DELFT UNIVERSITY OF TECHNOLOGY

REPORT 16-01

OPTIMAL CONFIGURATION OF FUTURE ELECTRICITY GRID

M. DE BEURS, P. DE GRAAF, P. HANSLER, S. HERMANS, K. VAN WALSTIJN, J. DE WINTER AND D.J.P. LAHAYE

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2016

TU Delft
Faculty of Electrical Engineering

Case study Evolvingphysics

Optimal configuration of future electricity grid

Authors:
Marc de Beurs 1529048

Paul de Graaf 4001990

Perry Hansler 4009665

Sophie Hermans 4172477

Koen van Walstijn 4168895

Jules de Winter 4166086

Supervisor and examiner:
Dr. D.J.P. Lahaye

May 1, 2014

Abstract

This paper considers the scenario of a big new energy source connecting to an existing network of generators and loads. To be able to maintain such a network at the correct settings, the well known load flow equations have to be solved. Since existing software has trouble with solving the load flow equations in the scenario of a very large generator, new software was created using Matlab's inherent fsolve function. This software was consequently used in combination with existing software of the MATPOWER package to calculate the optimal way of connecting a new large generator to an existing network. This was done for a relatively small case using a brute force method which iterated over all possible options.

Contents

Chapter 1

Introduction

More renewable sources of energy have to be used in the future. One speculated method of generating such sustainable energy is the use of solar energy generated in the desert. The goal of this study is to model the addition of such a resource to the current power grid and to compute the optimal configuration of this combination. Rather than using actual data of a potential new energy source and the actual European power grid, this report uses a model of a smaller network and a fictional extra generator to look at the *methods* of finding the optimal connection points. Given detailed actual data, the methods explored in this paper can then be applied to finding the optimal configuration of a real power grid.

The goals of adding such an energy source to the grid can vary. Reducing costs by adding a cheap energy source could be one objective, for example, but investing in cleaner energy could be another, which might be more expensive. Both of these scenarios are discussed, as the optimal connections to the grid are the same in both cases.

This study uses Matlab as computational software and the publicly available MATPOWER package to model the electrical network. The network that will be used is based on case118, an existing case in the MATPOWER package.

This report is structured as follows. First we will briefly discuss the theory of load flow computations and the implementation in Matlab in the form of the MATPOWER package. In chapter 3, we will discuss a way of solving the load flow problem using a fsolve function in Matlab. In chapter 4, the implementation of the extra source in MATPOWER and the computation of the optimal configuration will be discussed.

Chapter 2

Load Flow Computation

In real life power distribution one is confronted with multiple users and multiple generators connected in a network. All these users have real time power needs and want their power right away. It is important for the generators to supply the exact amount of required power given certain loads. A load flow problem consists of a network of generators and loads connected through branches. A node in the network of branches that hosts a generator or load is called a bus. In Figure 2.1 a very easy example is shown with a generator bus on the left connected by one branch to a load bus on the right.

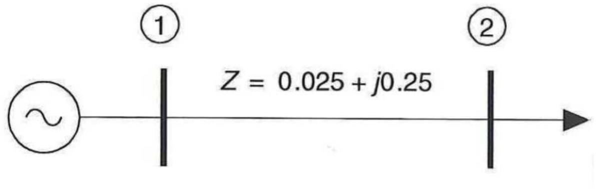

Figure 2.1: Very easy example of a load flow problem with one generator bus and one load bus. [1]

2.1 Bustypes

In a load flow problem the load buses have certain power needs and the generators supply a given amount of power. The goal is to find the correct voltage settings at each bus, such that the generators supply the right power. To accomplish this, the voltage angle and magnitude at each bus has to be determined.

Each bus is fully described by four parameters, namely: voltage magnitude $|V|$, voltage angle δ, real power P and reactive power Q. For each load bus the real and reactive power needs are specified, whilst the voltage angle and magnitude are unknown. These buses are also referred to as PQ-buses.

A generator bus has a known voltage magnitude and real power, but the voltage angle and reactive power are unknown. Generator buses are thus referred to as PV-buses.
To be able to cope with power losses and fluctuating loads one generator is designated as a slack generator. This generator has unspecified power, so it is able to fill in the gaps where this is needed. This slack bus does have a specified voltage magnitude and angle. The details of a buses have been combined in Table 2.1.

