DELFT UNIVERSITY OF TECHNOLOGY

REPORT 10-22

Scalable Newton-Krylov Solver for Very Large Power Flow Problems

R. Idema, D.J.P. Lahaye, C. Vuik, and L .van der Sluis

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2010

Scalable Newton-Krylov Solver for Very Large Power Flow Problems

R. Idema, D.J.P. Lahaye, C. Vuik* and L. van der Sluis†

Abstract

The power flow problem is generally solved by the Newton-Raphson method with a sparse direct solver for the linear system of equations in each iteration. While this works fine for small power flow problems, we will show that for very large problems the direct solver is very slow and we present alternatives that scale much better in the problem size. For the largest test problem, with around one million busses, the presented alternatives are over 120 times faster than using a direct solver.

1 Introduction

Electrical power systems can be regarded to be among the most complex systems designed, constructed, and operated by humans. The consumers are supplied with the requested amount of active and reactive power at a constant frequency and with a constant voltage. Loads are switched on and off continuously, and because electricity cannot be efficiently stored in large quantities, the balance between the amount of generated and consumed electricity has to be maintained by control actions. This has brought forward the basic design of these power systems: a solid, centralized, vertical structure. The production of electrical energy or, more correctly, the conversion of the primary energy source into electricity takes place in a central core consisting of a number of large power plants, completely controlled in terms of their output. Tree-structured distribution networks transport this energy towards the distributed consumers.

The sophisticated control of the large power plants, combined with the efficient design of the networks, has created the present power systems, based on the principles of robustness, reliability, security, stability, and quality of energy supply. These basic principles have always been the system design goals and were traditionally interwoven with the centralized, vertical structure. Aberration from this structure has often been treated with skepticism, implying violation of these principles. But in the recent years two socio-economic factors, namely the stimulation of the use of renewable energy as source of primary energy and the liberalization of the energy markets, created a boost towards a new concept. The systems of the future should have a non-centralized, distributed structure.

In such distributed systems, the generation of electrical energy takes place in a large number of small- to medium-scale geographically distributed power plants. But this horizontal operation is associated with a basic restricting feature: the output of these power plants cannot be regulated, either due to their operation in a liberalized market environment, or due to unavailability of the primary energy mover when renewable generation is concerned. Moreover, these generators are

*Delft University of Technology, Delft Institute of Applied Mathematics, Mekelweg 4, 2628 CD Delft, The Netherlands. E-mail: r.idema@tudelft.nl, d.j.p.lahaye@tudelft.nl, c.vuik@tudelft.nl

†Delft University of Technology, Power Systems Department, Mekelweg 4, 2628 CD Delft, The Netherlands. E-mail: l.vandersluis@tudelft.nl

connected at the lower voltage levels (in contrast to the current layout of the system where most of the generation is connected at transmission level), they are connected by means of power electronic interfaces (instead of the classic synchronous generator paradigm), and their output presents an intermittent character (leading to strong fluctuations in the power injected in the grid).

Resulting from these radical structural changes, increased uncertainty in electrical energy generation penetrates system operation. The system design engineers are therefore forced to deal with new problems, such as finding out under which conditions the new structure does not violate the operational principles of the power system, and what the design methodologies for the transition from the present to the future structure are.

A possible consequence of the transition from the current vertically-operated power system into a future horizontally-operated power system for the power system is that the transmission network becomes less active, while the distribution network becomes a lot more active. The power is not only consumed but also generated in the distribution network. The direction of power flows in the network become less predictable, and one-way traffic becomes two-way traffic. Voltage stability becomes an issue in this case, as the voltage is no longer controlled by the generators in the large centralized power stations and tap-changing transformers alone. The decentralized generation has to play a role in controlling the voltage.

The power flow, or load flow, computation is the most important network computation in power systems. It calculates the voltage magnitude and angle in each bus of a power system, under specified system operation conditions. Other quantities, such as current values, power values, and power losses, can be calculated easily when the bus voltages are known.

Power flow computations bring insight in the steady-state behavior of a power system. This is needed in many control and planning applications. For example, whenever power system components have to be taken out of service, say for maintenance purposes, it is crucial to know if the power system will still function within system limits without these particular components, or whether additional measures have to be taken. Moreover, a power system has to be at least n?1 secure, meaning that if any one component fails the power system still functions properly. These are typical problems that can be solved by doing load flow computations on the power system network.

