*, 4 (1994), pp. 393–422.
- [28] V. EYERT, *A comparative study on methods for convergence acceleration of iterative vector sequences*, *J. Comput. Phys.*, 124 (1996), pp. 271–285.
- [29] H.-R. FANG AND Y. SAAD, *Two classes of multisecond methods for nonlinear acceleration*, *Numer. Linear Algebra Appl.*, 16 (2009), pp. 197–221.
- [30] R. FLETCHER, *Practical Methods of Optimization, Volume 1*, Wiley, New York, 1987.
- [31] R. FLETCHER AND C. M. REEVES, *Function minimization by conjugate gradients*, *Comput. J.*, 7 (1964), pp. 149–154.
- [32] A. A. GOLDSTEIN, *On steepest descent*, *J. Soc. Indust. Appl. Math. Ser. A Control*, 3 (1965), pp. 147–151.
- [33] W. HACKBUSCH, *Comparison of different multi-grid variants for nonlinear equations*, *Z. Angew. Math. Mech.*, 72 (1992), pp. 148–151.
- [34] V. E. HENSON, *Multigrid methods for nonlinear problems: An overview*, *Proc. SPIE*, 5016 (2003), pp. 36–48.
- [35] M. R. HESTENES AND E. STEIFEL, *Methods of conjugate gradients for solving linear systems*, *J. Research Nat. Bur. Standards*, 49 (1952), pp. 409–436.
- [36] J. D. HUNTER, *Matplotlib: A 2d graphics environment*, *Comput. Sci. Engrg.*, 9 (2007), pp. 90–95.

- [37] F.-N. HWANG AND X.-C. CAI, *A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier-Stokes equations*, J. Comput. Phys., 204 (2005), pp. 666–691.
- [38] C. T. KELLEY, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, 1995.
- [39] C. T. KELLEY AND D. E. KEYES, *Convergence analysis of pseudo-transient continuation*, SIAM J. Numer. Anal., 35 (1998), pp. 508–523.
- [40] H. KLIE AND M. F. WHEELER, *Nonlinear Krylov-Secant Solvers*, Tech. Rep., Department of Mathematics and Institute of Computational Engineering and Sciences, University of Texas at Austin, 2006.
- [41] D. A. KNOLL AND P. R. MCHUGH, *Enhanced nonlinear iterative techniques applied to a nonequilibrium plasma flow*, SIAM J. Sci. Comput., 19 (1998), pp. 291–301.
- [42] D. A. KNOLL AND D. E. KEYES, *Jacobian-free Newton-Krylov methods: A survey of approaches and applications*, J. Comput. Phys., 193 (2004), pp. 357–397.
- [43] P. LADEVÈZE, J.-C. PASSIEUX, AND D. NÉRON, *The LATIN multiscale computational method and the proper generalized decomposition*, Comput. Methods Appl. Mech. Engrg., 199 (2010), pp. 1287–1296.
- [44] P. LADEVÈZE AND J. SIMMONDS, *Nonlinear Computational Structural Mechanics: New Approaches and Non-incremental Methods of Calculation*, Mechanical Engineering Series, Springer, New York, 1999.
- [45] P. J. LANZKRON, D. J. ROSE, AND J. T. WILKES, *An analysis of approximate nonlinear elimination*, SIAM J. Sci. Comput., 17 (1996), pp. 538–559.
- [46] D. C. LIU AND J. NOCEDAL, *On the limited memory BFGS method for large scale optimization*, Math. Programming, 45 (1989), pp. 503–528.
- [47] P. A. LOTT, H. F. WALKER, C. S. WOODWARD, AND U. M. YANG, *An accelerated Picard method for nonlinear systems related to variably saturated flow*, Adv. Water Res., 38 (2012), pp. 92–101.
- [48] J. M. MARTÍNEZ, *SOR-secant methods*, SIAM J. Numer. Anal., 31 (1994), pp. 217–226.
- [49] H. MATTHIES AND G. STRANG, *The solution of nonlinear finite element equations*, Internat. J. Numer. Methods Engrg., 14 (1979), pp. 1613–1626.
- [50] S. G. NASH, *A multigrid approach to discretized optimization problems*, Optim. Methods Softw., 14 (2000), pp. 99–116.
