SIAM J. SCI. COMPUT. © 2018 Society for Industrial and Applied Mathematics
Vol. 40, No. 3, pp. B812–B833

A NONLINEAR ALGEBRAIC MULTIGRID FRAMEWORK FOR THE POWER FLOW EQUATIONS[*]

C. PONCE[†], D. S. BINDEL[‡], AND P. S. VASSILEVSKI[†]

Abstract. Multigrid is a highly scalable class of methods most often used for solving large linear systems. In this paper we develop a nonlinear algebraic multigrid framework for the *power flow equations*, a complex quadratic system of the form $\mathrm{diag}(\boldsymbol{v})\overline{Y\boldsymbol{v}} = \boldsymbol{s}$, where Y is approximately a complex scalar rotation of a real graph Laplacian. This is a standard problem that needs to be solved repeatedly during power grid simulations. A key difference between our multigrid framework and typical multigrid approaches is the use of a novel multiplicative coarse-grid correction to enable a dynamic multigrid hierarchy. We also develop a new type of smoother that allows one to coarsen together the different types of nodes that appear in power grid simulations. In developing a specific multigrid method, one must make a number of choices that can significantly affect the method's performance, such as how to construct the restriction and interpolation operators, what smoother to use, and how aggressively to coarsen. In this paper, we make simple but reasonable choices that result in a scalable and robust power flow solver. Experiments demonstrate this scalability and show that it is significantly more robust to poor initial guesses than current state-of-the-art solvers.

Key words. power flow, algebraic multigrid, nonlinear multigrid, multiplicative correction

AMS subject classifications. 15-04, 15B99

DOI. 10.1137/16M1109965

1. Introduction.

The power flow problem, also known as the load flow problem, is a nonlinear algebraic system of equations at the heart of power grid simulations. The power flow equations are a set of complex, quadratic equations that relate the voltages at each node in a power network to the amount of power entering or leaving at each node [15]. The simulation of a power network in "steady state" requires the solution of the power flow equations, while time-domain simulation requires solving the power flow equations at each time step in a differential-algebraic problem [39]. Power flow is an inherently nonlinear problem, and although power networks are physical networks, modeling data often does not include physical locations. Thus,

[*]Submitted to the journal's Computational Methods in Science and Engineering section December 29, 2016; accepted for publication (in revised form) March 8, 2018; published electronically May 24, 2018.

http://www.siam.org/journals/sisc/40-3/M110996.html

Funding: The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award DE-AR0000230. The information, data, or work presented herein was funded in part by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. The first author's research was conducted with United States Government support under and awarded by DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. The third author's work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.

[†]Lawrence Livermore National Laboratory, Livermore, CA 94550 (ponce11@llnl.gov, vassilevski1@llnl.gov).

[‡]Department of Computer Science, Cornell University, Ithaca, NY 14853 (bindel@cs.cornell.edu).

it is typically an algebraic problem rather than a geometric one. In this paper, we present a method for solving the power flow problem based on the ideas of algebraic multigrid, a methodology typically used for solving linear systems of equations.

The earliest methods for solving the power flow problem were nonlinear Gauss–Seidel methods, known as *Y-Matrix methods*, developed in the 1950s [38]. These methods converge slowly but were suitable for early computers with limited memory. They were surpassed in the 1960s by Newton–Raphson methods with the development of efficient sparse Gaussian elimination techniques that made Jacobian factorization practical [41].

Today Newton–Raphson is still the workhorse method for the power flow problem [15], and many variants of the method are used today. The *Fast-Decoupled Load Flow* is a quasi-Newton method that makes assumptions about typical operating conditions to decouple the Jacobian into two smaller matrices [40]. The *DC Load Flow* method makes further assumptions by completely neglecting voltage magnitude changes to reduce the problem to a single linear solve [15]. The development of continuation methods enabled power flow computation near points of system instability [2].

A great deal of literature exists on solving the power flow equations, typically focusing on specific issues. Some work, for example, focuses on solving the power flow equations in the context of the distribution network, where high resistance/reactance ratios make the problem ill-conditioned [20, 42, 8]. Other work focuses on improving the efficiency of Newton–Raphson's Jacobian solve through developments in sparse direct solvers [23, 29, 35] and Krylov solvers [22, 18, 48]. In [18], the authors suggest algebraic multigrid as a preconditioner in a Newton–Krylov iteration.

One problem that often plagues Newton–Raphson is that its region of convergence is smaller than one might hope, especially when the system is heavily loaded. Furthermore, its region of convergence is a fractal [11], making it difficult to predict when Newton–Raphson will converge. This is problematic not only because it makes simulation difficult, but also because often one does not know if the solver diverged because the problem has no solution (implying a system collapse), or simply because Newton–Raphson didn't find the solution. Various robust solvers have been developed in response to this problem. In some cases, using a method such as the DC Load Flow can provide an initial starting guess within the region of convergence [37]. One common class of approaches is homotopy-based methods, in which a series of problems are solved that shift gradually between an easy problem and the desired, harder problem [31, 2, 32]. Other approaches typically involve some robust variation of Newton–Raphson. In [19], the authors present a variation of Newton–Raphson in which they optimally scale the update to minimize the residual norm. Others involve a Runge–Kutta-based variant [31] or a Levenberg–Marquardt method [27]. The methods that we present in this paper are particularly robust to poor starting conditions, making them desirable methods for ill-conditioned problems.

The framework and methods that we develop in this paper for solving the power flow problem are based on the ideas of algebraic multigrid methods. Multigrid methods work by constructing a hierarchy of "coarse" problems such that each problem is a reduced-size approximation of the one before it. By passing the current iterate between various levels of the hierarchy and performing some simple iteration step (such as Gauss–Seidel) on each level, one creates an efficient solver that makes global changes at the coarse levels and local changes at the fine levels [7]. Multigrid methods have seen great success as a highly scalable tool for the solution of linear systems of equations, especially for the ones that arise from elliptic partial differential equations [5].

Linear multigrid methods have also been applied to linear systems that do not arise from partial differential equations, and often do not even have an underlying geometry [7]. This has been most successful for graph Laplacian matrices, which are in many ways an algebraic analogue to the geometric Poisson problem [16, 33, 34, 10]. Algebraic multigrid, with its ever expanding area of application and intent for extreme scale parallel scalability, is still an active area of research [14, 6, 46, 17, 4].

Multigrid methods have been used in somewhat more limited fashion to solve nonlinear problems as well. The nonlinear Full Approximation Scheme (FAS) is a well-known multigrid framework for solving nonlinear partial differential equations [7]. Nonlinear multigrid methods have also been used to solve eigenvalue problems [24, 9]. Recent results for solving nonlinear finite element elliptic equations by specialized element based AMG (referred to as AMGe) are found in [26].

A related method, also exploiting the divide-and-conquer approach, similarly to multigrid (which can be viewed as an implicit divide-and-conquer method in the *frequency* domain) is the domain decomposition approach. Some recent work has made use of domain decomposition approaches to accelerate power flow solvers. In [1], the authors present an overlapping restricted additive Schwarz preconditioner for use in a Newton-GMRES solver of the power flow equations. In [3, 21, 30], researchers exploit domain decomposition in dynamic simulations.

However, a defining feature of multigrid methods is the construction of a hierarchy of “coarse” approximate problems that trade off between problem size and fidelity of approximation. While multigrid is not the only class of methods with this feature (e.g., see [36] for graph Laplacian linear solvers), achieving algorithmic scalability typically requires the use of some coarse problem hierarchy. With the exception of the AMG preconditioner in [18], the solvers in the above papers do not make use of a hierarchy of coarse problems. Recently, [28] also proposed the use of multigrid in the time domain for dynamic power grid simulations.

The nonlinear algebraic multigrid framework and methods developed in this paper do not use algebraic multigrid to solve the Jacobian of a Newton–Raphson iterate as in [18]. Rather, it builds a multigrid hierarchy for the power flow problem directly. Because of this, the method differs from most modern power flow solvers in that it is unrelated to Newton–Raphson; in fact, it is closer to the earlier Gauss–Seidel method. It builds a hierarchy of coarse power flow problems and uses a variant of the standard Gauss–Seidel method to smooth at each level. These methods also differ from most multigrid methods in that they work on a nonlinear problem, and in that it performs its updates using a novel multiplicative correction rather than the standard additive correction.

Multigrid is really a methodology rather than a method, and one can make a variety of choices, such as the smoother or the coarsening approach, to construct methods within a multigrid framework that have different advantages and disadvantages. We present a few simple possibilities for some of these choices here. These simple choices result in a method that is both highly scalable and significantly more robust to poor initial guesses than Newton–Raphson and its variants. The choice of a Jacobi smoother also results in, to the best of the authors’ knowledge, the first successful use of a Jacobi iteration in solving the power flow problem.

The rest of this paper is organized as follows. In section 2, we introduce the power flow equations. In section 3, we present our multigrid method. In section 4, we present numerical experiments, and in section 5, we conclude.

2. The power flow equations.

The power flow problem is a standard problem in power engineering [15]. One can represent a power network as a graph; power enters or leaves a network at nodes and flows between nodes along edges (with some "line loss"; power is not conserved in edges). These edges are transmission lines and are weighted by their electrical admittance. When simulating a power network that is in "steady state," power engineers typically make a set of simplifying assumptions.