More and more nation wide power systems are being connected to be able to exchange cheap excess power. This results in huge connected power systems with many times the busses and transmission lines of the classical systems. A small set of simultaneous failures could propagate through the entire system, causing a massive blackout. Therefore providing security against overloading is more important than ever. Furthermore, in a larger power system it is much more likely to have multiple failures at the same time. Therefore n?2 security analysis is already being done regularly, taking huge amounts of computational effort.

Currently, power flow calculations are done on the transmission network while the distribution network is aggregated at busses in the power system model. However, with the transition to horizontally-operated power systems with distributed generation it will no longer be possible to simply work on the transmission network alone. The power flow computations will have to be extended to the distribution network, possibly down to as low as the 10kV systems. This would result in power flow problems of sheer massive size. Motivated by these developments, we explore mathematical techniques for power flow problems that are particularly well equipped to deal with very large problem sizes.

Traditionally the power flow problem is solved by use of the Newton-Raphson method, with a direct solver for the linear systems [23, 24]. It has been recognized that iterative linear solvers can offer advantages over sparse direct solvers for large power systems [10, 5, 3, 12, 29]. The question arises when iterative methods are better than direct methods for power flow problems. The key to answering this question is in the scaling of the computational time of the power flow algorithm in the problem size.

In this paper we will answer that question for a test set of power flow problems, ranging up to one million busses. We will show how the Newton-Raphson method with a direct solver scales, and compare it with the scaling behavior of the Newton-Raphson method with an iterative linear solver with a selection of preconditioners.

We will see that direct solvers, or any other method using a complete LU factorization, scale very badly in the problem size. The alternatives that we propose in this paper show near linear scaling, and are therefore much faster for large power flow problems. Using a direct solver the largest problem takes over an hour to solve, while our solver can solve it in less than 30 seconds, that is 120 times faster.

The implementation of our solver is done in C++ using PETSc (Portable, Extensible Toolkit for Scientific Computation) [2], a state of the art C library for scientific computing. The Newton-Raphson method is part of the SNES framework within PETSc. PETSc can be used to produce both sequential programs, and programs running in parallel on multiple processors. The results that we present in this paper were all executed on a single processor core.

The test set of power flow problems used is based on the UCTE[1] winter 2008 study model, which consists of 4253 busses and 7191 lines. The smallest problem is the UCTE winter 2008 study model itself, while the larger problems are all constructed by copying the model multiple times, and interconnecting the copies with new transmission lines.

2 Power Flow Problem

The power flow problem is the problem to determine the voltage at each bus of a power system, given the supply at each generator and the demand at each load in the network.

Let $Y = G + jB$ denote the network admittance matrix of the power system. Then the power flow problem can be formulated as the nonlinear system of equations

$$\sum_{k=1}^{N} |V_i| \, |V_k| \, (G_{ik} \cos \delta_{ik} + B_{ik} \sin \delta_{ik}) = P_i, \tag{1}$$

$$\sum_{k=1}^{N} |V_i| \, |V_k| \, (G_{ik} \sin \delta_{ik} - B_{ik} \cos \delta_{ik}) = Q_i, \tag{2}$$

where $|V_i|$ is the voltage magnitude, δ_i is the voltage angle, with $\delta_{ij} = \delta_i - \delta_j$, P_i is the active power, and Q_i is the reactive power at bus i. For details see for example [19, 11].

Define the power mismatch function as

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{bmatrix} P_i - \sum_{k=1}^{N} |V_i| \, |V_k| \, (G_{ik} \cos \delta_{ik} + B_{ik} \sin \delta_{ik}) \\ Q_i - \sum_{k=1}^{N} |V_i| \, |V_k| \, (G_{ik} \sin \delta_{ik} - B_{ik} \cos \delta_{ik}) \end{bmatrix} \tag{3}$$

where $\boldsymbol{x}$ is the vector of voltage angles and magnitudes. Then we can reformulate the power flow problem (1), (2), to finding a solution vector $\boldsymbol{x}$ such that

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}. \tag{4}$$

This is the system of non-linear equations that we will solve to find the solution of the power flow problem.

