Newton-Krylov Methods in Power Flow and Contingency Analysis

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
vrijdag 23 november 2012 om 12:30 uur

door

Reijer IDEMA
wiskundig ingenieur

geboren te Broek op Langedijk

Dit proefschrift is goedgekeurd door de promotoren:

Prof.dr.ir. C. Vuik
Prof.ir. L. van der Sluis

Copromotor: Dr. D.J.P. Lahaye

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof.dr.ir. C. Vuik,	Technische Universiteit Delft, promotor
Prof.ir. L. van der Sluis,	Technische Universiteit Delft, promotor
Dr. D.J.P. Lahaye,	Technische Universiteit Delft, copromotor
Prof.dr.ir. C.W. Oosterlee,	Technische Universiteit Delft
Prof.dr.ir. J.G. Slootweg,	Technische Universiteit Eindhoven
Prof.dr. W.H.A. Schilders,	Technische Universiteit Eindhoven
Dr. I. Livshits,	Ball State University
Prof.dr.ir. H.X. Lin,	Technische Universiteit Delft, reservelid

Newton-Krylov Methods in Power Flow and Contingency Analysis.
Dissertation at Delft University of Technology.

ISBN 978-94-6191-538-2

SUMMARY

Newton-Krylov Methods in Power Flow and Contingency Analysis

Reijer Idema

A power system is a system that provides for the generation, transmission, and distribution of electrical energy. Power systems are considered to be the largest and most complex man-made systems. As electrical energy is vital to our society, power systems have to satisfy the highest security and reliability standards. At the same time, minimising cost and environmental impact are important issues.

Steady state power system analysis plays a very important role in both operational control and planning of power systems. Essential tools are power flow (or load flow) studies and contingency analysis. In power flow studies, the bus voltages in the power system are calculated given the generation and consumption. In contingency analysis, equipment outages are simulated to determine whether the system can still function properly if some piece of equipment were to break down unexpectedly.

The power flow problem can be mathematically expressed as a nonlinear system of equations. It is traditionally solved using the Newton-Raphson method with a direct linear solver, or using Fast Decoupled Load Flow (FDLF), an approximate Newton method designed specifically for the power flow problem. The Newton-Raphson method has good convergence properties, but the direct solver solves the linear system to a much higher accuracry than needed, especially in early iterations. In that respect the FDLF method is more efficient, but convergence is not as good. Both methods are slow for very large problems, due to the use of the LU decomposition.

We propose to solve power flow problems with Newton-Krylov methods. Newton-Krylov methods are inexact Newton methods that use a Krylov subspace method as linear solver. We discuss which Krylov method to use, investigate a range of preconditioners, and examine different methods for choosing the forcing terms. We also investigate the theoretical convergence of inexact Newton methods.

The resulting power flow solver offers the same convergence properties as the Newton-Raphson method with a direct linear solver, but eliminates both the need for oversolving, and the need for an LU factorisation. As a result, the method is slightly faster for small problems while scaling much better in the problem size, making it much faster for very large problems.

Contingency analysis gives rise to a large number of very similar power flow problems, which can be solved with any power flow solver. Using the solution of the base case as initial iterate for the contingency cases can help speed up the process. FDLF further allows the reuse of the LU factorisation of the base case for all contingency cases, through factor updating or compensation techniques. There is no equivalent technique for Newton power flow with a direct linear solver. We show that Newton-Krylov power flow does allow such techniques, through the use of a single preconditioner for all contingency cases. Newton-Krylov power flow thus allows very fast contingency analysis with Newton-Raphson convergence.

SAMENVATTING

Newton-Krylov Methoden in Loadflow en Contingency Analyse

Reijer Idema

Het energievoorzieningssysteem is het systeem dat zorgt voor de opwekking, transmissie en distributie van elektrische energie. Energievoorzieningssystemen vormen de grootste en ingewikkeldste systemen die door de mens zijn gemaakt. Omdat elektriciteit van cruciaal belang is voor onze samenleving, moeten energievoorzieningssystemen aan de hoogste veiligheids- en betrouwbaarheidseisen voldoen. Tegelijkertijd moet er rekening gehouden worden met de kosten en het milieu.

De analyse van een energievoorzieningssysteem in stationaire toestand is zeer belangrijk voor de planning en het operationele beheer van het systeem. Loadflow-studies en contingency analyse zijn hierbij essentieel. Gegeven het opgewekte en verbruikte vermogen kan de spanning in elk knooppunt van het systeem worden berekend door het loadflow-probleem op te lossen. Bij contingency analyse wordt het uitvallen van materieel gesimuleerd, om te bepalen of het systeem nog steeds naar behoren kan functioneren bij ongeplande uitval van dat materieel.

Het loadflow-probleem kan wiskundig worden beschreven als een niet-lineair stelsel van vergelijkingen. Het wordt gewoonlijk opgelost met behulp van de Newton-Raphson methode met een directe methode voor de lineaire vergelijkingen, of door middel van Fast Decoupled Load Flow (FDLF), een speciaal voor het loadflow-probleem ontwikkelde benadering van de Newton-Raphson methode. De Newton-Raphson methode heeft goede convergentie-eigenschappen, maar de directe methode lost de lineaire stelsels op tot een

veel hogere nauwkeurigheid dan nodig is. De FDLF methode is in dat opzicht efficiënter, maar de convergentie is minder goed. Beide methodes zijn traag voor zeer grote problemen omdat ze gebruik maken van de LU-decompositie.

Wij stellen voor het loadflow-probleem op te lossen met Newton-Krylov methoden. Newton-Krylov methoden zijn inexacte Newton methoden die een Krylov-deelruimte methode gebruiken om de lineaire stelsels op te lossen. We bespreken welke Krylov-methode gebruikt dient te worden, onderzoeken een scala aan preconditioneringen, en bekijken verschillende methoden voor de keuze van de nauwkeurigheid waarmee de lineaire stelsels opgelost moeten worden. Daarnaast onderzoeken we ook de theoretische convergentie van inexacte Newton methoden.

Het resultaat is een oplosmethode voor loadflow-problemen met dezelfde convergentie-eigenschappen als de Newton-Raphson methode, die de lineaire stelsels niet tot een hogere nauwkeurigheid dan nodig op hoeft te lossen en die geen LU decompositie nodig heeft. Hierdoor is de methode iets sneller voor kleine problemen en schaalt deze veel beter in de probleemgrootte, waardoor hij veel sneller is voor zeer grote problemen.

Contingency analyse leidt tot een groot aantal, zeer op elkaar gelijkende loadflow-problemen, die onafhankelijk van elkaar kunnen worden opgelost. Het proces kan vaak worden versneld door de oplossing van het basisprobleem te gebruiken als startoplossing voor de afgeleide problemen. FDLF staat verder toe dat de LU-decompositie van het basisprobleem wordt hergebruikt voor de afgeleide problemen, door middel van het bijwerken van de factoren of met behulp van compensatietechnieken. Bij gebruik van de Newton-Raphson methode kunnen deze technieken niet worden benut. Wij laten zien dat de Newton-Krylov methode zulke technieken wel toelaat, door dezelfde preconditionering te gebruiken voor alle afgeleide problemen in de contingency analyse. Newton-Krylov loadflow maakt het daardoor mogelijk contingency analyse zeer snel uit te voeren, met behoud van de goede Newton-Raphson convergentie-eigenschappen.

CONTENTS

CHAPTER 1

Introduction

Electricity is a vital part of modern everyday life. We plug our electronic devices into wall sockets and expect them to get power. This power is mostly generated in large power plants, in remote locations. Power generation is often in the news. Developments in wind and solar power generation, as well as other renewables, are hot topics. But also the issue of the depletion of natural resources, and the risks of nuclear power, are often discussed. Much less discussed is the transmission and distribution of electrical power, an incredibly complex task that needs to be executed very reliably and securely, and highly efficiently. To achieve this, both operation and planning require complex computational simulations of the power system network.

In this work we investigate the base computational problem in steady-state power system simulations—the power flow problem. The power flow (or load flow) problem is a nonlinear system of equations that relates the bus voltages to the power generation and consumption. For given generation and consumption, the power flow problem can be solved to reveal the associated voltages. The solution can be used to assess whether the power system can function properly for the given generation and consumption. Power flow is the main ingredient of many computations in power system analysis.

Monte Carlo simulations with power flow calculations for many different generation and consumption inputs, can be used to analyse the stochastic behaviour of a power system. This type of simulation is becoming especially important due to the uncontrollable nature of wind and solar power.

Contingency analysis simulates equipment outages in the power system, and solves the associated power flow problems to assess the impact on the power system. Contingency analysis is vital to identify possible problems, and solve them before they have a chance to occur. Many countries require their power system to operate in such a way that no single equipment outage causes interruption of service.

Operation and planning of power systems further lead to many kinds of optimisation problems. What power plants should be generating how much power at any given time? Where to best build a new power plant? Which buses to connect with a new line or cable? All these questions require the solution of an optimisation problem, where the set of feasible solutions is determined by power flow problems, or even contingency analysis and Monte Carlo simulations.

Traditionally, power generation is centralized in large power plants that are connected directly to the transmission system. The high voltage transmission system then transports the generated power to the lower voltage local distribution systems. In recent years decentralized power generation is emerging, for example in the form of small wind farms connected directly to the distribution network, or solar panels on the roofs of residential houses. It is expected that the future will bring a much more decentralized power system. This leads to many new computational challenges in power system operation and planning.

Meanwhile, national power systems are being interconnected more and more, and with it the energy markets. The resulting continent-wide power systems lead to much larger power system simulations.

Both these developments have the potential to lead to a whole new scale of power flow problems. For such problems, current power flow solution methods are not viable. Therefore, research into new solution techniques is very important.

In this work, we develop a Newton-Krylov solver that is much faster for large power flow problems than traditional solvers. Further, we use the contingency analysis problem to demonstrate how a Newton-Krylov solver can be used to speed up the computation of many slightly different power flow problems, as found not only in contingency analysis, but also in Monte Carlo simulations and some optimisation problems.

The research presented in this work was also published in [26, 28, 27, 29]. Further research on the subject of Newton-Krylov power flow for large power flow problems is presented in [30].

CHAPTER 2

Solving Linear Systems of Equations

A linear equation in n variables $x_1, \ldots, x_n \in \mathbb{R}$, is an equation of the form

$$a_1 x_1 + \ldots + a_n x_n = b, \tag{2.1}$$

with given constants $a_1, \ldots, a_n, b \in \mathbb{R}$. If there is at least one coefficient a_i not equal to 0, then the solution set is an $(n-1)$-dimensional affine hyperplane in $\mathbb{R}^n$. If all coefficients are equal to 0, then there is either no solution if $b \neq 0$, or the solution set is the entire space $\mathbb{R}^n$ if $b = 0$.

A linear system of equations is a collection of linear equations in the same variables, that have all to be satisfied simultaneously. Any linear system of m equations in n variables can be written as

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{2.2}$$

where $A \in \mathbb{R}^{m \times n}$ is called the coefficient matrix, $\boldsymbol{b} \in \mathbb{R}^m$ the right-hand side vector, and $\boldsymbol{x} \in \mathbb{R}^n$ the vector of variables or unknowns.

If there exists at least one solution vector $\boldsymbol{x}$ that satisfies all linear equations at the same time, then the linear system is called consistent; otherwise, it is called inconsistent. If the right-hand side vector $\boldsymbol{b} = \boldsymbol{0}$, then the system of equations is always consistent, because the trivial solution $\boldsymbol{x} = \boldsymbol{0}$ satisfies all equations independent of the coefficient matrix.

We focus on systems of linear equations with a square coefficient matrix:

$$A\boldsymbol{x} = \boldsymbol{b}, \text{ with } A \in \mathbb{R}^{n \times n} \text{ and } \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n. \tag{2.3}$$

If all equations are linearly independent, i.e, if $\operatorname{rank}(A) = n$, then the matrix A is invertible and the linear system (2.3) has a unique solution $\boldsymbol{x} = A^{-1}\boldsymbol{b}$.

If not all equations are linearly independent, i.e., if $\text{rank}\,(A) < n$, then A is singular. In this case the system is either inconsistent, or the solution set is a hyperplane of dimension $n - \text{rank}\,(A)$ in $\mathbb{R}^n$. Note that whether there is exactly one solution or not can be deduced from the coefficient matrix alone, while both coefficient matrix and right-hand side vector are needed to distinguish between no solutions or infinitely many solutions.

A solver for systems of linear equations can either be a direct method, or an iterative method. Direct methods calculate the solution to the problem in one pass. Iterative methods start with some initial vector, and update this vector in every iteration until it is close enough to the solution. Direct methods are very well-suited for smaller problems, and for problems with a dense coefficient matrix. For very large sparse problems, iterative methods are generally much more efficient than direct solvers.

2.1 Direct Solvers

A direct solver can consist of a method to calculate the inverse coefficient matrix A^{-1}, after which the solution of the linear system (2.3) can simply be found by calculating the matvec $\boldsymbol{x} = A^{-1}\boldsymbol{b}$. In practice, it is generally more efficient to build a factorisation of the coefficient matrix into triangular matrices, which can be used to easily derive the solution. For general matrices, the factorisation of choice is the LU decomposition.

2.1.1 LU Decomposition

The LU decomposition consists of a lower triangular matrix L, and an upper triangular matrix U, such that

$$LU = A. \tag{2.4}$$

The factors are unique if the requirement is added that all the diagonal elements of either L, or of U, are ones.

Using the LU decomposition, the system of linear equations (2.3) can be written as

$$LU\boldsymbol{x} = \boldsymbol{b}, \tag{2.5}$$

and solved by consecutively solving the two linear systems

$$L\boldsymbol{y} = \boldsymbol{b}, \tag{2.6}$$

$$U\boldsymbol{x} = \boldsymbol{y}. \tag{2.7}$$

Because L and U are triangular, these systems are quickly solved using forward and backward substitution respectively.

The rows and columns of the coefficient matrix A can be permuted freely without changing the solution of the linear system (2.3), as long as the vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ are per mutated accordingly. Using such permutations during the factorisation process is called pivoting. Allowing only row permutations is often referred to as partial pivoting.

Every invertible matrix A has an LU decomposition if partial pivoting is allowed. For some singular matrices an LU decomposition also exists, but for many there is no such factorisation possible. In general, direct solvers have problems with solving linear systems with singular coefficient matrices.

More information on the LU decomposition can be found in [19, 23, 25].

2.1.2 Solution Accuracy

Direct solvers are often said to calculate the exact solution, unlike iterative solvers, which calculate approximate solutions. Indeed, the algorithms of direct solvers lead to an exact solution in exact arithmetic. However, though the algorithms may be exact, the computers that execute them are not. Finite precision arithmetic may still introduce errors in the solution calculated by a direct solver.

During the factorisation process, rounding errors may lead to substantial inaccuracies in the factors. Errors in the factors can, in turn, lead to errors in the solution vector calculated by forward and backward substitution. Stability of the factorisation can be improved by using a good pivoting strategy during the process. The accuracy of the factors L and U can also be improved afterwards, by simple iterative refinement techniques [23].

2.1.3 Algorithmic Complexity

Forward and backward substitution operations have complexity $O(\mathrm{nnz}(A))$. For full coefficient matrices, the complexity of the LU decomposition is $O(n^3)$. For sparse matrix systems, special sparse methods improve on this, by exploiting the sparsity structure of the coefficient matrix. However, in general these methods still do not scale as well in the system size as iterative solvers can. Therefore, good iterative solvers will always be more efficient than direct solvers for very large sparse coefficient matrices.

To solve multiple systems of linear equations with the same coefficient matrix but different right-hand side vectors, it suffices to calculate the LU decomposition once at the start. Using this factorisation, the linear problem can be solved for each unique right-hand side by forward and backward substitution. Since the factorisation is far more time consuming than the substitution operations, this saves a lot of computational time compared to solving each linear system individually.

2.1.4 Fill-In and Matrix Ordering

In the LU decomposition of a sparse coefficient matrix A, there will be a certain amount of fill-in. Fill-in is the number of nonzero elements in L and U, of which the corresponding element in A is zero. Fill-in not only increases the amount of memory needed to store the factors, but also increases the complexity of the LU decomposition, as well as the forward and backward substitution operations.

The ordering of rows and columns—controlled by pivoting—can have a strong influence on the amount of fill-in. Finding the ordering that minimises fill-in has been proven to be NP-hard [57]. However, many methods have been developed that quickly find a good reordering, see for example [13, 19].

2.1.5 Incomplete LU decomposition

An incomplete LU decomposition [33, 34], or ILU decomposition, is a factorisation of A into a lower triangular matrix L, and an upper triangular matrix U, such that

$$LU \approx A. \tag{2.8}$$

The aim is to reduce computational cost by reducing the fill-in compared to the complete LU factors.

One method simply calculates the LU decomposition, and then drops all entries that are below a certain tolerance value. Obviously, this method does not reduce the complexity of the decomposition operation. However, the fill-in reduction saves memory, and reduces the computational cost of forward and backward substitution operations.

The ILU(k) method determines which entries in the factors L and U are allowed to be nonzero, based on the number of levels of fill $k \in \mathbb{N}$. ILU(0) is an incomplete LU decomposition such that $L + U$ has the same nonzero pattern as the original matrix A. For sparse matrices, this method is often much faster than the complete LU decomposition.

With an ILU(k) factorisation, the row and column ordering of A may still influence the number of nonzeros in the factors, although much less drastically than with the LU decomposition. Further, it has been observed in practice that the ordering also influences the quality of the approximation of the original matrix. A reordering that reduces the fill-in, often also reduces the approximation error for the ILU(k) factorisation.

It is clear that ILU factorisations are not suitable to be used in a direct solver, unless the approximation is very close to the original. In general, there is no point in using an ILU decomposition over the LU decomposition unless only a rough approximation of A is needed. ILU factorisations are often used a preconditioners for iterative linear solvers, see Section 2.2.4.

2.2 Iterative Solvers

Iterative solvers start with an initial iterate $\boldsymbol{x}_0$, and calculate a new iterate in each step, or iteration, thus producing a sequence of iterates $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots\}$. The aim is that at some iteration i, the iterate $\boldsymbol{x}_i$ will be close enough to the solution to be used as approximation of the solution. Since the true solution is not known, $\boldsymbol{x}_i$ cannot simply be compared with that solution to decide if it is close enough; a different measure of the error in $\boldsymbol{x}_i$ is needed.

The residual vector in iteration i is defined by

$$\boldsymbol{r}_i = \boldsymbol{b} - A\boldsymbol{x}_i. \tag{2.9}$$

Let $\boldsymbol{e}_i$ denote the difference between $\boldsymbol{x}_i$ and the true solution. Then it is clear that the norm of the residual

$$\|\boldsymbol{r}_i\| = \|\boldsymbol{b} - A\boldsymbol{x}_i\| = \|A\boldsymbol{e}_i\| = \|\boldsymbol{e}_i\|_{A^TA} \tag{2.10}$$

is a measure for the error in $\boldsymbol{x}_i$. The relative residual error $\frac{\|\boldsymbol{r}_i\|}{\|\boldsymbol{b}\|}$ can be used as a measure of the relative error in the iterate $\boldsymbol{x}_i$.

2.2.1 Krylov Subspace Methods

The Krylov subspace of dimension i, belonging to A and $\boldsymbol{r}_0$, is defined as

$$\mathcal{K}_i\left(A, \boldsymbol{r}_0\right) = \text{span}\left\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{i-1}\boldsymbol{r}_0, \right\}. \tag{2.11}$$

Krylov subspace methods are iterative linear solvers that generate iterates

$$\boldsymbol{x}_i \in \boldsymbol{x}_0 + \mathcal{K}_i\left(A, \boldsymbol{r}_0\right). \tag{2.12}$$

The simplest Krylov method consists of the Richardson iterations,

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{r}_i. \tag{2.13}$$

Basic iterative methods like the Jacobi, Gauss-Seidel, and Successive Over-Relaxation (SOR) iterations, can all be seen as preconditioned versions of the Richardson iterations. Preconditioning is treated in Section 2.2.4. More information on basic iterative methods can be found in [23, 40, 55].

Krylov subspace methods generally have no problem finding a solution for a consistent linear system with a singular coefficient matrix A. Indeed, the dimension of the Krylov subspace needed to describe the full column space of A is equal to $\text{rank}\,(A)$, and is therefore lower for singular matrices than for invertible matrices.

Popular iterative linear solvers for general coefficient matrices include GMRES [41], Bi-CGSTAB [54, 44], and IDR(s) [45]. These methods are more complex than the basic iterative methods, but generally converge a lot faster to a solution. All these iterative linear solvers can also be characterised as Krylov subspace methods. For an extensive treatment of Krylov subspace methods see [40].

2.2.2 Optimality and Short Recurrences

Two important properties of Krylov methods are the optimality property, and short recurrences. The first is about minimising the number of iterations needed to find a good approximation of the solution, while the second is about limiting the amount of computational work per iteration.

A Krylov method is said to have the optimality property, if in each iteration the computed iterate is the best possible approximation of the solution within current the Krylov subspace, i.e., if the residual norm $\|\boldsymbol{r}_i\|$ is minimised within the Krylov subspace. An iterative solver with the optimality property, is also called a minimal residual method.

An iterative process is said to have short recurrences if in each iteration only data from a small fixed number of previous iterations is used. If the needed amount of data and work keeps growing with the number of iterations, the algorithm is said to have long recurrences.

It has been proven that Kylov methods for general coefficient matrices can not have both the optimality property and short recurrences [21, 56]. Therefore, the Generalised Minimal Residual (GMRES) method necessarily has long recurrences. Using restarts or truncation, GMRES can be made into a short recurrence method without optimality. Bi-CGSTAB and IDR(s) have short recurrences, but do not meet the optimality property.

2.2.3 Algorithmic Complexity

The matrix and vector operations used in Krylov subspace methods are generally restricted to matvecs, vector updates, and inner products (see Sections A.2 and A.3). Of these operations, matvecs have the highest complexity with $O(\mathrm{nnz}(A))$. Therefore, the complexity of Krylov methods is $O(\mathrm{nnz}(A))$, provided convergence is reached in a limited number of steps.

The computational work for a Krylov method is often measured in the number matvecs, vector updates, and inner products used to increase the dimension of the Krylov subspace by one and find the new iterate within the expanded Krylov subspace. For short recurrence methods these numbers are fixed, while the computational work for methods with long recurrences grow with the iteration count.

2.2.4 Preconditioning

No Krylov subspace method can produce iterates that are better than the best approximation of the solution within the progressive Krylov subspaces, which are the iterates attained by minimal residual methods. In other words, the convergence of a Krylov subspace method is limited by the Krylov subspace. Preconditioning uses a preconditioner matrix M to change the Krylov subspace, in order to improve convergence of the iterative solver.

Left Preconditioning

The system of linear equations (2.3) with left preconditioning becomes

$$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}. \tag{2.14}$$

The preconditioned residual for this linear system of equations is

$$\boldsymbol{r}_i = M^{-1}\left(\boldsymbol{b} - A\boldsymbol{x}_i\right), \tag{2.15}$$

and the new Krylov subspace is

$$\mathcal{K}_i\left(M^{-1}A, M^{-1}\boldsymbol{r}_0\right), \tag{2.16}$$

Right Preconditioning

The system of linear equations (2.3) with right preconditioning becomes

$$AM^{-1}\boldsymbol{y} = \boldsymbol{b}, \text{ and } \boldsymbol{x} = M^{-1}\boldsymbol{y}. \tag{2.17}$$

The preconditioned residual is the same as the unpreconditioned residual:

$$\boldsymbol{r}_i = \boldsymbol{b} - A\boldsymbol{x}_i. \tag{2.18}$$

The Krylov subspace for this linear system of equations is

$$\mathcal{K}_i\left(AM^{-1}, \boldsymbol{r}_0\right). \tag{2.19}$$

However, this Krylov subspace is used to generate iterates $\boldsymbol{y}_i$, which are not solution iterates like $\boldsymbol{x}_i$. Solution iterates $\boldsymbol{x}_i$ can be produced by multiplying $\boldsymbol{y}_i$ by M^{-1}. This leads to vectors $\boldsymbol{x}_i$ that are in the Krylov subspace as with left preconditioning.

Split Preconditioning

Split preconditioning assumes some factorisation $M = M_L M_R$ of the preconditioner. The system of linear equations (2.3) then becomes

$$M_L^{-1}AM_R^{-1}\boldsymbol{y} = M_L^{-1}\boldsymbol{b}, \text{ and } \boldsymbol{x} = M_R^{-1}\boldsymbol{y}. \tag{2.20}$$

The preconditioned residual for this linear system of equations is

$$\boldsymbol{r}_i = M_L^{-1}\left(\boldsymbol{b} - A\boldsymbol{x}_i\right). \tag{2.21}$$

The Krylov subspace for the iterates $\boldsymbol{y}_i$ now is

$$\mathcal{K}_i\left(M_L^{-1}AM_R^{-1}, M_L^{-1}\boldsymbol{r}_0\right). \tag{2.22}$$

Transforming to solution iterates $\boldsymbol{x}_i = M_R^{-1}\boldsymbol{y}_i$, again leads to the same Krylov subspace as with left and right preconditioning.

Choosing the Preconditioner

Note that the explanation below assumes left preconditioning, but can be easily extended to right and split preconditioning.

To improve convergence, the preconditioner M needs to resemble the coefficient matrix A such that the preconditioned coefficient matrix $M^{-1}A$ resembles the identity matrix. At the same time, there should be a computationally cheap method available to evaluate $M^{-1}\boldsymbol{v}$ for any vector $\boldsymbol{v}$, because such an evaluation is needed in every preconditioned matvec in the Krylov subspace method.

A much used method is to create an LU decomposition of some matrix M that resembles A. In particular, an ILU decomposition of A can be used as a preconditioner. With such a preconditioner it is important to control the fill-in of the factors, so that the overall complexity of the method does not increase by much.

Another method of preconditioning, is to use an iterative linear solver to calculate a rough approximation of $\tilde{A}^{-1}\boldsymbol{v}$, and use this approximation instead of the explicit solution of $M^{-1}\boldsymbol{v}$. Here $\tilde{A}$ can be either the coefficient matrix A itself, or some convenient approximation of A. A stationary iterative linear solver can be used to precondition any Krylov subspace method, but nonstationary solvers require special flexible methods such as FGMRES [39].

2.2.5 Starting and Stopping

To start an iterative solver, an initial iterate $\boldsymbol{x}_0$ is needed. If some approximation of the solution of the linear system of equations is known, using it as initial iterate usually leads to fast convergence. If no such approximation is known, then usually the zero vector is chosen:

$$\boldsymbol{x}_0 = \boldsymbol{0}. \tag{2.23}$$

Another common choice is to use a random vector as initial iterate.

To stop the iteration process, some criterion is needed that indicates when to stop. By far the most common choice is to test if the relative residual error has become small enough, i.e., if for some choice of $\delta < 1$

$$\frac{\|\boldsymbol{r}_i\|}{\|\boldsymbol{b}\|} < \delta. \tag{2.24}$$

If left or split preconditioning is used, it is important to think about whether the true residual or the preconditioned residual should be used in the stopping criterion.

CHAPTER 3

Solving Nonlinear Systems of Equations

A nonlinear equation in n variables $x_1, \ldots, x_n \in \mathbb{R}$, is an equation

$$f(x_1, \ldots, x_n) = 0, \tag{3.1}$$

that is not a linear equation.

A nonlinear system of equations is a collection of equations of which at least one equation is nonlinear. Any nonlinear system of m equations in n variables can be written as

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}, \tag{3.2}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the vector of variables or unknowns, and $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a vector of m functions in $\boldsymbol{x}$, i.e.,

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{bmatrix} F_1(\boldsymbol{x}) \\ \vdots \\ F_m(\boldsymbol{x}) \end{bmatrix}. \tag{3.3}$$

A solution of a nonlinear system of equations (3.2), is a vector $\boldsymbol{x}^* \in \mathbb{R}^n$ such that $F_k(\boldsymbol{x}^*) = 0$ for all $k \in \{1, \ldots, m\}$ at the same time. In this work, we restrict ourselves to nonlinear systems of equations with the same number of variables as there are equations, i.e., $m = n$.

It is not possible to solve a general nonlinear equation analytically, let alone a general nonlinear system of equations. However, there are iterative methods to find a solution for such systems. The Newton-Raphson algorithm is the standard method to solve nonlinear systems of equations. Most, if not all, other well-performing methods can be derived from the Newton-Raphson algorithm. In this chapter the Newton-Raphson method is treated, as well as some common variations.

3.1 Newton-Raphson Methods

The Newton-Raphson method is an iterative process used to solve nonlinear systems of equations

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}, \tag{3.4}$$

where $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable. In each iteration, the method solves a linearisation of the nonlinear problem around the current iterate, to find an update for that iterate. Algorithm 3.1 shows the basic Newton-Raphson process.

Algorithm 3.1 Newton-Raphson Method

1: $i := 0$
2: given initial iterate $\boldsymbol{x}_0$
3: **while** not converged **do**
4: solve $-J(\boldsymbol{x}_i)\,\boldsymbol{s}_i = \boldsymbol{F}(\boldsymbol{x}_i)$
5: update iterate $\boldsymbol{x}_{i+1} := \boldsymbol{x}_i + \boldsymbol{s}_i$
6: $i := i + 1$
7: **end while**

In Algorithm 3.1, the matrix J represents the Jacobian of $\boldsymbol{F}$, i.e.,

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix}. \tag{3.5}$$

The Jacobian system

$$-J(\boldsymbol{x}_i)\,\boldsymbol{s}_i = \boldsymbol{F}(\boldsymbol{x}_i) \tag{3.6}$$

can be solved using any linear solver. When a Krylov subspace method is used, we speak of a Newton-Krylov method.

The Newton process has local quadratic convergence. This means that if the iterate $\boldsymbol{x}_I$ is close enough to the solution, then there is a $c \geq 0$ such that for all $i \geq I$

$$\|\boldsymbol{x}_{i+1} - \boldsymbol{x}^*\| \leq c\|\boldsymbol{x}_i - \boldsymbol{x}^*\|^2. \tag{3.7}$$

The basic Newton method is not globally convergent, meaning that there are problems for which it does not converge to a solution from every initial iterate $\boldsymbol{x}_0$. Line search and trust region methods can be used to augment the Newton method, to improve convergence if the initial iterate is far away from the solution, see Section 3.2.

As with iterative linear solvers, the distance of the current iterate to the solution is not known. The vector $\boldsymbol{F}(\boldsymbol{x}_i)$ can be seen as the nonlinear residual vector of iteration i. Convergence of the method is therefore mostly measured in the residual norm $\|\boldsymbol{F}(\boldsymbol{x}_i)\|$, or relative residual norm $\frac{\|\boldsymbol{F}(\boldsymbol{x}_i)\|}{\|\boldsymbol{F}(\boldsymbol{x}_0)\|}$.