1. *Complex numbers for voltages and currents.* Power grids use alternating current, in which all voltages and currents in a network oscillate at the same frequency (e.g., 60 Hz in the United States). We may represent these sine waves as complex phasors.
2. *Dimensionless units.* All quantities in a power network model are normalized, resulting in a unitless representation of electrical voltage, current, and power. The result is that all nodes have voltage magnitudes near 1.0. Voltage phase angles vary more than magnitudes, but still remain near 0.0.[1]

Under these assumptions, we can model power networks by representing nodal voltages, transmission line admittances, and network power injections as complex numbers.

To set up the model equations, we first define the *admittance matrix* $Y \in \mathbb{C}^{n \times n}$. The off-diagonal element Y_{ij} is the negative of electrical admittance of the transmission line connecting nodes i and j, or zero if there is no such line. The diagonal elements Y_{ii} are the sum of all connected electrical admittances plus a small additional *shunt admittance* term h_i [15]. It is nearly symmetric, except on lines with transformers, which introduce asymmetries in phase angle but not magnitude. The result is that Y is nearly a complex graph Laplacian.

Let $\boldsymbol{v} \in \mathbb{C}^n$ denote a vector of complex voltages at each node and let $\boldsymbol{s} \in \mathbb{C}^n$ denote a vector of complex power injections at each node. In addition let $D_{\boldsymbol{v}} = \operatorname{diag}(\boldsymbol{v})$ and let $\overline{\boldsymbol{x}}$ denote the complex conjugate of a vector $\boldsymbol{x}$. Then, under the assumptions stated above, the power flow equations are

$$D_{\boldsymbol{v}} \overline{Y \boldsymbol{v}} = \boldsymbol{s}. \tag{1}$$

Associated with each node is a nodal voltage v_i and a power injection s_i. There are three types of nodes in a power flow model, each associated with a different set of unknowns.

1. *PQ nodes.* In a PQ node i, the power injection s_i is known a priori, and we wish to solve for the unknown voltage v_i. Approximately 90% of the nodes in a network are PQ nodes [25].
2. *PV nodes.* In a PV node i, the real component of the power injection, $\operatorname{Re}(s_i)$, is known, as well as the voltage magnitude, which we denote $|v_i^{sp}|$. Here we wish to solve for the unknown voltage phase angle $\theta_i = \arg(v_i)$. PV nodes represent generators and usually make up approximately 10% of the nodes in a network [25].
3. *The slack node.* There is always one slack node. The complex voltage v_i is known a priori. Without a slack node, the power flow equations would be invariant to scalar complex rotations of $\boldsymbol{v}$ and consequently underdetermined.

Note that the power injection of the slack node and the imaginary component of the power injections of the PV nodes (that is, the reactive power injection) are also unknown. However, they are simple to compute once the nodal voltages are found.

[1]For this reason, many iterative solvers use a starting iterate of $\boldsymbol{v} = \mathbf{1} + \imath \mathbf{0}$.

3. Multigrid framework and solver. The guiding principle behind the development of our multigrid framework comes from the above problem's similarity to the real linear problem $L\boldsymbol{x} = \boldsymbol{f}$ when L is a graph Laplacian. In the linear case, a smoother such as Gauss–Seidel or Jacobi is effective at reducing error modes associated with large eigenvalues of L, while those modes associated with small eigenvalues of L tend to be slowly varying; that is, small-eigenvalue error modes tend to be locally nearly constant.

Problem (1) can be rewritten as $Y\boldsymbol{v} = \overline{D_{\boldsymbol{v}}^{-1}\boldsymbol{s}}$, so it is qualitatively similar to the linear problem, especially given that v_i is near 1.0 for each i. Thus, we establish the heuristic that, like the linear case, we expect Gauss–Seidel and Jacobi smoothers to be complementary to coarsening approaches that assume nearly constant local error.

We first present two simplifying observations that we will find useful:

- Most off-diagonal nonzeros are the result of transmission lines. Transmission lines tend to be physically similar but of varying lengths, so these off-diagonal entries have similar complex phase angles with varying magnitudes. The result is that

$$Y \approx zL, \tag{2}$$

where z is a complex scalar rotation and L is a real-valued graph Laplacian.
- Voltage phase angles between neighboring nodes typically are quite close. Therefore, if i and j are neighbors,

$$\exp(\imath(\theta_i - \theta_j)) \approx 1 + \imath(\theta_i - \theta_j). \tag{3}$$

3.1. Coarsening. Our multigrid framework must allow us to coarsen such that the coarse problem has the same structure as the original problem, enabling recursion. That is, the coarse problem must

- be of the form $D_{\boldsymbol{v}}\overline{Y\boldsymbol{v}} = \boldsymbol{s}$,
- have a matrix Y that is approximately zL, where $z \in \mathbb{C}$ and L is a real-valued graph Laplacian, though it need not be exactly symmetric, and
- have PQ nodes, PV nodes, and a single slack node.

We present a multiplicative approach to coarsening instead of the standard additive approach. For the linear problem $L\boldsymbol{x} = \boldsymbol{f}$, one typically constructs restriction and interpolation matrices R and P, and then computes a coarse-grid correction by solving the problem

$$RLP\boldsymbol{x}_C = R\boldsymbol{f} - RL\boldsymbol{x} \tag{4}$$

for $\boldsymbol{x}_C$ and performing the update

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + P\boldsymbol{x}_C. \tag{5}$$

If L is a graph Laplacian and one chooses $R = P^T$, the matrix P^TLP is also symmetric and positive-definite, and if P is piecewise-constant, then P^TLP is a graph Laplacian as well. For general conditions for preserving the structure of graph Laplacian, see sections 3 and 4 of [10].

The choice of restriction and interpolation matrices is one of the many choices in developing a specific multigrid method, and a great deal of literature exists on this question (see, e.g., [44, 7, 13]). The restriction and interpolation matrices determine what subspace of the original problem is captured in the coarse problems, and this can significantly impact the effectiveness of a method. A deep exploration of possible restriction and interpolation matrices is outside the scope of this paper; here we use $R = P^T$ and choose P to consist of ones and zeros (often called a *piecewise constant* interpolation matrix). These choices enable convenient algebraic tricks discussed below, though more general choices (such as blocked operators) are likely possible as well.

In our case, the traditional additive update in (5) presents a problem. Applying the above coarse-grid correction to the power flow equations results in the coarse problem

$$P^T\overline{\mathrm{diag}(\boldsymbol{v} + P\boldsymbol{v}_C)}Y(\boldsymbol{v} + P\boldsymbol{v}_C) = P^T\overline{\boldsymbol{s}}.$$

A simple expansion of terms shows that we now have a problem with two linear terms, one in $\boldsymbol{v}_C$ and one in $\overline{\boldsymbol{v}_C}$, so this coarse problem no longer has the same structure as problem (1). Therefore, this construction violates the requirement that the coarse problem be of the form $D_{\boldsymbol{v}}\overline{Y\boldsymbol{v}} = \boldsymbol{s}$.

To avoid this problem, we introduce the *multiplicative coarse-grid correction.* Rather than perform a correction of the form

$$\boldsymbol{v} \leftarrow \boldsymbol{v} + P\boldsymbol{v}_C, \tag{6}$$

we perform one of the form

$$\boldsymbol{v} \leftarrow D_{\boldsymbol{v}}P\boldsymbol{v}_C. \tag{7}$$

To analyze this type of update, we first present a useful lemma.

LEMMA 1. *Let $P \in \{0,1\}^{n\times\ell}$, $\ell \le n$, denote a matrix with exactly one* 1 *in each row and let $\boldsymbol{x} \in \mathbb{C}^\ell$. Then*

$$P^T \mathit{diag}(P\boldsymbol{x}) = \mathit{diag}(\boldsymbol{x})P^T.$$

Proof. Assume without loss of generality that the rows of P are ordered by the columns of their nonzero elements; that is, assume that

$$P^T = \begin{bmatrix} 1 & \cdots & 1 & & & & & & \\ & & & 1 & \cdots & 1 & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \cdots 1 \end{bmatrix}.$$

Then

$$P\boldsymbol{x} = \begin{bmatrix} x_1 & \cdots x_1 & \cdots & x_\ell & \cdots & x_\ell \end{bmatrix}^T,$$

and so

$$P^T\mathrm{diag}(P\boldsymbol{x}) = \begin{bmatrix} x_1 & \cdots & x_1 & & & & & & \\ & & & x_2 & \cdots & x_2 & & & \\ & & & & & & \ddots & & \\ & & & & & & & x_\ell & \cdots x_\ell \end{bmatrix} = \mathrm{diag}(\boldsymbol{x})P^T. \quad \square \tag{8}$$

Now, given the multiplicative update in (7), the coarse-grid problem to solve is

$$\begin{aligned} & P^T\overline{D_{\boldsymbol{v}}D_{P\boldsymbol{v}_C}}YD_{\boldsymbol{v}}P\boldsymbol{v}_C = P^T\overline{\boldsymbol{s}} \\ \Longrightarrow\ & \overline{D_{\boldsymbol{v}_C}}P^T\overline{D_{\boldsymbol{v}}}YD_{\boldsymbol{v}}P\boldsymbol{v}_C = P^T\boldsymbol{s} \qquad \text{(Lemma (1))} \\ \Longrightarrow\ & \overline{D_{\boldsymbol{v}_C}}Y_C\boldsymbol{v}_C = \overline{\boldsymbol{s}_C}, \end{aligned} \tag{9}$$

where $Y_C = \overline{P^T D_{\boldsymbol{v}}} Y D_{\boldsymbol{v}} P$ and $\boldsymbol{s}_C = P^T \overline{\boldsymbol{s}}$.