[1] UCTE is a former association of transmission system operators in Europe. As of July 2009, the European Network of Transmission System Operators for Electricity (ENTSO-E), a newly formed association of 42 TSOs from 34 countries in Europe, has taken over all operational tasks of the existing European TSO associations, including UCTE. See `http://www.entsoe.eu/`

3 Power Flow Solution

Our solver is based on the Newton-Raphson method for systems of nonlinear equations like (4). This iterative method updates the approximate solution $\boldsymbol{x}_i$ in each iteration with a Newton step $\boldsymbol{s}_i$, calculated from the linear system of equations

$$J_i \boldsymbol{s}_i = -\boldsymbol{F}_i, \tag{5}$$

where J_i is the Jacobian of the power mismatch $\boldsymbol{F}$. For details on the Newton-Raphson method see for example [7].

In power flow analysis, the classical way to solve the linear system (5) is with a direct solver. Effectively, this means that J_i is factorized into a lower triangular factor L_i, and an upper triangular factor U_i, such that $L_i U_i = J_i$. Then the linear problem $L_i U_i \boldsymbol{s}_i = -\boldsymbol{F}_i$ can be solved using forward and backward substitution, which are both very fast operations.

The LU factorization was originally designed for full matrices, and has complexity n^3 in the problem size n. It was adapted very efficiently for sparse matrices, like the Jacobian matrix J_i in the power flow problem. Although the scaling of the LU factorization's computational time in the problem size is much more complex for sparse matrices, it is well-known that in general it does not scale linearly for large problem sizes. This is something that is also clearly illustrated by our numerical experiments in Section 6. For more information on sparse direct methods see for example [8, 4].

To get a power flow solver that scales better for large problems we will look at alternative linear solvers for problem (5) in the form of iterative methods, in particular the Generalized Minimal Residual method (GMRES) [18]. The reason to use GMRES is that it, being a minimal residual method, solves the problem in the least number of matrix-vector multiplications. From the performance of GMRES we can then judge whether we should try other iterative linear solvers, like Bi-CGSTAB [27, 20] or IDR(s) [21].

With the Newton-Raphson method there is only a certain accuracy that we can expect to reach in each iteration. Solving the linear system to an accuracy higher than that needed to reach the best Newton-Raphson accuracy in the current step would be a waste of computational time. Therefore, when using an iterative linear solver we should not solve each linear system to full precision. Instead we should determine forcing terms η_i and use the iterative linear solver to solve up to

$$\|J_i \boldsymbol{s}_i + \boldsymbol{F}_i\| \leq \eta_i \|\boldsymbol{F}_i\|. \tag{6}$$

We then speak of an inexact Newton-Krylov method [6].

In [12] we discussed three methods of choosing the forcing terms η_i. Here we will use the method proposed by Eisenstat and Walker [9]. This method has been successfully used in practice on many different problems, and also provided very good results for us.

4 Preconditioning

Essential to the performance of a Krylov method like GMRES is a good preconditioner. See for example [17] for information on preconditioning. In our solver we use right preconditioning, meaning that we iteratively solve the linear system

$$J_i P_i^{-1} \boldsymbol{z}_i = -\boldsymbol{F}_i, \tag{7}$$

and then get the Newton step $\boldsymbol{s}_i$ by solving $P_i \boldsymbol{s}_i = \boldsymbol{z}_i$. For fast convergence the preconditioner matrix P_i should resemble the coefficient matrix J_i. At the same time we need a fast way to solve

linear systems of the form $P_i \boldsymbol{u}_i = \boldsymbol{v}_i$, as such a system has to be solved in each iteration of the linear solver.

In this paper we will construct preconditioners in the form of a product of a lower and upper triangular matrix $P_i = L_i U_i$. Any linear system with coefficient matrix P_i can then simply be solved using forward and backward substitution. To come to such a preconditioner we choose a target matrix Q_i and then construct either the LU factorization $L_i U_i = Q_i$ or an incomplete LU (ILU) factorization [13, 14] $L_i U_i$ of Q_i. In the case of the full LU factorization we get a preconditioner $P_i = L_i U_i = Q_i$, whereas with the ILU factorization the preconditioner $P_i = L_i U_i$ only resembles the target matrix Q_i. The trade-off here is that an ILU factorization is cheaper to build than an LU factorization, whereas the full LU factorization will generally results in a better preconditioner.

There are three choices that we will consider for the target matrix Q_i. These are the Jacobian matrix $Q_i = J_i$, the initial Jacobian matrix $Q_i = J_0$, and $Q_i = \Phi$, where

$$\Phi = \begin{bmatrix} B' & 0 \\ 0 & B'' \end{bmatrix}, \tag{8}$$

with B' and B'' as in the BX scheme of the Fast Decoupled Load Flow (FDLF) [22] method as proposed by van Amerongen [26].