3.1.1 Inexact Newton

Inexact Newton methods [15] are Newton-Raphson methods in which the Jacobian system (3.6) is not solved to full accuracy. Instead, in each Newton iteration the Jacobian system is solved such that

$$\frac{\|\boldsymbol{r}_i\|}{\|\boldsymbol{F}(\boldsymbol{x}_i)\|} \leq \eta_i, \tag{3.8}$$

where

$$\boldsymbol{r}_i = \boldsymbol{F}(\boldsymbol{x}_i) + J(\boldsymbol{x}_i)\,\boldsymbol{s}_i. \tag{3.9}$$

The values η_i are called the forcing terms.

The most common form of inexact Newton methods, is with an iterative linear solver to solve the Jacobian systems. The forcing terms then determine the accuracy to which the Jacobian system is solved in each Newton iteration. However, approximate Jacobian Newton methods and Jacobian-free Newton methods, treated in Section 3.1.2 and Section 3.1.3 respectively, can also be seen as inexact Newton methods. The general inexact Newton method is shown in Algorithm 3.2.

Algorithm 3.2 Inexact Newton Method

1: $i := 0$
2: given initial solution $\boldsymbol{x}_0$
3: **while** not converged **do**
4: solve $-J(\boldsymbol{x}_i)\,\boldsymbol{s}_i = \boldsymbol{F}(\boldsymbol{x}_i)$ such that $\|\boldsymbol{r}_i\| \leq \eta_i \|\boldsymbol{F}(\boldsymbol{x}_i)\|$
5: update iterate $\boldsymbol{x}_{i+1} := \boldsymbol{x}_i + \boldsymbol{s}_i$
6: $i := i + 1$
7: **end while**

The convergence behaviour of the method strongly depends on the choice of the forcing terms. Convergence results derived in [15] are summarised in Table 3.1. In Chapter 4 we present our own theoretical results on local convergence for inexact Newton methods, proving that the local convergence factor is arbitrarily close to η_i in each iteration, for properly chosen forcing terms. This result is reflected by the final row of Table 3.1, where $\alpha > 0$ can be chosen arbitrarily small. The specific conditions under which these convergence results hold, can be found in [15] and Chapter 4 respectively.

If a forcing term is chosen too small, then the nonlinear error generally reduces much less than the linear error in that iteration. This is called oversolving. In general, the closer the current iterate is to the solution, the smaller the forcing terms can be chosen without oversolving. Over the years, a lot of effort has been invested in finding good strategies for choosing the forcing terms. Some examples can be found in [16], [20], [24].

forcing terms	local convergence
$\eta_i < 1$	linear
$\limsup_{i\to\infty} \eta_i = 0$	superlinear
$\limsup_{i\to\infty} \frac{\eta_i}{\|\boldsymbol{F}_i\|^p} < \infty,\ p \in (0,1)$	order at least $1+p$
$\eta_i < 1$	factor $(1+\alpha)\,\eta_i$

Table 3.1: Local convergence for inexact Newton methods

3.1.2 Approximate Jacobian Newton

The Jacobian of the function $\boldsymbol{F}(\boldsymbol{x})$ is not always available in practice. For example, it is possible that $\boldsymbol{F}(\boldsymbol{x})$ can be evaluated in any point by some method, but no analytical formulation is known. Then it is impossible to calculate the derivatives analytically. Or, if an analytical form is available, calculating the derivatives may simply be too computationally expensive.

In such cases, the Newton method may be used with appropriate approximations of the Jacobian matrices. The most widely used Jacobian matrix approximation is based on finite differences:

$$J_{ij}(\boldsymbol{x}) = \frac{\partial F_i}{\partial x_j}(\boldsymbol{x}) \approx \frac{F_i(\boldsymbol{x} + \delta \boldsymbol{e}_j) - F_i(\boldsymbol{x})}{\delta}, \tag{3.10}$$

where $\boldsymbol{e}_j$ is the vector with element j equal to 1, and all other elements equal to 0. For small enough δ, this is a good approximation of the derivative.

3.1.3 Jacobian-Free Newton

In some Newton-Raphson procedures the use of an explicit Jacobian matrix can be avoided. If done so, the method is called a Jacobian-free Newton method. A Jacobian-free Newton method is needed if the nonlinear problem is too large for the Jacobian to be stored in memory explicitly. Jacobian-free Newton methods can also be used as an alternative to approximate Jacobian Newton methods, if no analytical formulation of $\boldsymbol{F}(\boldsymbol{x})$ is known, or if the Jacobian is too computationally expensive to calculate.

Consider Newton-Krylov methods, where the Krylov solver only uses the Jacobian in matrix-vector products of the form $J(\boldsymbol{x})\,\boldsymbol{v}$. These products can be approximated by the directional finite difference scheme

$$J(\boldsymbol{x})\,\boldsymbol{v} \approx \frac{\boldsymbol{F}(\boldsymbol{x} + \delta \boldsymbol{v}) - \boldsymbol{F}(\boldsymbol{x})}{\delta}, \tag{3.11}$$

removing the need to store the Jacobian matrix explicitly. For more information see [31], and the references therein.

3.2 Newton-Raphson with Global Convergence

Line search and trust region methods are iterative processes that can be used to find a local minimum in unconstrained optimisation. Both methods have global convergence to such a minimiser.

Unconstrained optimisation techniques may be used to find roots of $\|\boldsymbol{F}\|$, which are the solutions of the nonlinear problem (3.2). Since line search and trust region methods ensure global convergence to a local minimum of $\|\boldsymbol{F}\|$, if all such minima are roots of $\boldsymbol{F}$, then these methods have global convergence to a solution of the nonlinear problem. However, if there is a local minimum that is not a root of $\|\boldsymbol{F}\|$, then the algorithm may terminate without finding a solution. In this case, the method is usually restarted from a different initial iterate, in the hope of finding a different local minimum that is a solution of the nonlinear system.

Near the solution, line search and trust region methods generally converge much slower than the Newton-Raphson method, but they can be used in conjunction with the Newton process to improve convergence farther from the solution. Both methods use their own criterion which the update vector has to satisfy. Whenever the Newton step satisfies this criterion then it is used to update the iterate normally. If the criterion is not satisfied, then some alternative update vector is calculated that does satisfy the criterion.

3.2.1 Line Search

The idea behind augmenting the Newton-Raphson method with line search is simple. Instead of updating the iterate $\boldsymbol{x}_i$ with the Newton step $\boldsymbol{s}_i$, it is updated with some vector $\lambda_i \boldsymbol{s}_i$ along the Newton step direction, i.e.,

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \lambda_i \boldsymbol{s}_i. \tag{3.12}$$

Ideally, λ_i is chosen such that $\|\boldsymbol{F}(\boldsymbol{x}_i + \lambda_i \boldsymbol{s}_i)\|$ is minimised over λ_i. Below a strategy is outlined for finding a good value for λ_i, starting with the introduction of a convenient mathematical description of the problem. Note that $\boldsymbol{F}(\boldsymbol{x}_i) \neq \boldsymbol{0}$, as otherwise the nonlinear problem is already solved with solution $\boldsymbol{x}_i$. In the remainder of this section, the iteration index i dropped for readability.

Define the positive function

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{F}(\boldsymbol{x})\|^2 = \frac{1}{2}\boldsymbol{F}(\boldsymbol{x})^T \boldsymbol{F}(\boldsymbol{x}), \tag{3.13}$$

and note that

$$\nabla f(\boldsymbol{x}) = J(\boldsymbol{x})^T \boldsymbol{F}(\boldsymbol{x}). \tag{3.14}$$

A vector $\boldsymbol{s}$ is called a descent direction of f in $\boldsymbol{x}$, if

$$\nabla f(\boldsymbol{x})^T \boldsymbol{s} < 0. \tag{3.15}$$

The Newton direction $\boldsymbol{s} = -J(\boldsymbol{x})^{-1}\boldsymbol{F}(\boldsymbol{x})$ is a descent direction, since

$$\nabla f(\boldsymbol{x})^T \boldsymbol{s} = -\boldsymbol{F}(\boldsymbol{x})^T J(\boldsymbol{x}) J(\boldsymbol{x})^{-1} \boldsymbol{F}(\boldsymbol{x}) = -\|\boldsymbol{F}(\boldsymbol{x})\|^2 < 0. \tag{3.16}$$

Now define the nonnegative function

$$g(\lambda) = f(\boldsymbol{x} + \lambda \boldsymbol{s}) = \frac{1}{2}\boldsymbol{F}(\boldsymbol{x} + \lambda \boldsymbol{s})^T \boldsymbol{F}(\boldsymbol{x} + \lambda \boldsymbol{s}). \tag{3.17}$$

A minimiser of g also minimises the value of $\|\boldsymbol{F}(\boldsymbol{x} + \lambda \boldsymbol{s})\|$. Thus the best choice for λ is given by

$$\hat{\lambda} = \arg\min_{\lambda} g(\lambda). \tag{3.18}$$

It is generally not possible to solve minimisation problem (3.18) analytically, but there are plenty methods to find a numerical approximation of $\hat{\lambda}$. In practice, a rough estimate suffices.

The decrease of f is regarded as sufficient, if λ satisfies the Armijo rule [5]

$$f(\boldsymbol{x} + \lambda \boldsymbol{s}) \leq f(\boldsymbol{x}) + \alpha \lambda \nabla f(\boldsymbol{x})^T \boldsymbol{s}, \tag{3.19}$$

where $\alpha \in (0, 1)$. A typical choice that often yields good results is $\alpha = 10^{-4}$. Note that for the Newton direction, we can write the Armijo rule (3.19) as

$$\|\boldsymbol{F}(\boldsymbol{x} + \lambda \boldsymbol{s})\|^2 \leq (1 - 2\alpha\lambda)\|\boldsymbol{F}(\boldsymbol{x})\|^2. \tag{3.20}$$

The common method to find a satisfactory value for λ, is to start with $\lambda^0 = 1$, and—while relation (3.19) is not satisfied—backtrack by setting

$$\lambda^{k+1} = \rho^k \lambda^k, \quad \rho^k \in [0.1, 0.5]. \tag{3.21}$$

The interval restriction on ρ^k is called safeguarding.

Since $\boldsymbol{s}$ is a descent direction, at some point the Armijo rule should be satisfied. The reduction factor ρ^k for λ^k, is chosen such that

$$\rho^k = \arg\min_{\rho^k \in [0.1, 0.5]} h\left(\rho^k \lambda^k\right), \tag{3.22}$$

where h is a quadratic polynomial model of f. This model h is made as a parabola through either the values $g(0)$, $g'(0)$, and $g\left(\lambda^k\right)$, or the values $g(0)$, $g\left(\lambda^{k-1}\right)$, and $g\left(\lambda^k\right)$. Note that for the Newton direction

$$g'(0) = \nabla f(\boldsymbol{x})^T \boldsymbol{s} = -\|\boldsymbol{F}(\boldsymbol{x})\|^2. \tag{3.23}$$

Further note that the second model can only be used from the second iteration onward, and λ^1 has to be chosen without the use of the model, for example by setting $\lambda^1 = 0.5$.

For more information on line search methods see for example [18]. For line search applied to inexact Newton-Krylov methods, see [8].

3.2.2 Trust Regions

Trust region methods define a region around the current iterate $\boldsymbol{x}_i$ that is trusted, and require the update step $\boldsymbol{s}_i$ to be such that the new iterate $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{s}_i$ lies within this trusted region. In this section again, the iteration index i is dropped for readability.

Assume the trust region to be a hypersphere, i.e.,

$$\|\boldsymbol{s}\| \leq \delta. \tag{3.24}$$

The goal is find the best possible update within the trust region.

Finding the update that minimises $\|\boldsymbol{F}\|$ within the trust region may be as hard as solving the nonlinear problem itself. Instead, the method searches for an update that satisfies

$$\min_{\|\boldsymbol{s}\| \leq \delta} q(\boldsymbol{s}), \tag{3.25}$$

with $q(\boldsymbol{s})$ the quadratic model of $\boldsymbol{F}(\boldsymbol{x} + \boldsymbol{s})$ given by

$$q(\boldsymbol{s}) = \frac{1}{2}\|\boldsymbol{r}\|^2 = \frac{1}{2}\|\boldsymbol{F} + J\boldsymbol{s}\|^2 = \frac{1}{2}\boldsymbol{F}^T\boldsymbol{F} + \left(J^T\boldsymbol{F}\right)^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T J^T J\boldsymbol{s}, \tag{3.26}$$

where $\boldsymbol{F}$ and J are short for $\boldsymbol{F}(\boldsymbol{x})$ and $J(\boldsymbol{x})$ respectively.

The global minimum of the quadratic model $q(\boldsymbol{s})$, is attained at the Newton step $\boldsymbol{s}^N = -J(\boldsymbol{x})^{-1}\boldsymbol{F}(\boldsymbol{x})$, with $q\left(\boldsymbol{s}^N\right) = 0$. Thus, if the Newton step is within the trust region, i.e., if $\|\boldsymbol{s}^N\| \leq \delta$, then the current iterate is updated with the Newton step. However, if the Newton step is outside the trust region, it is not a valid update step.

It has been proven that problem (3.25) is solved by

$$\boldsymbol{s}(\mu) = \left(J(\boldsymbol{x})^T J(\boldsymbol{x}) + \mu I\right)^{-1} J(\boldsymbol{x})^T \boldsymbol{F}(\boldsymbol{x}), \tag{3.27}$$

for the unique μ for which $\|\boldsymbol{s}(\mu)\| = \delta$. See for example [18, Lemma 6.4.1], or [10, Theorem 7.2.1].

Finding this update vector $\boldsymbol{s}(\mu)$ is very hard, but there are fast methods to get a useful estimate, such as the hook step and the (double) dogleg step. The hook step method uses an iterative process to calculate update steps $\boldsymbol{s}(\mu)$ until $\|\boldsymbol{s}(\mu)\| \approx \delta$. Dogleg steps are calculated by making a piecewise linear approximation of the curve $\boldsymbol{s}(\mu)$, and taking the new iterate as the point where this approximation curve intersects the trust region boundary.

An essential part of making trust region methods work, is using suitable trust regions. Each time a new iterate is calculated it has to be decided if it is acceptable, and the size of the trust region has to be adjusted accordingly.

For an extensive treatment of trust regions methods see [10]. For trust region methods applied to inexact Newton-Krylov methods, see [8].

CHAPTER 4

Convergence Theory

The Newton-Raphson method (see Chapter 3) is usually the method of choice to solve systems of nonlinear equations. In power system analysis, power flow computations lead to systems of nonlinear equations, which are also mostly solved using Newton methods (see Chapters 5 and 6). In our research into improving power flow computations for large power systems, we have investigated the application of inexact Newton-Krylov methods to power flow problems (see Chapter 7).

In the analysis of our numerical power flow experiments, some interesting behaviour surfaced. Our method converged quadratically in the Newton iterations, as expected from Newton convergence theory. At the same time, however, the convergence was approximately linear in the total number of linear solver iterations performed during the Newton iterations. This observation led us to investigate the theoretical convergence of inexact Newton methods. The results of this investigation are presented in this chapter.

In Section 4.1 the theoretical convergence of general inexact iterative methods is investigated. In Section 4.2 the result is formalised for the inexact Newton method, which also allows the explanation of the linear convergence observed in our power flow experiments. In Section 4.3 some numerical experiments are presented to illustrate how the theoretical results translate to practice. Finally, in Section 4.4 some applications are discussed.

4.1 Convergence of Inexact Iterative Methods

Assume an iterative method that, given current iterate $\boldsymbol{x}_i$, has some way to exactly determine a unique new iterate $\hat{\boldsymbol{x}}_{i+1}$. If instead an approximation $\boldsymbol{x}_{i+1}$ of the exact iterate $\hat{\boldsymbol{x}}_{i+1}$ is used to continue the process, we speak of an inexact iterative method. Inexact Newton methods (see Section 3.1.1) are

examples of inexact iterative methods. Figure 4.1 illustrates a single step of an inexact iterative method.

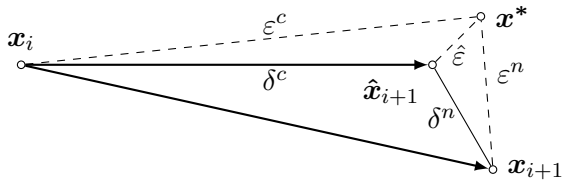

Figure 4.1: Inexact iterative step

Note that

$$\delta^c = \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_{i+1}\| > 0, \tag{4.1}$$

$$\delta^n = \|\boldsymbol{x}_{i+1} - \hat{\boldsymbol{x}}_{i+1}\| \geq 0, \tag{4.2}$$

$$\varepsilon^c = \|\boldsymbol{x}_i - \boldsymbol{x}^*\| > 0 \tag{4.3}$$

$$\varepsilon^n = \|\boldsymbol{x}_{i+1} - \boldsymbol{x}^*\|, \tag{4.4}$$

$$\hat{\varepsilon} = \|\hat{\boldsymbol{x}}_{i+1} - \boldsymbol{x}^*\| \geq 0. \tag{4.5}$$

Further, define γ as the distance of the exact iterate $\hat{\boldsymbol{x}}_{i+1}$ to the solution, relative to the length δ^c of the exact update step, i.e.,

$$\gamma = \frac{\hat{\varepsilon}}{\delta^c} > 0. \tag{4.6}$$

The ratio $\frac{\varepsilon^n}{\varepsilon^c}$ is a measure for the improvement of the inexact iterate $\boldsymbol{x}_{i+1}$ over the current iterate $\boldsymbol{x}_i$, in terms of the distance to the solution $\boldsymbol{x}^*$. Likewise, the ratio $\frac{\delta^n}{\delta^c}$ is a measure for the improvement of the inexact iterate $\boldsymbol{x}_{i+1}$, in terms of the distance to the exact iterate $\hat{\boldsymbol{x}}_{i+1}$. As the solution is unknown, so is the ratio $\frac{\varepsilon^n}{\varepsilon^c}$. Assume, however, that some measure for the ratio $\frac{\delta^n}{\delta^c}$ is available, and that it can be controlled. For example, for an inexact Newton method the relative linear residual norm $\frac{\|\boldsymbol{r}_k\|}{\|\boldsymbol{F}(\boldsymbol{x}_i)\|}$, controlled by the forcing term η_i, can be used as a measure for $\frac{\delta^n}{\delta^c}$.

The aim is to have an improvement in the controllable error translate into a similar improvement in the distance to the solution, i.e., to have

$$\frac{\varepsilon^n}{\varepsilon^c} \leq (1+\alpha)\,\frac{\delta^n}{\delta^c} \tag{4.7}$$

for some reasonably small $\alpha > 0$.

The worst case scenario can be identified as

$$\max \frac{\varepsilon^n}{\varepsilon^c} = \frac{\delta^n + \hat{\varepsilon}}{|\delta^c - \hat{\varepsilon}|} = \frac{\delta^n + \gamma\delta^c}{|1-\gamma|\,\delta^c} = \frac{1}{|1-\gamma|}\frac{\delta^n}{\delta^c} + \frac{\gamma}{|1-\gamma|}. \tag{4.8}$$

To guarantee that the inexact iterate $\boldsymbol{x}_{i+1}$ is an improvement over $\boldsymbol{x}_i$, using equation (4.8), it is required that

$$\frac{1}{|1-\gamma|}\frac{\delta^n}{\delta^c} + \frac{\gamma}{|1-\gamma|} < 1 \Leftrightarrow \frac{\delta^n}{\delta^c} + \gamma < |1-\gamma| \Leftrightarrow \frac{\delta^n}{\delta^c} < |1-\gamma| - \gamma. \quad (4.9)$$

If $\gamma \geq 1$ this would mean that $\frac{\delta^n}{\delta^c} < -1$, which is impossible. Therefore, to guarantee a reduction of the distance to the solution, it is required that

$$\frac{\delta^n}{\delta^c} < 1 - 2\gamma \Leftrightarrow 2\gamma < 1 - \frac{\delta^n}{\delta^c} \Leftrightarrow \gamma < \frac{1}{2} - \frac{1}{2}\frac{\delta^n}{\delta^c}. \quad (4.10)$$

As a result, the absolute operators can be dropped from equation (4.8).

Note that if the iterative method converges to the solution superlinearly, then γ goes to 0 with the same rate of convergence. Thus, at some point in the iteration process equation (4.10) is guaranteed to hold. This is in particular the case for an inexact Newton method, if it converges, as convergence is quadratic once the iterate is close enough to the solution.

Figure 4.2 shows plots of equation (4.8) on a logarithmic scale for several values of γ. The horizontal axis shows the number of digits improvement in the distance to the exact iterate: $d_\delta = -\log \frac{\delta^n}{\delta^c}$. The vertical axis depicts the resulting minimum number of digits improvement in the distance to the solution: $d_\varepsilon = -\log\left(\max \frac{\varepsilon^n}{\varepsilon^c}\right)$.

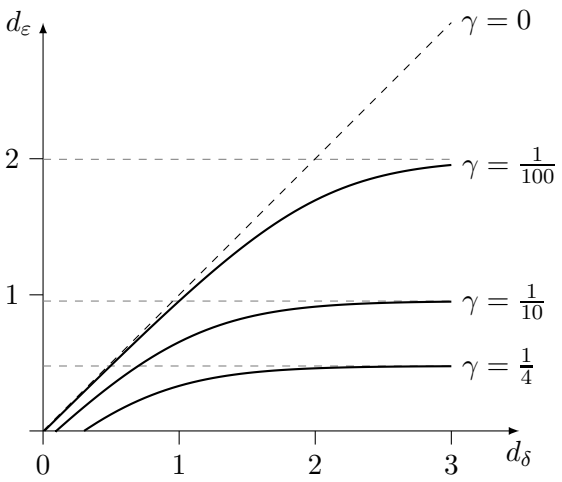

Figure 4.2: Number of digits improvement

For fixed d_δ, the smaller the value of γ, the better the resulting d_ε is. For $\gamma = \frac{1}{10}$, there is a significant start-up cost on d_δ before d_ε becomes positive, and a full digit improvement on the distance to the solution can never be guaranteed. Making more than a 2 digit improvement in the distance to the exact iterate results in a lot of effort with hardly any return at $\gamma = \frac{1}{10}$. However, when $\gamma = \frac{1}{100}$ there is hardly any start-up cost on d_δ any more,

and the guaranteed improvement in the distance to the solution can be taken up to about 2 digits.

The above mentioned start-up cost can be derived from equation (4.10) to be $d_\delta = -\log(1-2\gamma)$. The asymptote to which d_ε approaches is given by $d_\varepsilon = -\log\left(\frac{\gamma}{1-\gamma}\right) = \log\left(\frac{1}{\gamma}-1\right)$, which is the improvement obtained when taking the exact iterate.

The value α, as introduced in equation (4.7), is a measure of how far the graph of d_ε deviates from the ideal $d_\varepsilon = d_\delta$, which is attained only in the fictitious case that $\gamma = 0$. Combining equations (4.7) and (4.8), the minimum value of α can be investigated that is needed for equation (4.7) to be guaranteed to hold:

$$\frac{1}{1-\gamma}\frac{\delta^n}{\delta^c} + \frac{\gamma}{1-\gamma} = (1+\alpha_{min})\frac{\delta^n}{\delta^c} \Leftrightarrow \tag{4.11}$$

$$\frac{1}{1-\gamma} + \frac{\gamma}{1-\gamma}\left(\frac{\delta^n}{\delta^c}\right)^{-1} = (1+\alpha_{min}) \Leftrightarrow \tag{4.12}$$

$$\alpha_{min} = \frac{\gamma}{1-\gamma}\left[\left(\frac{\delta^n}{\delta^c}\right)^{-1} + 1\right] \tag{4.13}$$

Figure 4.3 shows α_{min} as a function of $\frac{\delta^n}{\delta^c} \in [0,1)$ for several values of γ. Left of the dotted line the equation (4.10) is satisfied, i.e., improvement of the distance to the solution is guaranteed, whereas right of the dotted line this is not the case.

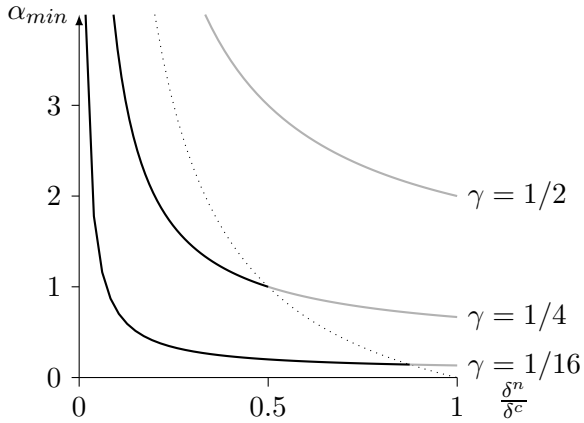

Figure 4.3: Minimum required value of α

For given γ, reducing $\frac{\delta^n}{\delta^c}$ increases α_{min}. Especially for small $\frac{\delta^n}{\delta^c}$, the value of α_{min} grows very rapidly. Thus, the closer the inexact iterate is brought to the exact iterate, the less the expected return in the distance to the solution is. For the inexact Newton method this translates into over-solving whenever the forcing term η_i is chosen too small.

Further, it is clear that if γ becomes smaller, then α_{min} is reduced also. If γ is small, $\frac{\delta^n}{\delta^c}$ can be made very small without compromising the return of investment on the distance to the solution. However, for γ nearing $\frac{1}{2}$, or more, any choice of $\frac{\delta^n}{\delta^c}$ no longer guarantees a similar great improvement, if any, in the distance to the solution. For such γ oversolving is therefore inevitable.

Recall that if the iterative method converges superlinearly, then γ rapidly goes to 0 also. Thus, for such a method, $\frac{\delta^n}{\delta^c}$ can be made smaller and smaller in later iterations, without oversolving. Or, in other words, for any choice of $\alpha > 0$ and $\frac{\delta^n}{\delta^c} \in [0, 1)$, there will be some point in the iteration process from which on forward equation (4.7) is satisfied.

For the inexact Newton method, equation (4.7) translates into

$$\|\boldsymbol{x}_{i+1} - \boldsymbol{x}^*\| \leq (1 + \alpha)\, \eta_i \|\boldsymbol{x}_i - \boldsymbol{x}^*\|. \tag{4.14}$$

In the next section this equation is formally proven to hold for the inexact Newton method, in a certain norm.

4.2 Convergence of Inexact Newton Methods

Consider the nonlinear system of equations $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$, where:

- there is a solution $\boldsymbol{x}^*$ such that $\boldsymbol{F}(\boldsymbol{x}^*) = \boldsymbol{0}$,
- the Jacobian matrix J of $\boldsymbol{F}$ exists in a neighbourhood of $\boldsymbol{x}^*$,
- $J(\boldsymbol{x}^*)$ is continuous and non-singular.

In this section, theory is presented that relates the convergence of the inexact Newton method for the above problem directly to the chosen forcing terms. The following theorem is a variation on the inexact Newton convergence theorem presented in [15, Thm. 2.3].

Theorem 4.2.1. *Let $\eta_i \in (0, 1)$ and choose $\alpha > 0$ such that $(1 + \alpha)\, \eta_i < 1$. Then there exists an $\varepsilon > 0$ such that, if $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \varepsilon$, the sequence of inexact Newton iterates $\boldsymbol{x}_i$ converges to $\boldsymbol{x}^*$, with*

$$\|J(\boldsymbol{x}^*)(\boldsymbol{x}_{i+1} - \boldsymbol{x}^*)\| < (1 + \alpha)\, \eta_i \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|. \tag{4.15}$$

Proof. Define

$$\mu = \max[\|J(\boldsymbol{x}^*)\|, \|J(\boldsymbol{x}^*)^{-1}\|] \geq 1. \tag{4.16}$$

Recall that $J(\boldsymbol{x}^*)$ is non-singular. Thus μ is well-defined and we can write

$$\frac{1}{\mu}\|\boldsymbol{y}\| \leq \|J(\boldsymbol{x}^*)\, \boldsymbol{y}\| \leq \mu \|\boldsymbol{y}\|. \tag{4.17}$$

Note that $\mu \geq 1$ because the induced matrix norm is submultiplicative.

Let

$$\gamma \in \left(0, \frac{\alpha\eta_i}{5\mu}\right) \tag{4.18}$$

and choose $\varepsilon > 0$ sufficiently small such that if $\|\boldsymbol{y} - \boldsymbol{x}^*\| \leq \mu^2\varepsilon$ then

$$\|J(\boldsymbol{y}) - J(\boldsymbol{x}^*)\| \leq \gamma, \tag{4.19}$$

$$\|J(\boldsymbol{y})^{-1} - J(\boldsymbol{x}^*)^{-1}\| \leq \gamma, \tag{4.20}$$

$$\|\boldsymbol{F}(\boldsymbol{y}) - \boldsymbol{F}(\boldsymbol{x}^*) - J(\boldsymbol{x}^*)(\boldsymbol{y} - \boldsymbol{x}^*)\| \leq \gamma\|\boldsymbol{y} - \boldsymbol{x}^*\|. \tag{4.21}$$

That such an ε exists follows from [36, Thm. 2.3.3 & 3.1.5].

$\star\star\star\star\star$

First we show that if $\|\boldsymbol{x}_i - \boldsymbol{x}^*\| < \mu^2\varepsilon$, then equation (4.15) holds. Write

$$J(\boldsymbol{x}^*)(\boldsymbol{x}_{i+1} - \boldsymbol{x}^*) = \left[I + J(\boldsymbol{x}^*)\left(J(\boldsymbol{x}_i)^{-1} - J(\boldsymbol{x}^*)^{-1}\right)\right] \cdot [\boldsymbol{r}_i + (J(\boldsymbol{x}_i) - J(\boldsymbol{x}^*))(\boldsymbol{x}_i - \boldsymbol{x}^*) - (\boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}^*) - J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*))]. \tag{4.22}$$

Taking norms gives

$$\begin{aligned}\|J(\boldsymbol{x}^*)(\boldsymbol{x}_{i+1} - \boldsymbol{x}^*)\| &\leq \left[1 + \|J(\boldsymbol{x}^*)\|\|J(\boldsymbol{x}_i)^{-1} - J(\boldsymbol{x}^*)^{-1}\|\right] \cdot [\|\boldsymbol{r}_i\| + \\ &\quad \|J(\boldsymbol{x}_i) - J(\boldsymbol{x}^*)\|\|\boldsymbol{x}_i - \boldsymbol{x}^*\| + \|\boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}^*) - J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|], \\ &\leq [1 + \mu\gamma] \cdot [\|\boldsymbol{r}_i\| + \gamma\|\boldsymbol{x}_i - \boldsymbol{x}^*\| + \gamma\|\boldsymbol{x}_i - \boldsymbol{x}^*\|], \\ &\leq [1 + \mu\gamma] \cdot [\eta_i\|\boldsymbol{F}(\boldsymbol{x}_i)\| + 2\gamma\|\boldsymbol{x}_i - \boldsymbol{x}^*\|]. \end{aligned} \tag{4.23}$$

Here the definitions of η_i (equation (3.8)) and μ (equation (4.16)) were used, together with equations (4.19)–(4.21).