	Known parameters	Unknown parameters	Bus-Type
Slack generator	$\|V\|,\ \delta$	$P,\ Q$	Slack-bus
Generator	$P,\ \|V\|$	$\delta,\ Q$	PV-bus
Load	$P,\ Q$	$\|V\|,\ \delta$	PQ-bus

Table 2.1: Summary of bustypes

2.2 Deriving the load flow equations

When a voltage difference is created across a branch a current starts to flow. Each branch has its own characteristics that have an effect on the caused current. These effects can be expressed in the impedance Z, which is the sum of the resistance R and reactance X:

$$Z = R + jX$$

In the load flow equations it is more convenient to work with the admittance Y, which is the inverse of the impedance:

$$Y = \frac{1}{Z} = G + jB$$

Here G is the inductance and B the susceptance. Loosely spoken: the bigger the admittance, the larger the current under a certain voltage difference.
The admittances for all the branches in a network consisting of N-buses can be conveniently collected in an [N x N] admittance matrix, which has the following form:

$$Y_{ij} = G_{ij} + jB_{ij} = |Y_{ij}|\ e^{j\theta_{ij}}$$

The voltage magnitude and angle at a node find their origin in a complex representation of the voltage:

$$V_i = |V_i|\ e^{j\delta_i} = |V_i|\,(\cos\delta_i + j\sin\delta_i)$$

The current at a given node can be calculated by summing the voltages of all the nodes multiplied by the admittances of the branches between them:

$$I_i = \sum_{n=1}^{N} Y_{in} Vn$$

When nodes i and j are not connected with each other the admittance Y_{ij} is equal to 0. Thus the current at a node is only dependent on the voltages of the nodes it is connected with.
The complex power S is the sum of the real power P and the complex reactive power Q. It can be calculated for a node by multiplying the current at this node with the corresponding voltage.

$$S_i = V_i\,(I_i)^* = P_i + jQ_i$$

We now can create an expression for the real and reactive power in terms of the admittance matrixes (which can be created from knowledge of the network) and the yet unknown voltage magnitude and angles.

$$S_i = V_i \sum_{n=1}^{N} (Y_{in})^*\,(Vn)^* = \sum_{n=1}^{N} |V_i|\,|Y_{in}|\,|V_n|\,e^{j(\delta_i - \delta_n - \theta_{in})} = \sum_{n=1}^{N} |V_i|\,|Y_{in}|\,|V_n|\,(\cos(\delta_i - \delta_n - \theta_{in}) + j\sin(\delta_i - \delta_n - \theta_{in}))$$

By separating the real and complex part and making use of basic trigonometry we get the well known load flow equations[1]:

$$
\begin{aligned}
P_i &= \sum_{n=1}^{N} |V_i| \, |Y_{in}| \, |V_n| \cos\left(\theta_{in} + \delta_n - \delta_i\right) \\
Q_i &= -\sum_{n=1}^{N} |V_i| \, |Y_{in}| \, |V_n| \sin\left(\theta_{in} + \delta_n - \delta_i\right)
\end{aligned}
$$

Chapter 3

Solving the loadflow equations using Fsolve

One way to solve the load flow equations is using MATPOWER, a package for Matlab. However for certain problems MATPOWER does not produce a solution or takes too long to solve a problem. For example: problems with convergence arise when a generator with a big power production is connected to a system. This directly translates to the scenario of a big power source in the desert which is connected to the current european network. It is therefore of quite some importance to have a different method capable of solving the load flow equations in such a scenario.

It was our goal to write a function using Matlab's standard function fsolve that can solve the problems where MATPOWER fails. We strived to make this program compatible with MATPOWER case structure and to make as much use of existing MATPOWER codes as possible. In this section we first provide the mathematical basis behind solving a load flow problem, then show how MATPOWER's operates and where this goes wrong. Finally we present our solution and explain how it intertwines with MATPOWER's structure.

3.1 Solving a load flow problem

When confronted by a load flow problem, one is given a system of generator and load buses. The load flow problem is considered solved when for each bus the voltage angle and magnitude is known, such that at each bus the (real and reactive) power calculated with the load flow equations is equal to the specified power. Mathematically:

$$P_{i,\,calc} = P_{i,\,spec} \rightarrow \sum_{n=1}^{N} |V_i|\,|Y_{in}|\,|V_n| \cos\left(\theta_{in} + \delta_n - \delta_i\right) - P_{i,\,spec} = 0$$
$$Q_{i,\,calc} = Q_{i,\,spec} \rightarrow -\sum_{n=1}^{N} |V_i|\,|Y_{in}|\,|V_n| \sin\left(\theta_{in} + \delta_n - \delta_i\right) - Q_{i,\,spec} = 0$$

As explained in chapter 2 a load bus provide us with two unknowns, $|V_i|$ and δ_i and has a specified real power $P_{i,\,spec}$ and a specified reactive power $Q_{i,\,spec}$. Since both $P_{i,\,spec}$ and $Q_{i,\,spec}$ are known we can use both of the two equations shown above to find the two unknowns.