The FDLF matrix Φ can be seen as an approximate Schur complement of the initial Jacobian matrix [15]. In previous studies Φ has already proven to be a good preconditioner [10], [12], while containing only half the non-zeros of the Jacobian matrix, thus providing benefits in computing time and memory.

Other preconditioners that are known to often work well for large problems are preconditioners based on iterative methods. Only stationary iterative methods can be used as a preconditioner for standard implementations of GMRES, Bi-CGSTAB, and IDR(s). Non-stationary iterative methods, like GMRES itself, can only be used with special flexible iterative methods, like FGMRES [16]. The use of FGMRES with a GMRES based preconditioner has been explored in [29].

Algebraic Multigrid (AMG) methods can also be used as a preconditioner. Running one cycle of AMG, with a stationary solve on the coarsest grid, leads to a stationary preconditioner. Such a preconditioner is very well suited for extremely large problems. For more information on AMG see [25, App. A].

5 Factorization

As mentioned in the previous section, we use complete and incomplete LU factorizations of the target matrix Q_i to construct our preconditioners. In this section we consider the quality and fill, i.e., the number of non-zeros, of the preconditioners and how to control these properties.

When using a full LU factorization the quality of the preconditioner is predetermined by the choice of the target matrix, as $P_i = Q_i$. However, when doing an LU decomposition on a sparse matrix we can expect the factors L_i and U_i to have more non-zeros than the original matrix Q_i. This is referred to as fill-in. The number of non-zeros in the factors divided by the number of non-zeros in the original matrix is called the fill-in ratio. The higher the fill-in ratio is, the more memory is needed, and the more computational time is needed for the factorization and the forward and backward substitutions. It is therefore paramount to try to minimize the fill-in of the LU factorization.

The fill-in can be controlled by reordering the rows and columns of the matrix that is to be factored. However, finding the ordering that minimizes fill-in has been proven to be NP-hard [28]. Many methods have been developed to quickly find a good reordering, see for example [8, 4].

The method of incomplete LU factorization that we use is ILU(k), where k is the number of levels of fill-in. This method determines the allowed non-zero positions in the factors based on the k-level, and subsequently calculates the best values for these non-zero positions.

As the fill-in is predetermined one might think that reordering is not important. However, even if the reordering does not influence the fill-in for an ILU(k) factorization, it does influence the quality with which L_iU_i approximates the target matrix Q_i.

In our experiments we have used the AMD reordering [1], which yielded very good results for both the LU and ILU(k) factorizations. To illustrate the impact of a good reordering, Table 1 shows the results of a single LU factorization of the initial Jacobian matrix J_0 for the uctew032 problem (see Table 2). The factorization time is measured in seconds.

reordering	none	AMD
factorization time	33.6	0.53
fill-in ratio	35	2.3

Table 1: LU factorization of J_0 for uctew032

The influence of reordering on the ILU(k) factorization is less drastic, but still very useful. For the uctew032 case, when solving the initial Jacobian system $J_0\boldsymbol{s}_0 = -\boldsymbol{F}_0$ with an ILU(k) factorization of J_0 as preconditioner, with different choices of k, we observed GMRES needing around 25% less iterations to converge when using the AMD reordering, compared to using no reordering.

6 Numerical Experiments

In this section we present the results of our numerical experiments. The test cases used are created using the UCTE winter 2008 model, as described in Section 1. Table 2 shows the number of busses and lines in the test problems, as well as the number of non-zeros in the Jacobian matrix nnz(J). The naming convention used is uctewXXX, where XXX is the number of times the model is copied and interconnected.

	busses	lines	nnz(J)
uctew001	4.25k	7.19k	62.7k
uctew002	8.51k	14.4k	125k
uctew004	17.0k	28.8k	251k
uctew008	34.0k	57.6k	502k
uctew016	68.0k	115k	1.00M
uctew032	136k	231k	2.01M
uctew064	272k	462k	4.02M
uctew128	544k	924k	8.05M
uctew256	1.09M	1.85M	16.1M

Table 2: Power flow test problems

All tests are run on a single core of a machine with Intel Core2 Duo 3GHz CPU and 16Gb memory. The problems are solved from a flat start, up to an accuracy of 10^{-6}.