This problem has the same structure as problem (1) and represents a valid power flow problem. The system is not quite symmetric, due to the complex conjugation in Y_C. However, it is still nearly symmetric, as neighboring nodes tend to have phase angles that are close to each other. As we show later, we never aggregate together two nodes of different types, so we have distinct PQ, PV, and slack nodes. Therefore, the coarse problem has the required elements to enable recursion and therefore a multigrid hierarchy.

We make a few remarks about the multiplicative coarse-grid correction:

- The multiplicative update corresponds to an *additive* update for the complex phase angles. In addition, because all nodes have magnitude near 1.0, it is approximately an additive update for voltage magnitudes as well.
- If $\boldsymbol{v}$ already solves the problem, then the solution to the coarse-grid problem is $\boldsymbol{v}_C = \mathbf{1}$. For this reason, any iterative method on the coarse grid should begin near $\mathbf{1}$ rather than $\mathbf{0}$. Many standard power flow solvers use a starting iterate of $\mathbf{1}$, so this behavior is similar to power flow problems as well.

3.2. Selection of coarse grid. Many coarsening procedures are possible within our multigrid framework. There are two requirements:

- The resulting interpolation matrix P must be a piecewise-constant operator.
- The coarsening procedure does not aggregate nodes of different types. However, for smoothing purposes later, we keep track of PQ and PV coarse nodes that we would "like" to have aggregated had they been of the same type. The result is a sequence of sets of coarse nodes of the form

$$\{\{i_1\}, \{i_2\}, \{i_3, i_4\}, \{i_5\}, \{i_6, i_7, i_8\}, \ldots\}, \tag{10}$$

  where the size of each set is determined by the number of types of nodes that we would have liked to coarsen together.

Within these framework requirements, many choices are possible. In this paper, we make a specific choice by following our guiding principle that problem (1) is qualitatively similar to a linear problem, so we choose to construct our coarse grid under the heuristic assumption that nodes i and j are likely to have similar values if $|y_{ij}|$ is large.

This is similar to the reasoning used in [43]. We use a related algorithm, the greedy matching algorithm presented in Algorithm 6.1 of [45], where we order edges by $|y_{ij}|$. This produces a piecewise-constant operator P. It is common in algebraic multigrid to then produce a smoothed version of operator P, but we do not do that here, as Lemma 1 relies on having a piecewise-constant interpolation operator.

Another option would be to use Algorithm 2 of [43] without the subsequent smoothing. Algorithm 6.1 of [45] doesn't require any parameter selection, however, so we use that.

One could also coarsen more aggressively by employing a strategy to aggregate more than two nodes at a time. There is typically a tradeoff to make here in terms of coarsening aggressiveness. More aggressive coarsening results in multigrid cycles that are less expensive but also potentially less effective. Optimal coarsening aggressiveness is often problem-dependent.

Rather than choosing the aggregates using Y, one could use $\overline{D_{\boldsymbol{v}}}YD_{\boldsymbol{v}}$. This would likely more accurately reflect how power is flowing. However, because $\boldsymbol{v}$ changes, this would involve repeating the hierarchy construction process each iteration and so would be expensive. Furthermore, because all $|v_i|$ are near 1.0, we do not expect the aggregation to change very often, so we do not do this.

While we need to recompute the entries of Y_C at each iteration, its sparsity structure does not change, so we can precompute the matrix structure for Y_C and simply fill in the matrix values at each iteration. Then this is only slightly more expensive than the traditional approach of recomputing the right-hand side at each iteration (which, in our case, remains constant).

3.3. Smoothing. As with coarsening approaches, many different smoothers are possible. The term "smoother" comes from linear geometric multigrid methods in which one employs a simple method that literally smooths out the error profile. More generally, a smoother is a simple iterative method that reduces the types of error that will be projected away at the next coarser level.

In this paper, we base our specific smoothing approach on the Gauss–Seidel method typically discussed in power engineering literature [38]. In the previous section we discussed pairs of nodes that we would like to update together, despite them being of different types. The traditional Gauss–Seidel method does not allow for this, as it treats PQ and PV nodes differently. We therefore modify that method to allow us to update together parts of the PQ and PV nodes that we would have liked to aggregate had they been the same type. The typical power flow Gauss–Seidel method cycles through each node i and solves

$$y_{ii}v_i^{new} + \sum_{j\neq i} y_{ij}v_j = \overline{s_i/v_i}$$

for v_i^{new}. If i is a PV node, power flow Gauss–Seidel pre- and postprocesses this calculation to estimate s_i and to maintain the known voltage magnitude.

Now, PV nodes have known voltage magnitude, so we cannot update PQ and PV nodes' voltage magnitude together; however, we can update their phase angles together. To that end, we develop a Gauss–Seidel smoother that updates a node's voltage magnitude and phase angle separately. We present the smoother here and discuss in section 3.4 how to apply it at a coarse level with paired PQ and PV nodes.

For each of these updates, we do not use the entire complex residual at node i, but rather project some rotation of that residual onto the real axis. That is, we update the phase angle θ_i or the magnitude $|v_i|$ so that

$$\operatorname{Re}(\beta r_i) = 0, \qquad r_i = (-\overline{s_i} + \overline{v_i}(Y\boldsymbol{v})_i),$$

for some $\beta \in \mathbb{C}$. The question, then, is what to use for β for the phase angle and for the magnitude of v_i.

Recalling observations (2) and (3), note that

$$\begin{aligned}
\overline{zv_i}\left(-\overline{s_i/v_i} + (Y\boldsymbol{v})_i\right) &\approx \overline{z}\left(-\overline{s_i} + \overline{v_i}(zL\boldsymbol{v})_i\right) \\
&= -\overline{zs_i} + \overline{v_i}(L\boldsymbol{v})_i \\
&= -\overline{zs_i} + \sum_{j=1}^{n} L_{ij}|v_i||v_j|e^{\imath(\theta_j-\theta_i)} \\
&\approx -\overline{zs_i} + \sum_{j=1}^{n} L_{ij}|v_i||v_j|(1+\imath(\theta_j-\theta_i)) \qquad \text{(observation (3))} \\
&= \left(-\operatorname{Re}(\overline{zs_i}) + \sum_{j=1}^{n} L_{ij}|v_i||v_j|\right) \\
&\quad + \imath\left(-\operatorname{Im}(\overline{zs_i}) + \sum_{j=1}^{n} L_{ij}|v_i||v_j|(\theta_j-\theta_i)\right).
\end{aligned}$$

The imaginary term in the above formula is (approximately) the component of the residual equation that is most strongly affected by the phase angles, and the real term is (approximately) the component of the residual equation that is least affected by phase angles. Therefore, we use $\beta = \imath\overline{zv_i}$ when updating phase angles and $\beta = \overline{zv_i}$ when updating voltage magnitudes.

The complex scalar z is not just a theoretical object; we must actually compute it, at least approximately. There are various ways to do this; we simply take the mean of the diagonal of Y.

We update separately a node's voltage phase angle and magnitude by setting the imaginary and real components of $\overline{zv_i}r_i$, respectively, to zero, and updating v_i to match that component of the current injection. Mathematically, we write this in the following way:

- To update the phase angle of v_i by an amount $\lambda \in \mathbb{R}$, we set

$$\operatorname{Im}\left[\overline{z}\left(\overline{s_i} - \overline{v_i}\sum_{j\neq i} Y_{ij}v_j - \overline{v_i}Y_{ii}v_i e^{\imath\lambda}\right)\right] = 0. \tag{11}$$

  Solving for $e^{\imath\lambda}$ gives two possibilities:

$$e^{\imath\lambda} = \frac{\imath c \pm \sqrt{|x|^2 - c^2}}{x}, \tag{12}$$

  where

$$x = \overline{z}|v_i|^2 Y_{ii}, \qquad c = \operatorname{Im}\left[\overline{z}\left(\overline{s_i} - \overline{v_i}\sum_{j\neq i} Y_{ij}v_j\right)\right].$$

  We select the option that causes the smallest angle change and then update v_i with the rotation

$$v_i \leftarrow v_i e^{\imath\lambda}. \tag{13}$$

Algorithm 1. Split Gauss–Seidel iteration.

```
1: procedure SplitGaussSeidelIteration($\boldsymbol{v}, \boldsymbol{s}, Y$)
2:     for $i = 1, \ldots, n$ do
3:         if $i$ is a PV node then
4:             $s_i \leftarrow P_i + \imath \operatorname{Im}\left(v_i \overline{(Y\boldsymbol{v})_i}\right)$        (17)
5:             Update $v_i$ using (12) and (13)
6:         else if $i$ is a PQ node then
7:             Update $v_i$ using (12) and (13)
8:             Update $v_i$ using (15) and (16)
9:         end if
10:    end for
11:    return $\boldsymbol{v}$.
12: end procedure
```

- To update the voltage magnitude of v_i by a scaling $\alpha \in \mathbb{R}^+$, we set

$$\operatorname{Re}\left[\overline{z}\left(\overline{s_i} - \overline{v_i}\sum_{j\neq i} Y_{ij}v_j - \overline{v_i}Y_{ii}(\alpha v_i)\right)\right] = 0. \tag{14}$$

Solving for α gives us

$$\alpha = \frac{\operatorname{Re}\left[\overline{z}(\overline{s_i} - \overline{v_i}\sum_{j\neq i} Y_{ij}v_j)\right]}{\operatorname{Re}[\overline{z}Y_{ii}||v_i|^2]} = 1 + \frac{\operatorname{Re}[\overline{z}\overline{v_i}r_i]}{\operatorname{Re}[\overline{z}Y_{ii}|v_i|^2]}. \tag{15}$$

We then update v_i with the scaling

$$v_i \leftarrow \alpha v_i. \tag{16}$$

We show the split Gauss–Seidel algorithm in Algorithm 1.