A generator bus has a known voltage magnitude, so the only unknown is the voltage angle: δ_i. At this bus only the specified real power $P_{i,\,spec}$ is given, so now we can use only one equation to find one unknown.

The voltage magnitude and angle of the slack bus are known and do not need to be calculated.

For each load bus we have two equations and two unknowns and for each generator bus we have one equation and one unknown. We are thus dealing with a square system of nonlinear equations: for n unknowns in the system we have an equal number of nonlinear equations. This implies we have a well-defined problem.

We now have the daunting task to solve this system of nonlinear equations by finding a set of voltage magnitude and angles such that the above equations are equal to zero. This can only be done iteratively and there are many different methods for this task.

The main principle behind all these methods is to start with an initial guess for the unknowns. The real and reactive powers (of the nodes where these are specified) are then calculated with the load flow equations and the specified powers are then subtracted from the corresponding calculated powers.

If the calculated powers of all nodes match the specified powers, the difference of all equations is zero and our initial guess was the solution. However this is most likely not the case and we have to improve our guess. After improvement, the powers are recalculated and the difference is checked again. This process is repeated until the norm of the difference between specified and calculated power is below a certain threshold, also called tolerance. The lower the tolerance, the more precise the solution of our system. Depending on the accuracy wanted by the user, the tolerance can thus be made smaller or larger. The main difference between the different methods of solving nonlinear systems is the way of improving the initial guess.

3.2 MATPOWER's way of solving a load flow problem

The MATPOWER-package uses a Newton-Raphson method to solve the load flow equations. This is an iterative method that makes use of derivatives to find a root of a function. A root is the coordinate where the function has value 0. The key is to use the direction of the derivative to make successive estimates of the location of the root.

It is most easily explained with a function of a single variable as illustrated in figure 3.1. Here a function $h(x)$ is sketched and the iterative steps towards the roots are graphed as well.

An initial (nonzero) point x^0 is supplied to the method. The tangent at this point is then calculated and the intersection of the tangent with the x-axis is the new estimate for the root. The value of h with this new estimate, $h(x^1)$, is then calculated. If $|h(x^1)| > \epsilon$, where ϵ is the tolerance, the procedure is repeated and a new estimate is retrieved that should be closer to the root. If all goes well, an x is found after n steps which satisfies $|h(x^n)| < \epsilon$ and the root is found with the desired accuracy.

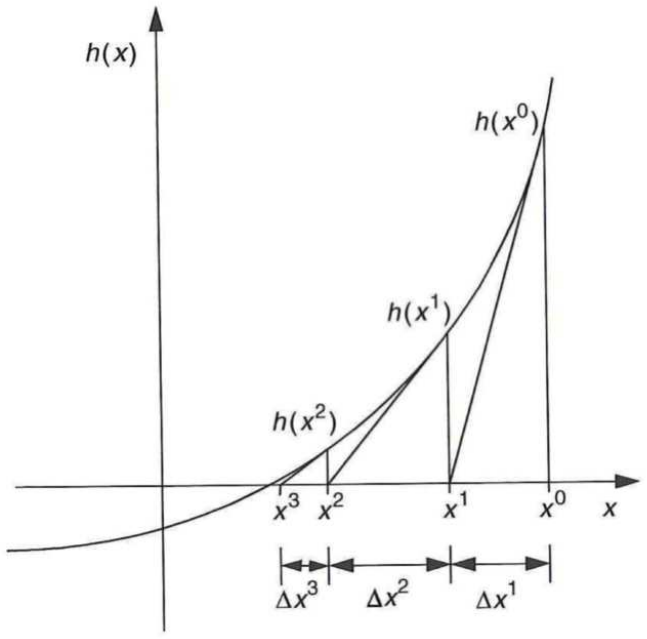

Figure 3.1: Example of the Newton-Raphson method. [1]

This method can be applied to a load flow problem by using a generalization to more equations. This algorithm makes use of a Jacobian, which is a matrix of all equations differentiated to all unknowns. The mathematical algorithm is for the next estimate is:

$$J\Delta x = \begin{bmatrix} P_{2,\,spec} - P_2(x) \\ \vdots \\ P_{N,\,spec} - P_N(x) \\ Q_{N_g+2,\,spec} - Q_{N_q+2}(x) \\ \vdots \\ Q_{N,\,spec} - Q_N(x) \end{bmatrix} \quad \text{with } x = \begin{bmatrix} \delta_2 \\ \vdots \\ \delta_N \\ |V_{N_g+2}| \\ \vdots \\ |V_N| \end{bmatrix}$$