Further write, using that by definition $\boldsymbol{F}(\boldsymbol{x}^*) = \boldsymbol{0}$,

$$\boldsymbol{F}(\boldsymbol{x}_i) = [J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)] + [\boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}^*) - J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)]. \tag{4.24}$$

Again taking norms gives

$$\begin{aligned}\|\boldsymbol{F}(\boldsymbol{x}_i)\| &\leq \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| + \|\boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}^*) - J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| \\ &\leq \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| + \gamma\|\boldsymbol{x}_i - \boldsymbol{x}^*\|. \end{aligned} \tag{4.25}$$

Substituting equation (4.25) into equation (4.23) then leads to

$$\begin{aligned}&\|J(\boldsymbol{x}^*)(\boldsymbol{x}_{i+1} - \boldsymbol{x}^*)\| \\ &\quad \leq (1 + \mu\gamma)\left[\eta_i\left(\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| + \gamma\|\boldsymbol{x}_i - \boldsymbol{x}^*\|\right) + 2\gamma\|\boldsymbol{x}_i - \boldsymbol{x}^*\|\right] \\ &\quad \leq (1 + \mu\gamma)\left[\eta_i(1 + \mu\gamma) + 2\mu\gamma\right]\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|. \end{aligned} \tag{4.26}$$

Here equation (4.17) was used to write $\|\boldsymbol{x}_i - \boldsymbol{x}^*\| \leq \mu\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|$.

Finally, using that $\gamma \in \left(0, \frac{\alpha\eta_i}{5\mu}\right)$, and that both $\eta_i < 1$ and $\alpha\eta_i < 1$—the latter being a result from the requirement that $(1+\alpha)\,\eta_i < 1$—gives

$$\begin{aligned}
(1+\mu\gamma)\left[\eta_i\,(1+\mu\gamma)+2\mu\gamma\right] &\leq \left(1+\frac{\alpha\eta_i}{5}\right)\left[\eta_i\left(1+\frac{\alpha\eta_i}{5}\right)+\frac{2\alpha\eta_i}{5}\right] \\
&= \left[\left(1+\frac{\alpha\eta_i}{5}\right)^2+\left(1+\frac{\alpha\eta_i}{5}\right)\frac{2\alpha}{5}\right]\eta_i \\
&= \left[1+\frac{2\alpha\eta_i}{5}+\frac{\alpha^2\eta_i^2}{25}+\frac{2\alpha}{5}+\frac{2\alpha^2\eta_i}{25}\right]\eta_i \\
&< \left[1+\frac{2\alpha}{5}+\frac{\alpha}{25}+\frac{2\alpha}{5}+\frac{2\alpha}{25}\right]\eta_i \\
&< (1+\alpha)\,\eta_i. \qquad (4.27)
\end{aligned}$$

Equation (4.15) follows by substituting equation (4.27) into equation (4.26).

$\star\star\star\star\star$

Given that equation (4.15) holds if $\|\boldsymbol{x}_i - \boldsymbol{x}^*\| < \mu^2\varepsilon$, we now proceed to prove Theorem 4.2.1 by induction.

For the base case

$$\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \varepsilon \leq \mu^2\varepsilon. \qquad (4.28)$$

Thus equation (4.15) holds for $i = 0$.

The induction hypothesis that equation (4.15) holds for $i = 0, \ldots, k-1$ then leads to

$$\begin{aligned}
\|\boldsymbol{x}_k - \boldsymbol{x}^*\| &\leq \mu\|J\,(\boldsymbol{x}^*)\,(\boldsymbol{x}_k - \boldsymbol{x}^*)\,\| \\
&< \mu\,(1+\alpha)^k\,\eta_{k-1}\cdots\eta_0\|J\,(\boldsymbol{x}^*)\,(\boldsymbol{x}_0 - \boldsymbol{x}^*)\,\| \\
&< \mu\|J\,(\boldsymbol{x}^*)\,(\boldsymbol{x}_0 - \boldsymbol{x}^*)\,\| \\
&\leq \mu^2\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| \\
&< \mu^2\varepsilon. \qquad (4.29)
\end{aligned}$$

Thus equation (4.15) also holds for $i = k$, completing the proof. □

In words, Theorem 4.2.1 states that for an arbitrarily small $\alpha > 0$, and any choice of forcing terms $\eta_i \in (0, 1)$, equation (4.15) will hold if the current iterate is close enough to the solution.

Note that this does not mean that for a certain iterate $\boldsymbol{x}_i$, one can choose α and η_i arbitrarily small and expect equation (4.15) to hold, as ε depends on the choice of α and η_i. On the contrary, a given iterate $\boldsymbol{x}_i$—close enough to the solution to guarantee convergence—imposes the restriction that, for Theorem 4.2.1 to hold, the forcing terms η_i cannot be chosen too small.

Recall that it was already shown in Section 4.1 that choosing η_i too small leads to oversolving.

If we define oversolving as using forcing terms η_i that are too small for a certain iterate $\boldsymbol{x}_i$, in the context of Theorem 4.2.1, then the theorem can be characterised by saying that a convergence factor $(1+\alpha)\,\eta_i$ is attained if η_i is chosen such that there is no oversolving. Using equation (4.18), $\eta_i > \frac{5\mu\gamma}{\alpha}$ can then be seen as a theoretical bound on the forcing terms that guards against oversolving.

Corollary 4.2.1. *Let $\eta_i \in (0,1)$ and choose $\alpha > 0$ such that $(1+\alpha)\,\eta_i < 1$. Then there exists an $\varepsilon > 0$ such that, if $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \varepsilon$, the sequence of inexact Newton iterates $\boldsymbol{x}_i$ converges to $\boldsymbol{x}^*$, with*

$$\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| < (1+\alpha)^i\,\eta_{i-1}\cdots\eta_0\|J(\boldsymbol{x}^*)(\boldsymbol{x}_0 - \boldsymbol{x}^*)\|. \tag{4.30}$$

Proof. The stated follows from the repeated application of Theorem 4.2.1. □

A relation between the nonlinear residual norm $\|\boldsymbol{F}(\boldsymbol{x}_i)\|$ and the error norm $\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|$ can be derived, within the neighbourhood of the solution where Theorem 4.2.1 holds.

Theorem 4.2.2. *Let $\eta_i \in (0,1)$ and choose $\alpha > 0$ such that $(1+\alpha)\,\eta_i < 1$. Then there exists an $\varepsilon > 0$ such that, if $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \varepsilon$, then*

$$\left(1 - \frac{\alpha\eta_i}{5}\right)\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| < \|\boldsymbol{F}(\boldsymbol{x}_i)\| < \left(1 + \frac{\alpha\eta_i}{5}\right)\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|. \tag{4.31}$$

Proof. Using that $\boldsymbol{F}(\boldsymbol{x}^*) = \boldsymbol{0}$ by definition, again write

$$\boldsymbol{F}(\boldsymbol{x}_i) = [J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)] + [\boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}^*) - J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)]. \tag{4.32}$$

Taking norms, and using equations (4.21) and (4.17), gives

$$\begin{aligned}\|\boldsymbol{F}(\boldsymbol{x}_i)\| &\le \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| + \|\boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}^*) - J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| \\ &\le \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| + \gamma\|\boldsymbol{x}_i - \boldsymbol{x}^*\| \\ &\le \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| + \mu\gamma\|J(\boldsymbol{x}^*)\,\boldsymbol{x}_i - \boldsymbol{x}^*\| \\ &= (1+\mu\gamma)\,\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|. \end{aligned} \tag{4.33}$$

Similarly, it holds that

$$\begin{aligned}\|\boldsymbol{F}(\boldsymbol{x}_i)\| &\ge \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| - \|\boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}^*) - J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| \\ &\ge \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| - \gamma\|\boldsymbol{x}_i - \boldsymbol{x}^*\| \\ &\ge \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| - \mu\gamma\|J(\boldsymbol{x}^*)\,\boldsymbol{x}_i - \boldsymbol{x}^*\| \\ &= (1-\mu\gamma)\,\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|. \end{aligned} \tag{4.34}$$

The theorem now follows from (4.18). □

For the nonlinear residual norm $\|\boldsymbol{F}(\boldsymbol{x}_i)\|$, a similar result can now be derived as presented in Theorem 4.2.1 for the error norm $\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|$.

Theorem 4.2.3. *Let $\eta_i \in (0,1)$ and choose $\alpha > 0$ such that $(1+2\alpha)\eta_i < 1$. Then there exists an $\varepsilon > 0$ such that, if $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \varepsilon$, the sequence $\|\boldsymbol{F}(\boldsymbol{x}_i)\|$ converges to 0, with*

$$\|\boldsymbol{F}(\boldsymbol{x}_{i+1})\| < (1+2\alpha)\,\eta_i \|\boldsymbol{F}(\boldsymbol{x}_i)\|. \tag{4.35}$$

Proof. Note that the conditions imposed in Theorem 4.2.3, are such that Theorems 4.2.1 and 4.2.2 hold. Define μ and γ again as in Theorem 4.2.1.

Using equation (4.33), Theorem 4.2.1, and equation (4.34), write

$$\begin{aligned}\|\boldsymbol{F}(\boldsymbol{x}_{i+1})\| &\leq (1+\mu\gamma)\,\|J(\boldsymbol{x}^*)(\boldsymbol{x}_{i+1} - \boldsymbol{x}^*)\| \\ &< (1+\mu\gamma)(1+\alpha)\,\eta_i \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| \\ &\leq \frac{(1+\mu\gamma)}{(1-\mu\gamma)}(1+\alpha)\,\eta_i \|\boldsymbol{F}(\boldsymbol{x}_i)\|. \end{aligned} \tag{4.36}$$

Further, using (4.18), write

$$\frac{1+\mu\gamma}{1-\mu\gamma} < \frac{1+\frac{\alpha\eta_i}{5}}{1-\frac{\alpha\eta_i}{5}} = \frac{1-\frac{\alpha\eta_i}{5}+\frac{2}{5}\alpha\eta_i}{1-\frac{\alpha\eta_i}{5}} = 1 + \frac{\frac{2}{5}\alpha\eta_i}{1-\frac{\alpha\eta_i}{5}} < 1 + \frac{\frac{2}{5}\alpha\eta_i}{\frac{4}{5}} = 1 + \frac{\alpha\eta_i}{2}.$$

Finally, using that both $\eta_i < 1$ and $2\alpha\eta_i < 1$—the latter being a result from the requirement that $(1+2\alpha)\,\eta_i < 1$—gives

$$\frac{1+\mu\gamma}{1-\mu\gamma}(1+\alpha) < \left(1+\frac{\alpha\eta_i}{2}\right)(1+\alpha) = 1 + \left(1+\frac{\eta_i}{2}\right)\alpha + \frac{1}{2}\eta_i\alpha^2 < 1 + 2\alpha.$$

Substitution into equation (4.36) completes the proof. □

Theorem 4.2.3 shows that the nonlinear residual norm $\|\boldsymbol{F}(\boldsymbol{x}_i)\|$ converges at similar rate as error norm $\|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\|$. This is important, because Newton methods use $\|\boldsymbol{F}(\boldsymbol{x}_i)\|$ to measure convergence of the iterate to the solution.

4.2.1 Linear Convergence

The inexact Newton method uses some iterative process in each Newton iteration, to solve the linear Jacobian system $J(\boldsymbol{x}_i)\,\boldsymbol{s}_i = -\boldsymbol{F}(\boldsymbol{x}_i)$ up to accuracy $\|J(\boldsymbol{x}_i)\,\boldsymbol{s}_i + \boldsymbol{F}(\boldsymbol{x}_i)\| \leq \eta_i \|\boldsymbol{F}(\boldsymbol{x}_i)\|$. In many practical applications, the convergence of the iterative linear solver turns out to be approximately linear. That is, for some convergence rate $\beta > 0$

$$\|\boldsymbol{r}_i^k\| \approx 10^{-\beta k}\|\boldsymbol{F}(\boldsymbol{x}_i)\|, \tag{4.37}$$

where $\boldsymbol{r}_i^k = \boldsymbol{F}(\boldsymbol{x}_i) + J(\boldsymbol{x}_i)\,\boldsymbol{s}_i^k$ is the linear residual after k iterations of the linear solver in Newton iteration i.

Suppose that the linear solver indeed converges linearly, with the same rate of convergence β in each Newton iteration. Let K_i be the number of linear iterations performed in Newton iteration i, i.e., K_i is minimum integer such that $10^{-\beta K_i} \leq \eta_i$. Further, let $N_i = \sum_{j=0}^{i-1} K_j$ be the total number of linear iterations performed up to the start of Newton iteration i. Then, using Corollary 4.2.1,

$$\begin{aligned} \|J(\boldsymbol{x}^*)(\boldsymbol{x}_i - \boldsymbol{x}^*)\| &< (1+\alpha)^i \eta_{i-1} \cdots \eta_0 \|J(\boldsymbol{x}^*)(\boldsymbol{x}_0 - \boldsymbol{x}^*)\| \\ &= (1+\alpha)^i 10^{-\beta N_i} \|J(\boldsymbol{x}^*)(\boldsymbol{x}_0 - \boldsymbol{x}^*)\|, \end{aligned} \tag{4.38}$$

Thus, if the linear solver converges approximately linearly, with similar rate of convergence in each Newton iteration, the forcing terms are such that there is no oversolving, and if α can be chosen small enough, i.e., the initial iterate is close enough to the solution, then the inexact Newton method will converge approximately linearly in the total number of linear iterations.

Note that this result is independent of the rate of convergence in the Newton iterations. If the forcing terms are chosen constant, the method will converge linearly in the number of Newton iterations, and linearly in the total number of linear iterations performed throughout those Newton iterations. If the forcing terms η_i are chosen properly, the method will converge quadratically in the Newton iterations, while converging linearly in the linear iterations. The amount of Newton iterations needed in these two scenarios may differ greatly, but the total amount of linear iterations should be approximately equal.

4.3 Numerical Experiments

Both the classical Newton-Raphson convergence theory [36, 18], and the inexact Newton convergence theory by Dembo et al. [15], require the current iterate to be close enough to the solution. What exactly is close enough is problem dependent, and generally too hard to calculate in practice. However, decades of practice have shown that the theoretical convergence is reached within a few Newton steps for most problems. Thus the theory is not just of theoretical, but also of practical importance.

In this section, practical experiments are presented to illustrate that Theorem 4.2.1 also has practical merit, despite the elusive requirement that current iterate has to be close enough to the solution. Moreover, instead of convergence relation (4.15), an idealised version is tested, in which the error norm is changed to the 2-norm, and α is neglected:

$$\|\boldsymbol{x}_{i+1} - \boldsymbol{x}^*\| < \eta_i \|\boldsymbol{x}_i - \boldsymbol{x}^*\|. \tag{4.39}$$

If relation (4.39) is satisfied, that means that any improvement of the linear residual norm in a certain Newton iteration, improves the error in the nonlinear iterate by an equal factor.

The experiments in this section are performed on a power flow problem. The power flow problem, and how to solve it, is treated in Chapters 5–7. The actual test case used is the uctew032 power flow problem (see Appendix B). The resulting nonlinear system has approximately 256k equations, and the Jacobian matrix has around 2M nonzeros. The linear Jacobian systems are solved using GMRES, preconditioned with a high quality ILU factorisation of the Jacobian.

In Figures 4.4–4.6, the uctew032 problem is solved with different amounts of GMRES iterations per Newton iteration. In all cases, two Newton steps with just a single GMRES iteration were performed at the start, but not shown in the figures. In each figure, the solid line represents the norm of the actual error $\|\boldsymbol{x}_i - \boldsymbol{x}^*\|$, while the dashed line depicts the expected error norm following the idealised theoretical relation (4.39).

Figure 4.4 shows the distribution of GMRES iterations for a typical choice of forcing terms that leads to a fast solution of the problem. The practical convergence nicely follows the idealised theory. This suggests that the two initial Newton iterations with a single GMRES iteration each, lead to an iterate $\boldsymbol{x}_2$ close enough to the solution for practice to follow theory, for the chosen forcing terms η_i. Note that $\boldsymbol{x}_2$ is in actuality still very far from the solution, and that it is unlikely that it satisfies the theoretical bound on the proximity to the solution required in Theorem 4.2.1.

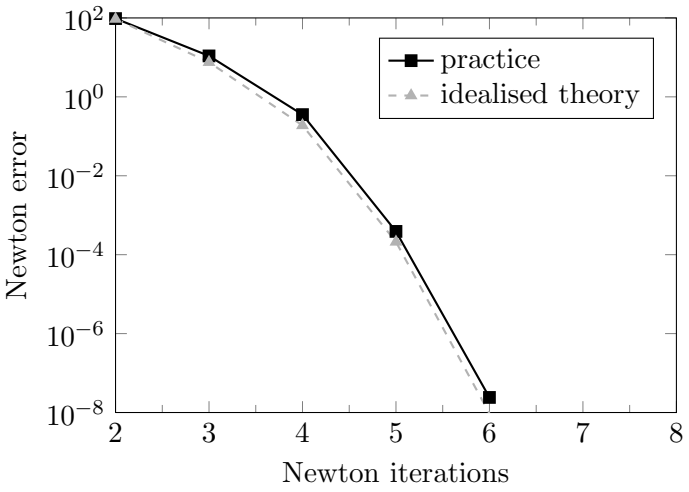

Figure 4.4: GMRES iteration distribution 1,1,4,6,10,14

Figure 4.5 has a more exotic distribution of GMRES iterations performed per Newton iteration, illustrating that practice can also follow theory nicely for such a scenario.

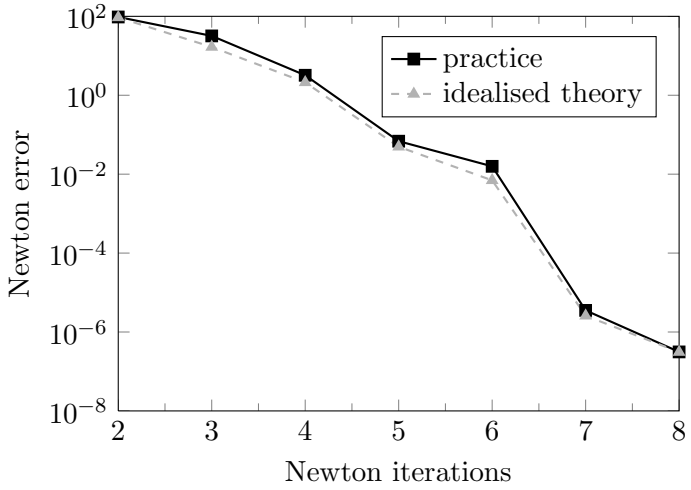

Figure 4.5: GMRES iteration distribution 1,1,3,4,6,3,11,3

Figure 4.6 illustrates the impact of oversolving. Practical convergence is nowhere near the idealised theory, because extra GMRES iterations are performed that do not further improve the nonlinear error. In terms of Theorem 4.2.1 this means that the iterates $\boldsymbol{x}_i$ are not close enough to the solution, to be able to take the forcing terms η_i as small as they were chosen in this example.

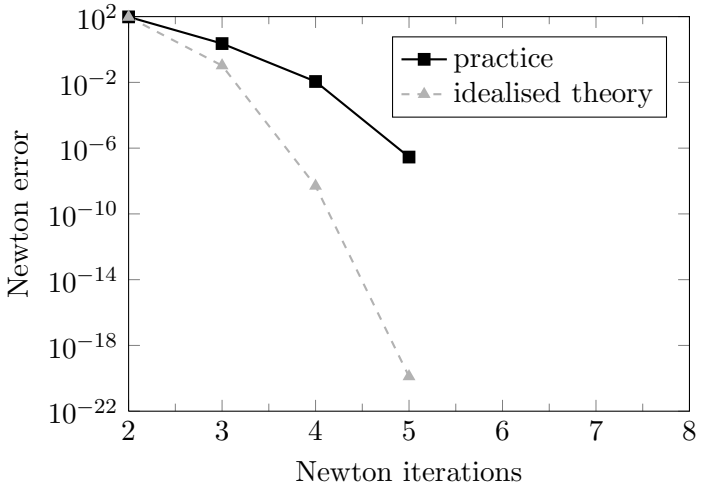

Figure 4.6: GMRES iteration distribution 1,1,9,19,30

In Figure 4.7, the convergence in the number of Newton iterations is compared with the convergence in the number of GMRES iterations. For the uctew032 test case, the convergence of the GMRES solves is approximately linear, with similar rate of convergence in each Newton iteration. Thus the same figure can be used to also illustrate the theory from Section 4.2.1.

The top figure shows the true error norm in the number of Newton iterations, for five different distributions of GMRES iterations per Newton iteration, i.e., for five different sets of forcing terms. The graphs are as expected; the more GMRES iteration are performed per Newton iteration, the better the convergence. A naive interpretation might conclude that option (A) is the best of the considered choices, and that option (E) is by far the worst. However, this is too simple a conclusion, as illustrated below by the bottom figure.

The bottom figure shows the convergence of the true error in the total number of GMRES iterations for the same five distributions. In this figure, the convergence of option (A) is worse than that of option (E), revealing that option (A) imposes a lot of oversolving. Option (E) is still the worst of the options that do not oversolve much, but it no longer seems as bad as suggested by the top figure. Options (B), (C), and (D) show approximately linear convergence, as predicted by the theory of Section 4.2.1. As the practical GMRES convergence is not exactly linear, nor exactly the same in each Newton iteration, the convergence of these options is not identical, and option (E) is still quite a bit worse. The strong influence of the near linear GMRES convergence is nonetheless very clear.

It is clear that neither the top figure, nor the bottom figure in Figure 4.7 tells the entire story on its own. If the set-up time of a Newton iteration—generally mostly determined by the calculation of J and $\boldsymbol{F}$—is very high compared to the computational cost of iterations of the linear solver, then the top figure approximates the convergence in the solution time. However, if these set-up costs are negligible compared to the linear solves, then it is the bottom figure that better approximates the convergence in the solution time. The practical truth is generally in between, but knowing which of these extremes a certain problem is closer to can be important to make the correct choice of forcing terms.

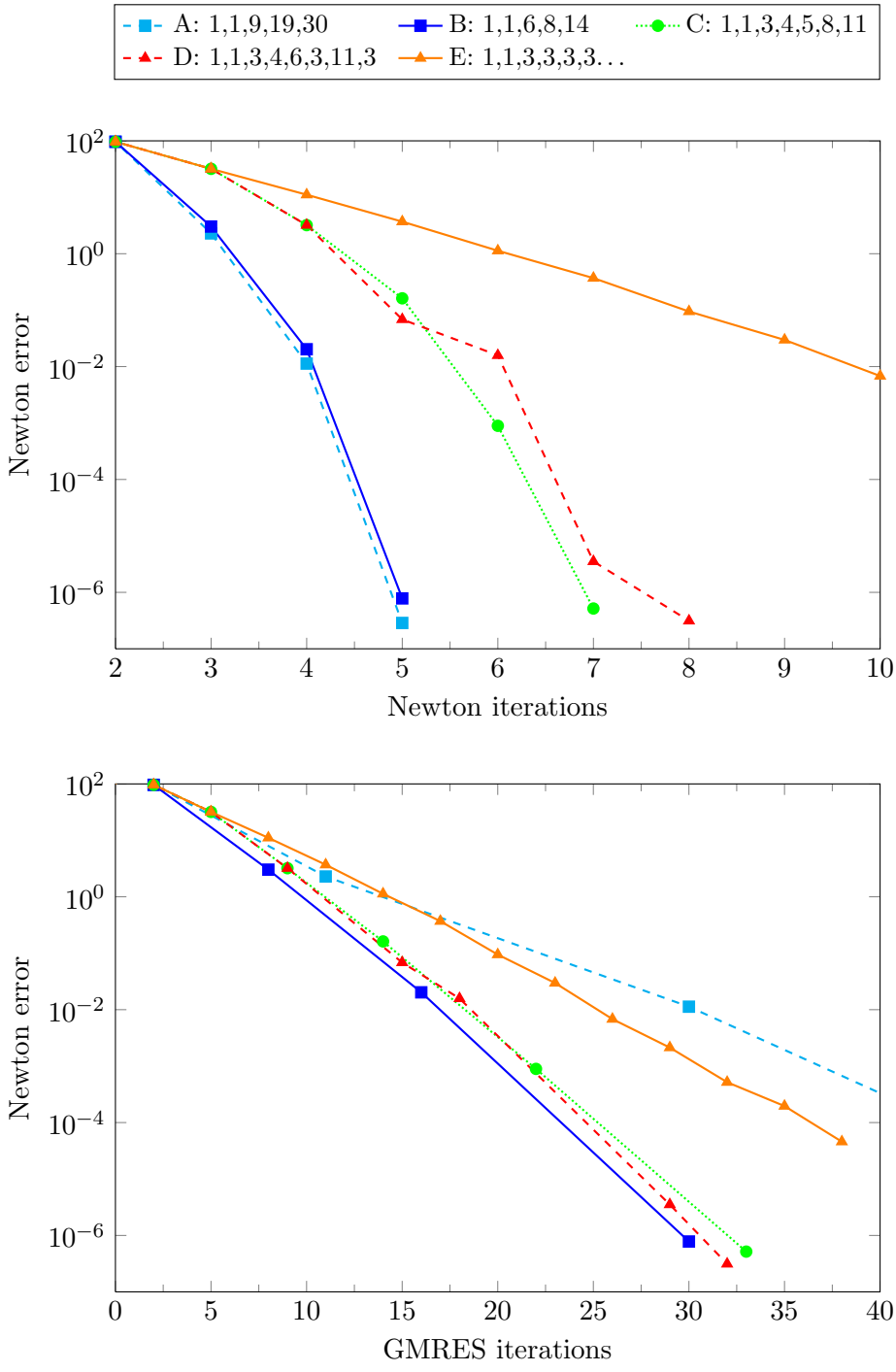

Figure 4.7: Convergence in Newton and GMRES iterations

4.4 Applications

In this section, ideas are presented to use the knowledge from the previous sections to design better inexact Newton algorithms. First, optimising the choice of the forcing terms is explored, and after that, possible adaptations of the linear solver within the Newton process are treated.

4.4.1 Forcing Terms

The ideas for the choice of the forcing terms η_i rely on the expectation that in Newton iteration i—provided that there is no oversolving—both the unknown true error, and its known measure $\|\boldsymbol{F}(\boldsymbol{x}_i)\|$, should reduce with an approximate factor η_i, as indicated by Theorem 4.2.1.

Theoretically, this knowledge can be used to choose the forcing terms adaptively by calculating $\|\boldsymbol{F}\left(\boldsymbol{x}_i + \boldsymbol{s}_i^k\right)\|$ in every linear iteration k, and checking whether the reduction in the norm of $\boldsymbol{F}$ is close enough to the reduction in the linear residual. Once the reduction in the norm of $\boldsymbol{F}$ starts lagging that of the linear residual, the linear solver is oversolving, and the next Newton iteration should be started. Obviously, this adaptive method only makes sense if $\|\boldsymbol{F}\left(\boldsymbol{x}_i + \boldsymbol{s}_i^k\right)\|$ can be evaluated cheaply, compared to the cost of doing extra linear iterations, which is often not the case.

Theorem 4.2.1 can also be used to set a lower bound for the forcing terms. Assume that the aim is to solve up to the nonlinear tolerance $\|\boldsymbol{F}\| \leq \tau$. A forcing term $\eta_i = \frac{\tau}{\|\boldsymbol{F}(\boldsymbol{x}_i)\|}$ should be sufficient to approximately reach the desired nonlinear tolerance, provided that there is no oversolving. Choosing η_i significantly smaller than that, always leads to a waste of computational effort. Therefore, it makes sense to enforce

$$\eta_i \geq \sigma \frac{\tau}{\|\boldsymbol{F}(\boldsymbol{x}_i)\|}, \tag{4.40}$$

in every Newton iteration, for some sensible choice of $\sigma \in (0, 1)$.

Knowledge of the computational cost to set up a new Newton iteration, and of the convergence behaviour of the used iterative linear solver, can further help to choose better forcing terms. If the set-up cost of a Newton iteration is very high, it then makes sense to choose smaller forcing terms to get the most out of each Newton iteration. Similarly, if the linear solver converges superlinearly slightly smaller forcing terms may be preferred, to maximise the benefit of this superlinear convergence. On the other hand if the set-up cost of a Newton iteration is low, then it may yield better results to keep the forcing terms a bit larger to prevent oversolving, especially if the linear solver does not converge superlinearly.

4.4.2 Linear Solver

Given a forcing term η_i, which linear solver is used may be adapted to the value of this forcing term. For example, if it is expected that only a few linear iterations are needed, then GMRES is often the best choice. On the other hand, if many linear iterations are anticipated it might be better to use Bi-CGSTAB or IDR(s). If the nonlinear problem is not too large, it may even be best to switch to a direct solver in later iterations, if η_i becomes very small. See Chapter 2 for information about linear solvers.

Instead of changing the entire linear solver between Newton iterations, it is also an option to change just the preconditioning. For example, higher quality preconditioners could be used in later iterations. Alternatively, a preconditioner can be kept through multiple Newton iterations, updating it depending on η_i.

CHAPTER 5

Power System Analysis

A power system is a system that provides for the generation, transmission, and distribution of electrical energy. Power systems are considered to be the largest and most complex man-made systems. As electrical energy is vital to our society, power systems have to satisfy the highest security and reliability standards. At the same time, minimising cost and environmental impact are important issues.

Thermal power plants generate electrical power using heat, mostly from the combustion of fossil fuels, or from a nuclear reaction in the case of nuclear power plants. Most thermal power stations heat water to produce steam, which is then used to power turbines. Kinetic energy from these rotating devices is converted into electrical power by means of electromagnetic induction. Hydroelectric power plants run water through water turbines (typically located in dams), wind farms use wind turbines, and photovoltaic plants use solar panels to generate electrical power. Hydroelectric, wind, and solar power are examples of renewable energy, as they are generated from naturally replenished resources.

The transmission network connects the generating plants to substations near the consumers. It also performs the function of connecting different power pools, to reduce cost and increase reliability. High voltage alternating current (AC) is used to reduce voltage drops and power losses, and to increase capacity of the transmission lines. The three-phase system is used to reduce conductor material.

Finally, the distribution network connects the transmission network to the consumers. The distribution network operates at lower voltages than the transmission network, supplying three-phase AC to industrial consumers, and single-phase AC for common household consumption. Figure 5.1 shows a schematic representation of a power system.