PV nodes. At a PV node the voltage magnitude is specified, so we skip the voltage magnitude update. Additionally, s_i is not fully specified at a PV node, so we must first update our guess of the reactive power injection with

$$s_i \leftarrow \operatorname{Re}(s_i) + \imath \operatorname{Im}\left(v_i\overline{(Y\boldsymbol{v})_i}\right). \tag{17}$$

This is the preprocessing step for PV nodes used in standard power flow Gauss–Seidel [38].

No solution to angle update. It is possible that there are no solutions to (11). This occurs if $|x|^2 < c^2$. If there are no solutions, we simply minimize the residual by choosing $e^{\imath\lambda}$ to either maximize or minimize $\operatorname{Im}[\overline{z}\overline{v_i}Y_{ii}v_ie^{\imath\lambda}]$, whichever is appropriate.

Updating with the small-angle approximation. We typically expect phase angles to only change by small amounts. Therefore, in solving for the phase angle update, one may make the approximation $e^{\imath\lambda} \approx 1 + \imath\lambda$. In this case, solving for λ gives the solution

$$\lambda = \frac{\operatorname{Im}[\overline{z}\overline{v_i}r_i]}{\operatorname{Im}[\imath\overline{z}Y_{ii}|v_i|^2]} \tag{18}$$

and the update formula

$$v_i \leftarrow (1 + \imath\lambda)v_i. \tag{19}$$

This update formula is less expensive than (12), and its use requires no branching. Note that, in this case, the voltage magnitude of a PV node is perturbed somewhat, and so it must be corrected with

$$v_i \leftarrow |v_i^{sp}| \frac{v_i}{|v_i|}.$$

3.4. Coarsening PQ and PV nodes. When applying the above smoother at a coarse level, we cycle through a sequence of coarse node groupings as in (10). If the group is of size 1, we simply apply the smoother as normal. Here we discuss how to apply the smoother to paired PQ-PV groups of size 2.

We perform the update as normal when updating the voltage magnitude, applying the update only to the PQ node of the pair. When updating the pair's phase angles, however, we essentially treat them as "aggregated" and update their phase angles in synchrony. That is, let $\mathcal{J} = \{i, i'\}$, where i is a PQ node and i' is a PV node. We solve

$$\operatorname{Im}\left(\sum_{i\in\mathcal{J}} \overline{zv_i}\left(\overline{s_i/v_i} - \sum_{j\notin\mathcal{J}} y_{ij}v_j - e^{\imath\lambda}\sum_{k\in\mathcal{J}} y_{ik}v_k\right)\right) = 0$$

for $e^{\imath\lambda}$, and then perform the update

$$\begin{aligned} v_i &\leftarrow v_i e^{\imath\lambda}, \\ v_{i'} &\leftarrow v_{i'} e^{\imath\lambda}. \end{aligned}$$

This is straightforward to compute in constant time. A similar formula can be written for voltage magnitude updates, but it is not needed as only the coarse PQ node is included in that update.

3.5. Variants. Coarse Newton's method. One option is to cut off a multigrid cycle before reaching the coarsest level of the hierarchy, and to fully or partially solve that coarse problem. Because the coarse problem is itself a power flow problem, one can use an existing power flow solver, which is typically highly effective at solving small- to medium-sized problems.

Using a Jacobi smoother. A Jacobi smoother makes all the same updates that a Gauss–Seidel smoother does, but makes those updates simultaneously rather than sequentially. The Jacobi iteration is popular in linear multigrid methods because it is highly parallelizable, but there is a tradeoff with stability and convergence. When using Jacobi in a multigrid method for a linear system of equations with a Jacobi smoother, one needs to *weight* or *damp* the Jacobi step to obtain convergence. That is, after computing a possible update $\tilde{\boldsymbol{v}}$, one then sets

$$\boldsymbol{v} \leftarrow \omega\tilde{\boldsymbol{v}} + (1-\omega)\boldsymbol{v}$$

for some $\omega \in (0,1)$. The optimal value of ω varies from problem to problem, but a selection of $\omega = 2/3$ is often effective [7].

As in the linear case, using a Jacobi smoother in multigrid power flow is less stable than a Gauss–Seidel smoother. We found experimentally that the weighting factor ω needed when the residual $\boldsymbol{r}$ is large is often much smaller than the ω needed when it is small. So, we use a *weighting schedule* with Jacobi iterations that allows ω to grow as the algorithm progresses. In particular, we define a set of bounds $b_1, \dots, b_k, b_1 = 0, b_k = \infty$, and a set of weights $\omega_1, \dots, \omega_{k-1}$. Then, if $b_i < \|\boldsymbol{r}\|_\infty \le b_{i+1}$, we set $\omega = \omega_i$.

We use some experimentally determined weighting schedules in the following section, but we leave the question of how best to choose this schedule as future work.

Table 1

Jacobi weighting schedules for various real-world networks. Also listed is the maximum number of hierarchy levels for each of the networks.

$\|\boldsymbol{r}\|_\infty$ in range	IEEE 57	IEEE 118	IEEE 300	Polish Winter Peak
$(0, 0.05]$	0.8	0.8	0.55	0.65
$(0.05, 0.1]$	0.75	0.75	0.55	0.65
$(0.1, 1.0]$	0.65	0.65	0.65	0.65
$(1.0, 5.0]$	0.3	0.3	0.3	0.3
$(5.0, 8.0]$	0.1	0.1	0.1	0.1
$(8.0, 50]$	0.05	0.05	0.05	0.05
$(50, 80]$	0.02	0.02	0.02	0.02
$(80, 200]$	0.01	0.01	0.01	0.01
$(200, \infty)$	0.01	0.01	0.001	0.01
# Hier. levels	6	6	7	10

4. Convergence experiments.

In this section we demonstrate the convergence properties of multigrid power flow. We explore several variants of multigrid power flow by changing the method in the following ways:

- whether to use Gauss–Seidel iterations or Jacobi iterations as the smoother,
- whether to use V-Cycles or Full Multigrid Cycles, and
- whether to use the complete multigrid hierarchy or switch to Newton's method at some level ℓ. This case is further subdivided into the level at which to switch to Newton's method, and whether to use a full Newton or quasi-Newton method in the inner iteration. In this paper, we consider full Newton's method, but consider not running the inner iteration to convergence.

We note that in some cases, the power flow problem has more than one solution. However, there is a single "high-voltage" solution that typically corresponds to the physically realistic solution. In all experiments here, we consider convergence to that high-voltage solution.

4.1. Real-world networks.

In this section we experiment with various real-world test networks under a variety of algorithm choices. We use here four real-world networks: the IEEE 57-bus, 118-bus, and 300-bus test networks, as well as the 1999–2000 Polish Winter Peak network (2,383 nodes) [49]. We run each test from a starting vector of $\mathbf{1}$, and continue until the relative residual drops below 10^{-6}. The Jacobi weighting schedules for the different networks were found by trial and error, and can be seen in Table 1. Also shown in Table 1 is the maximum number of hierarchy levels for each of the networks. Note that we do not use the small-angle approximation in (18), but use the exact angle update.

Although the real-world networks shown here have a variety of sizes, one should not interpret the results in this section as a rigorous scaling study. Each network has its own characteristics aside from size that can affect convergence. For example, the Polish Winter Peak network is a snapshot of the Polish power grid while it was under heavy load, which can hinder convergence.

Figure 1 shows convergence plots for the four real-world test networks. We compare V-Cycles against FMG-Cycles and Gauss–Seidel smoothers against Jacobi smoothers. As expected, FMG-Cycles and Gauss–Seidel smoothers have stronger convergence properties. However, the V-Cycle is typically about half as expensive as the FMG-Cycle, and the Jacobi smoother is parallelizable while the Gauss–Seidel smoother is not. We note in these plots that many of the plots drop dramatically, and then "bend right" at some point as convergence slows somewhat. This is particularly noticeable in the Polish Network: If a relative error of 10^{-3} is acceptable, then the