In this equation there are N buses of which 1 is the slack bus and N_g are other generators. The slack bus is numbered one, and the buses are order such that the generators are on buses 2 to $N_g + 1$. All unknowns are gathered in a vector x and the correction of this vector for the next estimate is noted as Δx. The power calculated with the load flow equations are functions of the unknowns and are thus denoted as $P_i(x)$ and $Q_i(x)$. J is the Jacobian and has the form:

$$J = \begin{bmatrix} \frac{\partial P_2(x)}{\partial \delta_2} & \cdots & \frac{\partial P_2(x)}{\partial |V_N|} \\ \vdots & \ddots & \vdots \\ \frac{\partial Q_N(x)}{\partial \delta_2} & \cdots & \frac{\partial Q_N(x)}{\partial |V_N|} \end{bmatrix}$$

Since both J and the vector of specified power minus the calculated power can be calculated the correction the estimate can be obtained by factorization or the inversion of the Jacobian. The new guess is then obtained by adding Δx to x. The mathematical derivation of this method depends on a Taylor expansion and we direct the interested reader to [1].

3.3 Problems with MATPOWER's convergence

Although the Newton-Raphson method works pretty well in most cases, it sometimes fails to converge. Again this is most easily shown with a function of one variable.

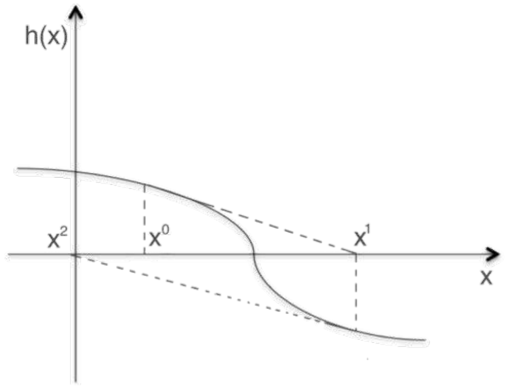

Figure 3.2: A function where the Newton-Raphson method has trouble converging.

As usual we start with an initial guess x^0. The tangent line overshoots the solution and we end up overcorrecting our initial guess and get x^1. This isn't a big problem if it happens just once, however our next guess also overshoots our solution and we end up with an estimate x^2 that is worse than our initial guess x^0. It is obvious that this iteration scheme is not going as planned.

3.3.1 Using 'Fsolve': a trust region method

A possible solution to this problem is to implement a trust region. A trust region limits the maximum correction of the estimate and may thus eliminate the possibility of overcorrecting the estimate. Intelligent trust region algorithms exist that allow for larger trust regions when it is "safe" to make big corrections, but that limit the trust region when this is not the case.

Matlab's fsolve function is able to solve a system on nonlinear equations, such as the load flow problem. It has three different algorithms to solve such systems, of which two use a trust region. Our goal was to use such a trust region algorithm that is built into Matlab by applying the fsolve function to the load flow equations.

To maximize efficiency and complementarity we tried to keep our program as closely related to MATPOWER as possible. Therefore we use the same structure for the input and we use a lot of functions that are part of the MATPOWER package. A schematic overview of how the program works is shown in Figure 3.3

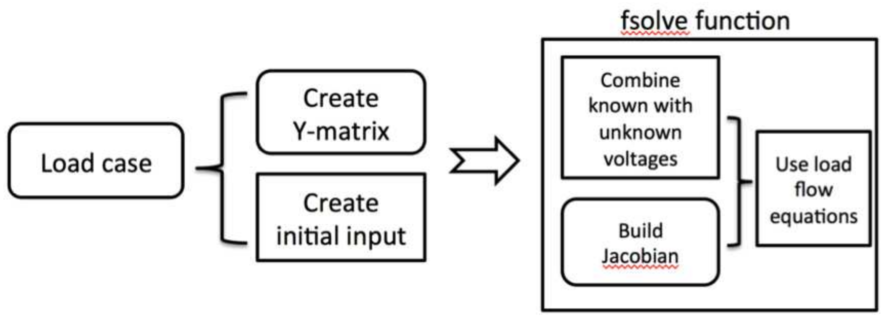

Figure 3.3: A schematic overview of the written program. Squares with rounded corners imply that the task was achieved using MATPOWER functions.

Here squares with rounded corners indicate that the task was achieved using a function that was part of the MATPOWER package.