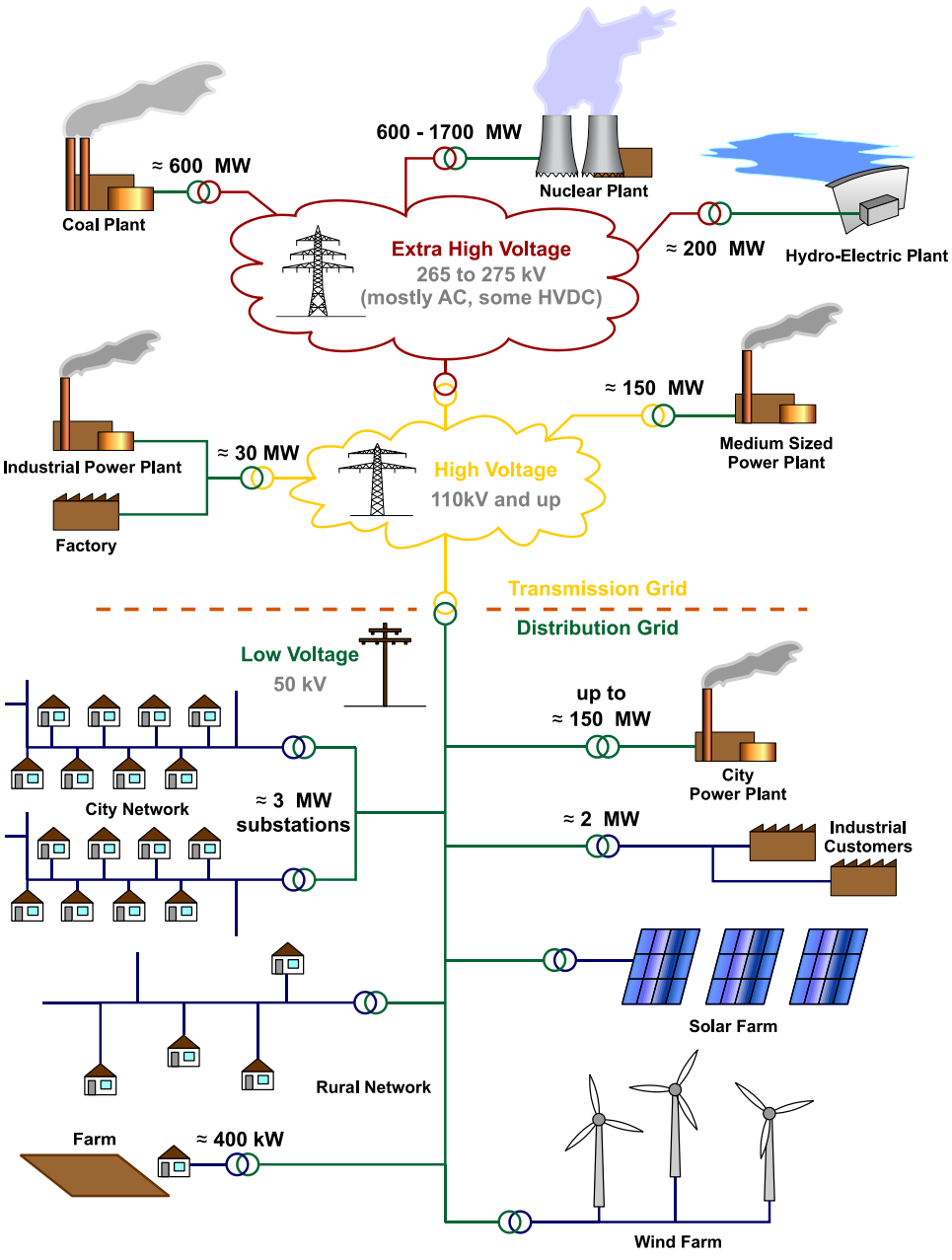

Figure 5.1: Schematic representation of a power system

Power systems have to operate very close to a fixed frequency, mostly 50Hz in Europe. Whenever an electrical appliance is turned on, the load on the power system increases. In the case of a thermal power plant, the extra power is taken from the kinetic energy of a rotating device, slowing down the rotation. Extra steam has to be fed to the turbines to keep the rotation at the desired frequency for the power system. Automated controls make it possible for the power system to operate at near fixed frequency, making steady state power system models—where the frequency is regarded constant—a useful approximation of reality.

Steady state power system analysis, by means of simulations on mathematical models, plays an important role in both operational control and planning. This chapter first treats the required mathematical models of electrical power, and power system components. Using these models, power flow (or load flow) study and contingency analysis are treated. Power flow study calculates the bus voltages in the power system, given the generation and consumption. Contingency analysis simulates equipment outages, to determine if the system can still function reliably if such a contingency were to occur.

5.1 Electrical Power

To model a power system, firstly models of the underlying quantities are needed, as well as mathematical relations between these quantities. This sections treats voltage, current, and power quantities in steady state power system analysis, as well as quantities related to electrical resistance. Using these quantities, Ohm's law, and Kirchhoff's laws for AC circuits are treated.

5.1.1 Voltage and Current

In a power system in steady state, the voltage and current can be assumed to be sinusoidal functions of time with constant frequency ω. It is conventional to use the cosine function to describe these quantities, i.e.,

$$v(t) = V_{\max} \cos(\omega t + \delta_V) = \mathrm{Re}\left(V_{\max} e^{\iota \delta_V} e^{\iota \omega t}\right), \tag{5.1}$$

$$i(t) = I_{\max} \cos(\omega t + \delta_I) = \mathrm{Re}\left(I_{\max} e^{\iota \delta_I} e^{\iota \omega t}\right), \tag{5.2}$$

where ι is the imaginary unit[1], and Re is the operator that takes the real part. See section A.1 for a short introduction to complex numbers.

[1]The imaginary unit is most commonly denoted by i in mathematics, and by j in electrical engineering because i is reserved for the current. In this work, the imaginary unit is sometimes part of a matrix or vector equation, where i and j are used as indices. To avoid ambiguity, the imaginary unit is therefore denoted by ι (iota).

Since the frequency ω is assumed constant in steady state analysis, the term $e^{\iota\omega t}$ is not needed to describe the voltage or current in a particular steady state system. The remaining complex quantities $V = V_{\max}e^{\iota\delta_V}$ and $I = I_{\max}e^{\iota\delta_I}$ are independent of the time t, and are called the phasor representation of the voltage and current respectively. These quantities are used to represent the voltage and current in circuit theory. In power system theory, instead the effective phasor representation is used:

$$V = |V|\, e^{\iota\delta_V}, \text{ with } |V| = \frac{V_{\max}}{\sqrt{2}}, \tag{5.3}$$

$$I = |I|\, e^{\iota\delta_I}, \quad \text{with } |I| = \frac{I_{\max}}{\sqrt{2}}. \tag{5.4}$$

Note that $|V|$ and $|I|$ are the RMS values of $v(t)$ and $i(t)$, and that the effective phasors differ from the circuit theory phasors by a factor $\sqrt{2}$.

This thesis is about power system calculations, and thus V and I will be used to denote the effective voltage and current phasors, as defined above.

5.1.2 Complex Power

Using the voltage and current equations (5.1) and (5.2), choose the reference time such that the voltage can be written as $v(t) = V_{\max}\cos(\omega t)$, and the current as $i(t) = I_{\max}\cos(\omega t - \phi)$. The quantity $\phi = \delta_V - \delta_I$ is called the power factor angle, and $\cos\phi$ the power factor.

The instantaneous power $p(t)$ then is given by

$$\begin{aligned}
p(t) &= v(t)\, i(t) \\
&= \sqrt{2}\,|V|\cos(\omega t)\,\sqrt{2}\,|I|\cos(\omega t - \phi) \\
&= 2\,|V|\,|I|\cos(\omega t)\cos(\omega t - \phi) \\
&= 2\,|V|\,|I|\cos(\omega t)\left[\cos\phi\cos(\omega t) + \sin\phi\sin(\omega t)\right] \\
&= |V|\,|I|\left[2\cos\phi\cos^2(\omega t) + 2\sin\phi\sin(\omega t)\cos(\omega t)\right] \\
&= |V|\,|I|\cos\phi\left[2\cos^2(\omega t)\right] + |V|\,|I|\sin\phi\left[2\sin(\omega t)\cos(\omega t)\right] \\
&= |V|\,|I|\cos\phi\left[1 + \cos(2\omega t)\right] + |V|\,|I|\sin\phi\left[\sin(2\omega t)\right] \\
&= P\left[1 + \cos(2\omega t)\right] + Q\left[\sin(2\omega t)\right],
\end{aligned} \tag{5.5}$$

where $P = |V|\,|I|\cos\phi$, and $Q = |V|\,|I|\sin\phi$.

Thus the instantaneous power is the sum of a unidirectional component that is sinusoidal with average value P and amplitude P, and a component of alternating direction that is sinusoidal with average 0 and amplitude Q. Note that integrating the instantaneous power over a time period $T = \frac{2\pi}{\omega}$ gives

$$\frac{1}{T}\int_0^T p(t)\, dt = P. \tag{5.6}$$

The magnitude P is called the active power, or real power, or average power, and is measured in W (watts). The magnitude Q is called the reactive power, or imaginary power, and is measured in var (volt-ampere reactive).

Using the complex representation of voltage and current, we can write

$$P = |V|\,|I| \cos\phi = \operatorname{Re}\left(|V|\,|I|\, e^{\iota(\delta_V - \delta_I)}\right) = \operatorname{Re}\left(V\overline{I}\right), \tag{5.7}$$

$$Q = |V|\,|I| \sin\phi = \operatorname{Im}\left(|V|\,|I|\, e^{\iota(\delta_V - \delta_I)}\right) = \operatorname{Im}\left(V\overline{I}\right), \tag{5.8}$$

where $\overline{I}$ is the complex conjugate of I. Thus we can define the complex power in AC circuits as

$$S = P + \iota Q = V\overline{I}, \tag{5.9}$$

where S is measured in VA (volt-ampere).

Note that strictly speaking VA and var are the same unit as W, however it is useful to use the different unit names to distinguish between the measured quantities.

5.1.3 Impedance and Admittance

Impedance is the extension of the resistance notion to AC circuits. It is a measure of opposition to a sinusoidal current. The impedance is denoted by

$$Z = R + \iota X, \tag{5.10}$$

and measured in ohms (Ω). The real part $R \geq 0$ is called the resistance, and the imaginary part X the reactance. If $X > 0$ the reactance is called inductive and we can write $\iota X = \iota\omega L$, where $L > 0$ is the inductance. If $X < 0$ the reactance is called capacitive and we write $\iota X = \frac{1}{\iota\omega C}$, where $C > 0$ is the capacitance.

The admittance

$$Y = G + \iota B \tag{5.11}$$

is the inverse of the impedance and is measured in siemens (S), i.e.,

$$Y = \frac{1}{Z} = \frac{R}{|Z|^2} - \iota\frac{X}{|Z|^2}. \tag{5.12}$$

The real part $G = \frac{R}{|Z|^2} \geq 0$ is called the conductance, while the imaginary part $B = -\frac{X}{|Z|^2}$ is called the susceptance.

The voltage drop over an impedance Z is equal to $V = ZI$. This is the extension of Ohm's law to AC circuits. Alternatively, using the admittance, we can write

$$I = YV. \tag{5.13}$$

Using Ohm's law, we find that the power consumed by an impedance Z is

$$S = V\overline{I} = Z I\overline{I} = |I|^2 Z = |I|^2 R + \iota\,|I|^2 X. \tag{5.14}$$

5.1.4 Kirchhoff's circuit laws

Kirchhoff's circuit laws are used to calculate the voltage and current in electrical circuits.

Kirchhoff's current law (KCL)

At any point in the circuit, the sum of currents flowing towards that point is equal to the sum of currents flowing away from that point, i.e., $\sum_k I_k = 0$.

Kirchhoff's voltage law (KVL)

The directed sum of the electrical potential differences around any closed circuit is zero, i.e., $\sum_k V_k = 0$.

5.2 Power System Model

Power systems are modelled as a network of buses (nodes) and branches (edges). At each bus i, four electrical quantities are of importance:

$$\begin{array}{rcl} |V_i| & : & \text{voltage magnitude,} \\ \delta_i & : & \text{voltage phase angle,} \\ P_i & : & \text{injected active power,} \\ Q_i & : & \text{injected reactive power.} \end{array}$$

Each bus can hold a number of electrical devices. The bus is named according to the electrical magnitudes specified at that bus, see Table 5.1.

bus type	known	unknown
load bus or PQ-bus	P_i, Q_i	$\|V_i\|$, δ_i
generator bus or PV-bus	P_i, $\|V_i\|$	Q_i, δ_i
slack bus or swing bus	δ_i, $\|V_i\|$	P_i, Q_i

Table 5.1: Bus types with electrical magnitudes

Local distribution networks are usually connected to the transmission network at a single bus. In steady state power system models, such networks generally get aggregated into that connecting bus, which then gets assigned the total load of the distribution network.

Further, balanced three-phase systems are represented by one-line diagrams of equivalent single-phase systems, and voltage and current quantities are represented in per unit. For more details see for example [7, 42].

5.2.1 Generators, Loads, and Transmission Lines

A physical generator usually has P and $|V|$ controls and thus specifies these magnitudes. Likewise, a load will have a negative injected active power P specified, as well as a reactive power Q. However, the name of the bus does not necessarily indicate what type of devices it consists of. A wind turbine, for example, is a generator but does not have PV controls. Instead, it is modelled as a load bus with positive injected active power P. When a PV generator and a PQ load are connected to the same bus, the result is a PV-bus with a voltage amplitude equal to that of the generator, and an active power equal to the sum of the active power of the generator and the load. Also, there may be buses without a generator or load connected, such as transmission substations, which are modelled as load with $P = Q = 0$.

In any practical power system there are system losses. These losses have to be taken into account, but since they depend on the power flow they are not known in advance. A generator bus has to be assigned that will compensate for the difference between the total generation specified, and the total specified load plus the losses. This bus is called the slack bus, or swing bus. Obviously it is not possible to specify the real power P for this bus. Instead the voltage magnitude $|V|$ and angle δ are specified. Note that δ is merely the reference phase to which the other phase angles are measured. As such, for the slack bus it is usually set that $\delta = 0$.

Branches are the network representation of the transmission lines, that connect the buses in the power system. From a modelling viewpoint, lines define how to relate buses through Kirchhoff's circuit laws. Lines generally incur losses on the transported power and must be modelled as such.

A transmission line from bus i to bus j has some impedance. This impedance is modelled as a single total impedance quantity z_{ij} on the branch. The admittance of that line is thus $y_{ij} = \frac{1}{z_{ij}}$. Further, there is a shunt admittance from the line to the neutral ground. This shunt admittance is modelled as a total shunt admittance quantity y_s that is split evenly between bus i and bus j. Figure 5.2 shows a schematic representation of the transmission line model.

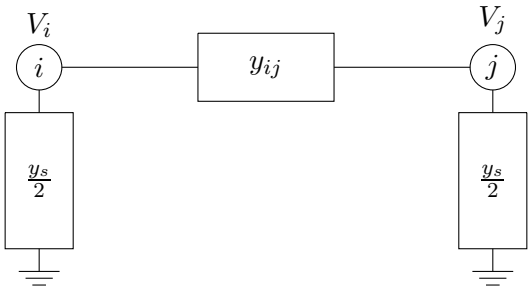

Figure 5.2: Transmission line model

It is usually assumed that there is no conductance from the line to the ground. This means that the shunt admittance is due only to the electrical field between line and ground, and is thus a capacitive susceptance, i.e., $y_s = \iota b_s$, with $b_s \geq 0$. For this reason, the shunt admittance y_s is also sometimes referred to as the shunt susceptance b_s. See also the notes about modelling shunts in Section 5.2.2.

5.2.2 Shunts and Transformers

Two other devices that are commonly found in power systems are shunts and transformers. Shunt capacitors can be used to inject reactive power, resulting in a higher node voltage, whereas shunt inductors consume reactive power, thus lowering the node voltage. Transformers are used to step-up the voltage to a higher level, or step-down to a lower level. A phase shifting transformers (PST) can also change the voltage phase angle.

A shunt is modelled as a reactance $z_s = \iota x_s$ between the bus and the ground, see Figure 5.3. The shunt admittance thus is $y_s = \frac{1}{z_s} = -\iota\frac{1}{x_s} = \iota b_s$. If $x_s > 0$ the shunt is inductive, if $x_s < 0$ the shunt is capacitive. Note that the shunt susceptance b_s has the opposite sign of the shunt reactance x_s.

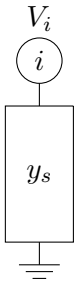

Figure 5.3: Shunt model

Transformers can be modelled as depicted in Figure 5.4, where $T : 1$ is the transformer ratio. The modulus of T determines the change in voltage magnitude. This value is usually around 1, because the better part of the differences in voltage levels are incorporated through the per unit system. The argument of T determines the shift of the voltage phase angle.

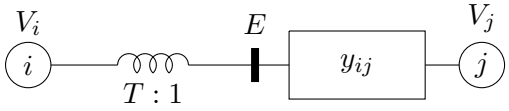

Figure 5.4: Transformer model

5.2.3 Admittance Matrix

The admittance matrix Y is a matrix that relates the injected current at each bus to bus voltages, such that

$$\boldsymbol{I} = Y\boldsymbol{V}, \tag{5.15}$$

where $\boldsymbol{I}$ is the vector of injected currents at each bus, and $\boldsymbol{V}$ is the vector of bus voltages. This is in fact Ohm's law (5.13) in matrix form. As such we can also define the impedance matrix $Z = Y^{-1}$.

To calculate the admittance matrix Y, we look at the injected current I_i at each bus i. Let I_{ij} denote the current flowing from bus i in the direction of bus $j \neq i$, or to the ground in case of a shunt. Applying Kirchhoff's current law now gives

$$I_i = \sum_k I_{ik}. \tag{5.16}$$

Let y_{ij} denote the admittance of the line between bus i and j, with $y_{ij} = 0$ if there is no line between these buses. For a simple transmission line from bus i to bus j — without shunt admittance — Ohm's law states that

$$I_{ij} = y_{ij}\left(V_i - V_j\right), \text{ and } I_{ji} = -I_{ij}, \tag{5.17}$$

or in matrix notation:

$$\begin{bmatrix} I_{ij} \\ I_{ji} \end{bmatrix} = y_{ij} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} V_i \\ V_j \end{bmatrix}. \tag{5.18}$$

Now suppose that there is a shunt s connected to bus i. Then, according to equation (5.16), an extra term I_{is} is added to the injected current I_i. From Figure 5.3, it is clear that

$$I_{is} = y_s\left(V_i - 0\right) = y_s V_i. \tag{5.19}$$

This means that in the admittance matrix an extra term y_s has to be added to Y_{ii}. Recall that $y_s = \imath b_s$, and that the sign of b_s depends on the shunt being inductive or capacitive.

Knowing how to deal with shunts, it is now easy to incorporate the line shunt model as depicted in Figure 5.2. For a transmission line between the buses i and j, half of the line shunt admittance of that line, i.e., $\frac{y_s}{2}$, has to be added to both Y_{ii} and Y_{jj} in the admittance matrix. For a transmission line with shunt admittance y_s, we thus find

$$\begin{bmatrix} I_{ij} \\ I_{ji} \end{bmatrix} = \left(y_{ij} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} + y_s \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \right) \begin{bmatrix} V_i \\ V_j \end{bmatrix}. \tag{5.20}$$

The influence on the admittance matrix of a transformer between the buses i and j, can be derived from the model depicted in Figure 5.4.

Let E be the voltage induced by the transformer, then

$$V_i = TE. \tag{5.21}$$

The current from bus j to the transformer device—and thus in the direction of bus i—then is

$$I_{ji} = y_{ij}\left(V_j - E\right) = y_{ij}\left(V_j - \frac{V_i}{T}\right). \tag{5.22}$$

Conservation of power within the transformer gives

$$V_i\overline{I}_{ij} = -E\overline{I}_{ji} \Leftrightarrow T\overline{I}_{ij} = -\overline{I}_{ji} \Leftrightarrow \overline{T}I_{ij} = -I_{ji}. \tag{5.23}$$

Therefore, the current from bus i to the transformer device—and thus in the direction of j—is

$$I_{ij} = -\frac{I_{ji}}{\overline{T}} = y_{ij}\left(\frac{V_i}{|T|^2} - \frac{V_j}{\overline{T}}\right). \tag{5.24}$$

The total contribution to the admittance matrix, of a branch between bus i and bus j, thus becomes

$$\begin{bmatrix} I_{ij} \\ I_{ji} \end{bmatrix} = \left(y_{ij} \begin{bmatrix} \frac{1}{|T|^2} & -\frac{1}{\overline{T}} \\ -\frac{1}{T} & 1 \end{bmatrix} + y_s \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \right) \begin{bmatrix} V_i \\ V_j \end{bmatrix}, \tag{5.25}$$

where $T = 1$ if the branch is not a transformer.

The admittance matrix Y can now be constructed as follows. Start with a diagonal matrix with the shunt admittance value on diagonal element i for each bus i that has a shunt device, and 0 on each diagonal element for which the corresponding bus has no shunt device. Then, for each branch add its contribution to the matrix according to equation (5.25).

5.3 Power Flow

The power flow problem, or load flow problem, is the problem of computing the flow of electrical power in a power system in steady state. In practice, this amounts to calculating the voltage in each bus of the power system. The problem arises in many applications in power system analysis and is treated in many books on power systems, see for example [7, 38, 42].

Mathematical equations for the power flow problem can be obtained by combining the complex power (5.9), with Ohm's law (5.15). This gives

$$S_i = V_i\overline{I}_i = V_i\left(\overline{Y\boldsymbol{V}}\right)_i = V_i\sum_{k=1}^{N}\overline{Y}_{ik}\overline{V}_k, \tag{5.26}$$

where S_i is the injected power at bus i, I_i the current through bus i, V_i the bus voltage, Y is the admittance matrix, and N is the number of buses in the power system

The admittance matrix Y is easy to obtain, and generally very sparse. Therefore a formulation using the admittance matrix has preference over formulations using the impedance matrix Z, which is generally a lot harder to obtain and not sparse.

In Chapter 6 two traditional methods to solve the power flow problem (5.26) are treated. In Chapter 7 we investigate power flow solvers based on inexact Newton-Krylov methods, and show such solvers scale much better in the problem size, making them much faster than the traditional methods for large power flow problems.

5.4 Contingency Analysis

Contingency analysis is the act of identifying changes in a power system that have some non-negligible chance of unplanned occurrence, and analysing the impact of these contingencies on power system operation. The most common contingencies are single generator and branch outages.

A power system that will still operate properly on the occurrence of any single contingency, is called $n-1$ secure. In some cases $n-2$ security analysis is desired, i.e., analysis of the impact of any two contingencies happening simultaneously.

Contingency analysis generally involves solving power flow problems in which the contingencies have been modelled. In Chapter 8 we investigate how the Newton-Krylov power flow solver can be exploited to speed up contingency analysis calculations.

CHAPTER 6

Traditional Power Flow Solvers

As long as there have been power systems, there have been power flow studies. This chapter discusses the two traditional methods to solve power flow problems: Newton power flow and Fast Decoupled Load Flow (FDLF).

Newton power flow is described in Section 6.1. The concept of the power mismatch function is treated, and the corresponding Jacobian matrix is derived. Further, it is detailed how to treat different bus types within the Newton power flow method.

Fast Decoupled Load Flow is treated in Section 6.2. The FDLF method can be seen as a clever approximation of Newton power flow. Instead of the Jacobian matrix, an approximation—based on the practical properties of power flow problems—is calculated once, and used throughout all iterations.

Finally, Section 6.3 discusses convergence and computational properties of the two methods, and Section 6.4 describes how Newton power flow and the FDLF method can be interpreted as basic Newton-Krylov methods, motivating how Newton-Krylov methods can be used to improve on these traditional power flow solvers.

6.1 Newton Power Flow

Newton power flow uses the Newton-Raphson method (see Chapter 3) to solve power flow problems. Traditionally, a direct solver is used to solve for the linear system of equations (3.6) that arises in each iteration of the Newton method [49, 50].

In order to use the Newton-Raphson method, the power flow equations have to be written in the form $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$. The common procedure to get such a form is described in Section 6.1.1. This procedure leads to a function $\boldsymbol{F}(\boldsymbol{x})$ called the power mismatch function. The power mismatch function contains the the injected active power P_i and reactive power Q_i at each bus, while the vector parameter $\boldsymbol{x}$ consists of the voltage angles δ_i and voltage magnitudes $|V_i|$.

Another element required for the Newton-Raphson method, is the Jacobian matrix $J(\boldsymbol{x})$. In Section 6.1.2 the Jacobian matrix of the power mismatch function is derived. Further, it is shown that this matrix can be computed cheaply from the building blocks used in the evaluation of the power mismatch function.

For load buses the voltage angle δ_i and voltage magnitude $|V_i|$ are the unknowns, see Table 5.1 (page 40). However, for generator buses the voltage magnitude δ_i is known, while the injected reactive power Q_i is unknown. And for the slack bus, the entire voltage phasor is known, while the injected power is unknown. Thus, the power mismatch function $\boldsymbol{F}(\boldsymbol{x})$ is not simply a known function in an unknown parameter. Section 6.1.3 deals with the steps needed for each of the different bus types, to be able to apply the Newton-Raphson method to the power mismatch function.

6.1.1 Power Mismatch Function

Recall from Section 5.3 that the power flow problem can be described by the equations

$$S_i = V_i \sum_{k=1}^{N} \overline{Y}_{ik} \overline{V}_k. \tag{6.1}$$

As it is not possible to treat the voltage phasors V_i as variables of the problem for the slack bus and generator buses, it makes sense to rewrite the N complex nonlinear equations of equation (5.26) as $2N$ real nonlinear equations in the quantities P_i, Q_i, $|V_i|$, and δ_i.

Substituting $V_i = |V_i| e^{\iota\delta_i}$, $Y = G + \iota B$, and $\delta_{ij} = \delta_i - \delta_j$ into the power flow equations (6.1) gives

$$\begin{aligned} S_i &= |V_i| e^{\iota\delta_i} \sum_{k=1}^{N} (G_{ik} - \iota B_{ik}) |V_k| e^{-\iota\delta_k} \\ &= \sum_{k=1}^{N} |V_i| |V_k| (\cos\delta_{ik} + \iota \sin\delta_{ik}) (G_{ik} - \iota B_{ik}). \end{aligned} \tag{6.2}$$

Now define the real vector $\boldsymbol{x}$ of voltage variables as

$$\boldsymbol{x} = [\ \delta_1, \ldots, \delta_N, |V_1|, \ldots, |V_N|\ ]^T. \tag{6.3}$$

For the purpose of notational comfort, further define the matrix functions $\mathcal{P}(\boldsymbol{x})$ and $\mathcal{Q}(\boldsymbol{x})$ with entries

$$\mathcal{P}_{ij}(\boldsymbol{x}) = |V_i|\,|V_j|\,(G_{ij}\cos\delta_{ij} + B_{ij}\sin\delta_{ij}), \tag{6.4}$$
$$\mathcal{Q}_{ij}(\boldsymbol{x}) = |V_i|\,|V_j|\,(G_{ij}\sin\delta_{ij} - B_{ij}\cos\delta_{ij}), \tag{6.5}$$

and the vector functions $\boldsymbol{P}(\boldsymbol{x})$ and $\boldsymbol{Q}(\boldsymbol{x})$ with entries

$$P_i(\boldsymbol{x}) = \textstyle\sum_k \mathcal{P}_{ik}(\boldsymbol{x}), \tag{6.6}$$
$$Q_i(\boldsymbol{x}) = \textstyle\sum_k \mathcal{Q}_{ik}(\boldsymbol{x}). \tag{6.7}$$

Note that $\mathcal{P}_{ij}(\boldsymbol{x}) = \mathcal{Q}_{ij}(\boldsymbol{x}) = 0$ for each pair of buses $i \neq j$ that is not connected by a branch.

Using the above definitions, equation (6.2) can be written as

$$\boldsymbol{S} = \boldsymbol{P}(\boldsymbol{x}) + \iota\boldsymbol{Q}(\boldsymbol{x}). \tag{6.8}$$

Now, the power mismatch function $\boldsymbol{F}$ is the real vector function

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{P} - \boldsymbol{P}(\boldsymbol{x}) \\ \boldsymbol{Q} - \boldsymbol{Q}(\boldsymbol{x}) \end{bmatrix}, \tag{6.9}$$

and the power flow problem can be written as the system of nonlinear equations

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}. \tag{6.10}$$

6.1.2 Jacobian Matrix

The Jacobian matrix $J(\boldsymbol{x})$ of a function $\boldsymbol{F}(\boldsymbol{x})$, is the matrix of all first order partial derivatives of that function. The Jacobian matrix of the power mismatch function has the structure, where $P_i(\boldsymbol{x})$ and $Q_i(\boldsymbol{x})$ are as in equations (6.6) and (6.7) respectively:

$$J(\boldsymbol{x}) = -\left[\begin{array}{ccc|ccc} \frac{\partial P_1}{\partial \delta_1}(\boldsymbol{x}) & \dots & \frac{\partial P_1}{\partial \delta_N}(\boldsymbol{x}) & \frac{\partial P_1}{\partial |V_1|}(\boldsymbol{x}) & \dots & \frac{\partial P_1}{\partial |V_N|}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_N}{\partial \delta_1}(\boldsymbol{x}) & \dots & \frac{\partial P_N}{\partial \delta_N}(\boldsymbol{x}) & \frac{\partial P_N}{\partial |V_1|}(\boldsymbol{x}) & \dots & \frac{\partial P_N}{\partial |V_N|}(\boldsymbol{x}) \\ \hline \frac{\partial Q_1}{\partial \delta_1}(\boldsymbol{x}) & \dots & \frac{\partial Q_1}{\partial \delta_N}(\boldsymbol{x}) & \frac{\partial Q_1}{\partial |V_1|}(\boldsymbol{x}) & \dots & \frac{\partial Q_1}{\partial |V_N|}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_N}{\partial \delta_1}(\boldsymbol{x}) & \dots & \frac{\partial Q_N}{\partial \delta_N}(\boldsymbol{x}) & \frac{\partial Q_N}{\partial |V_1|}(\boldsymbol{x}) & \dots & \frac{\partial Q_N}{\partial |V_N|}(\boldsymbol{x}) \end{array}\right]. \tag{6.11}$$

Note that the Jacobian matrix (6.11) consist of the negated first order derivatives of $P_i(\boldsymbol{x})$ and $Q_i(\boldsymbol{x})$, but that the Newton-Raphson method uses the negated Jacobian. Therefore, the coefficient matrix of the linear system solved in each iteration of the Newton method consists of the first order derivatives of $P_i(\boldsymbol{x})$ and $Q_i(\boldsymbol{x})$. These partial derivatives are derived below, where it is assumed that $i \neq j$ whenever applicable.

$$\frac{\partial P_i}{\partial \delta_j}(\boldsymbol{x}) = |V_i||V_j|(G_{ij}\sin\delta_{ij} - B_{ij}\cos\delta_{ij}) = \mathcal{Q}_{ij}(\boldsymbol{x}), \tag{6.12}$$

$$\begin{aligned}\frac{\partial P_i}{\partial \delta_i}(\boldsymbol{x}) &= \sum_{k\neq i}|V_i||V_k|(-G_{ik}\sin\delta_{ik} + B_{ik}\cos\delta_{ik}) \\ &= -\sum_{k\neq i}\mathcal{Q}_{ik}(\boldsymbol{x}) = -Q_i(\boldsymbol{x}) - |V_i|^2 B_{ii},\end{aligned} \tag{6.13}$$

$$\frac{\partial Q_i}{\partial \delta_j}(\boldsymbol{x}) = |V_i||V_j|(-G_{ij}\cos\delta_{ij} - B_{ij}\sin\delta_{ij}) = -\mathcal{P}_{ij}(\boldsymbol{x}), \tag{6.14}$$

$$\begin{aligned}\frac{\partial Q_i}{\partial \delta_i}(\boldsymbol{x}) &= \sum_{k\neq i}|V_i||V_k|(G_{ik}\cos\delta_{ik} + B_{ik}\sin\delta_{ik}) \\ &= \sum_{k\neq i}\mathcal{P}_{ik}(\boldsymbol{x}) = P_i(\boldsymbol{x}) - |V_i|^2 G_{ii},\end{aligned} \tag{6.15}$$

$$\frac{\partial P_i}{\partial |V_j|}(\boldsymbol{x}) = |V_i|(G_{ij}\cos\delta_{ij} + B_{ij}\sin\delta_{ij}) = \frac{\mathcal{P}_{ij}(\boldsymbol{x})}{|V_j|}, \tag{6.16}$$

$$\begin{aligned}\frac{\partial P_i}{\partial |V_i|}(\boldsymbol{x}) &= 2|V_i|G_{ii} + \sum_{k\neq i}|V_k|(G_{ik}\cos\delta_{ik} + B_{ik}\sin\delta_{ik}) \\ &= 2|V_i|G_{ii} + \sum_{k\neq i}\frac{\mathcal{P}_{ik}(\boldsymbol{x})}{|V_i|} = \frac{P_i(\boldsymbol{x}) + |V_i|^2 G_{ii}}{|V_i|},\end{aligned} \tag{6.17}$$

$$\frac{\partial Q_i}{\partial |V_j|}(\boldsymbol{x}) = |V_i|(G_{ij}\sin\delta_{ij} - B_{ij}\cos\delta_{ij}) = \frac{\mathcal{Q}_{ij}(\boldsymbol{x})}{|V_j|}, \tag{6.18}$$

$$\begin{aligned}\frac{\partial Q_i}{\partial |V_i|}(\boldsymbol{x}) &= -2|V_i|B_{ii} + \sum_{k\neq i}|V_k|(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik}) \\ &= -2|V_i|B_{ii} + \sum_{k\neq i}\frac{\mathcal{Q}_{ik}(\boldsymbol{x})}{|V_i|} = \frac{Q_i(\boldsymbol{x}) - |V_i|^2 B_{ii}}{|V_i|}.\end{aligned} \tag{6.19}$$

Observe that the Jacobian matrix entries consist of the same building blocks $\mathcal{P}_{ij}$ and $\mathcal{Q}_{ij}$ as the power mismatch function $\boldsymbol{F}$. This means that whenever the power mismatch function is evaluated, the Jacobian matrix can be calculated at relatively little extra computational cost.