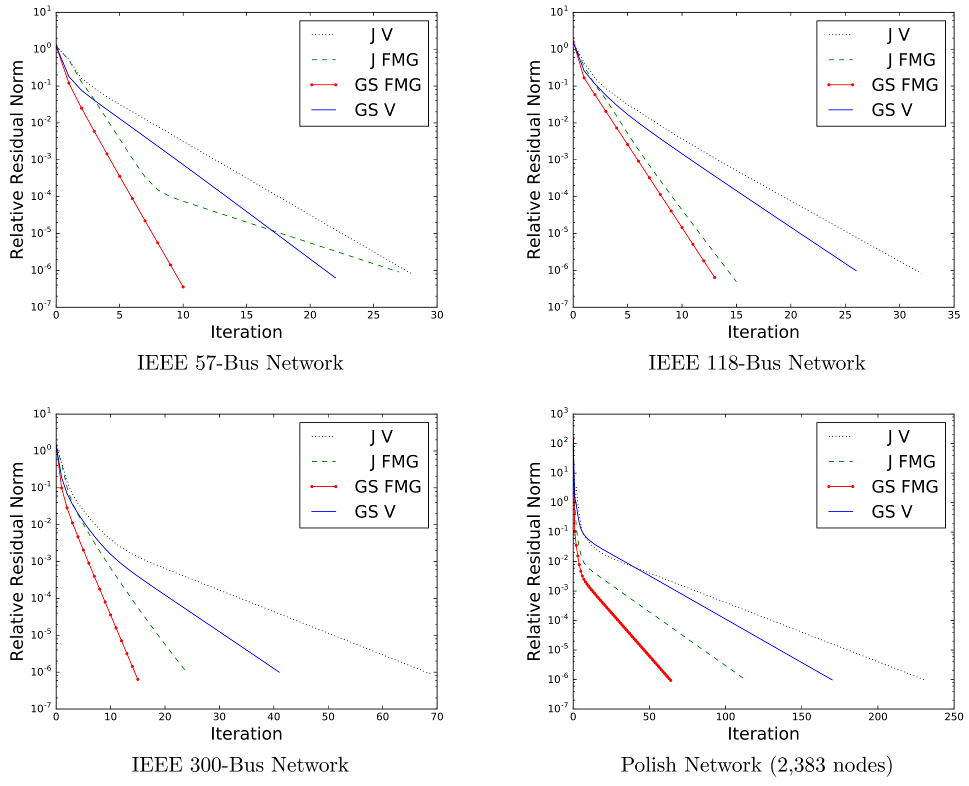

Fig. 1. *Convergence plots for various real-world test networks that compare V-Cycles against FMG-Cycles and Gauss–Seidel smoothers against Jacobi smoothers.*

FMG-Cycle with a Gauss–Seidel smoother is highly effective.

Next we show a plot of iterations required to converge under a greater variety of algorithm choices. We consider the same cases as above, but here we also consider "cutting off" the traversal up the hierarchy at some point and applying a Newton solver. We investigate the effects of using a single Newton iterations, two iterations, and running the Newton solver to convergence (which we implemented as either convergence or 25 iterations, whichever came first).

Figures 2 and 3 show the results for the IEEE 57 and 118-bus test networks. Again, FMG-Cycles and Gauss–Seidel smoothers have stronger convergence. We see here that convergence rates are not drastically affected by using a Newton solver, with perhaps some modest improvements if using a Newton solver at coarse level $\ell = 1$ (that is, a two-grid method). Running the Newton solver for more than a single iteration does not provide any benefit, which suggests that a quasi-Newton iteration might also be effective here. The Jacobi experiments show that the price of parallelism is worse convergence, but not drastically so.

We show the experiments with the IEEE 300-bus network in Figure 4. Here the decision where and if to use a Newton solver has a much bigger effect on the convergence rate when using a V-Cycle, especially for the Jacobi smoother. Again, we see that more than a single Newton iteration does not offer any improvement in rate of convergence.

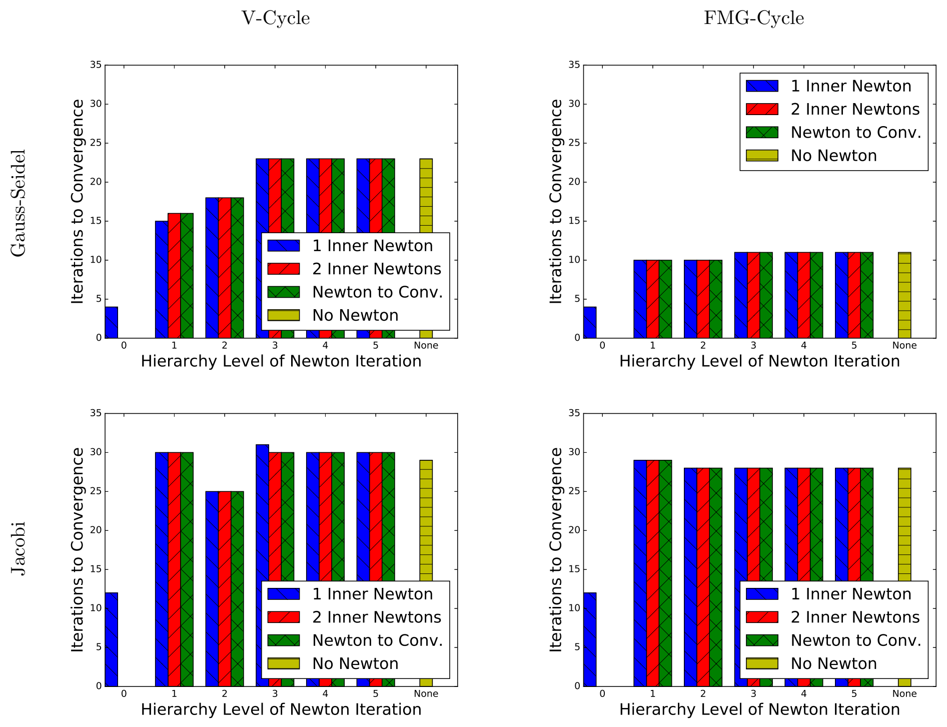

FIG. 2. *Multigrid power flow on the IEEE* 57*-bus test network with switches to Newton's method partway up the hierarchy. The bar group shows the hierarchy level at which we switch, while the bar within the group shows the number of inner Newton iterations performed. In blue is the test of the full hierarchy with no switch to Newton. Performing a single Newton iteration on hierarchy level* 0 *with* $\omega = 1$ *(as in the Gauss–Seidel row) is equivalent to a full Newton iteration.*

The experiments with the 1999–2000 Polish Winter Peak network, shown in Figure 5, show still-greater impact of the Newton solver. Unlike in the IEEE 300-bus case, however, a Newton solver at coarse level $\ell = 1$ (that is, a two-grid method) helps the most with the Gauss–Seidel iteration, and in fact makes the Jacobi iteration worse.

4.2. Synthetic networks. In this section we present scaling studies of multigrid power flow. We cannot simply scale up real-world problems, as those problems come from specific networks, and as discussed above the various real-world networks are not necessarily comparable. Instead, we generate synthetic networks in the following manner: we define some domain D and smooth functions $v(\boldsymbol{x}), \theta(\boldsymbol{x})$ representing the voltage magnitude and phase angle values at each point $\boldsymbol{x} \in D$. We can then place an arbitrary number of simulated nodes within D, and the functions $v(), \theta()$ give us the voltage at each node. An application of (1) then allows us to determine the power injection at each node. We then simply choose the type of each node (PQ, PV, or slack) to complete our synthetic network construction. In this way, we can construct a sequence of comparable problems of varying sizes.

Here we define D to be a circle of radius 1 centered at the origin in the plane. The voltage magnitude and phase angle are defined separately as solutions to the Laplace

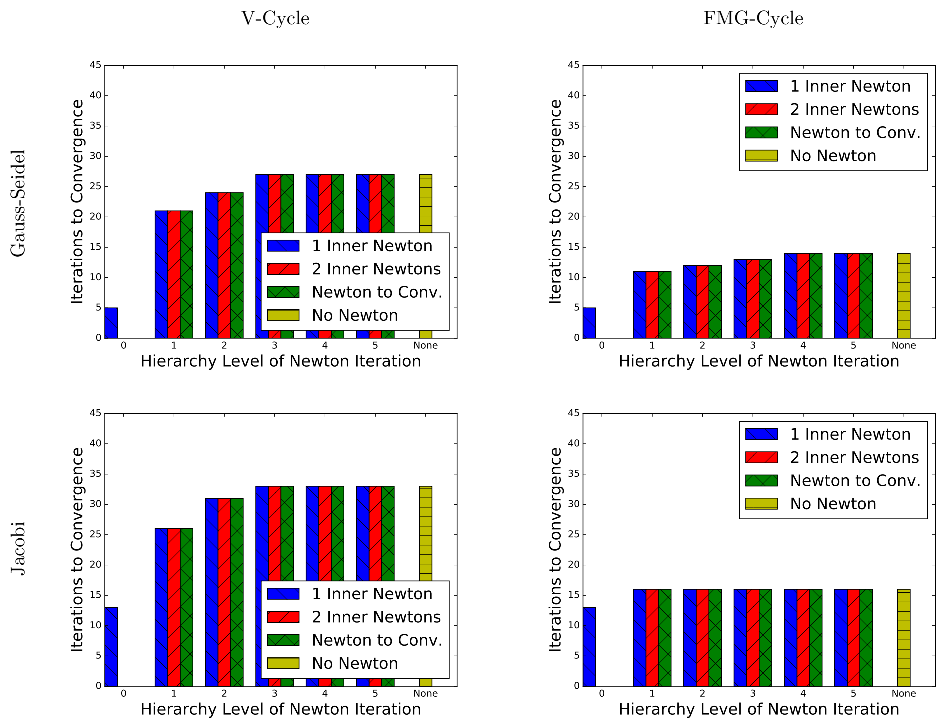

Fig. 3. *Multigrid power flow on the IEEE* 118*-bus test network with switches to Newton's method partway up the hierarchy.*

equation:

$$\begin{aligned}\Delta u(\boldsymbol{x}) &= 0, \quad \boldsymbol{x} \in D,\\ u(\boldsymbol{x}) &= g, \quad \boldsymbol{x} \in \partial D.\end{aligned}$$

The solution to this problem is well known (see, e.g., [12]) and is given by

$$u(\boldsymbol{x}) = u(r,\theta) = \int_{\partial D} \frac{1-r^2}{2\pi\|\boldsymbol{x}-\boldsymbol{y}\|_2^2} g(\boldsymbol{y}) dS(\boldsymbol{y}).$$

We use the solution to the Laplace equation because it is harmonic, which implies that it is smooth and also guarantees that all function values in D lie within the extrema of g on ∂D. In addition, we have the option of producing a variety of different smooth functions simply by changing the boundary values g.