3.4 Calculating the Jacobian

3.4.1 Analytical calculating the Jacobian with the load flow equations

The Jacobian for solving the problem can be calculated with the load flow equations. The active and reactive power are given by:

$$P_i = \sum_{n=1}^{N} |V_i|\,|Y_{in}|\,|V_n| \cos(\theta_{in} + \delta_n - \delta_i)$$
$$Q_i = -\sum_{n=1}^{N} |V_i|\,|Y_{in}|\,|V_n| \sin(\theta_{in} + \delta_n - \delta_i)$$

By deriving these with respect to V and δ, one can obtain the Jacobian:

$$J = \begin{bmatrix} \frac{\partial P}{\partial \delta} & \frac{\partial P}{\partial V} \\ \frac{\partial Q}{\partial \delta} & \frac{\partial Q}{\partial V} \end{bmatrix}$$

3.4.2 Fast decoupled method

The fast decoupled method makes use of a simplified Jacobian. The assumption is made that for small values of δ the values of $\frac{\partial P}{\partial V}$ and $\frac{\partial Q}{\partial \delta}$ are so small that they are assumed to be zero, because there is a weak coupling. Consequently less computations are needed per iteration. When the result converges, it converges to the correct solution. However, the simplification can invoke more iterations. The simplified Jacobian takes the following form.

$$J = \begin{bmatrix} \frac{\partial P}{\partial \delta} & 0 \\ 0 & \frac{\partial Q}{\partial V} \end{bmatrix}$$

When the derivatives are taken, the simplified Jacobian becomes:

$$\frac{\partial P_i}{\partial \delta_i} = \sum_{n=1}^{N} |V_i| |Y_{in}| |V_n| \sin(\theta_{in} + \delta_n - \delta_i)$$
$$\frac{\partial Q_i}{\partial V_i} = -\sum_{n=1}^{N} \frac{V_i V_n^2 Y_{in}^2}{|V_i| |Y_{in}| |V_n|} \sin(\theta_{in} + \delta_n - \delta_i)$$

3.4.3 The Jacobian in MATPOWER

In MATPOWER the Jacobian is not calculated numerically. The Jacobian is calculated as a matrix with real and imaginary elements of the following equations.

$$\boxed{\begin{aligned} \frac{\partial S}{\partial V_m} &= \operatorname{diag}(V) \cdot \operatorname{conj}\left(Y_{bus} \cdot \operatorname{diag}\left(\frac{V}{|V|}\right)\right) + \operatorname{conj}(\operatorname{diag}(I_{bus})) \cdot \operatorname{diag}\left(\frac{V}{|V|}\right) \\ \frac{\partial S}{\partial V_a} &= j \cdot \operatorname{diag}(V) \cdot \operatorname{conj}(\operatorname{diag}(I_{bus}) - Y_{bus} \cdot \operatorname{diag}(V)) \end{aligned}}$$

With S being the S_{bus}, V_m and V_a the voltage magnitude and angle, respectively. The Y_{bus} is the admittance matrix and the I_{bus} is defined as $I_{bus} = Y_{bus}V$.

When fsolve is not supplied with a Jacobian it generates a Jacobian after each iteration by finite differencing. This is computationally expensive, therefore we have chosen to supply the Jacobian calculated using the method mentioned in this subparagraph.

3.5 An analysis of Residual plots for fsolve and MATPOWER's Newton Method

This section depicts a comparison of the solving power of Matpower's Newton method and our fsolve program described in the previous subsection for a variety of cases. It has been found that even though our fsolve program does not make up in speed in terms of residual reduction per iteration, it does solve a larger or different subset of problems. Note that all results shown below were interpreted with a required residual minimum of 10^{-15}.

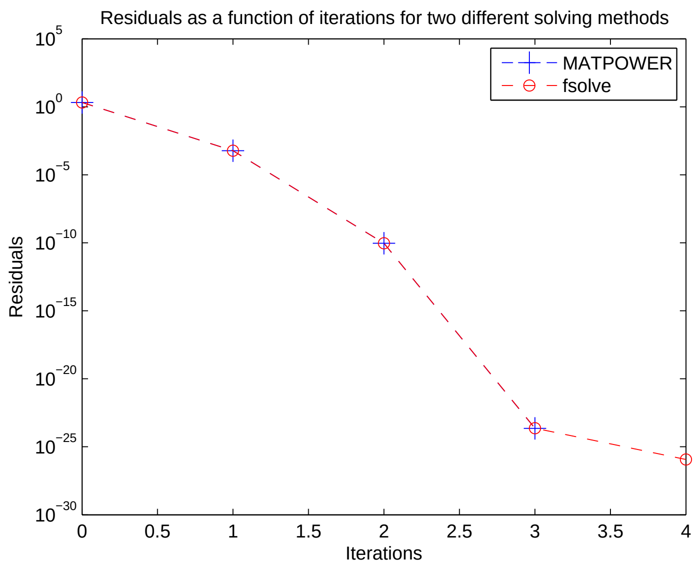

Figure 3.4: Plot of the residuals versus iterations for both Newton and fsolve solvers for the often-mentioned 'case118' as given by MATPOWER. It can be seen and confirmed that there is no difference between solvers for this particular size.