First we consider the results using preconditioners based on the Jacobian matrix, i.e., $Q_i = J_i$. Fig. 1 shows the solve time in seconds versus the number of busses. Note that using a full LU factorization in this case is effectively the same as using a direct solver.

It is clear that the direct solver does not scale well in the problem size. For our test cases it is competitive up to about 150k busses, but deteriorates rapidly for larger problems. The largest problem took over an hour to solve with a direct linear solver, whereas the ILU(4), ILU(8), or ILU(12) factorization of J_i as preconditioner all solved the problem in less than a minute.

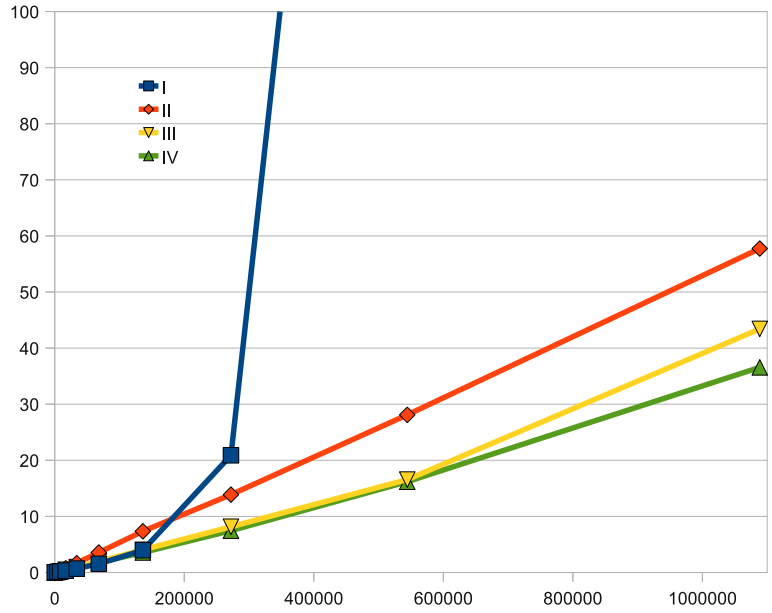

Figure 1: Solve time in seconds with Jacobian preconditioner
I: LU of J_i, II: ILU(4) of J_i, III: ILU(8) of J_i, IV: ILU(12) of J_i

We also tested k-levels higher than 12, but found that the factorization then started taking too much time, and the solver became slower. For our test data ILU(12) provided the optimum in terms of solve time on the largest cases.

Next we test preconditioners based on the initial Jacobian matrix, i.e., $Q_i = J_0$. Fig. 2 shows the solve times for these preconditioners.

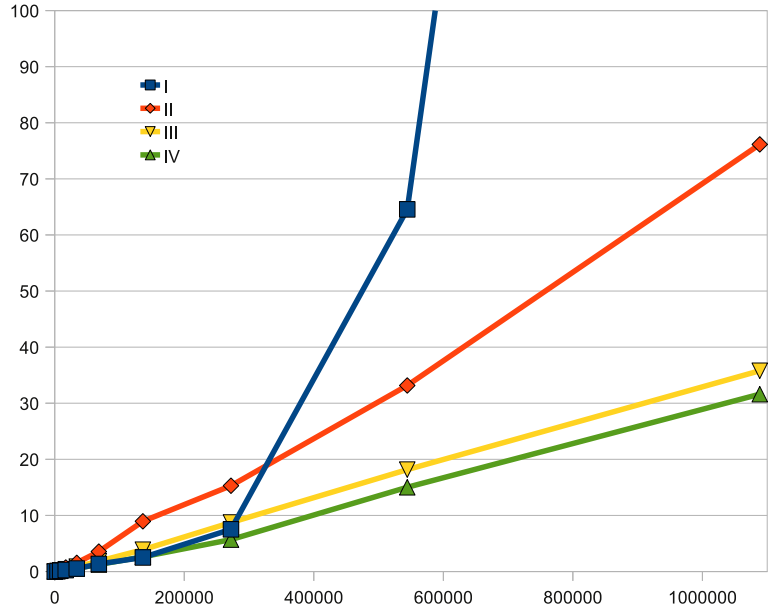

Figure 2: Solve time in seconds with initial Jacobian preconditioner
I: LU of J_0, II: ILU(4) of J_0, III: ILU(8) of J_0, IV: ILU(12) of J_0

Since the preconditioner is the same in each iteration, we only have to make a factorization once at the start. Therefore using the LU factorization, where the factorization is by far the most expensive computation in the entire solve, is much faster than for the direct solver, where we had to make a new factorization in each iteration.