- [51] J. NOCEDAL, *Updating quasi-Newton matrices with limited storage*, Math. Comp., 35 (1980), pp. 773–782.
- [52] J. NOCEDAL AND S. J. WRIGHT, *Numerical Optimization*, Springer, New York, 1999.
- [53] C. W. OOSTERLEE AND T. WASHIO, *Krylov subspace acceleration of nonlinear multigrid with application to recirculating flow*, SIAM J. Sci. Comput., 21 (2000), pp. 1670–1690.
- [54] E. POLAK AND G. RIBIERE, *Note sur la convergence de méthodes de directions conjuguées*, Rev. Française Informat. Recherche Opérationnelle, 3 (1969), pp. 35–43.
- [55] P. PULAY, *Convergence acceleration of iterative sequences: The case of SCF iteration*, Chem. Phys. Lett., 73 (1980), pp. 393–398.
- [56] A. QUARTERONI AND A. VALLI, *Domain Decomposition Methods for Partial Differential Equations*, Oxford Science Publications, Oxford, UK, 1999.
- [57] P. RAMACHANDRAN AND G. VAROQUAUX, *Mayavi: 3D Visualization of Scientific Data*, Comput. Sci. Engrg., 13 (2011), pp. 40–51.
- [58] Y. SAAD, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.
- [59] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
- [60] M. H. SCOTT AND G. L. FENVES, *A Krylov subspace accelerated Newton algorithm*, in Proceedings of the 2003 ASCE Structures Congress, L. N. Lowes and G. R. Miller, eds., 2003, pp. 45–55.
- [61] D. F. SHANNO, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp., 24 (1970), pp. 647–656.
- [62] J. R. SHEWCHUK, *An Introduction to the Conjugate Gradient Method without the Agonizing Pain*, Tech. Rep., Carnegie Mellon University, Pittsburgh, 1994.
- [63] B. SMITH, *Domain decomposition methods for partial differential equations*, ICASE LARC Interdisciplinary Ser. Sci. Engrg., 4 (1997), pp. 225–244.
- [64] B. F. SMITH, P. BJØRSTAD, AND W. D. GROPP, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, Cambridge, UK, 1996.
- [65] B. F. SMITH AND X. TU, *Domain decomposition*, in Encyclopedia of Applied and Computational Mathematics, B. Engquist, ed., Springer, New York, 2015.

- [66] M. SMOOKE AND R. MATTHEIJ, *On the solution of nonlinear two-point boundary value problems on successively refined grids*, Appl. Numer. Math., 1 (1985), pp. 463–487.
- [67] A. TOSELLI AND O. B. WIDLUND, *Domain Decomposition Methods: Algorithms and Theory*, Springer Ser. Comput. Math. 34, B. Engquist, ed., Springer, New York, 2005.
- [68] U. TROTTEMBERG, C. OOSTERLEE, AND A. SCHÜLLER, *Multigrid*, Academic Press, San Diego, CA, 2001.
- [69] R. S. TUMINARO, H. F. WALKER, AND J. N. SHADID, *On backtracking failure in Newton–GMRES methods with a demonstration for the Navier–Stokes equations*, J. Comput. Phys., 180 (2002), pp. 549–558.
- [70] H. F. WALKER AND P. NI, *Anderson acceleration for fixed-point iterations*, SIAM J. Numer. Anal., 49 (2011), pp. 1715–1735.
- [71] H. F. WALKER, C. S. WOODWARD, AND U. M. YANG, *An accelerated fixed-point iteration for solution of variably saturated flow*, in Proceedings of the 18th International Conference on Water Resources, J. Carrera, ed., Barcelona, 2010.
- [72] T. WASHIO AND C. OOSTERLEE, *Krylov subspace acceleration for nonlinear multigrid schemes*, Electron. Trans. Numer. Anal., 6 (1997), pp. 271–290.
- [73] P. WESSELING, *An Introduction to Multigrid Methods*, R. T. Edwards, Philadelphia, 2004.
- [74] P. WOLFE, *Convergence conditions for ascent methods*, SIAM Rev., 11 (1969), pp. 226–235.
- [75] P. WRIGGERS, *Nonlinear Finite Element Methods*, Springer, New York, 2008.