For the voltage magnitude, we set

$$g(\boldsymbol{x}) = g(1,\theta) = 1 + m_M \cos(m_f \theta),$$

where m_M and m_f are chosen constants. Similarly, for the voltage phase angle we set

$$g(\boldsymbol{x}) = g(1,\theta) = a_M \cos(a_f \theta),$$

where a_M and a_f are chosen constants. Here we select

$$m_M = 0.05, \quad m_f = 1.0, \quad a_M = \pi/4, \quad a_f = 1.5.$$

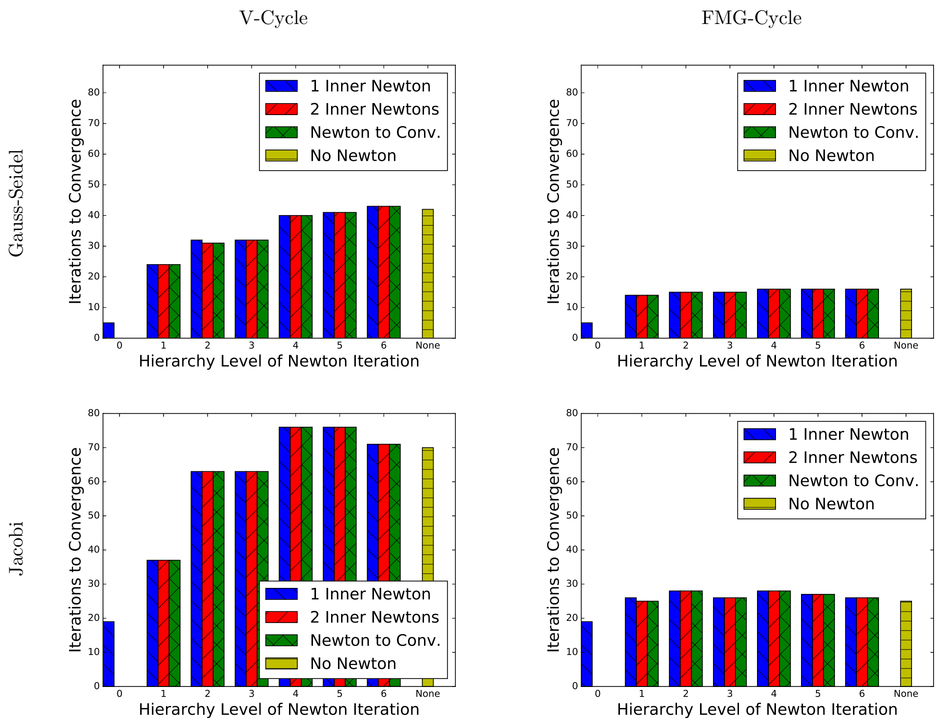

FIG. 4. *Multigrid power flow on the IEEE* 300*-bus test network with switches to Newton's method partway up the hierarchy.*

Plots of these functions over the unit circle can be seen in Figure 6.

We choose nodes in a rectangular grid within the circle such that all neighboring nodes are a constant distance away. All lines have impedance $10 - \imath 20$ per unit. All lines are also given a line charging susceptance of 0.002 per unit. We place the slack node at the center of the circle, and all other nodes are independently selected as PV nodes with probability 0.05, and PQ otherwise.

As above, we perform all tests with a flat start, where all elements of the starting voltage vector are $1 + \imath 0$, except for PV nodes, whose magnitudes are known, and the slack nodes, whose voltage is known. We then test the number of cycle iterations required for convergence. We perform the following eight tests:

- With Gauss–Seidel smoothing and V-Cycles, no Newton iteration or a full Newton solve at level $\ell = 1$.
- With Gauss–Seidel smoothing and FMG-Cycles, no Newton iteration or a full Newton solve at level $\ell = 1$.
- With Jacobi smoothing and V-Cycles, no Newton iteration or a full Newton solve at level $\ell = 1$.
- With Jacobi smoothing and FMG-Cycles, no Newton iteration or a full Newton solve at level $\ell = 1$.

When using a Jacobi smoother, we use the ω schedule that is used for the IEEE 57-bus and 118-bus networks in Table 1.

Of our synthetic tests, the smallest network, with 16 nodes, has 5 hierarchy levels and a mean coarsening ratio of 1.22. The largest network, with 17,647 nodes, has 14 hierarchy levels and a mean coarsening ratio of 1.96. This is expected, as we use a

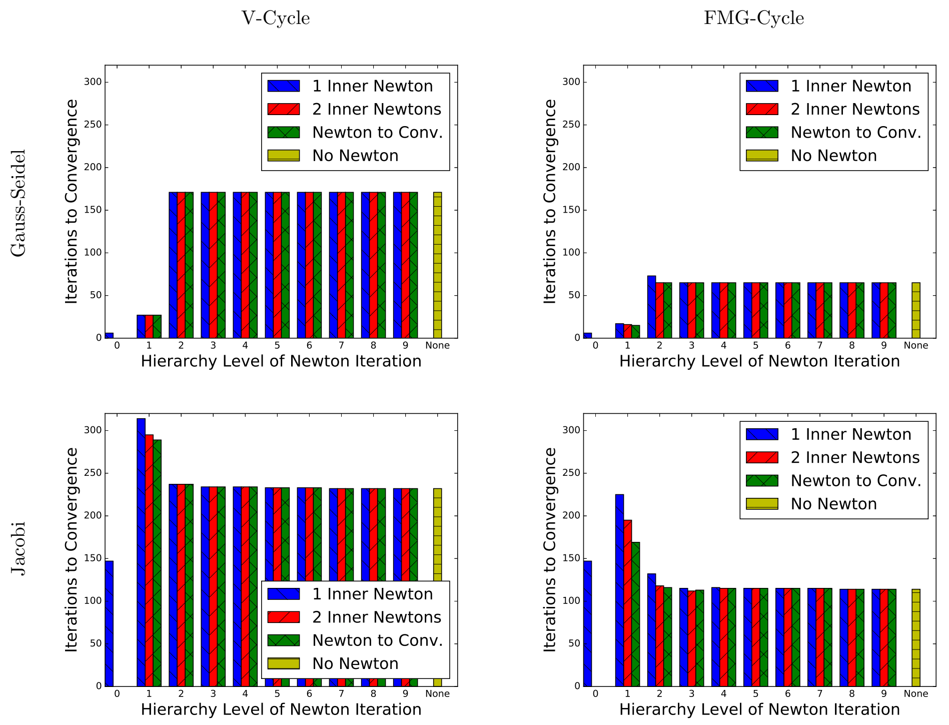

Fig. 5. *Multigrid power flow on the* 1999–2000 *Polish Winter Peak network with switches to Newton's method partway up the hierarchy.*

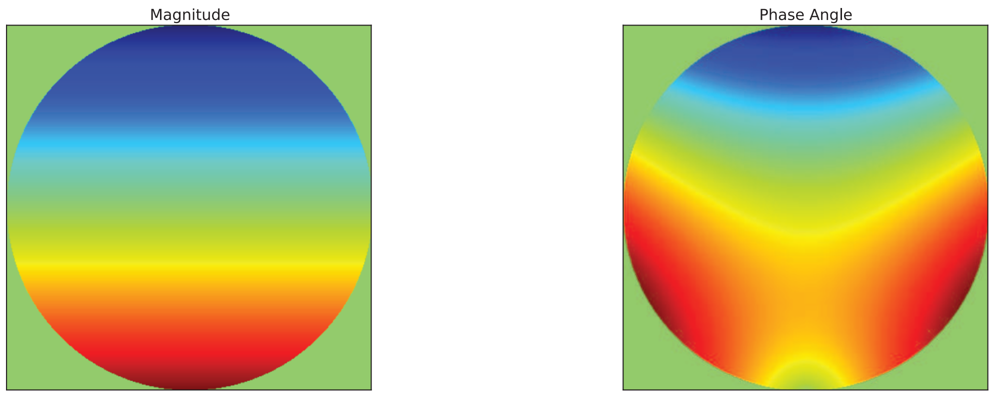

Fig. 6. *Analytic functions on the unit circle from which we sample voltage magnitude and phase angle to create synthetic networks.*

matching-based coarsening procedure, so the coarsening ratio is bounded from above by 2.0.