6.1.3 Handling Different Bus Types

Which of the values P_i, Q_i, $|V_i|$, and δ_i are specified, and which are not, depends on the associated buses, see Table 5.1 (page 40).

Dealing with the fact that some elements in $\boldsymbol{P}$ and $\boldsymbol{Q}$ are not specified is easy. The equations corresponding to these unknowns can simply be dropped from the problem. The unknown voltages in $\boldsymbol{x}$ can be calculated from the remaining equations, after which the unknown power values follow by evaluating the corresponding entries of $\boldsymbol{P}(\boldsymbol{x})$ and $\boldsymbol{Q}(\boldsymbol{x})$.

Dealing with specified voltage values is less straight-forward. Recall that the Newton-Raphson method is an iterative process that, in each iteration, calculates a vector $\boldsymbol{s}_i$ and sets the new iterate to be $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{s}_i$. Now, if some entries of $\boldsymbol{x}$ are known—as is the case for generator buses and the slack bus—then the best value for the corresponding entry of the update vector $\boldsymbol{s}_i$ is clearly 0.

To ensure that the update for known voltage values is indeed 0, these entries in the update vector, and the corresponding columns of the coefficient matrix, can simply be dropped. Thus for every generator bus, one unknown in the update vector and one column in the coefficient matrix are dropped, whereas for the slack bus two of each are dropped.

The amount of nonlinear equations dropped from the problem, is always equal to the amount of variables, and corresponding columns, dropped from the linear systems. Therefore, the linear systems that are actually solved have a square coefficient matrix of size $2N - N_G - 2 = 2N_L + N_G$, where N_L is the number of load buses, and N_G is the number of generator buses in the power system.

Another method to deal with different bus types is not to eliminate any rows or columns from the problem. Instead the linear systems are built normally, except for the linear equations that correspond to power values that are not specified. For these equations, the right-hand side value and all off-diagonal entries are set to 0, while the diagonal entry is set to 1. Or, the diagonal entry can be set to some very large number, in which case the off-diagonal entries can be kept as they would have been.

This method also ensures that the update for known voltage values is 0 in each iteration. The linear systems that have to be solved are of size $2N$, and thus larger than in the previous method. However, the structure of the matrix can be made independent of the bus types. This mean that the matrix structure can be kept between runs that change the type of one or more buses. Bus-type switching is used for example to ensure that reactive power limits of generators are satisfied.

6.2 Fast Decoupled Load Flow

Fast Decoupled Load Flow (FDLF) is an approximation of Newton power flow, based on practical properties of power flow problems. The general FDLF method is shown in Algorithm 6.1.

Algorithm 6.1 Fast Decoupled Load Flow

1: calculate the matrices B' and B''
2: calculate LU factorisation of B' and B''
3: given initial iterates $\boldsymbol{\delta}$ and $|\boldsymbol{V}|$
4: **while** not converged **do**
5: solve $B'\boldsymbol{\Delta\delta} = \boldsymbol{\Delta P}(\boldsymbol{\delta}, |\boldsymbol{V}|)$
6: update $\boldsymbol{\delta} := \boldsymbol{\delta} + \boldsymbol{\Delta\delta}$
7: solve $B''\boldsymbol{\Delta}|\boldsymbol{V}| = \boldsymbol{\Delta Q}(\boldsymbol{\delta}, |\boldsymbol{V}|)$
8: update $|\boldsymbol{V}| := |\boldsymbol{V}| + \boldsymbol{\Delta}|\boldsymbol{V}|$
9: **end while**

The original derivation of the method is presented in Section 6.2.1, and in Section 6.2.2 notes on dealing with shunts and transformers are added. Finally, Section 6.2.3 treats different choices for the matrices B' and B'', called schemes, and explains how the BX and XB schemes can be interpreted as an approximation of Newton power flow using the Schur complement. The techniques described in Section 6.1.3 can again be used to handle the different bus types.

6.2.1 Classical Derivation

In Fast Decoupled Load Flow, the assumption is made that for all i, j

$$\delta_{ij} \approx 0, \tag{6.20}$$
$$|V_i| \approx 1. \tag{6.21}$$

In the original derivation in [46] it is further assumed that

$$|G_{ij}| \ll |B_{ij}|. \tag{6.22}$$

Using assumption (6.20), the following approximations can be made:

$$\mathcal{P}_{ij}(\boldsymbol{x}) = |V_i||V_j|(G_{ij}\cos\delta_{ij} + B_{ij}\sin\delta_{ij}) \approx +|V_i||V_j|G_{ij}, \tag{6.23}$$
$$\mathcal{Q}_{ij}(\boldsymbol{x}) = |V_i||V_j|(G_{ij}\sin\delta_{ij} - B_{ij}\cos\delta_{ij}) \approx -|V_i||V_j|B_{ij}. \tag{6.24}$$

Note that for $i = j$ these approximations are exact, since $\delta_{ii} = 0$.

From assumption (6.22) it then follows that

$$|G_{ij}| \approx |\mathcal{P}_{ij}(\boldsymbol{x})| \ll |\mathcal{Q}_{ij}(\boldsymbol{x})| \approx |B_{ij}|. \tag{6.25}$$

This leads to the idea of decoupling, i.e., neglecting the off-diagonal blocks of the Jacobian matrix, which are based on G_{ij} and $\mathcal{P}_{ij}$, as they are small compared to the B_{ij} and $\mathcal{Q}_{ij}$ based diagonal blocks.

By the above assumptions, the first order derivatives that constitute the Jacobian matrix of the Newton power flow process can be approximated as follows. Note that it is assumed that $i \neq j$ whenever applicable, and that assumption (6.21) is used in the first two equations, though only on $|V_j|$.

$$\frac{\partial P_i}{\partial \delta_j}(\boldsymbol{x}) = \mathcal{Q}_{ij}(\boldsymbol{x}) \approx -|V_i||V_j|B_{ij} \approx -|V_i|B_{ij}, \tag{6.26}$$

$$\frac{\partial P_i}{\partial \delta_i}(\boldsymbol{x}) = -\sum_{k\neq i}\mathcal{Q}_{ik}(\boldsymbol{x}) \approx \sum_{k\neq i}|V_i||V_k|B_{ik} \approx |V_i|\sum_{k\neq i}B_{ik}, \tag{6.27}$$

$$\frac{\partial Q_i}{\partial \delta_j}(\boldsymbol{x}) = -\mathcal{P}_{ij}(\boldsymbol{x}) \approx 0, \tag{6.28}$$

$$\frac{\partial Q_i}{\partial \delta_i}(\boldsymbol{x}) = \sum_{k\neq i}\mathcal{P}_{ik}(\boldsymbol{x}) \approx 0, \tag{6.29}$$

$$\frac{\partial P_i}{\partial |V_j|}(\boldsymbol{x}) = \frac{\mathcal{P}_{ij}(\boldsymbol{x})}{|V_j|} \approx 0, \tag{6.30}$$

$$\frac{\partial P_i}{\partial |V_i|}(\boldsymbol{x}) = 2|V_i|G_{ii} + \sum_{k\neq i}\frac{\mathcal{P}_{ik}(\boldsymbol{x})}{|V_i|} \approx 0, \tag{6.31}$$

$$\frac{\partial Q_i}{\partial |V_j|}(\boldsymbol{x}) = \frac{\mathcal{Q}_{ij}(\boldsymbol{x})}{|V_j|} \approx -|V_i|B_{ij}, \tag{6.32}$$

$$\frac{\partial Q_i}{\partial |V_i|}(\boldsymbol{x}) = -2|V_i|B_{ii} + \sum_{k\neq i}\frac{\mathcal{Q}_{ik}(\boldsymbol{x})}{|V_i|} \approx -2|V_i|B_{ii} - \sum_{k\neq i}|V_k|B_{ik}. \tag{6.33}$$

The last equation (6.33) still requires some work. To this purpose, define the negated row sum $\mathcal{D}_i$ of the imaginary part B of the admittance matrix:

$$\mathcal{D}_i = \sum_k -B_{ik} = -B_{ii} - \sum_{k\neq i}B_{ik}. \tag{6.34}$$

Note that, if the diagonal elements of B are negative and the off-diagonal elements are nonnegative, then $\mathcal{D}_i$ is the diagonal dominance of row i. In a system with only generators, loads, and transmission lines without line shunts, $\mathcal{D}_i = 0$ for all i.

Now, use assumption (6.21) on equation (6.33) to approximate $|V_k|$ by $|V_i|$ for all k. This gives

$$\begin{aligned}\frac{\partial Q_i}{\partial |V_i|}(\boldsymbol{x}) &\approx -2\,|V_i|\,B_{ii} - \sum_{k\neq i} |V_k|\,B_{ik} \\ &\approx -2\,|V_i|\,B_{ii} - |V_i| \sum_{k\neq i} B_{ik} \\ &= |V_i| \sum_{k\neq i} B_{ik} - 2\,|V_i| \left(B_{ii} + \sum_{k\neq i} B_{ik} \right) \\ &= |V_i| \sum_{k\neq i} B_{ik} + 2\,|V_i|\,\mathcal{D}_i. \end{aligned} \tag{6.35}$$

The only term left, in the approximated Jacobian matrix, that depends on the current iterate, is $|V_i|$. Because of assumption (6.21) this term can be simple set to 1. Another common strategy to remove the dependence on the current iterate from the approximated Jacobian matrix, is to divide each linear equation i by $|V_i|$ in every iteration of the FDLF process. In both cases, the coefficient matrices are the same and constant throughout all iterations. The off-diagonal blocks of these matrices are 0. The upper and lower diagonal blocks are referred to as B' and B'' respectively:

$$B'_{ij} = -B_{ij} \quad (i \neq j), \tag{6.36}$$

$$B'_{ii} = \sum_{k\neq i} B_{ik}, \tag{6.37}$$

$$B''_{ij} = -B_{ij} \quad (i \neq j), \tag{6.38}$$

$$B''_{ii} = \sum_{k\neq i} B_{ik} + 2\mathcal{D}_i. \tag{6.39}$$

Note that, in a system with only generators, loads and transmission lines, B' is equal to $-B$ without any line shunts incorporated, while B'' is equal to $-B$ with double line shunt values.

Summarising, the FDLF method calculates the update for the iterate in iteration k by solving the following linear systems:

$$B'\boldsymbol{\Delta\delta}^k = \boldsymbol{\Delta P}^k, \tag{6.40}$$

$$B''\boldsymbol{\Delta}\,|\boldsymbol{V}|^k = \boldsymbol{\Delta Q}^k, \tag{6.41}$$

with either

$$\Delta P_i^k = P_i - P_i\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right) \text{ and } \Delta Q_i^k = Q_i - Q_i\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right), \tag{6.42}$$

or

$$\Delta P_i^k = \frac{P_i - P_i\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right)}{|V_i|} \text{ and } \Delta Q_i^k = \frac{Q_i - Q_i\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right)}{|V_i|}. \tag{6.43}$$

6.2.2 Shunts and Transformers

A few additional notes can be made with respect to shunts and transformers, the treatment of which is different between B' and B''.

Shunts have the same influence on the system as transmission line shunts, i.e., they only change the diagonal entries of the admittance matrix. Thus, shunts are left out in B', and doubled in B''.

The modulus $|T|$ of the transformer ratio changes the voltage magnitude, and is therefore generally simply set to 1 in the calculation of B', which works on the voltage phase angle. Likewise, the argument $\arg(T)$ changes the voltage phase angle, and is usually set to 0 for the calculation of B'', which works on the voltage magnitude.

6.2.3 BB, XB, BX, and XX

The Fast Decoupled Load Flow method derived in Section 6.2.1 is commonly referred to as the BB version, because the susceptance values

$$B_{ij} = \operatorname{Im}\left(\frac{1}{R_{ij} + \iota X_{ij}}\right) = \frac{-X_{ij}}{R_{ij}^2 + X_{ij}^2}. \tag{6.44}$$

are used for both B' and B''.

Stott and Alsac [46] already reported improved convergence in many power flow problems, if the series resistance R was neglected in B', i.e., if for B' the values

$$B_{ij}^X = \operatorname{Im}\left(\frac{1}{\iota X_{ij}}\right) = \frac{-1}{X_{ij}} \tag{6.45}$$

are used instead of the full susceptance. This methods is called the XB scheme, because B' is derived from the reactance values X_{ij}, and B'' from the susceptance values B_{ij}.

Van Amerongen [52] found that the BX scheme, where B' is derived from the susceptance values B_{ij}, and B'' from the reactance values X_{ij}, yields convergence that is comparable to XB in most cases, and considerably better in some. Further, he noted that an XX scheme is never better than the BX and XB schemes.

Monticelli, et al. [35] presented mathematical support for the good results obtained with the XB and BX schemes. Their idea is the following. Starting with assumptions (6.20) and (6.21), the Jacobian system of the Newton power flow method can be approximated by

$$\begin{bmatrix} -B & G \\ -G & -B \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta\delta} \\ \boldsymbol{\Delta}\,|\boldsymbol{V}| \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Delta P} \\ \boldsymbol{\Delta Q} \end{bmatrix}. \tag{6.46}$$

For simplicity, the differences between the diagonals of the upper-left and lower-right blocks, as well as those of the lower-left and upper-right blocks, are neglected.

It should be noted, that the remarks on the incorporation of line shunts described in Section 6.2.1, and those on shunts and transformers described in Section 6.2.2, remain useful to improve convergence.

Using downward block Gaussian elimination on the Jacobian system approximation (6.46) gives

$$\begin{bmatrix} -B & G \\ 0 & -\left(B + GB^{-1}G\right) \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta\delta} \\ \boldsymbol{\Delta}\left|\boldsymbol{V}\right| \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Delta P} \\ \boldsymbol{\Delta Q} - GB^{-1}\boldsymbol{\Delta P} \end{bmatrix}. \quad (6.47)$$

This linear system is solved in three steps, that are then combined into the two steps of the BX scheme.

Step 1: Calculate the partial voltage angle update $\boldsymbol{\Delta\delta}_B^k$ from

$$-B\boldsymbol{\Delta\delta}_B^k = \boldsymbol{\Delta P}\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right) \Rightarrow \boldsymbol{\Delta\delta}_B^k = -B^{-1}\boldsymbol{\Delta P}\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right), \quad (6.48)$$

where k is the current FDLF iteration.

Step 2: Calculate the voltage magnitude update $\boldsymbol{\Delta}\left|\boldsymbol{V}\right|^k$ from

$$-\left(B + GB^{-1}G\right)\boldsymbol{\Delta}\left|\boldsymbol{V}\right|^k \approx \boldsymbol{\Delta Q}\left(\boldsymbol{\delta}^k + \boldsymbol{\Delta\delta}_B^k, |\boldsymbol{V}|^k\right). \quad (6.49)$$

This is an approximation of the lower block of linear equations in (6.47), since the first order Taylor expansion can be used to write

$$\begin{aligned} \boldsymbol{\Delta Q}\left(\boldsymbol{\delta}^k + \boldsymbol{\Delta\delta}_B^k, |\boldsymbol{V}|^k\right) &\approx \boldsymbol{\Delta Q}\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right) + \frac{\partial \boldsymbol{\Delta Q}}{\partial \boldsymbol{\delta}}\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right)\boldsymbol{\Delta\delta}_B^k \\ &\approx \boldsymbol{\Delta Q}\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right) - GB^{-1}\boldsymbol{\Delta P}\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right). \quad (6.50) \end{aligned}$$

Here it is used that the partial derivative of $\boldsymbol{\Delta Q}$ with respect to $\boldsymbol{\delta}$ is in the bottom-left block of the Jacobian matrix (6.11), which is approximated by the matrix $-G$ in accordance with equation (6.46).

Step 3: Calculate a second partial voltage angle update $\boldsymbol{\Delta\delta}_G^k$ from

$$B\boldsymbol{\Delta\delta}_G^k = G\boldsymbol{\Delta}\left|\boldsymbol{V}\right|^k \Rightarrow \boldsymbol{\Delta\delta}_G^k = B^{-1}G\boldsymbol{\Delta}\left|\boldsymbol{V}\right|^k. \quad (6.51)$$

Then the solution of the upper block of equations in (6.47) is given by

$$\boldsymbol{\Delta\delta}^k = -B^{-1}\boldsymbol{\Delta P}\left(\boldsymbol{\delta}^k, |\boldsymbol{V}|^k\right) + B^{-1}G\boldsymbol{\Delta}\left|\boldsymbol{V}\right|^k = \boldsymbol{\Delta\delta}_B^k + \boldsymbol{\Delta\delta}_G^k. \quad (6.52)$$

The next step would be step 1 of the next iteration, i.e.,

$$\boldsymbol{\Delta\delta}_B^{k+1} = -B^{-1}\boldsymbol{\Delta P}\left(\boldsymbol{\delta}^{k+1}, |\boldsymbol{V}|^{k+1}\right). \tag{6.53}$$

However, note that

$$\begin{aligned}
&\boldsymbol{\Delta\delta}_B^{k+1} + \boldsymbol{\Delta\delta}_G^{k} && (6.54)\\
&\quad = -B^{-1}\boldsymbol{\Delta P}\left(\boldsymbol{\delta}^{k+1}, |\boldsymbol{V}|^{k+1}\right) + B^{-1}G\boldsymbol{\Delta}\,|\boldsymbol{V}|^{k} && (6.55)\\
&\quad = -B^{-1}\left(\boldsymbol{\Delta P}\left(\boldsymbol{\delta}^{k} + \boldsymbol{\Delta\delta}_B^{k} + \boldsymbol{\Delta\delta}_G^{k}, |\boldsymbol{V}|^{k+1}\right) - G\boldsymbol{\Delta}\,|\boldsymbol{V}|^{k}\right) && (6.56)\\
&\quad \approx -B^{-1}\left(\boldsymbol{\Delta P}\left(\boldsymbol{\delta}^{k} + \boldsymbol{\Delta\delta}_B^{k}, |\boldsymbol{V}|^{\boldsymbol{k+1}}\right) + B\boldsymbol{\Delta\delta}_G^{k} - G\boldsymbol{\Delta}\,|\boldsymbol{V}|^{k}\right) && (6.57)\\
&\quad = -B^{-1}\boldsymbol{\Delta P}\left(\boldsymbol{\delta}^{k} + \boldsymbol{\Delta\delta}_B^{k}, |\boldsymbol{V}|^{k+1}\right), && (6.58)
\end{aligned}$$

where a first order Taylor expansion—similar to that in (6.50)—is used to go from equation (6.56) to (6.57). Thus, instead of calculating $\boldsymbol{\delta}_G^k$ to update the voltage angle with it, and then calculating $\boldsymbol{\delta}_B^{k+1}$ from equation (6.53) to again update the voltage angle, instead a single combined voltage angle update $\boldsymbol{\Delta\delta}_B^{k+1} + \boldsymbol{\Delta\delta}_G^k$ can be calculated from equation (6.58).

The above observations lead to the following iteration scheme:

$$\begin{array}{ll}
\text{solve} & -B\boldsymbol{\Delta\delta} = \boldsymbol{\Delta P}\left(\boldsymbol{\delta}, |\boldsymbol{V}|\right),\\
\text{update} & \boldsymbol{\delta} := \boldsymbol{\delta} + \boldsymbol{\Delta\delta},\\
\text{solve} & -\left(B + GB^{-1}G\right)\boldsymbol{\Delta}\,|\boldsymbol{V}| = \boldsymbol{\Delta Q}\left(\boldsymbol{\delta}, |\boldsymbol{V}|\right),\\
\text{update} & |\boldsymbol{V}| := |\boldsymbol{V}| + \boldsymbol{\Delta}\,|\boldsymbol{V}|.
\end{array}$$

Note that $\boldsymbol{\Delta\delta}$ here denotes the combined update from equation (6.54).

It thus remains to show that the matrix $-\left(B + GB^{-1}G\right)$ is properly represented in the FDLF method. To this purpose, write $B = A^T dBA$ and $G = A^T dGA$, where A is the incidence matrix of the associated graph (see Section A.4) and the matrices dB and dG are the diagonal matrices of edge susceptances and edge conductances respectively.

There are two special cases in which this notation can be used to simplify the matrix $-\left(B + GB^{-1}G\right)$. First, if the network is radial then A can be set up as a square nonsingular matrix, see [35], and

$$\begin{aligned}
-\left(B + GB^{-1}G\right) &= -A^T dBA - A^T dGA\left(A^T dBA\right)^{-1} A^T dGA\\
&= -A^T dBA - A^T dGAA^{-1}dB^{-1}A^{-T}A^T dGA\\
&= -A^T dBA - A^T\left(dG^2 dB^{-1}\right)A. \qquad (6.59)
\end{aligned}$$

For the second case, note that

$$B_{ij} = \frac{-X_{ij}}{R_{ij}^2 + X_{ij}^2} = -\frac{X_{ij}}{R_{ij}}\frac{R_{ij}}{R_{ij}^2 + X_{ij}^2} = -\frac{X_{ij}}{R_{ij}}G_{ij}. \tag{6.60}$$

Therefore, if the R/X ratio $\rho = \frac{R_{ij}}{X_{ij}}$ is equal on all branches of the power system, then $dB = -\frac{1}{\rho} dG$, and

$$\begin{aligned} -\left(B + GB^{-1}G\right) &= -A^T dBA - A^T dGA \left(A^T dBA\right)^{-1} A^T dGA \\ &= -A^T dBA + \rho A^T dGA \left(A^T dGA\right)^{-1} A^T dGA \\ &= -A^T dBA + \rho A^T dGA \\ &= -A^T dBA - A^T \left(dG^2 dB^{-1}\right) A. \end{aligned} \tag{6.61}$$

Both cases lead to the same result, which can be further simplified to

$$\begin{aligned} -\left(B + GB^{-1}G\right) &= -A^T dBA - A^T \left(dG^2 dB^{-1}\right) A \\ &= -A^T \left(dB^2 + dG^2\right) dB^{-1} A \\ &= -A^T \left(dX^{-1}\right) A, \end{aligned} \tag{6.62}$$

where $A^T \left(dX^{-1}\right) A$ is equal to the matrix B^X, as defined in equation (6.45).

For general networks, if the R/X ratios do not vary a lot, the matrix constructed from the inverse reactances X_{ij}^{-1} can therefore be used as an approximation of the Schur complement matrix $\left(B + GB^{-1}G\right)$. This leads to the BX scheme of the Fast Decoupled Load Flow method.

Similar to the above derivation, starting with the linear system (6.46), and applying Gaussian elimination upward instead of downward, the XB scheme can be derived. However, when there are PV buses, the convergence of this scheme becomes less reliable than that of the BX scheme. This can be understood by analysing what happens to the XB scheme if all buses are PV buses. In this case the vector $|\boldsymbol{V}|$ is known, and the linear system from equation (6.46) reduces to

$$-B\boldsymbol{\Delta\delta} = \boldsymbol{\Delta P}. \tag{6.63}$$

In the BX scheme, this is indeed the system that is solved. However, in the XB scheme, the coefficient matrix $-B^X$ is used instead of $-B$, leading to unnecessary extra approximation errors.

Summarising, with the assumptions that $\delta_{ij} \approx 0$ and $|V_i| \approx 1$, and the assumption that the R/X ratio does not vary too much between different branches in the network, the BX and XB schemes of the Fast Decoupled Load Flow method can be derived. The assumption on the R/X ratios replaces the original assumption (6.22). The BX and XB schemes of the Fast Decoupled Load Flow method are not decoupled in the original meaning of the term, because the off-diagonal blocks are not disregarded, but are incorporated in the method. As such, these schemes generally have better convergence properties than the BB scheme.

6.3 Convergence and Computational Properties

The convergence of Newton power flow is generally better than that of Fast Decoupled Load Flow, since the FDLF method is a direct approximation of Newton power flow. The Newton-Raphson method has quadratic convergence when the iterate is close enough to the solution. Fast Decoupled Load Flow often exhibits convergence that is approximately linear. FDLF convergence may be close to the Newton power flow convergence in early iterations, when the iterate is still relatively far from the solution, but closer to the solution Newton power flow converges much faster. Furthermore, in some cases FDLF may fail to converge, while Newton power flow can still find a solution.

Newton power flow and Fast Decoupled Load Flow both evaluate the power mismatch equations in every iteration. The FDLF method calculates the coefficient matrices B' and B'' only once at the start. In the case of Newton power flow, the Jacobian matrix has to be calculated in every iteration. However, the Jacobian matrix can be computed at relatively little extra cost when evaluating the power mismatch function, see Section 6.1.2. Thus, there is no significant computational difference in terms of the evaluation of the power mismatch and coefficient matrices.

Both algorithms traditionally use a direct method to solve the linear systems of equations. Newton power flow needs to make an LU decomposition of the Jacobian in each iteration. In case of the FDLF method, the LU decomposition of B' and B'' can be made once at the start. Then, in every iteration, only forward and backward substitutions are needed to solve the linear systems, reducing computational cost (see Section 2.1.3). Furthermore, the FDLF coefficient matrices B' and B'' each hold about a quarter of the number of nonzeros that the Jacobian matrix has, reducing memory requirements and computational cost compared to Newton power flow.

Summarising, the choice between Newton power flow and Fast Decoupled Load Flow is about reducing computational and memory cost per iteration, at the cost of convergence speed and robustness.

In practice, Newton power flow is usually preferred because of the improved robustness. In the discussion of [11], it was also agreed upon that for the large complex power flow problems of the future, the focus should be on Newton power flow, rather than Fast Decoupled Load Flow. As discussed—both in theory and experiments—in the remainder of this work, Newton-Krylov power flow methods offer the best of both.

6.4 Interpretation as Elementary Newton-Krylov Methods

Both traditional Newton power flow and Fast Decoupled Load Flow can be interpreted as simple Newton-Krylov power flow solvers, that perform a single Richardson iteration (see Section 2.2.1) in each Newton step. In the case of Newton power flow the Richardson iteration is preconditioned using an LU factorisation of the Jacobian matrix. For Fast Decoupled Load Flow the preconditioner instead is the FDLF operator, consisting of LU factorisations of B' and B''. This interpretation shows a clear path of investigation, towards improving on the traditional power flow solvers. The single Richardson iteration should be replaced by the combination of a more efficient Krylov method, like GMRES, Bi-CGSTAB, or IDR(s), and a good strategy for choosing the forcing terms.

For Fast Decoupled Load Flow this directly leads to a proper Newton-Krylov method, preconditioned with the FDLF operator. Provided that the used Krylov method converges linearly or better, the total amount of linear iterations performed is no larger than the total amount of Richardson iterations needed for FDLF (see Chapter 4), while the amount of nonlinear iterations goes down, and the convergence and robustness improve to the level of Newton power flow.

Newton power flow needs some more work. Since the preconditioner is a direct solve on the coefficient matrix, a single linear iteration leads to convergence independent of the Krylov method and the forcing terms. Thus the preconditioner has to be relaxed. Obvious candidates are using an incomplete LU factorisation of the Jacobian matrix in each Newton iteration, or a single LU or ILU factorisation of the initial Jacobian J_0 throughout all Newton iterations. A relaxed preconditioner leads to more linear iterations being needed. However, if the calculation of the relaxed preconditioner is sufficiently faster than the direct solves of the traditional method, the method may be faster overall.

In Chapter 7 we investigate the use of Newton-Krylov solvers for power flow problems in detail, and compare the performance of such methods with that of traditional Newton power flow.

CHAPTER 7

Newton-Krylov Power Flow Solver

Newton power flow solvers traditionally use a direct solver for the linear systems [49, 50] (see also Chapter 6). Iterative linear solvers are generally more efficient than direct solvers for large linear systems of equations with a sparse coefficient matrix (see Chapter 2). Using a Krylov method to solve the Jacobian systems in the Newton-Raphson method, leads to an inexact Newton-Krylov method (see Chapter 3). It has been recognised that such solvers can offer advantages over the traditional implementation for large power systems [43, 37, 22, 14, 9, 58].

In this chapter, all aspects of an inexact Newton-Krylov power flow solver are discussed, using the numerical experiments in Chapter 9 as a reference. Section 7.1 focuses on which Krylov method to use, followed by a discussion on preconditioning in Section 7.2, and a treatment of different strategies to choose the forcing terms in Section 7.3. Section 7.4 gives an overview of the speed and scaling of the Newton-Krylov power flow solver, and Section 7.5 discusses the robustness of the solver.

We show that direct solvers, and other methods using the LU factorisation, scale very badly in the problem size. The alternatives proposed in this chapter are faster for all tested problems and show near linear scaling, thus being much faster for large power flow problems. The largest test problem, with one million buses, takes over an hour to solve using a direct solver, while our Newton-Krylov solver can solve it in less than 30 seconds. That is 120 times faster.

Furthermore, the Newton-Krylov power flow solver offers more options to tweak settings and reuse information when solving many closely related power flow problems, as is done for example in contingency analysis. This can lead to a significant reduction of computation time (see Chapter 8).