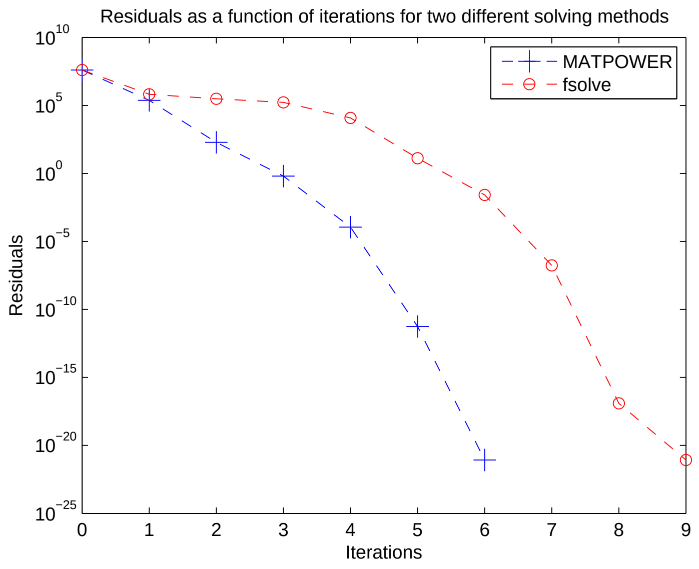

Figure 3.5: Plot of the residuals versus iterations for both Newton and fsolve solvers for 'case2383wp' as given by MATPOWER. One can see that MATPOWER's Newton method converges more quickly in terms of residual reduction per iteration than our Fsolve program. In absolute time MATPOWER won by about 0.2 seconds. One can argue that since this case is 'safe', fsolve's trust region is working against itself in terms of speed, leaving Newton's method to be the winner.

With the knowledge that fsolve will not have an advantage in terms of speed, cases included with the

MATPOWER package have been edited to make them more extreme; increasing generator output or load demand. This to see if fsolve could cope with a wider range of scenarios than MATPOWER. The results are shown in the figures below. Having tested far more cases than shown in these results, the difference in speed started to appear in cases involving approximately 2000 or more nodes.

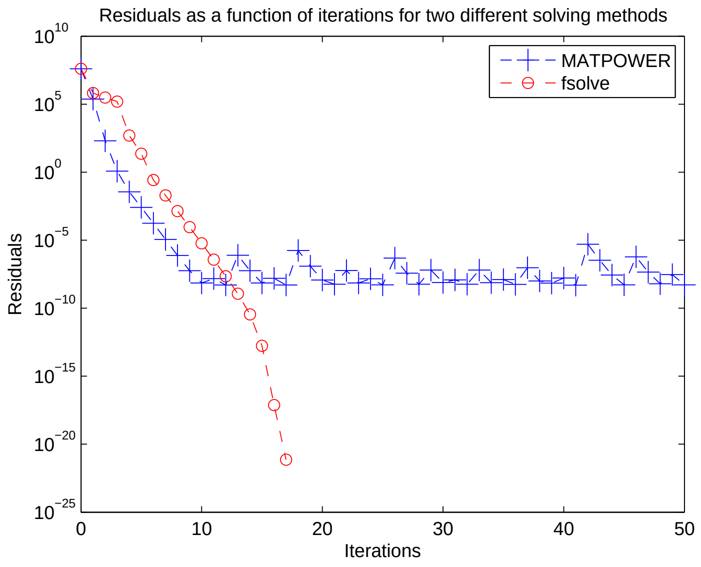

Figure 3.6: Plot of the residuals versus iterations for both Newton and fsolve solvers for an edited version of 'case2383wp', known as 'case2383wpcrash'. The difference is that one of the load buses has had their active load demand significantly increased, however the number to make Fsolve work and MATPOWER fail was rather specific and hints towards a significant sensitivity. Once again, one can argue that MATPOWER wins in terms of speed, but has problems dipping below a residual of roughly 10^{-10}, starting to oscillate. The fsolve program described in the previous subsection does not show a problem to reach the residual limit of 10^{-15}.