We show the results in Figure 7. As expected, the Gauss–Seidel smoother has better convergence than the Jacobi smoother. Above network size approximately 2^9, the FMG-Cycle shows high scalability, with iteration growth rate of approximately $O(n^{0.3})$. This implies an algorithmic time complexity of about $O(n^{1.3})$, as each iteration takes time $O(n)$. A single Newton iteration at coarse level $\ell = 1$ with

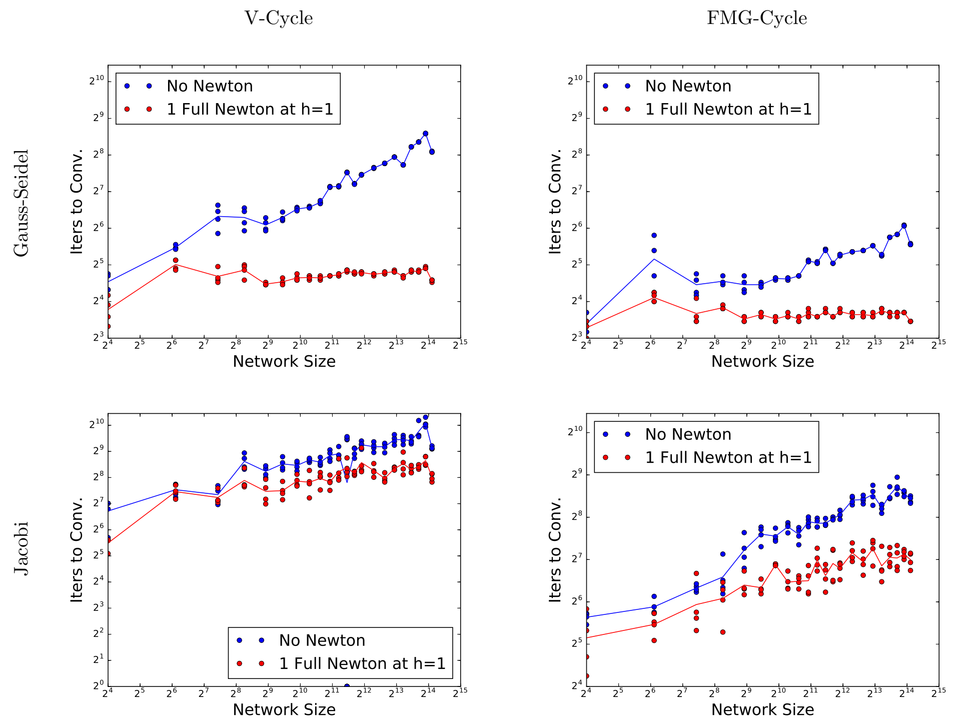

FIG. 7. *Convergence rates using a synthetic network. Each network size is sampled four times. The solid lines represent the mean at each network size. A Newton solve at coarse level $\ell = 1$ with Gauss–Seidel smoothing performs best, showing no decay in convergence as network size grows. An FMG-Cycle with either Jacobi or Gauss–Seidel smoothing also shows little decay in convergence rate.*

Gauss–Seidel smoothing shows no decay in rate of convergence as network size grows, regardless of cycle type. It is also interesting that in most cases, convergence rate variance decreases with increasing network size.

4.3. Robustness experiments. As discussed in section 1, while fast for small and medium-sized problems, Newton–Raphson suffers from a small region of convergence. In this section we present experiments demonstrating the robustness of our power flow solver to poor initial starting conditions and comparing it to a number of existing robust solution methods.

For these experiments, we take the initial guess for an iterative solver from the data file itself, add Gaussian random noise separately to the magnitudes and phase angles of that initial guess, and then run the iterative solver to see if it finds the correct solution. As a practical matter, voltage magnitudes never stray far from 1.0, so if we add noise to the phase angles with a standard deviation greater than 0.16, the voltage magnitudes only receive noise with standard deviation equal to 0.16. Otherwise, voltage magnitudes and phase angles receive the same amount of noise.

We test our method against three other solvers. The first is Newton–Raphson with a bisecting line search. This algorithm first calculates a typical Newton–Raphson update. If the update causes the norm of the residual to decrease, it takes that step; if it causes the residual to increase, it cuts the step size in half and tries again. It repeats this procedure until it either finds a step size that results in a reduction in residual norm, or until the step size reaches machine precision, at which point the solver exits. This method is guaranteed not to diverge, and is generally more stable than traditional Newton–Raphson. It can be viewed as an approximation to an optimal line search.

Next, we test our method against the continuation power flow solver of [2]. This is the most well known of the class of homotopy methods and is most effective when the system is near its voltage stability limit, a setting in which Newton–Raphson typically has a very small region of convergence. It first solves a power flow problem in which the system is more lightly loaded than the desired problem. Given that starting point then, it solves a series of power flow problems in which the loading level of the system is gradually increased, using the solution to the previous problem as an initial guess for the next one. The loading level is added to the problem as its own variable, which prevents the Jacobian from becoming singular at the voltage stability limit as it normally would.

Finally, we test our method against the Adjusted Levenberg–Marquardt method developed in [27]. If one calculates the squared 2-norm of the residual, $\boldsymbol{r}^T\boldsymbol{r}$, and then uses a first-order Taylor expansion to estimate the residual norm minimizer, the result is the typical Newton–Raphson method. However, if rather than minimizing $\boldsymbol{r}^T\boldsymbol{r}$ one instead attempts to minimize

$$\boldsymbol{r}^T\boldsymbol{r} + \lambda(\boldsymbol{x} - \boldsymbol{x}_{prev})^T(\boldsymbol{x} - \boldsymbol{x}_{prev}),$$

this problem penalizes taking steps that are too large, with the penalty controlled by the coefficient λ. This is the Levenberg–Marquardt approach. The authors of [27] further enhance this using a two-step procedure akin to the midpoint method for solving ODEs. In these tests, we use $\lambda = 1$.

We show the results on the robustness experiments for the IEEE 57-, 118-, and 300-bus networks in Figure 8. For these experiments we used the Gauss–Seidel smoother with a V-Cycle and no Newton cutoff. We test using both (1,1)-cycles and (5,5)-cycles, in which we apply the smoother once or five times at each step in the hierarchy while coarsening and again while refining. As we see in these figures, while the adjusted Levenberg–Marquardt method is indeed more robust than both Newton–Raphson with bisecting line search and continuation power flow, our multigrid solver is still significantly more robust than either, easily handling far more noise in the starting iterate than any of the other methods. Furthermore, the (5,5)-cycle is even more robust than the (1,1)-cycle; sufficient noise in the Polish Winter Peak network causes the (1,1)-cycle to fail, whereas the (5,5)-cycle remains stable throughout.

5. Conclusions.

In this paper we introduced a nonlinear algebraic multigrid framework for solving the power flow equations, a standard problem in power engineering. We did this by developing a novel multiplicative coarse-grid correction for the state variables in the problem, and demonstrating that the resulting coarse problem is also a power flow problem, enabling multigrid recursion.

We then presented a number of specific choices for constructing specific multigrid methods from that framework. We showed that both Gauss–Seidel and Jacobi smoothers are effective, resulting in the first (to the best of the authors' knowledge) successful use of a Jacobi iteration in solving the power flow equations. We also showed that one can partially or completely solve a coarse power flow problem at some level of the hierarchy instead of making use of the entire hierarchy, and that this tends to accelerate convergence. We demonstrated the scalability of our method using experiments on models of both real-world and synthetic power grids. We demonstrated its

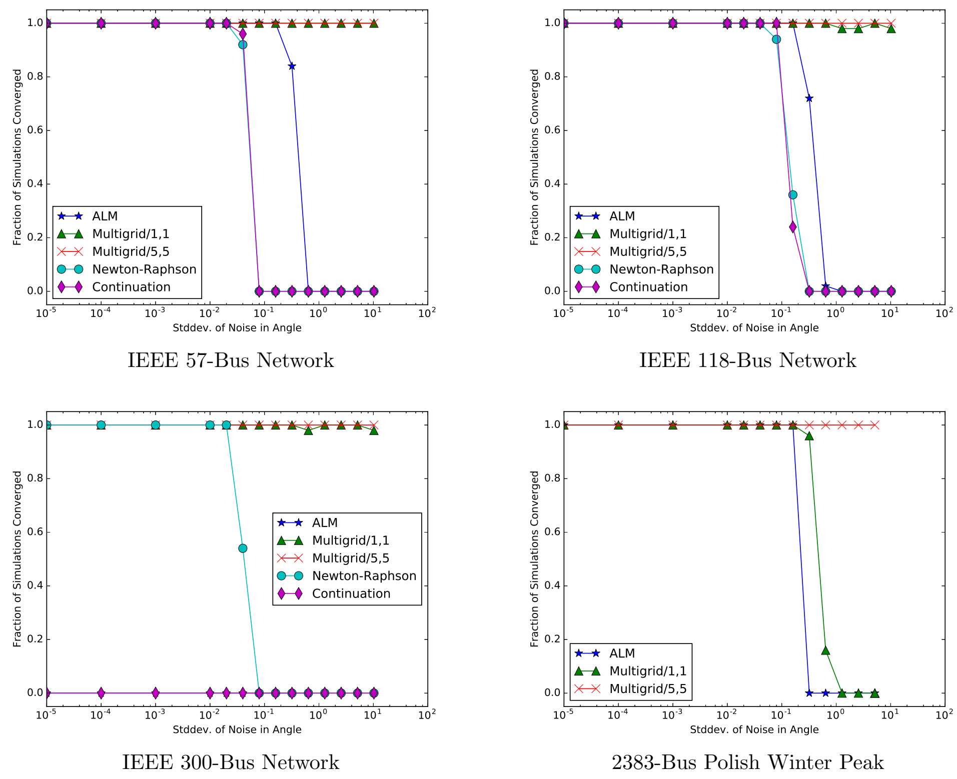

Fig. 8. *Robustness experiments comparing Newton–Raphson, Continuation, Adjusted Levenberg–Marquardt, and our multigrid solver using* (1,1)*-V cycles and* (5,5)*-V cycles. For each experiment in the IEEE networks,* 50 *tests were run. In the Polish Winter Peak network, due to computational cost we ran* 10 *tests for each ALM experiment, and* 25 *for each multigrid experiment. The voltage magnitude starting points received the same amount of noise as the voltage angles, up to a standard deviation of* 0.16*.*

robustness to poor initial conditions and showed that it is more robust than many state-of-the-art power flow solvers.