After the first iteration, J_0 will not be as good a preconditioner as J_i, thus more GMRES iterations will be needed. We see that using ILU(4) of J_0 is slower than using ILU(4) of J_i. This is because the ILU(4) factorization is very fast, and most of the time is spend on GMRES iterations. With J_0 as preconditioner the number of GMRES iteration goes up by more than is saved by having to do only one factorization. However, for higher k-levels like ILU(12), the extra GMRES iterations needed when using J_0 cost less time than the extra factorizations needed for J_i.

Finally we experiment with the FDLF matrix as basis for our preconditioners, i.e., $Q_i = \Phi$. Fig. 3 shows the results.

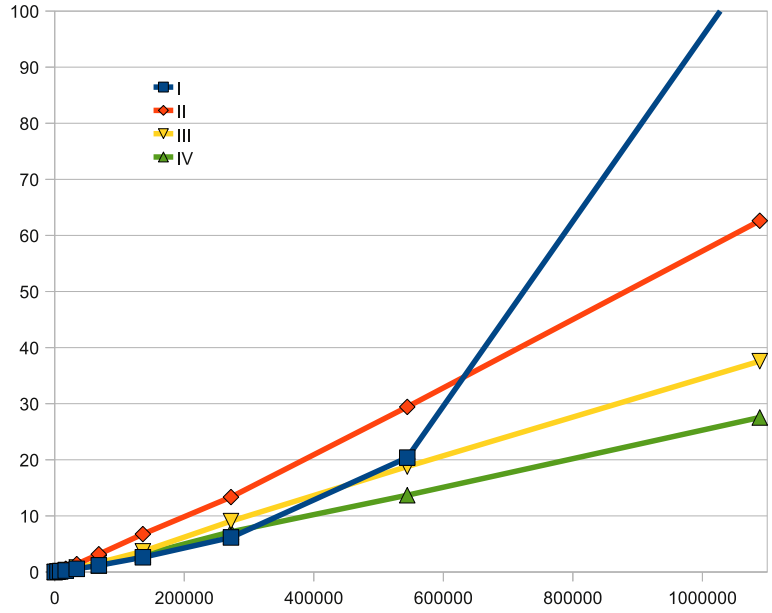

Figure 3: Solve time in seconds with FDLF preconditioner
I: LU of Φ, II: ILU(4) of Φ, III: ILU(8) of Φ, IV: ILU(12) of Φ

Again we only have to make a single factorization, and the target matrix has about half the non-zeros of the Jacobian. Because of this, using the LU factorization can stay competitive with the ILU(k) alternatives longer, up to problem sizes of around 300k busses. However, for larger problems the LU factorization again starts to show bad scaling behavior, and the ILU(12) factorization is a clear winner.

Inspection of the solve times showed that for our test problems using preconditioners based on J_i, while competitive for smaller problems, was never better than using $Q_i = J_0$ or $Q_i = \Phi$. In Fig. 4 we therefore compare the LU and ILU(12) factorizations of J_0 and Φ.

Obviously, for the largest problems the ILU(12) factorizations perform the best. The variant using ILU(12) of Φ is slightly faster, but close inspection of the solver output shows this is actually due to the inexact Newton-Krylov solver needing 8 Newton-Raphson iterations when using J_0, against 6 iterations for Φ. This has to do with being a bit lucky, or unlucky, at how the Newton step exactly turns out for a certain problem under the forcing term condition (6), much more than it has to do with the quality of the preconditioner.