With these results, it has been attempted to 'break' one of the cases involving less nodes, to see if fsolve can handle more extreme cases in that regime as well.

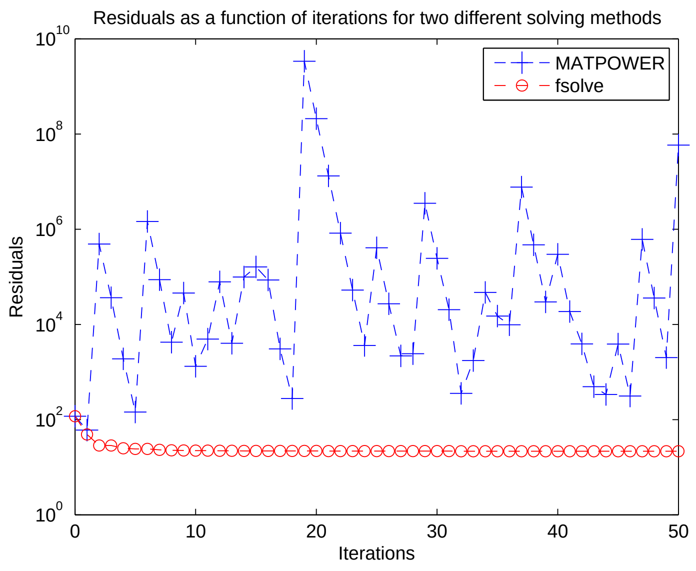

Figure 3.7: Plot of the residuals versus iterations for both Newton and fsolve solvers for an edited version of 'case9', known as 'case9crash'. The difference is made by making one bus require much more reactive load than usual (factor 10). Here one can see that while both solvers experience severe issues while solving, fsolve seems to converge slowly towards a certain limit while MATPOWER's solver experiences extremely volatile behaviour in a higher regime of residuals

3.5.1 Conclusions

This section will summarize the results found above. The main point is that fsolve does indeed solve a different set of problems than MATPOWER's Newton method does. The size of fsolve's set hints at being bigger than MATPOWER's Newton method, however the boundaries of these different subsets are difficult to probe due to the high amount of variables, especially in high-node configurations. Further research for probing these boundaries is therefore encouraged.
Few other points can be made, when both solvers don't seem to converge to an acceptable threshold, MATPOWER's residual behaviour per iterations shows extremely volatile behaviour whereas fsolve converges to a certain unacceptable threshold. Another point which can be made is that fsolve does not show signs of beating MATPOWER's speed in terms of residual reduction per iteration, but sometimes wins in terms in absolute time. This means that in those cases fsolve requires less time per iteration.

Chapter 4

Implementation of an extra source

We now have a method of solving the load flow equations when we connect a huge generator to an existing network. This means we can solve the proposed scenario of a large generator in the desert that is connected to the European network. Even though it is practical, and most likely imperative, to be able to calculate the right settings for all the generators in such a situation, it is also important to consider other facets of this scenario. One important question that needs to be answered is where you will connect said generator to the existing network. With the large number of buses, there is an even larger number of possible ways to connect the "desert-generator" to the network. This section explores a way of finding the optimal network in terms of cost reduction. By using the MATPOWER function runopf for particular case (case118), all possible scenarios of adding a generator have been compared.

4.1 The characteristics of an optimal configuration

A network is optimal when the costs of the generator are the lowest of all possible configurations. Costs associated with power generation and maintenance have to be taken into account. These are modeled as the cost functions of the generators. Energy losses of the cables make that the generators need to work harder: the costs associated with this are therefore taken into account indirectly.

Also, the stability of the system has to be guaranteed. A system is stable when the voltage is between 0.9 and 1.1 p.u. [2]. This condition has to be met, so possibly the configuration with lowest costs will not be a good option.

4.2 Optimal power flow computation in MATPOWER

Optimal power flow (runopf) is the main function that is used to handle a case. The input of the runopf function is a network in the format suited to MATPOWER. It calculates the optimal configuration by changing generator outputs and voltages. The input of runopf can be any case (in the right format). The calculation of an optimal configuration is done by minimizing four conditions to calculate the optimal power flow of a given system: feasibility (equality), cost, gradient and complementarity condition (inequality) [3]. Especially the cost condition is interesting for this case study. However, the other three conditions have to be met as well to get the right results for a specific configuration. The tolerance for these conditions is 10^{-6}.

Both the DC and AC problem are solved with the CCV approach (Constrained Cost Variable). The cost function is then replaced by a helper variable and a set of linear constraints. These constraints form a convex set, as a consequence the cost variable have to lie in the epigraph of the cost function. [3].
In the vector mpoption the options for the MATPOWER problem are set. One of these options is the solver type (which solver is used to solve the problem). A specific solver can be set or the default solver is used.