Possible future directions include further investigation into smoothing and coarsening techniques and exploring the scenarios in which different choices are most effective. The method might also be further improved by the application of acceleration methods such as Anderson Acceleration [47] or Nonlinear GMRES. Possible future directions also include developing an efficient parallel implementation of our method.

REFERENCES

[1] S. Abhyankar, B. Smith, and E. Constantinescu, *Evaluation of overlapping restricted additive Schwarz preconditioning for parallel solution of very large power flow problems*, in International Conference for High Performance Computing, Network, Storage and Analysis (SC 2013), ACM, New York, 2013.

[2] V. Ajjarapu and C. Christy, *The continuation power flow: A tool for steady state voltage stability analysis*, IEEE Trans. on Power Systems, 7 (1992), pp. 416–423.

[3] P. Aristidou, S. Lebeau, and T. V. Cutsem, *Power system dynamic simulations using a parallel two-level Schur-complement decomposition*, IEEE Trans. on Power Systems, 31 (2015), pp. 3984–3995.

[4] A. Bienz, R. D. Falgout, W. Gropp, L. N. Olson, and J. B. Schroder, *Reducing parallel communication in algebraic multigrid through sparsification*, SIAM J. Sci. Comput., 38 (2016), pp. S332–S357, https://doi.org/10.1137/15M1026341.

[5] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.

[6] M. Brezina, P. Vaněk, and P. S. Vassilevski, *An improved convergence analysis of smoothed aggregation algebraic multigrid*, Numer. Linear Algebra Appl., 19 (2012), pp. 441–469.

[7] W. L. Briggs, V. E. Hensen, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed., SIAM, Philadelphia, 2000, https://doi.org/10.1137/1.9780898719505.

[8] G. Cespedes, *New method for the analysis of distribution networks*, IEEE Trans. Power Deliv., 5 (1990), pp. 391–396.

[9] H. De Sterck, T. A. Manteuffel, S. F. McCormick, K. Miller, J. Ruge, and G. Sanders, *Algebraic multigrid for Markov chains*, SIAM J. Sci. Comput., 32 (2010), pp. 544–562, https://doi.org/10.1137/090753589.

[10] P. Dell'Acqua, A. Frangioni, and S. Serra-Capizzano, *Accelerated multigrid for graph Laplacian operators*, Appl. Math. Comput., 270 (2015), pp. 193–215, https://doi.org/10.1016/j.amc.2015.08.033.

[11] J.-J. Deng, T.-Q. Zhao, H.-D. Chiang, Y. Tang, and Y. Wang, *Convergence regions of Newton method in power flow studies: Numerical studies*, in Proceedings of the 2013 IEEE International Symposium on Circuits and Systems, 2013, pp. 1532–1535.

[12] L. C. Evans, *Partial Differential Equations*, AMS, Providence, RI, 1998.

[13] R. D. Falgout, *An introduction to algebraic multigrid*, Comput. Sci. Eng., 8 (2006), pp. 24–33, https://doi.org/10.1109/MCSE.2006.105.

[14] R. D. Falgout and J. B. Schroder, *Non-Galerkin coarse grids for algebraic multigrid*, SIAM J. Sci. Comput., 36 (2014), pp. 309–334, https://doi.org/10.1137/130931539.

[15] J. D. Glover, M. S. Sarma, and T. J. Overbye, *Power System Analysis and Design*, Cengage Learning, Stamford, CT, 2012.

[16] D. Hoske, D. Lukarski, H. Meyerhenke, and M. Wegner, *Engineering a combinatorial Laplacian solver: Lessons learned*, Algorithms, 9 (2016), 72.

[17] X. Hu, P. S. Vassilevski, and J. Xu, *A two-grid SA-AMG convergence bound that improves when increasing the polynomial degree*, Numer. Linear Algebra Appl., 23 (2016), pp. 746–771.

[18] R. Idema, G. Papaefthymiou, D. Lahaye, C. Vuik, and L. van der Sluis, *Towards faster solution of large power flow problems*, IEEE Trans. on Power Systems, 28 (2013), pp. 4918–4925.

[19] S. Iwamoto and Y. Tamura, *A load flow calculation method for ill-conditioned power systems*, IEEE Trans. Power App. Syst., PAS-100 (1981), pp. 1736–1743.

[20] R. A. Jabr, *Radial distribution load flow using conic programming*, IEEE Trans. on Power Systems, 21 (2006), pp. 1458–1459.

[21] V. Jalili-Marandi and V. Dinavahi, *Instantaneous relaxation-based real-time transient stability simulation*, IEEE Trans. on Power Systems, 24 (2009), pp. 1327–1336.

[22] S. K. Khaitan and J. D. McCalley, *A class of new preconditioners for linear solvers used in power system time-domain simulation*, IEEE Trans. on Power Systems, 25 (2010), pp. 1835–1844.

[23] S. K. Khaitan, J. D. McCalley, and Q. Chen, *Multifrontal solver for online power system time-domain simulation*, IEEE Trans. on Power Systems, 23 (2008), pp. 1727–1737.

[24] A. Knyazev and K. Neymeyr, *Efficient solution of symmetric eigenvalue problems using multigrid preconditioners in the locally optimal block conjugate gradient method*, Electron. Trans. Numer. Anal., 15 (2003), pp. 38–55.

[25] D. P. Kothari and I. J. Nagrath, *Power System Engineering*, Tata McGraw-Hill, West Patel Nagar, New Delhi, India, 2008.

[26] M. la Cour Christensen, P. S. Vassilevski, and U. Villa, *Nonlinear multigrid solvers exploiting AMGe coarse spaces with approximation properties*, J. Comput. Appl. Math., in press, 2017.

[27] P. J. Lagace, M. H. Vuong, and I. Kamwa, *Improving power flow convergence by Newton-Raphson with a Levenberg-Marquardt method*, in Power and Energy Society General Meeting, IEEE, 2008, pp. 1–6.

[28] M. Lecouvez, R. Falgout, C. Woodward, and P. Top, *A parallel multigrid reduction in time method for power systems*, in Power and Energy Society General Meeting (PESGM), IEEE, 2016.

[29] X. Li, F. Li, and J. M. Clark, *Exploration of multifrontal method with GPU in power flow computation*, in Power and Energy Society General Meeting, IEEE, 2013, pp. 1–5.

[30] Y. Liu and Q. Jiang, *Two-stage parallel waveform relaxation method for large-scale power system transient stability simulation*, IEEE Trans. on Power Systems, 31 (2016), pp. 153–162.

[31] F. Milano, *Continuous Newton's method for power flow analysis*, IEEE Trans. on Power Systems, 24 (2009), pp. 50–57.

[32] W. Murray, T. T. De Rubira, and A. Wigington, *Improving the robustness of Newton-based power flow methods to cope with poor initial points*, in North American Power Symposium, IEEE, 2013.

[33] A. Napov and Y. Notay, *An efficient multigrid method for graph Laplacian systems*, Electron. Trans. Numer. Anal., 45 (2016), pp. 201–218.

[34] C. Ponce and P. S. Vassilevski, *Solving graph Laplacian systems through recursive partitioning and two-grid preconditioning*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 621–648, https://doi.org/10.1137/15M1050872.

[35] S. Smith, C. Woodward, L. Min, C. Jing, and A. Del Rosso, *On-line transient stability analysis using high performance computing*, in Innovative Smart Grid Technologies Conference (ISGT), IEEE, 2014, pp. 1–5.

[36] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 2004, pp. 81–90.

[37] B. Stott, *Effective starting process for Newton-Raphson load flows*, in Proceedings of the Institution of Electrical Engineers, 118 (1971), pp. 983–987.

[38] B. Stott, *Review of load-flow calculation methods*, Proceedings of the IEEE, 62 (1974), pp. 916–929.

[39] B. Stott, *Power system dynamic response calculations*, Proceedings of the IEEE, 67 (1979), pp. 219–241.

[40] B. Stott and O. Alsaç, *Fast decoupled load flow*, IEEE Trans. Power App. Syst., 93 (1974), pp. 859–869.

[41] W. F. Tinney and C. E. Hart, *Power flow solution by Newton's method*, IEEE Trans. Power App. Syst., 86 (1967), pp. 1449–1460.

[42] S. Tripathy, G. Durga Prasa, O. Malik, and G. Hope, *Load-flow solutions for ill-conditioned power systems by a Newton-like method*, IEEE Trans. Power App. Syst., PAS-101 (1982), pp. 3648–3657.

[43] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[44] P. Vassilevski, *Multilevel Block Factorization Preconditioners*, Springer, Livermore, CA, 2008.

[45] P. Vassilevski and L. Zikatanov, *Commuting projections on graphs*, Numer. Linear Algebra Appl., 21 (2014), pp. 297–315.

[46] P. S. Vassilevski and U. Yang, *Reducing communication in algebraic multigrid using additive variants*, Numer. Linear Algebra Appl., 21 (2014), pp. 275–296.

[47] H. F. Walker and P. Ni, *Anderson acceleration for fixed-point iterations*, SIAM J. Numer. Anal., 49 (2011), pp. 1715–1735, https://doi.org/10.1137/10078356X.

[48] Y.-S. Zhang and H.-D. Chiang, *Fast Newton-FGMRES solver for large-scale power flow study*, IEEE Trans. on Power Systems, 25 (2010), pp. 769–776.

[49] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, *MATPOWER: Steady-state operations, planning and analysis tools for power systems research and education*, IEEE Trans. on Power Systems, 26 (2011), pp. 12–19.