7.1 Linear Solver

In every Newton iteration, a linear system of the form

$$-J_i \boldsymbol{s}_i = \boldsymbol{F}_i \tag{7.1}$$

has to be solved. The Jacobian matrix J_i can be calculated at very little extra cost when evaluating the power mismatch $\boldsymbol{F}_i$ (see Section 6.1.2). Therefore, there is no need to resort to an approximate Jacobian, or Jacobian-free method (see Section 3.1).

The linear solver considered in this chapter is preconditioned GMRES (see Section 2.2). GMRES has the optimality property, and thus solves the linear problem in the minimal number of iterations generally needed by a Krylov method. The number of iterations needed to converge says something about the quality of the preconditioner. Whether restarted GMRES, or other Krylov methods—like Bi-CGSTAB or IDR(s)—should also be considered, can be derived from the performance of GMRES.

In our experiments, the best results were attained with high quality preconditioners. Then, only a low number of GMRES iterations are needed per Newton iteration. At such low iteration counts there is no reason to restart GMRES, nor to drop the optimality property for a method with short recurrences. Bi-CGSTAB proved faster than GMRES for lower quality preconditioners; however, GMRES with a high quality preconditioner was still faster than Bi-CGSTAB with lower quality preconditioners. IDR(s) can be expected to lead to a similar result for the tested power flow problems.

7.2 Preconditioning

Preconditioning is essential to the performance of Krylov methods such as GMRES, see Section 2.2.4. Our solver uses right preconditioning, meaning that the linear system

$$J_i P_i^{-1} \boldsymbol{z}_i = -\boldsymbol{F}_i, \tag{7.2}$$

is solved and the Newton step $\boldsymbol{s}_i$ follows from 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, a fast way to solve linear systems of the form $P_i \boldsymbol{u}_i = \boldsymbol{v}_i$ is needed, as such a system has to be solved in each iteration of the linear solver.

In this work we consider 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 get such a preconditioner, we choose a target matrix Q_i and construct either the LU factorisation (see Section 2.1.1) or an ILU(k) factorisation (see Section 2.1.5). Section 7.2.1 presents the different choices

for the target matrix. In Section 7.2.2 quality and fill-in are discussed for the two factorisation methods.

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). Nonstationary iterative methods, like GMRES itself, can only be used with special flexible iterative methods, like FGMRES [39]. The use of FGMRES with a GMRES based preconditioner was explored in [58].

Algebraic Multigrid (AMG) methods can also be used for as preconditioner. A cycle of AMG, with a stationary solver 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 [51, App. A].

7.2.1 Target Matrices

The target matrix Q_i is the matrix used to derive the preconditioner from. In this work three options are considered. These are the Jacobian matrix $Q_i = J_i$, the initial Jacobian matrix $Q_i = J_0$, and $Q_i = \Phi$, where

$$\Phi = \begin{bmatrix} B' & 0 \\ 0 & B'' \end{bmatrix}, \tag{7.3}$$

with B' and B'' as in the BX scheme of the Fast Decoupled Load Flow method (see Section 6.2).

The FDLF matrix Φ can be seen as an approximate Schur complement of the initial Jacobian matrix, as discussed in Section 6.2.3. This matrix has already been shown to be a good preconditioner [22]. It is a lower quality preconditioner than the initial Jacobian matrix itself. However, it consists of two independent blocks with each about a quarter of the nonzeros found in the Jacobian matrix. The structure, and the lower nonzero count, provide benefits in computing time and memory usage.

7.2.2 Factorisation

The preconditioners used in our solver are LU factorisations and ILU(k) factorisations of the target matrices discussed in Section 7.2.1. For the LU factorisation this leads to preconditioners $P_i = L_i U_i = Q_i$, whereas with the ILU factorisation the preconditioners $P_i = L_i U_i$ only resembles the target matrix Q_i. An ILU factorisation is generally cheaper to build than the LU factorisation, whereas the LU factorisation will result in a higher quality preconditioner.

With the LU factorisation the quality of the preconditioner is predetermined by the target matrix, since $P_i = Q_i$. The LU decomposition of a sparse matrix generally leads to a certain amount of fill-in (see Section 2.1.4).

The more fill-in, the more memory and computational time are needed for the factorisation and the forward and backward substitutions. It is therefore important to try to minimise the fill-in of the LU factorisation, by choosing a proper ordering of the rows and columns of the target matrix Q_i.

With the ILU(k) factorisation, the fill-in ratio is influenced both by the number of levels k, and the row and column ordering. The effect of the matrix ordering on the fill-in is much less pronounced than with the LU decomposition. However, the ordering also influences the quality with which the ILU factorisation approximates the target matrix Q_i (see Section 2.1.5).

In our experiments, using no reordering was compared with the PETSc implementations of Nested Dissection (ND), One-way Dissection (1WD), Reverse Cuthill-McKee (RCM), Quotient Minimum Degree (QMD), and Approximate Minimum Degree (AMD). Of these, the AMD reordering [3] was a clear winner, being the fastest to compute while yielding the least fill-in with a full LU factorisation and the best quality ILU factorisations at the same time. Reordering methods provided in UMFPACK [12], SuperLU [17], SuperLU_DIST [32], and MUMPS [4] were also tested, but none yielded an improvement over the PETSc AMD reordering for our test problems. See Section 9.1 for an overview of the experiments that led to these conclusions.

7.3 Forcing Terms

Inexact Newton methods solve the Jacobian system in iteration i such that

$$\|\boldsymbol{F}(\boldsymbol{x}_i) + J(\boldsymbol{x}_i)\,\boldsymbol{s}_i\| \leq \eta_i \|\boldsymbol{F}(\boldsymbol{x}_i)\|, \tag{7.4}$$

where η_i is called the forcing term (see Section 3.1.1). Choosing the forcing terms correctly is very important, as discussed in detail in Chapter 4. Below, three strategies for choosing the forcing terms are discussed.

The first strategy is based on work by Dembo and Steihaug [16]:

$$\eta_i = \min\left\{\frac{1}{2^i}, \|\boldsymbol{F}_i\|\right\}. \tag{7.5}$$

This method allows for superlinear convergence when the iterate is far from the solution, switching to quadratic convergence when nearing the solution.

The second strategy is by Eisenstat and Walker [20]. This method starts with some choice of initial forcing term η_0, and for $i > 0$ sets

$$\eta_i = \left|\frac{\|\boldsymbol{F}_i\| - \|\boldsymbol{F}_{i-1} + J_{i-1}\boldsymbol{s}_{i-1}\|}{\|\boldsymbol{F}_{i-1}\|}\right|, \tag{7.6}$$

while safeguarding from oversolving by adding the rule

$$\text{if } \eta_{i-1}^{\frac{1}{2}+\frac{1}{2}\sqrt{5}} > \frac{1}{10}, \text{ then } \eta_i = \max\left\{\eta_i, \eta_{i-1}^{\frac{1}{2}+\frac{1}{2}\sqrt{5}}\right\}. \tag{7.7}$$

In our experiments $\eta_0 = 0.1$ was used.

The final strategy discussed here, is the affine contravariant strategy derived in the work of Hohmann [24]:

$$\eta_i = \frac{\varepsilon_i}{1+\varepsilon_i}, \tag{7.8}$$

where

$$\varepsilon_i = \frac{\beta}{2}\min\left\{1, h_i\right\}, \tag{7.9}$$

with

$$h_i = \begin{cases} \frac{2-\beta}{1+\beta} & i = 0, \\ \frac{1+\beta}{2(1-\varepsilon_{i-1})} h_{i-1}^2 & i > 0. \end{cases} \tag{7.10}$$

For our experiments we used $\beta = 1$. Note that the forcing terms of this strategy—unlike the other two strategies—do not depend on the problem. This strategy was also applied to power flow problems in [14].

Some authors have used fixed forcing terms throughout all Newton iterations for power flow problems [43, 37, 22, 9]. This is generally not a good idea. If the chosen forcing terms are small, then a lot of oversolving is done in early iterations, leading to many extra GMRES iterations. If the forcing terms are large, then there is a lot of undersolving in later iterations, leading to many extra Newton iterations. And anything in between leads to both oversolving in early iterations, and undersolving in later iterations.

The strategy by Eisenstat and Walker is successfully being used in practice on many different types of problems. It also provided very good results in our power flow experiments. The method based on the work of Dembo and Steihaug also gave good results for our test cases, but generally led to undersolving. The Hohmann strategy also performed quite well, but generally yielded smaller forcing terms, resulting in oversolving. For more details see Section 9.2, and the numerical experiments in Section 9.3.

7.4 Speed and Scalability

This section gives an overview of the speed and scalability of our Newton-Krylov power flow solver. The numerical experiments, on which this section is based, can be found Section 9.3.

For the smaller test cases, using the LU factorisation of the J_0 target matrix led to the best results for all forcing term strategies. The Eisenstat and Walker forcing terms performed slightly better than the tested alternatives. Table 7.1 compares the resulting solution times in seconds, with those of the traditional implementation with a direct solver.

Solving a single power flow problem of these sizes is so fast, that all tested methods are acceptable. However, when using the power flow solver as part of a larger system that has to solve many power flow problems, as for example in contingency analysis, using the LU factorisation of J_0 can lead to a significant reduction of computing time.

problem	uctew001	uctew002	uctew004	uctew008
direct	0.077	0.16	0.33	0.69
LU of J_0	0.060	0.12	0.25	0.52

Table 7.1: Power flow for small test cases

For the larger test cases, any method using the LU decomposition is slow due to the bad scaling of that operation (see Section 9.1.1). The best results were attained using ILU(12) factorisations of the J_0 and Φ target matrices. Again, the Eisenstat and Walker forcing terms generally performed slightly better than the other tested methods. Figure 7.1 shows the solution time in seconds for LU and ILU(12) factorisations of the J_0 and Φ target matrices, using Eisenstat and Walker forcing terms.

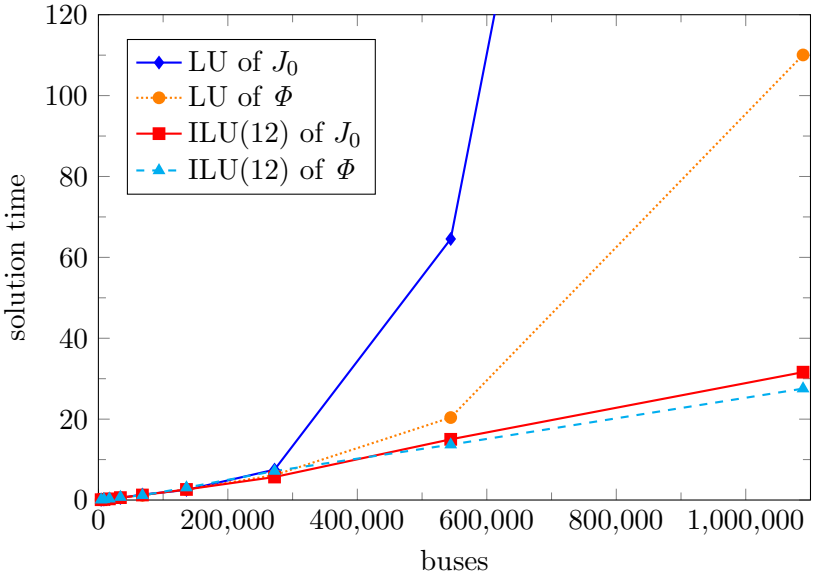

Figure 7.1: Power flow for all test cases

The bad scaling of the LU factorisation is clearly visible. It is worse for J_0 than for the Φ target matrix, because the Φ matrix consists of two independent blocks of half the dimension, see also Section 9.1.1.

The ILU(12) factorisation leads to near linear scaling. Which of the two target matrices leads to the fastest solution, using ILU(12), differs per test case. This has to do with being a bit lucky, or unlucky, at how the inexact Newton step turns out exactly for a certain problem and forcing term, much more than it has to do with the quality of the preconditioner.

7.5 Robustness

This section investigates the robustness of the Newton-Krylov power flow solver, and compares it with that of traditional Newton power flow. Convergence of exact and inexact Newton methods are generally very close. Therefore, to compare the robustness of the methods, the linear solvers should be compared. Direct linear solvers are very robust. Thus the question is how robust the used iterative linear solver is.

The convergence of Krylov solvers depends on the Krylov subspace, which is determined by the coefficient matrix and starting solution (see Section 2.2.1). The condition number of the coefficient matrix can give some indication on how fast Krylov solvers will converge. The Jacobian systems of our test cases are quite ill-conditioned. For example, the condition number of the initial Jacobian J_0 of the uctew001 test case is 1.2×10^9. This is exactly why preconditioning is such an important part of the solution process. Due to the large condition number, low quality preconditioners lead to very slow convergence. However, with the preconditioners P_i based on LU and ILU(12) factorisations, the condition numbers of the preconditioned coefficient matrices $J_i P_i^{-1}$ drop below 10, leading to very fast convergence.

To test the robustness of the methods used in this work in the context of the power flow problem, experiments were conducted on the uctew032 test case under different loading levels. Both the Newton-Raphson method with direct solver and the inexact Newton methods were able to solve the problem up to a loading level of 160%, but failed to converge at 170% without the help of line search or trust region techniques. It should be noted here that the solution of the power flow problem at the highest loading levels had such large voltage angles not to be of practical value, indicating that the solvers are well up for any practical loading levels of the power system.

Table 7.2 shows test results for the uctew032 problem at different loading levels, using the LU factorisation of J_0 as preconditioner. Presented are the number of Newton iterations N, the GMRES iterations (total amount and amount per Newton iteration), and an estimate $\tilde{\sigma}$ of the condition number of the preconditioned coefficient matrix in the last Newton iteration.

load	N	GMRES iterations	$\tilde{\sigma}$
100%	6	21 [1,3,2,3,5,7]	3.5
110%	6	21 [1,3,2,3,5,7]	3.5
120%	6	22 [1,3,2,3,5,8]	4.1
130%	7	35 [1,3,2,3,6,7,13]	4.6
140%	7	37 [1,3,2,4,5,8,14]	5.4
150%	7	35 [1,3,2,4,6,7,12]	6.8
160%	7	34 [1,3,2,4,6,5,13]	10.4

Table 7.2: Convergence at different loading levels

The solution of the problem with higher loading levels lies farther away from the flat start than with lower load. Since the preconditioner is based on the Jacobian at a flat start, the Jacobian near the solution also differs more from the preconditioner for high loading levels. As a result the condition number of the preconditioned coefficient matrix in the last Newton iteration goes up with the loading level. However, overall the condition number stays very small and GMRES convergence does not deteriorate visibly. It does take longer to solve the systems with high load, but this is due to Newton convergence suffering and not the linear solver, and is thus the same when using a direct linear solver. Similar results hold for the other preconditioners suggested in this work.

CHAPTER 8

Contingency Analysis

Secure operation of a power system requires not only that the power system operates within specified system operating conditions, but also that proper operation is maintained when contingencies occur. Contingency analysis simulates credible contingencies to analyse their impact.

Contingency analysis consists of three phases: definition, selection, and evaluation. In the definition phase, a list of contingencies that have a non-negligible chance of occurring is constructed. These contingencies mostly consist of single or multiple generator and branch outages. Then, in the selection phase, fast approximation techniques are used to cheaply identify contingency cases that will not violate system operation conditions. These contingencies can be eliminated from the list made in the definition phase. Finally, in the evaluation phase, the power flow problem for the remaining contingencies is solved, and the solution analysed. For more detailed information, see for example [47].

In this work we focus on the evaluation phase. In particular we focus on using the Newton-Krylov power flow solver to speed up the calculation of many closely related power flow problems. Speeding up consecutive solves generally involves reusing information from earlier solves. The information that can be reused in contingency analysis, that is not available in traditional Newton power flow, is the preconditioner. We show that this can lead to a significant reduction of the computational time.

The methodology proposed in this chapter for branch outages in contingency analysis, can be used whenever power flow problems have to be solved that only differ slightly from each other. This includes other outages, Monte Carlo simulations, and optimization problems, but also handling reactive power limits of generators and tap changing transformers. Similar techniques were proposed for contingency screening in [2].

8.1 Simulating Branch Outages

A branch outage can be simulated by removing the branch from the power system model, and then solving the associated power flow problem normally. In contingency analysis, this leads to a large amount of power flow problems to be solved. Solving all these problems can take a huge computational effort. Therefore, it is important to look at ways to speed up this process, beyond the efforts of speeding up the power flow solver itself. The general methodology here, is to treat the branch outage as an update of the base case—that is, the case without any outages—instead of treating it as a separate problem.

A simple improvement that can be made, is to update the admittance matrix Y, instead of recalculating it. Let i and j be the buses that the removed branch used to connect. Then only the values y_{ii}, y_{ij}, y_{ji}, and y_{jj} in the admittance matrix need to be updated.

Since the power system with branch outage very closely resembles the base case, it seems logical to assume to solutions of the associated power flow problems are relatively close to each other. Therefore, instead of using a flat start, using the solution of the base case as initial iterate for the contingency case can be expected to significantly reduce the number of Newton iterations needed to converge.

The above two techniques can be equally exploited in classical Newton power flow, Fast Decoupled Load Flow, and Newton-Krylov power flow. Fast Decoupled Load Flow further allows the reuse of the factorisation of the base case coefficient matrices B' and B'' for the contingency cases, either through updates of the factors, or compensation [48, 1, 53].

Newton-Krylov power flow offers some extra options in the form of preconditioning and forcing terms. A preconditioner based on the base case can be used for all contingency cases, eliminating the need to perform a factorisation for each contingency. This preconditioner will generally not be as good for the contingency cases as one derived from the contingency cases themselves, but the resulting extra GMRES iterations are relatively cheap. The techniques used to update the coefficient matrices for simulating branch outages using Fast Decoupled Load Flow, can also be used to update the preconditioner for each contingency case. The convergence of the base case may be analysed, to derive an educated guess of forcing terms for the contingency cases.

Numerical experiments, simulating branch outages using Newton-Krylov power flow, can be found in Section 9.4. These experiments focus on reusing the LU factorisation of some Jacobian matrix as preconditioner. Updates of the factorisation, as used in FDLF, have not been tested. Analysis of the forcing terms yielded only minor improvements over the Eisenstat and Walker forcing terms, as these forcing terms already adapt to the problem very well.

Figure 8.1 gives an overview of the results for the UCTE winter 2008 study model test case, using Eisenstat and Walker forcing terms. PCSetUp is the time spent on LU factorisations, PCApply is the time spent on forward and backward substitutions, and KSPRest is the remaining time spent on linear solves. CalcJac stand for the the calculation of the Jacobian system, i.e., the power mismatch and the Jacobian matrix, and CARest is whatever computing time remains.

The left three bars represent methods using a flat start, while the right three bars use the solution of the base case as initial iterate for the contingency cases. The methods used are:

A : Newton power flow with a direct linear solver,
B : Newton-GMRES with the contingency cases preconditioned with their own initial Jacobian,
C : Newton-GMRES with the contingency cases preconditioned with the base case Jacobian evaluated in the vector that is used as initial iterate for the contingency cases.

Classical Newton power flow (A) is chosen here to serve as a reference. Method B is chosen because it proved the fastest for a single UCTE test case in Section 9.3. Finally, method C illustrates the benefits of using a single preconditioner for all contingency cases.

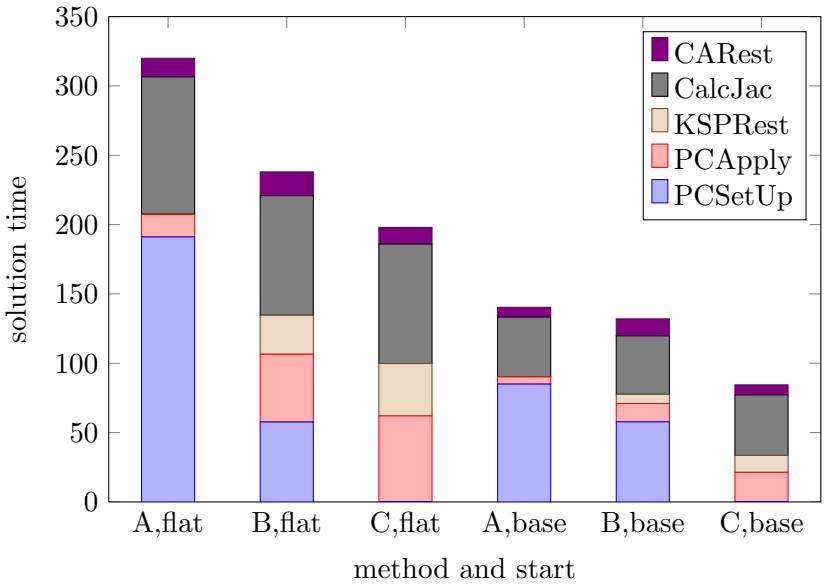

Figure 8.1: Contingency analysis

With the classical Newton power flow, most of the computational effort goes into the LU factorisation of the Jacobian matrices (PCSetUp). Such a factorisation has to be made in every Newton iteration. The rest of the linear solution time consists of a single forward-backward substitution per Newton iteration (PCApply). With Newton-GMRES, preconditioned with the initial Jacobian of the case that is being solved, only one LU factorisation has to be made per contingency case, at the cost of some other computational effort provided by the GMRES iterations. When preconditioning Newton-GMRES with the base case Jacobian, the time spent on LU factorisations becomes negligible, at the cost of some extra GMRES iterations.

Starting with the solution of the base case has a clear advantage over a flat start for this test case. Many fewer Newton iterations are needed when using the base case solution as initial iterate, leading to a significant computational speed-up.

The Newton-Krylov method B outperforms classical Newton power flow. However, the difference is much less distinct when starting from the base case solution than when using a flat start. Because there are less Newton iterations needed per contingency case, the benefit of doing only a single LU factorisation per case is less pronounced. Also, because the solution of the base case provides a much better initial iterate the Newton process is started closer to the solution, and the direct solver is doing less oversolving in the initial steps than when using a flat start. This further lessens the advantage of the inexact Newton method over Newton with a direct solver.

The best results are attained using the base case solution as start, and the base case Jacobian in that solution as preconditioner, for the contingency cases (C,base). This method is about 3.8 times faster than classical Newton power flow with a flat start, and 1.7 times faster than classical Newton power flow started with the solution of the base case. Note that the methods only differ in the linear solver. Looking purely at the time spent in the linear solver, this method is 6.2 and 2.7 times faster than classical Newton power flow with a flat start, and with the base case solution start, respectively.

8.2 Other Simulations with Uncertainty

The methodology described in Section 8.1 can also be used in many other power flow simulations that include some element of uncertainty. This includes optimisation problems and Monte Carlo simulations, but also the handling of tap transformers and reactive power limits of generators. Two examples are outlined below.

The most obvious example is dealing with uncertainty in load. This is a very topical problem, as wind turbines—which have a high uncertainty in the amount of generated power—are often modeled as negative loads. Using Monte Carlo simulations to handle these stochastic factors requires

the solution of a lot of power flow problems with different load values, but the same network topology. Since the load values do not influence the Jacobian matrix, all these power flow problems have the same initial Jacobian matrix when started with the same initial iterate. As such, that Jacobian matrix can be expected to be a very good preconditioner for all problems within the Monte Carlo simulation.

The other example is bus-type switching, due to the violation of reactive power limits of generators. Again, the preconditioner of the base case can be used to solve the derived case. However, this does require an implementation of the power flow solver, in which bus-type switching does not lead to a change in the dimension of the linear system (see Section 6.1.3).

CHAPTER 9

Numerical Experiments

In this chapter, numerical experiments with our Newton-Krylov power flow solver are discussed. Section 9.1 treats experiments with the LU and ILU(k) factorisation, using different reordering methods, to determine the relevant options. Section 9.2 analyses some experiments with different forcing term strategies. Section 9.3 discusses experiments with the power flow solver for all target matrices and forcing terms. Finally, section 9.4 treats contingency analysis experiments.

The implementation of our solver is done in C++ using PETSc (Portable, Extensible Toolkit for Scientific Computation) [6], a state of the art, award winning C library for scientific computing. PETSc can be used to produce both sequential programs, and programs running in parallel on multiple processors.

All experiments are performed on a single core of a machine with Intel Core2 Duo 3GHz CPU and 16Gb memory, running a Slackware 13 64-bit Linux distribution. For info on the used test set of power flow problems see Appendix B.

9.1 Factorisation

This section discusses numerical experiments with LU and ILU(k) factorisations, and different row and column ordering methods. In Section 9.1.1 different ordering methods are tested for the LU decomposition, and the scaling of this factorisation is investigated. Section 9.1.2 treats the impact of matrix ordering on the ILU(k) factorisation, and tests the speed and scaling for different levels k.

All experiments conducted in this section consist of solving the Jacobian system of the first Newton iteration, $J_0 \boldsymbol{s}_0 = \boldsymbol{F}_0$, using preconditioned GMRES up to an accuracy of 10^{-5}. The scaling experiments solve all test cases. The other experiments all use the uctew032 problem, but similar results were obtained for all test cases.

The discussion of the experiments includes computation times spend on the relevant PETSc functions. See Table 9.1 for an explanation of these PETSc functions. All computational times are measured in seconds. Further, the notation $\mathrm{nnz}\,(L+U)$ is used for the number of nonzeros in the factors L and U combined, and the term fill ratio is used for the number of nonzeros in the factors divided by the number of nonzeros in the target matrix.

MatGetOrdering	:	matrix reordering
MatLUFactorSym	:	symbolic LU factorisation
MatILUFactorSym	:	symbolic ILU factorisation
MatLUFactorNum	:	numeric factorisation for LU or ILU
PCSetUp	:	reordering and factorisation
PCApply	:	forward and backward substitution
KSPSolve	:	linear solve

Table 9.1: PETSc functions

9.1.1 LU Factorisation

Table 9.2 shows the results of the direct solution of the first Jacobian system of the uctew032 test case. The direct solver of PETSc was tested together with the provided matrix ordering methods. PETSc was also used to call the UMFPACK, MUMPS, SuperLU, and SuperLU_Dist solvers, with their respective ordering methods. Note that for the SuperLU and SuperLU_Dist packages the natural row ordering was used, as the alternative yielded no improvement. Further note that PETSc, UMFPACK, and SuperLU only provide sequential implementations of the LU factorisation. If parallel computing is desired, MUMPS and SuperLU_Dist can be used. For details on the tested ordering methods, see the manuals of the respective packages.

package	PETSc						SuperLU			
ordering	none	ND	1WD	RCM	QMD	AMD	none	MMD A^TA	MMD A^T+A	COLAMD
MatGetOrdering	0.01	0.20	0.04	0.05	5.91	0.09	0.20	0.20	0.20	0.20
MatLUFactorSym	1.63	2.52	7.56	7.56	0.18	0.13	0	0	0	0
MatLUFactorNum	31.92	287.44	925.83	922.26	0.56	0.31	32.83	1.79	3.98	1.37
PCSetUp	33.56	290.16	933.43	929.87	6.65	0.53	33.03	1.99	4.18	1.57
PCApply	0.22	0.35	1.02	1.03	0.02	0.02	0.29	0.06	0.09	0.06
KSPSolve	33.78	290.51	934.45	930.90	6.67	0.55	33.32	2.05	4.27	1.63
nnz $(L+U)$	70.4M	110M	328M	328M	5.28M	4.67M	71.8M	8.21M	15.4M	8.46M
fill ratio	35.03	54.89	163.10	163.26	2.63	2.32	35.72	4.08	7.68	4.21

package	UMFPACK		MUMPS					SuperLU_Dist		
ordering	AMD	A^T+A	AMD	AMF	PORD	METIS	QAMD	none	MMD A^TA	MMD A^T+A
MatGetOrdering	0.20		0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
MatLUFactorSym	0.48		0	0	0	0	0	0	0	0
MatLUFactorNum	0.90		1.26	1.21	2.07	1.90	1.27	84.07	2.23	1.27
PCSetUp	1.59		1.46	1.41	2.27	2.11	1.48	84.28	2.44	1.48
PCApply	0.03		0.12	0.12	0.12	0.12	0.13	0.66	0.17	0.13
KSPSolve	1.62		1.59	1.54	2.40	2.23	1.60	84.94	2.61	1.61
nnz $(L+U)$	4.63M		4.91M	4.64M	5.02M	5.56M	4.91M	70.8M	8.09M	5.26M
fill ratio	2.30		2.44	2.31	2.50	2.77	2.44	35.24	4.02	2.61

Table 9.2: Direct linear solve using different solver packages and ordering methods

From the fill ratio, it is clear that AMD (and related methods) provide the best reordering in terms of fill-in. Using such a method reduces the fill-in from a factor 35, to around 2.3. Some of the methods lead to a fill ratio much worse than that for the original ordering. These methods generally expect a much more structured matrix, as arises for example from the discretisation of differential equations on structured grids.

In terms of computational time, the PETSc solver with AMD ordering performed the best. The difference with AMD ordering of other packages may well be due to the overhead of calling that package from PETSc. It is possible to use external packages for the matrix reordering only, and then solve the problem with the PETSc solver. However, with the AMD reordering of PETSc leading to such good results, there is no need to do so.

Figure 9.1 shows the factorisation time for the initial Jacobian matrix and the FDLF matrix Φ of the uctew032 test case. The bad scaling of the LU decomposition is clearly visible. Recall from Section 7.2.1, that the matrix Φ consists of two independent blocks with each about a quarter of the nonzeros found in the Jacobian matrix. These blocks—which have half the dimension of the Jacobian matrix—can be factorised independently. As a result, the LU decomposition of Φ scales similar to that of the Jacobian matrix of a problem of half the size.

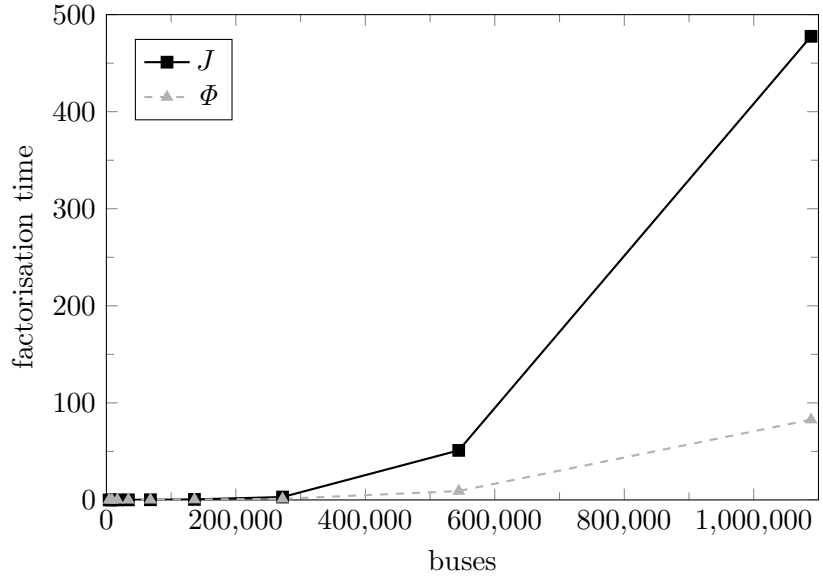

Figure 9.1: LU factorisation of J and Φ

9.1.2 ILU Factorisation

Table 9.3 shows the results of numerical experiments on the effect of matrix reordering on the ILU(k) factorisation method. The initial Jacobian system of the uctew032 test case was solved using GMRES, preconditioned with an ILU(8) factorisation of the coefficient matrix, using different ordering methods.

ordering	none	ND	1WD	RCM	QMD	AMD
MatGetOrdering	0.01	0.20	0.04	0.05	5.90	0.09
MatILUFactorSym	1.81	0.69	0.45	0.46	0.27	0.21
MatLUFactorNum	0.94	0.41	0.25	0.25	0.18	0.12
PCSetUp	2.76	1.31	0.74	0.77	6.36	0.42
PCApply	0.58	0.42	0.31	0.32	0.22	0.16
KSPSolve	3.54	2.01	1.29	1.32	6.76	0.74
GMRES iterations	12	15	13	13	11	10
nnz $(L+U)$	15.0M	7.62M	5.84M	5.96M	3.71M	3.56M
fill ratio	7.47	3.79	2.90	2.97	1.85	1.77

Table 9.3: Linear solve with ILU(8) and different orderings

The fill ratio clearly illustrates that the ordering method influences the fill-in of the ILU(k) factorisation much less drastically than it did for the LU decomposition. Further note that all the tested reorderings improve the fill-in compared to the natural ordering, whereas with the LU factorisation some ordering methods led to more fill-in. However, the ordering methods that led to worse fill-in here need more GMRES iterations to converge. This indicates that the ILU(8) preconditioner is of lesser quality for these ordering methods.