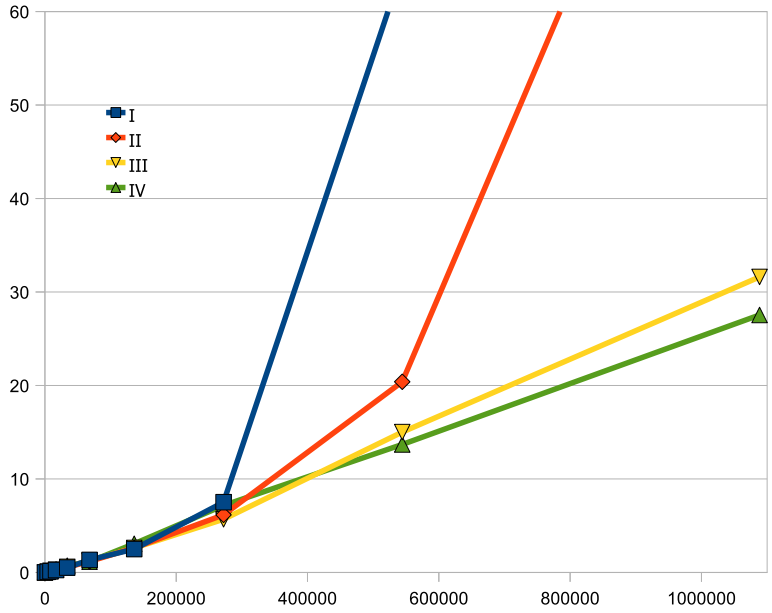

Figure 4: Solve time for all test problems with the best performing methods
I: LU of J_0, II: LU of Φ, III: ILU(12) of J_0, IV: ILU(12) of Φ

Note that under 150k busses it seems there is not much between these methods. Closer inspection reveals that this is indeed true, see Fig. 5. Differences here are again more due to the vagaries of the inexact Newton-Krylov method, than to the quality of the preconditioners.

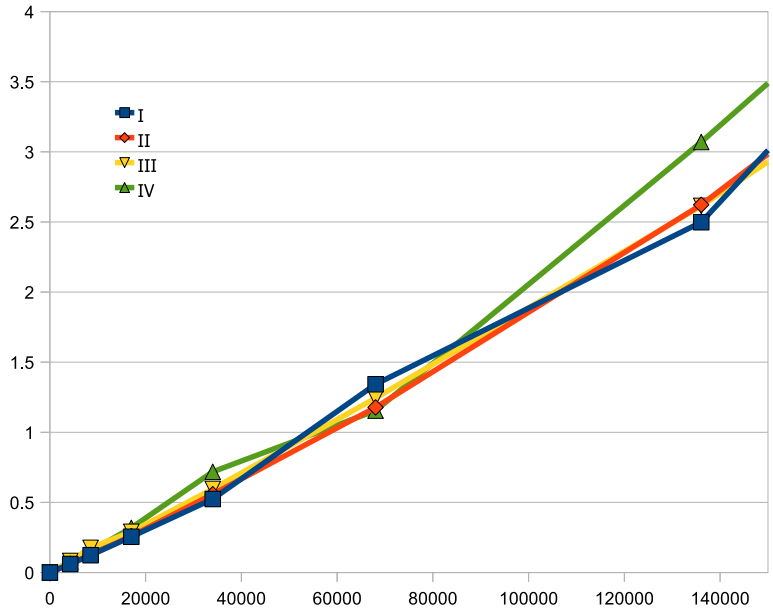

Figure 5: Solve time for the smaller problems with the best performing methods
I: LU of J_0, II: LU of Φ, III: ILU(12) of J_0, IV: ILU(12) of Φ

To validate our choice for GMRES as iterative solver, we have also tested Bi-CGSTAB with the same preconditioners. Bi-CGSTAB should be expected to outperform GMRES when a lot of

GMRES iterations are needed, meaning that Bi-CGSTAB will become a better alternative when the preconditioner becomes worse.

Our tests showed that for the largest test problem Bi-CGSTAB was faster than GMRES when using ILU(4) preconditioners, and slightly slower than GMRES with ILU(12). Using ILU(4) with Bi-CGSTAB was still significantly slower than using ILU(12) with either iterative linear solver. The results for ILU(8) were hard to compare because an extra Newton iteration was needed when using Bi-CGSTAB.

7 Conclusions

The experiments of the previous section clearly illustrates the bad scaling of the LU factorization in the problem size. Because of that bad scaling direct sparse solvers, but also Krylov methods with preconditioners based on the LU factorization, are not viable options for the solution of the linear systems that arise when solving power flow problems with the Newton-Raphson method. For the same reason the classical implementation of the Fast Decoupled Load Flow method is also not viable for very large problems.

To solve very large power flow problems, we need a scalable solver to solve the linear systems. In this paper we have proposed to use an iterative linear solver, in particular GMRES, with an ILU(k) factorization of the Jacobian matrix J_i, the initial Jacobian matrix J_0, or the FDLF matrix Φ, as preconditioner.

We have shown that a good reordering strategy is essential to reduce the fill-in of the LU factorization, but also to improve the quality of ILU factorizations. In [12] we already showed the importance of choosing good forcing terms for the inexact Newton-Krylov method. Using the AMD reordering, and the Eisenstat and Walker forcing terms, we have run experiments on power flow problems up to a million busses.

The experiments show that the ILU(k) preconditioners scale very well for the tested problem sizes. Factorizing a preconditioner once at the start, i.e., using J_0 or Φ, generally performed better than refactorizing in each iteration, i.e., using J_i. The best results were attained with an ILU(12) factorization of either J_0 or Φ as a preconditioner, solving the largest problem with over a million busses in around 30 seconds.

Experiments with Bi-CGSTAB showed to be very competitive. When using lower quality preconditioners, i.e., ILU factorization with lower k-levels, Bi-CGSTAB led to a faster power flow solver than GMRES. However, for the fastest ILU(12) based preconditioners GMRES was slightly faster than Bi-CGSTAB.

Acknowledgment

The authors would like to thank Robert van Amerongen for his many contributions, providing hands-on experience, and invaluable insight on the subject of power flow analysis. Further thanks are due to Barry Smith for his help with the PETSc package, and UCTE/ENTSO-E for providing the UCTE study model, and their assistance on using it.

References

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17(4):886–905, October 1996.

[2] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.1, Argonne National Laboratory, 2010. http://www.mcs.anl.gov/petsc/.

[3] D. Chaniotis and M. A. Pai. A new preconditioning technique for the GMRES algorithm in power flow and $P-V$ curve calculations. *Electrical Power and Energy Systems*, 25:239–245, 2003.

[4] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006.

[5] F. de León and A. Semlyen. Iterative solvers in the Newton power flow problem: preconditioners, inexact solutions and partial Jacobian updates. *IEE Proc. Gener. Transm. Distrib*, 149(4):479–484, 2002.

[6] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19(2):400–408, 1982.

[7] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, New Jersey, 1983.

[8] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, New York, 1986.

[9] S. C. Eisenstat and H. F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM J. Sci. Comput.*, 17(1):16–32, 1996.

[10] A. J. Flueck and H. D. Chiang. Solving the nonlinear power flow equations with an inexact Newton method using GMRES. *IEEE Transactions on Power Systems*, 13(2):267–273, 1998.

[11] R. Idema, D. J. P. Lahaye, and C. Vuik. Load flow literature survey. Report 09-04, Delft Institute of Applied Mathematics, Delft University of Technology, 2009.

[12] R. Idema, D. J. P. Lahaye, C. Vuik, and L. van der Sluis. Fast Newton load flow. In *Transmission and Distribution Conference and Exposition, 2010 IEEE PES*, pages 1–7, April 2010.

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

[14] J. A. Meijerink and H. A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *Journal of Computational Physics*, 44(1):134–155, 1981.

[15] A. J. Monticelli, A. Garcia, and O. R. Saavedra. Fast decoupled load flow: Hypothesis, derivations, and testing. *IEEE Transactions on Power Systems*, 5(4):1425–1431, 1990.

[16] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, March 1993.

[17] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, second edition, 2000.

[18] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[19] P. Schavemaker and L. van der Sluis. *Electrical Power System Essentials*. John Wiley & Sons, Chichester, 2008.

[20] G. L. G. Sleijpen, H. A. van der Vorst, and D. R. Fokkema. BiCGstab(ℓ) and other hybrid Bi-CG methods. *Numerical Algorithms*, 7:75–109, 1994.

[21] P. Sonneveld and M. B. van Gijzen. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM J. Sci. Comput.*, 31(2):1035–1062, 2008.

[22] B. Stott and O. Alsac. Fast decoupled load flow. *IEEE Transactions on Power Apparatus and Systems*, PAS-93(3):859–869, 1974.

[23] W. F. Tinney and C. E. Hart. Power flow solution by Newton's method. *IEEE Transactions on Power Apparatus and Systems*, PAS-86(11):1449–1449, 1967.

[24] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55(11):1801–1809, 1967.

[25] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.

[26] R. A. M. van Amerongen. A general-purpose version of the fast decoupled loadflow. *IEEE Transactions on Power Systems*, 4(2):760–770, 1989.

[27] H. A. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[28] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2(1):77–79, March 1981.

[29] Y.-S. Zhang and H.-D. Chiang. Fast Newton-FGMRES solver for large-scale power flow study. *IEEE Transactions on Power Systems*, 25(2):769–776, May 2010.