The AMD ordering again performs the best. It leads to the lowest fill ratio and the lowest amount of GMRES iterations. Thus it produces the highest quality preconditioner, with the least amount of nonzeros. Furthermore, both the calculation and the application of the AMD reordered preconditioner are faster than any of the alternatives, leading to the fastest overall solution time.

Table 9.4 holds the results of numerical experiments with different ILU levels. Again, the initial Jacobian system of the uctew032 test case was solved using ILU(k) preconditioned GMRES. The AMD reordering was used in all cases because it gave the best results, as was illustrated above for ILU(8). The last column holds the data of a direct solve for comparison.

Using less than 4 levels leads to a preconditioner of too low quality. The factorisation is fast and the fill ratio is low, but due to the amount of GMRES iterations needed the solution time is much higher than with more levels.

Using more than 16 levels leads to a very high quality preconditioner. With 64 levels the factorisation even becomes the same as the LU decomposition. Few GMRES iterations are needed to solve the linear problem, but the factorisation takes more time and the fill ratio is larger, making the overall solution time higher than with 4 to 16 levels.

factorisation	ILU(0)	ILU(2)	ILU(4)	ILU(8)
Mat(I)LUFactorSym	0.08	0.14	0.17	0.21
MatLUFactorNum	0.08	0.10	0.11	0.12
PCSetUp	0.24	0.33	0.37	0.42
PCApply	2.30	0.71	0.35	0.16
KSPSolve	24.94	2.94	1.28	0.74
GMRES iterations	215	55	25	10
$\mathrm{nnz}\,(L+U)$	2.01M	2.85M	3.21M	3.56M
fill ratio	1	1.42	1.60	1.77

factorization	ILU(16)	ILU(32)	ILU(64)	LU
Mat(I)LUFactorSym	0.26	0.47	0.85	0.31
MatLUFactorNum	0.14	0.26	0.31	0.13
PCSetUp	0.50	0.81	1.25	0.53
PCApply	0.09	0.07	0.04	0.04
KSPSolve	0.67	0.94	1.31	0.59
GMRES iterations	5	3	1	1
$\mathrm{nnz}\,(L+U)$	3.93M	4.52M	4.67M	4.67M
fill ratio	1.96	2.25	2.32	2.32

Table 9.4: Linear solve with different factorisations using AMD

Figure 9.2 show the scaling behaviour of different ILU(k) levels. The ILU(2) and ILU(8) factorisations both scale very well in the problem size, but ILU(8) is approximately twice as fast as ILU(2). Higher ILU levels start to lose the linear scaling behaviour, as illustrated by the ILU(32) graph, which is nearing that of the LU factorisation.

Figure 9.3 inspects the scaling of ILU factorisations with around 8 levels. ILU(4), ILU(8), and ILU(12) all scale approximately linearly, with ILU(4) being slightly slower than the other two. The graphs of ILU(8) and ILU(12) are practically on top of each other. ILU(16) is still very fast, but no longer scales linearly. Therefore, in the remainder of this chapter only 4, 8, and 12 levels of ILU factorisations are considered.

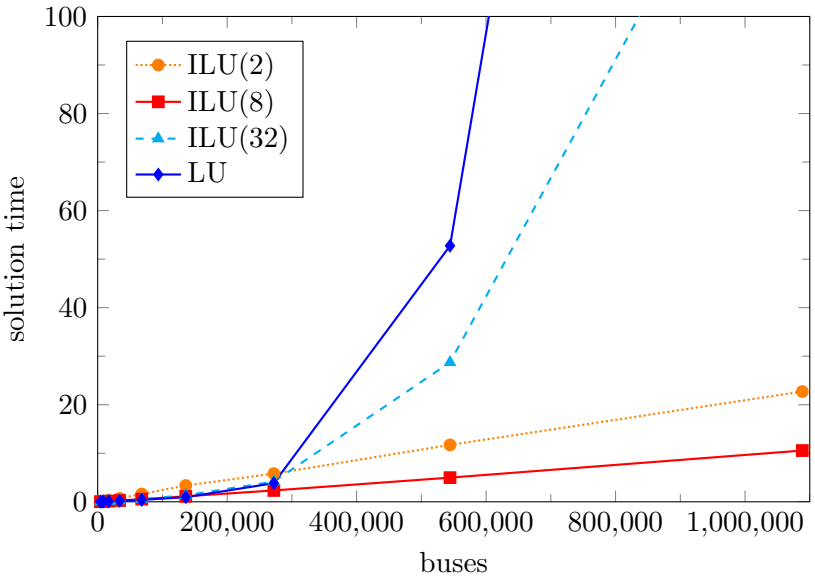

Figure 9.2: Linear solve for different problem sizes

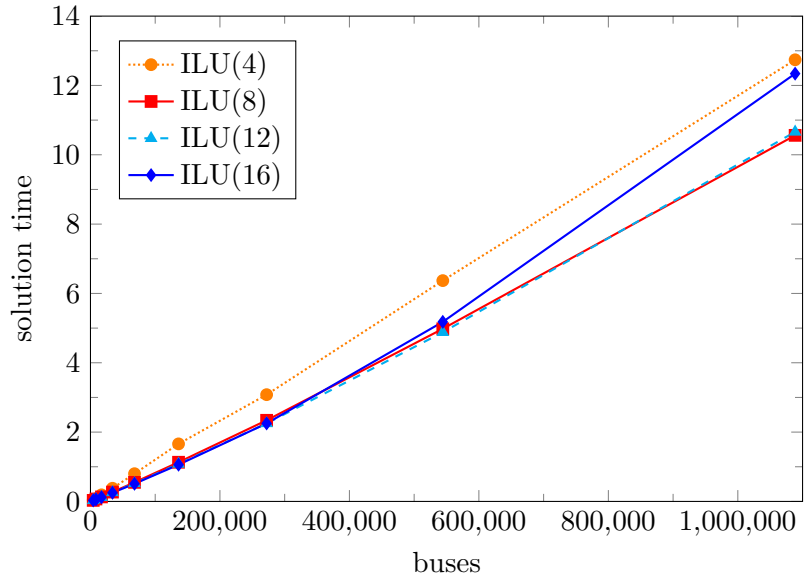

Figure 9.3: Linear solve for different problem sizes

9.2 Forcing Terms

This section takes a look at the general behaviour of the three forcing term strategies presented in Section 7.3. Their performance is illustrated by analysing a representative case, solved with each of the forcing term strategies. The uctew032 test problem is solved using GMRES, preconditioned with the ILU(8) factorisation of the FDLF matrix Φ.

Figures 9.4, 9.5, and 9.6 show the resulting nonlinear convergence in the total number of GMRES iterations, for Dembo and Steihaug, Eisenstat and Walker, and Hohmann forcing terms respectively. The legend of these figures is explained in Table 9.5. If the forcing residual mark is below the actual residual, then the method is oversolving in that iteration. If the best residual mark is below the actual residual, then it is undersolving. Note that when there is no oversolving, the forcing residual mark is mostly on top of the actual residual norm, as expected from the theory in Chapter 4.

actual	:	actual nonlinear residual norm in each Newton iteration
forcing	:	nonlinear residual norm resulting from multiplying the previous nonlinear residual norm by the forcing term used in the current Newton iteration
best	:	nonlinear residual norm resulting from doing a full accuracy linear solve in the current Newton iteration

Table 9.5: Nonlinear residual norm legend

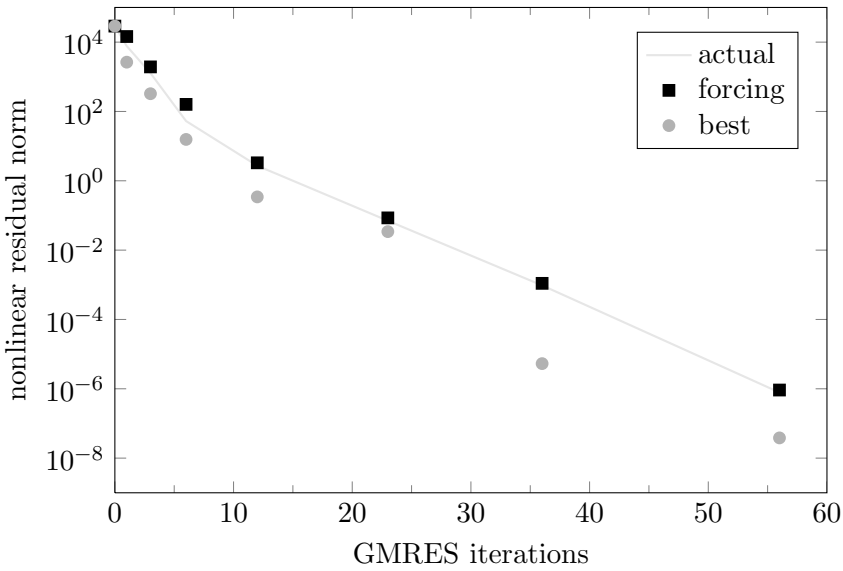

Figure 9.4: Dembo and Steihaug forcing terms

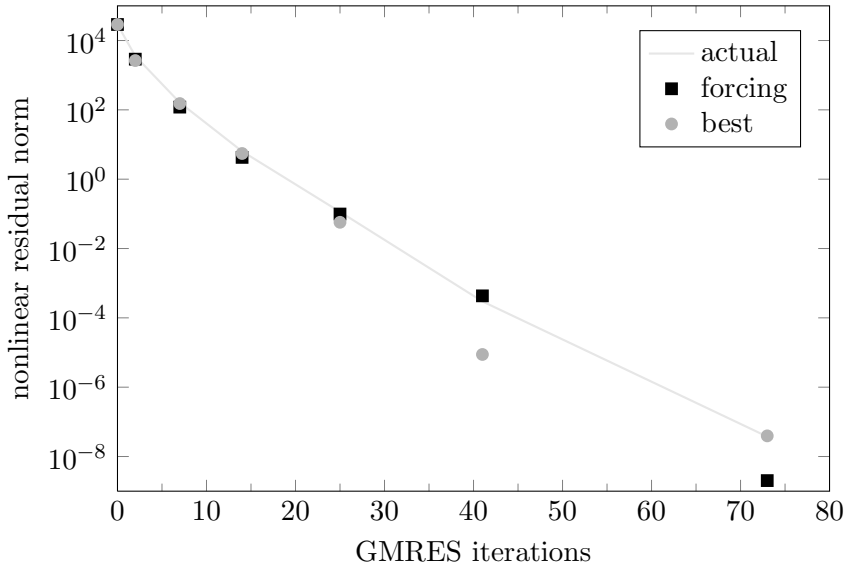

Figure 9.5: Eisenstat and Walker forcing terms

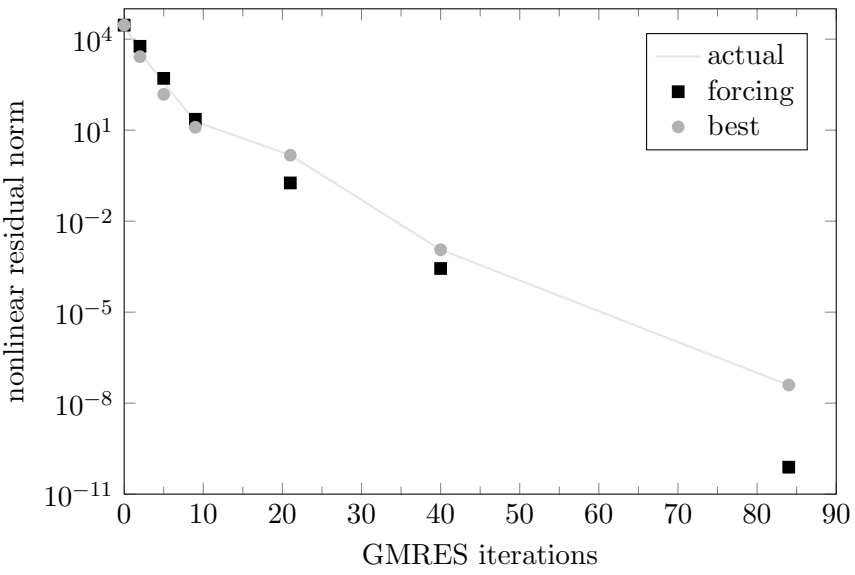

Figure 9.6: Hohmann forcing terms

The following conclusions can be drawn from the above experiments. The Dembo and Steihaug forcing terms lead to undersolving. The Eisenstat and Walker forcing terms are spot-on for the first 4 Newton iterations, while there is some undersolving in iteration 5, and oversolving in iteration 6. The Hohmann forcing terms lead to oversolving in the later Newton iterations. These conclusions are consistent with the general behaviour of these forcing terms strategies in our power flow experiments, see Section 9.3.

The Eisenstat and Walker strategy includes the nonlinear residual norm of the current and previous Newton iterations, and the latest linear residual norm in its calculation; all the ingredients needed to determine whether there was undersolving or oversolving in the previous iteration. Using this information, the method tends to correct itself. If there is a lot of oversolving in the previous iteration, then the forcing term will be chosen larger, often leading to some undersolving, and vice versa.

9.3 Power Flow

This section reports on numerical experiments with the inexact Newton-Krylov power flow solver, solving the full nonlinear problem. In each Newton iteration, the linear system is solved using preconditioned GMRES. LU, ILU(4), ILU(8), and ILU(12) factorisations of the target matrices J_i, J_0, and Φ are all tested as preconditioner. The problems are solved from a flat start, up to an accuracy of 10^{-6} p.u.

Tables 9.6, 9.7, and 9.8 shows the results for Dembo and Steihaug forcing terms, Eisenstat and Walker forcing terms, and Hohmann forcing terms respectively (see Section 7.3). For each of the experiments, the number of Newton iterations p and GMRES iterations q are given in the format p/q, and the solution time in seconds. In each table, the number of Newton iterations and solution time when using a direct solver are added as a reference.

For the smaller test cases, the best results were generally attained using an LU decomposition of J_0 as preconditioner. For the largest problems, the LU decomposition becomes too slow due to the bad scaling, as demonstrated in Section 9.1.1. For these cases, ILU(12) factorisations of J_i and Φ gave the best results.

The Dembo and Steihaug forcing terms generally led to undersolving, as can be seen from the higher amount of Newton iterations needed. The Hohmann forcing terms on the other hand mostly led to a minimal amount of Newton iterations, but higher amounts of GMRES iterations, indicating oversolving. The Eisenstat and Walker forcing terms usually showed behaviour somewhere in between the other two strategies. For the largest test cases, the Hohmann forcing terms sometimes were smaller than machine precision allowed. Some form of safeguarding would be needed to catch such cases.

For each individual test case, the smallest solution time was attained using some J_i or Φ based preconditioner together with the Eisenstat and Walker forcing terms. These best solution times are marked with a grey background in the tables. Note that in all the cases where a preconditioner based on Φ gave the best result, this is due to the J_i based alternative needing one or two extra Newton iterations.

9.3.1 Scaling

The number of Newton iterations needed to solve the problem generally did not increase much when the problem size increased. For some combinations of preconditioner and forcing terms the largest problems required a few more iterations, but for other combinations the number of Newton iterations stayed constant. This suggests that any increased number of iterations are more due to getting a bit unlucky with the Newton steps, than it being a fundamental result of the increased problem size.

Similarly, for each combination of preconditioner and forcing terms, the total number of GMRES iterations was fairly constant for increasing problem sizes. Whenever a significantly higher amount of GMRES iterations was needed, it was generally due to an extra Newton iteration being used.

The basic operations of the Newton-Krylov power flow solver, are all vector operations and operations on the Jacobian matrix. A larger power system has more buses, but generally not more connections per bus. As a result, the number of nonzeros per row in the Jacobian matrix do not grow for larger problems. Thus the total number of nonzeros scales linearly in the problem size, and so does the computation time of operations on the Jacobian matrix like the calculation of that matrix and the multiplication with a vector. The exception is the factorisation operation, see Section 9.1.

When the number of Newton and GMRES iterations are constant in the problem size, the scaling of the Newton-Krylov power flow solver can therefore be expected to be linear if the factorisation scales linearly. If the factorisation scales badly, as is the case for the LU decomposition, the power flow solver also scales badly.

Figures 9.7, 9.8, and 9.9 show the scaling behaviour of the solution time using different factorisations of, respectively, J_i, J_0, and Φ as preconditioner. Indeed, the solver exhibits approximately linear scaling when using ILU(k) factorisations with 4 to 8 levels, which were shown to scale linearly up to a million buses in Section 9.1.2. The LU factorisation leads to bad scaling, as expected from the results of Section 9.1.1.

power flow problem			uctew001	uctew002	uctew004	uctew008	uctew016	uctew032	uctew064	uctew128	uctew256
direct		iterations	7	7	7	7	7	7	7	6	8
		time	0.077	0.16	0.33	0.69	1.54	3.96	20.89	305.41	3810.80
J_i	ILU(4)	iterations	7/76	7/79	7/75	7/79	7/80	7/77	7/72	8/108	8/107
		time	0.12	0.26	0.53	1.14	2.40	4.81	9.43	27.09	54.91
J_i	ILU(8)	iterations	7/43	7/47	7/38	7/42	7/39	7/39	7/39	8/46	9/49
		time	0.10	0.21	0.42	0.89	1.76	3.62	7.42	18.43	42.78
J_i	ILU(12)	iterations	7/24	7/28	7/27	7/27	7/26	7/24	7/24	7/22	9/32
		time	0.089	0.19	0.39	0.81	1.64	3.27	6.75	15.07	42.25
J_0	LU	iterations	7/23	7/22	7/22	7/19	7/21	7/23	7/28	8/40	9/47
		time	0.068	0.14	0.28	0.57	1.21	2.70	7.89	65.13	517.90
J_0	ILU(4)	iterations	7/83	7/84	7/81	7/79	7/82	8/136	8/118	8/128	8/135
		time	0.11	0.23	0.49	1.01	2.14	7.08	12.56	27.73	58.90
J_0	ILU(8)	iterations	7/40	7/38	7/35	7/35	8/58	8/55	8/68	8/72	9/88
		time	0.082	0.16	0.33	0.68	1.84	3.66	8.51	18.03	42.93
J_0	ILU(12)	iterations	7/27	7/27	7/24	7/24	7/24	8/36	7/32	8/56	9/65
		time	0.073	0.15	0.30	0.62	1.25	3.08	5.75	16.09	36.87
Φ	LU	iterations	7/34	7/33	7/33	7/33	7/35	7/33	7/35	8/59	8/56
		time	0.081	0.13	0.28	0.58	1.22	2.54	5.84	23.22	113.18
Φ	ILU(4)	iterations	7/107	7/101	8/160	8/161	8/164	8/162	8/142	8/146	8/178
		time	0.12	0.23	0.78	1.68	3.68	7.77	13.76	29.29	73.77
Φ	ILU(8)	iterations	7/61	7/54	7/50	7/52	7/56	7/55	8/93	8/95	8/94
		time	0.083	0.16	0.32	0.70	1.49	3.11	9.65	20.08	40.20
Φ	ILU(12)	iterations	7/44	7/40	7/40	7/41	7/44	7/45	7/47	8/60	8/69
		time	0.073	0.14	0.30	0.63	1.32	2.81	5.97	14.57	32.57

Table 9.6: Power flow experiments using the Dembo and Steihaug forcing terms

power flow problem			uctew001	uctew002	uctew004	uctew008	uctew016	uctew032	uctew064	uctew128	uctew256
direct		iterations	7	7	7	7	7	7	7	6	8
		time	0.077	0.16	0.33	0.69	1.54	3.96	20.89	305.41	3810.80
J_i	ILU(4)	iterations	7/102	7/113	7/116	7/123	7/128	7/127	7/117	7/115	7/114
		time	0.15	0.34	0.74	1.65	3.56	7.34	13.87	28.10	57.74
J_i	ILU(8)	iterations	7/44	7/44	7/53	7/50	7/50	7/50	7/48	7/40	9/50
		time	0.10	0.21	0.48	0.96	1.97	4.04	8.15	16.55	43.42
J_i	ILU(12)	iterations	7/25	7/28	7/27	7/31	7/32	7/32	7/33	7/29	8/22
		time	0.090	0.19	0.39	0.84	1.75	3.56	7.45	16.18	36.57
J_0	LU	iterations	6/18	6/18	6/18	6/18	7/29	6/21	6/26	7/40	8/37
		time	0.060	0.12	0.25	0.52	1.34	2.50	7.51	64.55	511.08
J_0	ILU(4)	iterations	7/112	7/121	7/126	7/128	7/141	7/167	7/139	7/150	7/165
		time	0.14	0.33	0.72	1.54	3.55	8.95	15.30	33.16	76.14
J_0	ILU(8)	iterations	7/53	7/59	7/55	7/56	7/61	7/63	7/73	7/73	8/67
		time	0.092	0.20	0.41	0.86	1.83	3.88	8.73	18.16	35.78
J_0	ILU(12)	iterations	7/39	7/43	6/26	6/26	6/28	6/30	6/35	7/52	8/50
		time	0.082	0.18	0.29	0.60	1.24	2.61	5.69	15.01	31.61
Φ	LU	iterations	6/35	6/35	6/35	6/36	6/37	6/40	6/44	6/49	6/54
		time	0.063	0.13	0.26	0.56	1.18	2.62	6.16	20.39	110.05
Φ	ILU(4)	iterations	6/119	6/115	6/121	6/137	6/143	6/141	6/135	6/145	6/152
		time	0.12	0.24	0.57	1.40	3.17	6.76	13.36	29.42	62.61
Φ	ILU(8)	iterations	6/68	6/71	6/64	6/70	6/70	6/75	6/89	6/90	6/91
		time	0.085	0.18	0.35	0.79	1.66	3.67	9.07	18.79	37.58
Φ	ILU(12)	iterations	6/55	6/55	6/54	6/60	5/41	6/58	6/68	6/62	6/61
		time	0.076	0.15	0.32	0.72	1.15	3.07	7.16	13.69	27.54

Table 9.7: Power flow experiments using the Eisenstat and Walker forcing terms

power flow problem			uctew001	uctew002	uctew004	uctew008	uctew016	uctew032	uctew064	uctew128	uctew256
direct		iterations	7	7	7	7	7	7	7	6	8
		time	0.077	0.16	0.33	0.69	1.54	3.96	20.89	305.41	3810.80
J_i	ILU(4)	iterations	6/95	6/104	6/102	6/107	6/107	6/107	6/96	6/95	6/95
		time	0.13	0.31	0.65	1.45	3.06	6.35	11.84	24.43	49.69
J_i	ILU(8)	iterations	6/42	6/46	6/45	6/47	6/48	6/48	6/48	6/42	7/434
		time	0.093	0.20	0.41	0.87	1.82	3.72	7.68	15.53	719.50
J_i	ILU(12)	iterations	6/26	6/28	6/28	6/29	6/30	6/30	6/30	6/27	7/423
		time	0.083	0.17	0.36	0.76	1.58	3.21	6.62	14.40	722.93
J_0	LU	iteration	6/26	6/26	6/24	6/27	6/27	6/29	6/34	7/437	7/441
		time	0.065	0.13	0.27	0.59	1.23	2.76	8.14	425.69	1297.50
J_0	ILU(4)	iterations	6/94	6/104	6/101	6/107	6/110	6/116	6/118	6/126	6/136
		time	0.12	0.28	0.60	1.34	2.96	6.44	13.23	29.28	64.69
J_0	ILU(8)	iterations	6/50	6/54	6/52	6/53	6/58	6/58	6/63	6/65	7/467
		time	0.085	0.18	0.38	0.79	1.76	3.57	7.80	16.43	720.85
J_0	ILU(12)	iterations	6/36	6/38	6/35	6/35	6/39	6/40	6/47	6/50	7/454
		time	0.075	0.16	0.32	0.66	1.43	2.95	6.57	14.16	721.61
Φ	LU	iterations	6/49	6/49	6/49	6/49	6/51	6/53	6/54	6/59	6/65
		time	0.071	0.14	0.31	0.64	1.41	3.10	7.05	22.35	115.54
Φ	ILU(4)	iterations	6/129	6/145	6/146	6/152	6/155	6/162	6/153	6/164	6/173
		time	0.14	0.35	0.75	1.69	3.78	8.69	16.85	37.80	82.07
Φ	ILU(8)	iterations	6/70	6/73	6/72	6/73	6/74	6/78	6/82	6/90	6/95
		time	0.086	0.18	0.39	0.83	1.80	3.99	8.63	19.27	41.22
Φ	ILU(12)	iterations	6/56	6/58	6/55	6/53	6/55	6/62	6/67	6/72	6/78
		time	0.077	0.16	0.33	0.67	1.46	3.34	7.34	15.87	34.71

Table 9.8: Power flow experiments using the Hohmann forcing terms

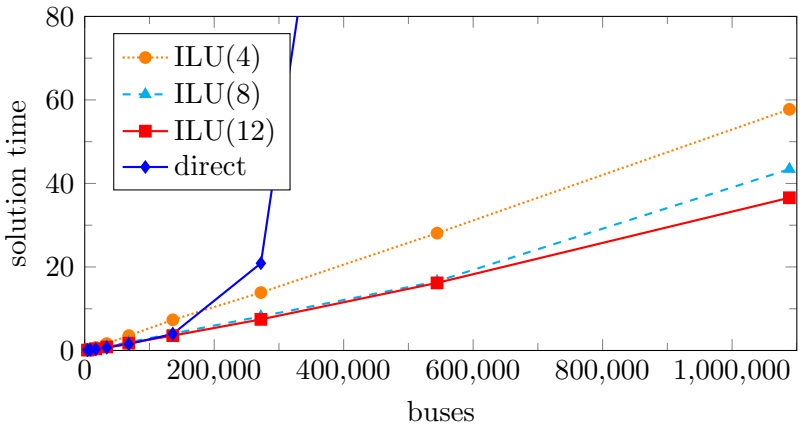

Figure 9.7: Power flow with J_i based preconditioning

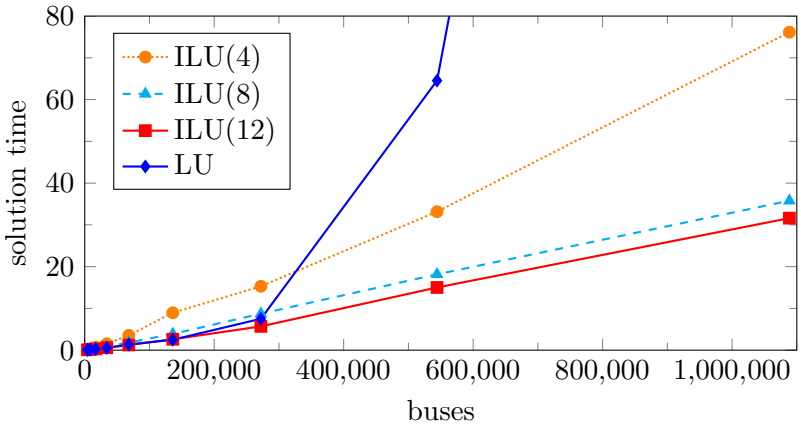

Figure 9.8: Power flow with J_0 based preconditioning

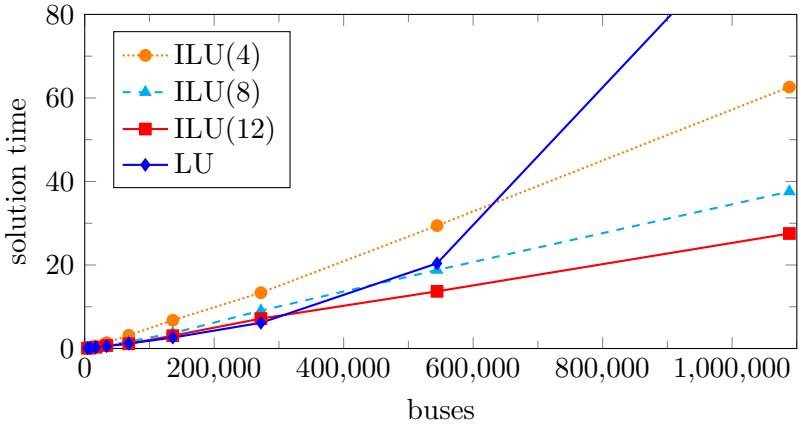

Figure 9.9: Power flow with Φ based preconditioning

9.4 Contingency Analysis

This section treats numerical experiments with the Newton-Krylov power flow solver, applied to the contingency analysis problem (see Section 5.4 and Chapter 8). The UCTE winter 2008 study model is used as base case. The contingency cases consist of the base case with a single pair of buses (that were connected in the base case) disconnected, simulating branch outages. This constitutes 6784 contingency cases, of which 95 cases could not be solved because they had two disconnected subnetworks, leaving 6689 contingency cases. The base case power flow problem is solved first, after which the power flow problem of each of the contingency cases is solved, making a total of 6690 power flow solves.

Table 9.9 presents the results using Eisenstat and Walker forcing terms. A maximum of 12 Newton iterations was allowed per case, and no line search was used to keep results as clean as possible. All cases were solved up to an accuracy of 10^{-4} p.u., and all times are measured in seconds.

The top half uses a flat start for all cases, while the bottom half solves the base case using a flat start, and then uses the solution of the base case as initial iterate for the contingency cases.

The left column shows the results using classical Newton power flow with a direct linear solver. The middle column solves each case with Newton-GMRES, preconditioned with the LU factorisation of the initial Jacobian of that case, which proved the fastest option for the base case in Section 9.3. The right column again uses the Newton-Krylov solver, but preconditions GMRES with the LU factorisation of the base case Jacobian evaluated in the vector that is used as initial solution for the contingency cases. With a flat start, the contingency cases are thus preconditioned with the initial Jacobian J_0 of the base case. And when starting with the solution of the base case, they are preconditioned with the base case Jacobian in the base case solution, denoted by J^*.

The converged and diverged rows, show the number of contingency cases that converged and diverged respectively, and the average amount of nonlinear iterations and linear iterations per case. A total of 23 contingency cases could not be solved by the Newton method. This is a common problem in contingency analysis, that we will not go into further here. One case was solved with some methods, but failed to converge with others.

For an explanation of PCSetUp, PCApply, and KSPSolve see Table 9.1. CalcJac stands for the calculation of the Jacobian system, i.e., the mismatch vector and the Jacobian matrix. The abbreviation CA is used for the entire contingency analysis process.

The Eisenstat and Walker forcing terms generally performed the best in this test, especially when using the base case solution as initial solution for the contingency cases. Their adaptive nature makes them very well-suited to handle the resulting varying initial residual norms.

initial solution	flat start					
preconditioning	direct		own J_0		base J_0	
	count	iter	count	iter	count	iter
converged	6665	7/7	6665	6/15	6666	6/20
diverged	24	12/12	24	12/73	23	12/88
	count	time	count	time	count	time
PCSetUp	46948	191	6690	57.6	2	0.02
PCApply	46948	16.2	142263	48.9	176899	62.0
KSPSolve	46948	208	40287	135	40360	99.8
CalcJac	53638	98.9	46977	86.2	47050	86.2
CA	1	320	1	238	1	198

initial solution	base case solution					
preconditioning	direct		own J_0		base J^*	
	count	iter	count	iter	count	iter
converged	6666	2.2/2.2	6666	2.3/3.3	6665	2.4/6.3
diverged	23	12/12	23	12/73	24	12/88
	count	time	count	time	count	time
PCSetUp	14975	85.0	6686	57.8	2	0.02
PCApply	14975	5.18	38335	13.2	60661	21.3
KSPSolve	14975	90.3	15472	77.6	16418	33.5
CalcJac	21665	43.0	22162	42.1	23108	43.7
CA	1	140	1	132	1	84.4

Table 9.9: Contingency analysis using Eisenstat and Walker forcing terms

We have looked at two methods to improve on these forcing terms. One is to reduce the initial forcing term value of the Eisenstat and Walker strategy when using the base case solution as initial iterate. Because this initial iterate is generally much closer to the solution than a flat start, it is expected that a greater improvement can be attained in the first Newton iteration than the default 0.1 that we have been using for the Eisenstat and Walker strategy. The other is to log the convergence of the base case, and use this as a model for the expected convergence of the contingency cases. Both methods showed only very minor improvements over using plain Eisenstat and Walker forcing terms.

CHAPTER 10

Conclusions

The power flow problem is a computational problem that arises in power system operation and planning. In contingency analysis and Monte Carlo simulations, many slightly varying power flow problems have to be solved. The trends of international connection of power systems and decentralised power generation, have the potential to lead to power flow problems of a whole new scale.

Traditionally, the power flow problem is solved using Newton power flow or the Fast Decoupled Load Flow method. Newton power flow possesses the quadratic convergence behaviour of the Newton-Raphson method, but needs a lot of computational work per iteration. FDLF needs relatively very little computational work per iteration, but convergence is only linear. In practice, Newton power flow is generally preferred, because for some power flow problems FDLF fails to converge at all, while Newton power flow can still solve the problem. Both these methods are not viable for very large power flow problems, due to the use of the LU decomposition.

In this work, a Newton-Krylov power flow solver has been developed that is much faster than traditional solvers for large power flow problems, and is also much faster when solving many slightly varying problems of any size.

The theory behind Newton-Krylov methods has been treated, and some convergence theory was developed for inexact iterative methods in general, and inexact Newton methods in particular. This theory provides a better understanding of Newton-Krylov methods, and helps to make good choices for the Krylov method, preconditioning, and forcing terms.

Newton-Krylov power flow solvers have been discussed and tested using many combinations of choices for the Krylov method, preconditioning, and forcing terms. For large power flow problems, the best results were obtained using GMRES, preconditioned with an ILU(12) factorisation of the initial

Jacobian matrix J_0 or the FDLF matrix Φ, in conjunction with forcing terms based on the work of Eisenstat and Walker. For smaller problems, the best results were obtained with the same method, but using a complete LU factorisation of the initial Jacobian matrix as preconditioner.

It was shown that the resulting solver is as fast as Newton power flow for small problems, and many times faster for large problems, while convergence and robustness is equally good. Furthermore, it was demonstrated that the Newton-Krylov solver allows the reuse of the preconditioner between solves of slightly varying problems, much like the FDLF method, but without the need for factor updating or compensation techniques.

Further, it was shown how the traditional power flow solvers—Newton power flow and FDLF—can be interpreted as Newton-Krylov methods. This revealed that the developed Newton-Krylov power flow solver can be seen as a direct theoretical improvement on these traditional solvers, within the class of Newton-Krylov methods.

Summarizing, the Newton-Krylov power flow solver has no drawbacks compared to Newton power flow in terms of speed and convergence. When solving very large power flow problems it is many times faster than Newton power flow, and for contingency analysis and Monte Carlo simulations on power systems of any size, it also offers a great computational advantage.

APPENDIX A

Fundamental Mathematics

A.1 Complex Numbers

A complex number $\alpha \in \mathbb{C}$, is a number

$$\alpha = \mu + \iota\nu, \tag{A.1}$$

with $\mu, \nu \in \mathbb{R}$, and ι the imaginary unit defined by $\iota^2 = -1$. The quantity $\operatorname{Re}\alpha = \mu$ is called the real part of α, whereas $\operatorname{Im}\alpha = \nu$ is called the imaginary part of the complex number. Note that any real number can be interpreted as a complex number with the imaginary part equal to 0.

Negation, addition, and multiplication are defined as

$$-(\mu + \iota\nu) = -\mu - \iota\nu, \tag{A.2}$$

$$\mu_1 + \iota\nu_1 + \mu_2 + \iota\nu_2 = (\mu_1 + \mu_2) + \iota(\nu_1 + \nu_2), \tag{A.3}$$

$$(\mu_1 + \iota\nu_1)(\mu_2 + \iota\nu_2) = (\mu_1\mu_2 - \nu_1\nu_2) + \iota(\mu_1\nu_2 + \mu_2\nu_1). \tag{A.4}$$

The complex conjugate is an operation that negates the imaginary part:

$$\overline{\mu + \iota\nu} = \mu - \iota\nu. \tag{A.5}$$

Complex numbers are often interpreted as points in complex plane, i.e., 2-dimensional space with a real and imaginary axis. The real and imaginary part are then the Cartesian coordinates of the complex point. The same point in complex space can be described by an angle and a length. The angle of a complex number is called the argument, while the length is called the modulus or absolute value:

$$\arg(\mu + \iota\nu) = \tan^{-1}\frac{\nu}{\mu}, \tag{A.6}$$

$$|\mu + \iota\nu| = \sqrt{\mu^2 + \nu^2}. \tag{A.7}$$

Using these definitions, any complex number $\alpha \in \mathbb{C}$ can be written as

$$\alpha = |\alpha| \, e^{\iota\varphi}, \tag{A.8}$$

where $\varphi = \arg \alpha$, and the complex exponential function is defined by

$$e^{\mu+\iota\nu} = e^{\mu} \left(\cos \nu + \iota \sin \nu\right). \tag{A.9}$$

A.2 Vectors

A vector $\boldsymbol{v} \in K^n$ is an element of the n-dimensional space of either real numbers ($K = \mathbb{R}$) or complex numbers ($K = \mathbb{C}$), generally denoted as

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}, \tag{A.10}$$

where $v_1, \ldots, v_n \in K$.

Scalar multiplication and vector addition are basic operations that are performed elementwise. That is, for $\alpha \in K$ and $\boldsymbol{v}, \boldsymbol{w} \in K^n$,

$$\alpha\boldsymbol{v} = \begin{bmatrix} \alpha v_1 \\ \vdots \\ \alpha v_n \end{bmatrix}, \; \boldsymbol{v} + \boldsymbol{w} = \begin{bmatrix} v_1 + w_1 \\ \vdots \\ v_n + w_n \end{bmatrix}. \tag{A.11}$$

The combined operation of the form $\boldsymbol{v} := \alpha\boldsymbol{v} + \beta\boldsymbol{w}$ is known as a vector update. Vector updates are of $O(n)$ complexity, and are naturally parallelisable.

A linear combination of the vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m \in K^n$ is an expression

$$\alpha_1\boldsymbol{v}_1 + \ldots + \alpha_m\boldsymbol{v}_m, \tag{A.12}$$

with $\alpha_1 \ldots \alpha_m \in K$. A set of m vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m \in K^n$ is called linearly independent, if none of the vectors can be written as a linear combination of the other vectors.

The dot product operation is defined for real vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$ as

$$\boldsymbol{v} \cdot \boldsymbol{w} = \sum_{i=1}^{n} v_i w_i. \tag{A.13}$$

The dot product is by far the most used type of inner product. In this work, whenever we speak of an inner product, we will be referring to the dot product unless stated otherwise. The operation is of $O(n)$ complexity, but not naturally parallelisable. The dot product can be extended to complex vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{C}$ as $\boldsymbol{v} \cdot \boldsymbol{w} = \sum_{i=1}^{n} \overline{v}_i w_i$.

A vector norm is a function $\|.\|$ that assigns a measure of length, or size, to all vectors, such that for all $\alpha \in K$ and $\boldsymbol{v}, \boldsymbol{w} \in K^n$

$$\|\boldsymbol{v}\| = 0 \Leftrightarrow \boldsymbol{v} = \boldsymbol{0}, \tag{A.14}$$

$$\|\alpha\boldsymbol{v}\| = |\alpha| \, \|\boldsymbol{v}\|, \tag{A.15}$$

$$\|\boldsymbol{v} + \boldsymbol{w}\| \leq \|\boldsymbol{v}\| + \|\boldsymbol{w}\|. \tag{A.16}$$

Note that these properties ensure that the norm of a vector is never negative. For real vectors $\boldsymbol{v} \in \mathbb{R}^n$ the Euclidean norm, or 2-norm, is defined as

$$\|\boldsymbol{v}\|_2 = \sqrt{\boldsymbol{v} \cdot \boldsymbol{v}} = \sqrt{\sum_{i=1}^{n} v_i^2}. \tag{A.17}$$

In Euclidean space of dimension n, the Euclidean norm is the distance from the origin to the point $\boldsymbol{v}$. Note the similarity between the Euclidean norm of a 2-dimensional vector and the modulus of a complex number. In this work we omit the subscripted 2 from the notation of Euclidean norms, and simply write $\|\boldsymbol{v}\|$.

A.3 Matrices

A matrix $A \in K^{m \times n}$ is a rectangular array of real numbers ($K = \mathbb{R}$) or complex numbers ($K = \mathbb{C}$), i.e.,

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}, \tag{A.18}$$

with $a_{ij} \in K$ for $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$.

A matrix of dimension $n \times 1$ is a vector, sometimes referred to as a column vector to distinguish it from a matrix of dimension $1 \times n$, which is referred to as a row vector. Note that the columns of a matrix $A \in K^{m \times n}$ can be interpreted as n (column) vectors of dimension m, and the rows as m row vectors of dimension n.

A dense matrix is a matrix that contains mostly nonzero values; all n^2 values have to be stored in memory. If most values are zeros the matrix is called sparse. For a sparse matrix A, the number of nonzero values is denoted by $\operatorname{nnz}(A)$. With special data structures, only the $\operatorname{nnz}(A)$ nonzero values have to be stored in memory.

The transpose of a matrix $A \in K^{m \times n}$, is the matrix $A^T \in K^{n \times m}$ with

$$\left(A^T\right)_{ij} = (A)_{ji}. \tag{A.19}$$

A square matrix that is equal to its transpose is called a symmetric matrix.

Scalar multiplication and matrix addition are elementwise operations, as with vectors. Let $\alpha \in K$ be a scalar, and $A, B \in K^{m \times n}$ matrices with columns $\boldsymbol{a}_i, \boldsymbol{b}_i \in K^m$ respectively, then scalar multiplication and matrix addition are defined as

$$\alpha \boldsymbol{A} = \begin{bmatrix} \alpha \boldsymbol{a}_1 & \dots & \alpha \boldsymbol{a}_n \end{bmatrix}, \tag{A.20}$$

$$\boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} \boldsymbol{a}_1 + \boldsymbol{b}_1 & \dots & \boldsymbol{a}_n + \boldsymbol{b}_n \end{bmatrix}. \tag{A.21}$$

Matrix-vector multiplication is the product of a matrix $A \in K^{m \times n}$ and a vector $\boldsymbol{v} \in K^n$, defined by

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} a_{1i} v_i \\ \vdots \\ \sum_{i=1}^{n} a_{mi} v_i \end{bmatrix}. \tag{A.22}$$

Note that the result is a vector in K^m. An operation of the form $\boldsymbol{u} := A\boldsymbol{v}$ is often referred to as a matvec. A matvec with a dense matrix has complexity $O\left(n^2\right)$, while with a sparse matrix the operation has $O\left(\mathrm{nnz}\left(A\right)\right)$ complexity. Both dense and sparse versions are naturally parallelisable.

Multiplication of matrices $A \in K^{m \times p}$ and $B \in K^{p \times n}$ can be derived as an extension of matrix-vector multiplication by writing the columns of B as vectors $\boldsymbol{b}_i \in K^p$. This gives

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} \boldsymbol{b}_1 & \dots & \boldsymbol{b}_n \end{bmatrix} = \begin{bmatrix} A\boldsymbol{b}_1 & \dots & A\boldsymbol{b}_n \end{bmatrix}. \tag{A.23}$$

The product AB is a matrix of dimension $m \times n$.

The identity matrix I is the matrix with values $I_{ii} = 1$, and $I_{ij} = 0$, $i \neq j$. Or, in words, the identity matrix is a diagonal matrix with every diagonal element equal to 1. This matrix is such, that $IA = A$ and $AI = A$ for any matrix $A \in K^{m \times n}$, and identity matrices I of appropriate size.

Let $A \in K^{n \times n}$ be a square matrix. If there is a matrix $B \in K^{n \times n}$ such that $BA = I$, then B is called the inverse of A. If the inverse matrix does not exist, then A is called singular. If it does exist, then it is unique and denoted by A^{-1}. Calculating the inverse is—with $O\left(n^3\right)$ complexity—very costly for large matrices.

The column rank of a matrix $A \in K^{m \times n}$ is the number of linearly independent column vectors in A. Similarly, the row rank is the number of linearly independent row vectors in A. For any given matrix, the row rank and column rank are equal, and can therefore simply be denoted as $\mathrm{rank}\left(A\right)$. A square matrix $A \in K^{n \times n}$ is invertible, if and only if $\mathrm{rank}\left(A\right) = n$.

A matrix norm is a function $\|.\|$ such that for all $\alpha \in K$ and $A, B \in K^{m\times n}$

$$\|A\| \geq 0, \tag{A.24}$$

$$\|\alpha A\| = |\alpha| \, \|A\|, \tag{A.25}$$

$$\|A + B\| \leq \|A\| + \|B\|. \tag{A.26}$$

Given a vector norm $\|.\|$, the corresponding induced matrix norm is defined for all matrices $A \in K^{m\times n}$ as

$$\|A\| = \max \{\|A\boldsymbol{v}\| : \boldsymbol{v} \in K^n \text{ with } \|\boldsymbol{v}\| = 1\}. \tag{A.27}$$

Every induced matrix norm is submultiplicative, meaning that

$$\|AB\| \leq \|A\|\|B\| \text{ for all } A \in K^{m\times p}, \ B \in K^{p\times n}. \tag{A.28}$$

A.4 Graphs

A graph is a collection of vertices, any pair of which may be connected by an edge. Vertices are also called nodes or points, and edges are also called lines. The graph is called directed if all edges have a direction, and undirected if they do not. Graphs are often used as the abstract representation of some sort of network. For example, a power system network can be modelled as an undirected graph, with buses as vertices and branches as edges.

Let $V = \{v_1, \ldots, v_N\}$ be a set of N vertices, and $E = \{e_1, \ldots, e_M\}$ a set of M edges, where each edge $e_k = (v_i, v_j)$ connects two vertices $v_i, v_j \in V$. The graph G of vertices V and edges E is then denoted as $G = (V, E)$. Figure A.1 shows a simple graph $G = (V, E)$ with vertices $V = \{1, 2, 3, 4, 5\}$ and edges $E = \{(2, 3), (3, 4), (3, 5), (4, 5)\}$.

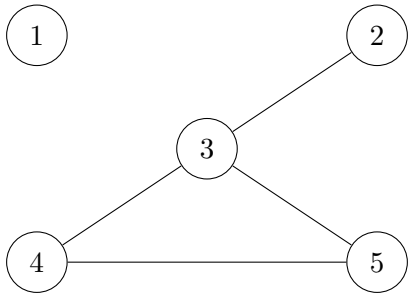

Figure A.1: A simple graph

The incidence matrix A of a graph $G = (V, E)$ is an $M \times N$ matrix in which each row i represents an edge $e_i = (p, q)$, and is defined as

$$a_{ij} = \begin{cases} -1 & \text{if } p = v_i, \\ 1 & \text{if } q = v_j, \\ 0 & \text{otherwise.} \end{cases} \tag{A.29}$$

In other words, row i has value -1 at index p and value 1 at index q. Note that this matrix is unique for a directed graph. For an undirected graph, some orientation has to be chosen. For example, the matrix

$$A = \begin{bmatrix} 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \tag{A.30}$$

is an incidence matrix of the graph shown in Figure A.1. Such a matrix is sometimes referred to as an oriented incidence matrix, to distinguish it from the unique unoriented incidence matrix, in which all occurrences of -1 are replaced with 1.

Note that some authors define the incidence matrix as the transpose of the matrix A defined here.

APPENDIX B

Power Flow Test Cases

For numerical experiments with power flow solvers, a test set of power flow problems is needed. Test problems with up to a few hundred busses are readily available on the web, but problems of realistic size are hard to come by. Transmission systems are vital to our way of life, and can be vulnerable to attacks if the attackers know where to strike. Therefore, the models of the actual transmission systems used in industry are not publicly available.

For our research, we were able to use the UCTE[1] winter 2008 study model, which consists of 4253 busses and 7191 lines. Larger test cases were constructed by copying the model and interconnecting the copies with new transmission lines, as detailed in Section B.1. This proved much easier than generating realistic models of virtual power systems from scratch.

The test cases are named uctewXXX, where XXX is the number of copies of the original problem used in the construction of the test case. Table B.1 shows the number of buses, branches, and nonzeros in the Jacobian matrix for each of the constructed test cases.

B.1 Construction

Each test case is constructed by connecting two copies of the previous test case. The important choices in this process are the choice of buses to connect and how to connect them, and how to deal with the slack buses. Figure B.1 shows a schematic representation of the construction process.

[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/`

name	buses	branches	nnz (J)
uctew001	4,253	7,191	62,654
uctew002	8,505	14,390	125,372
uctew004	17,009	28,796	250,872
uctew008	34,017	57,624	502,000
uctew016	68,033	115,312	1,004,512
uctew032	136,065	230,752	2,010,048
uctew064	272,129	461,760	4,022,144
uctew128	544,257	924,032	8,048,384
uctew256	1,088,513	1,849,088	16,104,960

Table B.1: Test cases

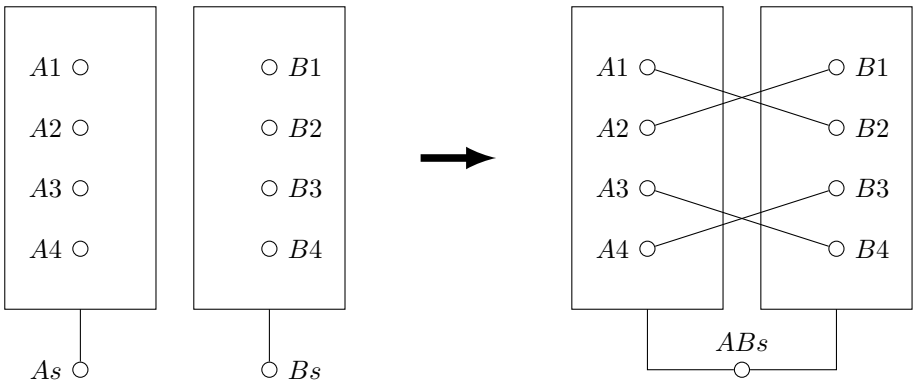

Figure B.1: Test case construction process

The two network copies A and B each have their own slack bus, denoted by As and Bs respectively. If one slack bus is simply removed, together with all branches connected to it, all the generation in that slack bus has to be provided for by the other slack bus. Because the other slack bus is in a totally different area of the network, this may lead to an imbalanced test case. Therefore, it is better to combine both slack busses into one new slack bus ABs, that is connected to all the buses that either of the old slack buses was connected to.

When two existing power systems are connected in practice, the network connection is generally made at the highest voltage level. Thus it makes sense to do the same when constructing test cases by connecting existing networks. We select a number of load busses at the highest voltage level, approximately uniformly distributed by bus index, with a small random element.

Connecting completely different regions of the network copies might lead to a serious imbalance. Thus, each bus in A should be connected to a bus in B that corresponds to a nearby bus in A. If each bus is connected directly to the corresponding bus in the other network, no current would be flowing between A and B. The solution of the newly constructed problem would simply consists of the original network solution in both A and B. Therefore, we choose to connect the buses per pair $A1$ and $A2$ close to each other, to the corresponding buses $B1$ and $B2$ in the other network, such that $A1$ is connected to $B2$, and $A2$ is connected to $B1$.

The number of buses connected between the two network copies is of some importance. In our test cases the number of buses selected in A is 8 times the amount of original UCTE models incorporated in A. If too few buses are chosen, the networks A and B are nearly decoupled. This results in an admittance matrix with two blocks on nonzeros on the diagonal, and only a few nonzeros outside of these blocks. This structure continues into the Jacobian matrix, and factorising such a Jacobian is similar to factorising the two diagonal blocks independently. Any issues with the scaling of the factorisation method would be lost.

ACKNOWLEDGEMENTS

As indicated by the single name on the cover of the thesis, a PhD is an individual achievement. However, it is by no means something you can achieve alone. Here, I would like to acknowledge those people that have been most important in my scientific journey.

First I would like to express my very great appreciation to my promotor Kees Vuik. You were already an inspiration when I was a student following your courses, and you have been an inspiration in many more ways since. You were always interested and involved in my work, while also giving me all the freedom that I could wish for. Thank you for giving me the opportunity to work in your Numerical Analysis group.

I would also like to offer special thanks to my daily supervisor Domenico Lahaye. You were always there, ready to help, and your enthusiasm and broad interest led me in directions that I might otherwise have missed.

Special thanks are also due to my promotor Lou van der Sluis from the group Electrical Power Systems. You brought the world of power systems so much closer, and helped ensure that our research was practically relevant.

I am particularly grateful for the assistance of Robert van Amerongen. You were vital in bridging the gap between applied mathematics and electrical engineering, and your sharp analysis of my work was immensely valuable.

My appreciation also goes to the members of the doctoral committee for their careful evaluation of my work.

Further, I would like to thank all my colleagues from the Numerical Analysis group. Thank you all for an unforgettable time. Special thanks go to my long-time office mates Tijmen, Sander, and Pavel. Our personal and professional conversations provided great entertainment and motivation.

From the Electrical Power Systems group, I want to thank Georgios, Zong Yu, and Nima. It was always a pleasure to share ideas.

My gratitude also goes to Barry Smith, for making our visit to Argonne National Laboratory such a pleasant one, and for his invaluable assistance with the PETSc library.

I want to thank all my good friends, especially everyone that I have lived with at CH10. You have provided both motivation and the necessary distraction. You have not just shaped my time in Delft, but also shaped me, and continue to do so. Thank you.

Finally, I want to thank my dear parents. You built the foundation for the person that I am. Thank you for your unconditional love and support.

Reijer Idema
Delft, November 2012

CURRICULUM VITAE

Reijer Idema was born on June 22, 1979, in Broek op Langedijk, The Netherlands. He received secondary eduction at the Adriaan Roland Holstschool (Bovenbouw Vrije School, 1993–1997) in Bergen. In 1998 he finished his secondary eduction (VWO) at the Cornelis Drebbel College in Alkmaar.

From 1998 to 2006, Reijer studied Applied Mathematics at the Delft University of Technology. As a Bachelor student he was active in several different committees of the study society Christiaan Huygens. For his Master he specialised in Computational Science and Engineering. His thesis research was conducted for FROG Navigation Systems in Utrecht, on the subject of constructing paths for automated guided

After finishing his thesis, Reijer briefly worked for FROG Navigation Systems, to implement the developed method for incorporation into their software framework. He further briefly worked at ORTEC in Gouda, on their order and inventory optimisation (ORION) software.

From 2008 to 2012, Reijer worked as a PhD student in the Numerical Analysis group of prof. Kees Vuik, within the Delft Institute of Applied Mathematics of the Delft University of Technology. His research on the use of Newton-Krylov methods for power flow and contingency analysis problems was conducted in collaboration with the Electrical Power Systems group of prof. Lou van der Sluis, and was supervised by Domenico Lahaye and prof. Kees Vuik, as well as prof. Lou van der Sluis. In addition to his research, Reijer lectured calculus and co-organised the PhDays 2010.

After finishing his PhD studies in 2012, Reijer took up a position as scientific software engineer at VORtech in Delft.

PUBLICATIONS

Journal Papers

R. Idema, D. J. P. Lahaye, C. Vuik, and L. van der Sluis. Scalable Newton-Krylov solver for very large power flow problems. *IEEE Transactions on Power Systems*, 27(1):390396, February 2012.

R. Idema, G. Papaefthymiou, D. J. P. Lahaye, C. Vuik, and L. van der Sluis. Towards faster solution of large power flow problems. *IEEE Transactions on Power Systems* (under review).

Conference Proceedings

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 17, April 2010.

Technical Reports

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.

R. Idema, D. J. P. Lahaye, and C. Vuik. On the convergence of inexact Newton methods. Report 11-14, Delft Institute of Applied Mathematics, Delft University of Technology, 2011.

BIBLIOGRAPHY

[1] O. Alsac, B. Stott, and W. F. Tinney. Sparsity-oriented compensation methods for modified network solutions. *IEEE Transactions on Power Apparatus and Systems*, PAS-102(5):1050–1060, May 1983. 70

[2] A. B. Alves, E. N. Asada, and A. Monticelli. Critical evaluation of direct and iterative methods for solving ax = b systems in power flow calculations and contingency analysis. *IEEE Transactions on Power Systems*, 14(2):702–708, May 1999. 69

[3] 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. 64

[4] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.*, 23(1):15–41, 2001. 64

[5] L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16(1):1–3, 1966. 16

[6] 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/. 75

[7] A. R. Bergen and V. Vittal. *Power Systems Analysis*. Prentice Hall, New Jersey, second edition, 2000. 40, 44

[8] P. N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM J. Sci. Stat. Comput.*, 11(3):450–481, 1990. 16, 17

[9] 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. 61, 65

[10] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods*. SIAM, Philadelphia, 2000. 17

[11] H. Dag and A. Semlyen. A new preconditioned conjugate gradient power flow. *IEEE Transactions on Power Systems*, 18(4):1248–1255, November 2003. 59

[12] T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):165–195, June 2004. 64

[13] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006. 6

[14] 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. 61, 65

[15] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19(2):400–408, 1982. 13, 23, 28

[16] R. S. Dembo and T. Steihaug. Truncated-Newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26:190–212, 1983. 13, 64

[17] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999. 64

[18] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, New Jersey, 1983. 16, 17, 28

[19] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, New York, 1986. 5, 6

[20] 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. 13, 64

[21] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 21:352–362, 1984. 8

[22] 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. 61, 63, 65

[23] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996. 5, 7

[24] A. Hohmann. *Inexact Gauss Newton Methods for Parameter Dependent Nonlinear Problems*. PhD thesis, Freie Universität Berlin, 1994. 13, 65

[25] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, third edition, 1990. 5

[26] 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. 2

[27] R. Idema, D. J. P. Lahaye, and C. Vuik. On the convergence of inexact Newton methods. Report 11-14, Delft Institute of Applied Mathematics, Delft University of Technology, 2011. 2

[28] 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. 2

[29] R. Idema, D. J. P. Lahaye, C. Vuik, and L. van der Sluis. Scalable Newton-Krylov solver for very large power flow problems. *IEEE Transactions on Power Systems*, 27(1):390–396, February 2012. 2

[30] R. Idema, G. Papaefthymiou, D. J. P. Lahaye, C. Vuik, and L. van der Sluis. Towards faster solution of large power flow problems. *IEEE Transactions on Power Systems*, 2012 (under review). 2

[31] D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *J. Comp. Phys.*, 193:357–397, 2004. 14

[32] X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw.*, 29(2):110–140, June 2003. 64

[33] 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. 6

[34] 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. 6

[35] 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. 55, 57

[36] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970. 24, 28

[37] M. A. Pai and H. Dag. Iterative solver techniques in large scale power system computation. In *Proceedings of the 36th Conference on Decision & Control*, pages 3861–3866, December 1997. 61, 65

[38] L. Powell. *Power System Load Flow Analysis*. McGraw-Hill, 2004. 44

[39] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, March 1993. 10, 63

[40] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, second edition, 2003. 7

[41] 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. 7

[42] P. Schavemaker and L. van der Sluis. *Electrical Power System Essentials*. John Wiley & Sons, Chichester, 2008. 40, 44

[43] A. Semlyen. Fundamental concepts of a Krylov subspace power flow methodology. *IEEE Transactions on Power Systems*, 11(3):1528–1537, August 1996. 61, 65

[44] 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. 7

[45] 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. 7

[46] B. Stott and O. Alsac. Fast decoupled load flow. *IEEE Transactions on Power Apparatus and Systems*, PAS-93(3):859–869, 1974. 52, 55

[47] B. Stott, O. Alsac, and A. J. Monticelli. Security analysis and optimization. *Proceedings of the IEEE*, 75(12):1623–1644, December 1987. 69

[48] W. F. Tinney. Compensation methods for network solutions by optimally ordered triangular factorization. *IEEE Transactions on Power Apparatus and Systems*, PAS-91(1):123–127, 1972. 70

[49] 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. 47, 61

[50] 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. 47, 61

[51] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001. 63

[52] R. A. M. van Amerongen. A general-purpose version of the fast decoupled loadflow. *IEEE Transactions on Power Systems*, 4(2):760–770, 1989. 55

[53] R. A. M. van Amerongen. A rank-oriented setup for the compensation algorithm. *IEEE Transactions on Power Systems*, 5(1):283–288, February 1990. 70

[54] 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. 7

[55] R. S. Varga. *Matrix Iterative Analysis*. Springer-Verlag, second edition, 2000. 7

[56] V. V. Voevodin. The problem of non-self-adjoint generalization of the conjugate gradient method is closed. *U.S.S.R. Comput. Math. and Math. Phys.*, 22:143–144, 1983. 8

[57] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2(1):77–79, March 1981. 6

[58] 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. 61, 63