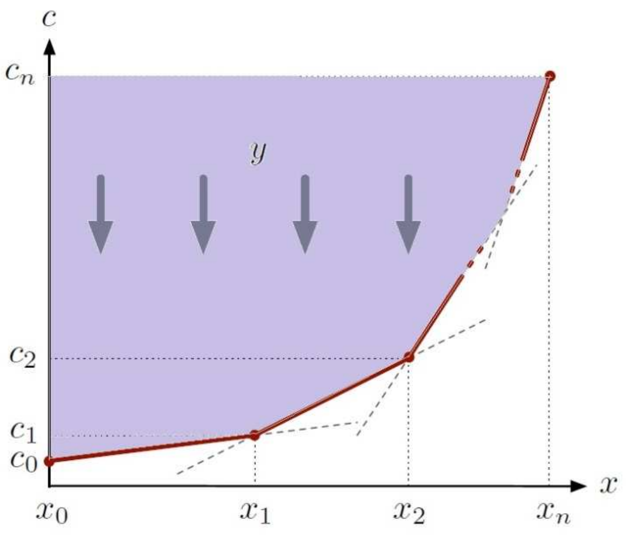

Figure 4.1: The purple area in the graph indicates the epigraph of the cost function in which the cost variable has to lie.

Table 4.1: Order of preference for the default solver for solving a DC or AC optimization problem. In the last column the MATPOWER codes for the different solver in the mpoptions vector.

Order of preference	Type of Solver	Code in the mpoptions vector
1	CPLEX	500
2	MOSEK	600
3	Gurobi	700
4	BPMPD_MEX	100
5	Tbx	300
6	MIPS	200

When the default solver is used its availability is checked. There is an order of preference for the different solvers.

4.3 Adding a generator to an existing case

To check which design is best, a brute force technique is used. An extra function is created that computes every possible design. It adds a generator (and branch) to every node of a given case in the right format. The characteristics of this generator and branch can be adjusted to match the existing model. The function added to check various configurations is built up as follows.

An iteration loop is created in which the generator is connected each time to a different node and the runopf function is ran. In each iteration the total cost of this new configuration is saved, in which the minimum cost is located. Therefore, the optimal configuration is found.

Unfortunately, some computations will not converge with the existing solving method used by runopf, this is where the fsolve technique from chapter 3 might come in. Even though runopf fails to converge, its still presents its calculated values. These values must not be considered as a possible outcome for the most optimal bus. An option is added to know whether a system has converged or not and to show a plot of the residue of the convergence (of all four conditons). This way, it can be seen with which setup the computation has failed and the result can be neglected as a possible outcome.

4.4 Possible variations in adding a generator

Connecting the generator through one branch to the network causes a huge load on a specific part of the network. Therefore, the created function can be adjusted to connect the generator through more than one branch. However, the computation time becomes even larger when a design is checked with a generator coupled with more than one branch. If a system contains N nodes and a generator will be added through number of branches (n_c), the number of possible designs is $N(N-1)(N-2)\cdots = \prod_{n_c=0}^{N-1}(N-n_c)$. The computation time becomes sky high when this option is used in large cases (with many nodes and branches).

To overcome this problem, the program is ran once to check for the best configurations. The best design is then fixed and the program is ran again to check every other possible connection with a second branch. Then the second branch is also fixed and so on. This can be repeated to reduce the costs even further. Using this technique, the assumption is made that the most optimum node from the first run, is also one of the nodes in the best configuration with more than one connection. This has been validated with small cases using the brute force method (iterating every possibility) upto 3 cables. But the assumption stays for larger cases and adding more cables. Lets call this technique from now on: **"The fixed node iteration method"**.

Another possibility is to interconnect nodes within the existing network. The philosophy behind this technique is to distribute the extra power (from the added generator) more equally over the network. The larger the case, the more possible interconnections there are. So another assumption is made in order to minimize the computation time: from the above mentioned technique (fixed node), a list of optimal node connections is obtained. Interconnecting branches between these nodes might reduce the costs further then interconnecting nodes not on this list.

4.5 Input data

Case118 is used to test the optimization model mentioned above. The case is based on a part of the American Electrical Power System and consists of 118 nodes; from which 54 are generators and 64 are load buses. All these buses are connected with a total of 186 branches. Below a schematic is shown of this network. Note however that this schematic gives no information about the distances between buses. It is not the topology, it merely gives an overview.

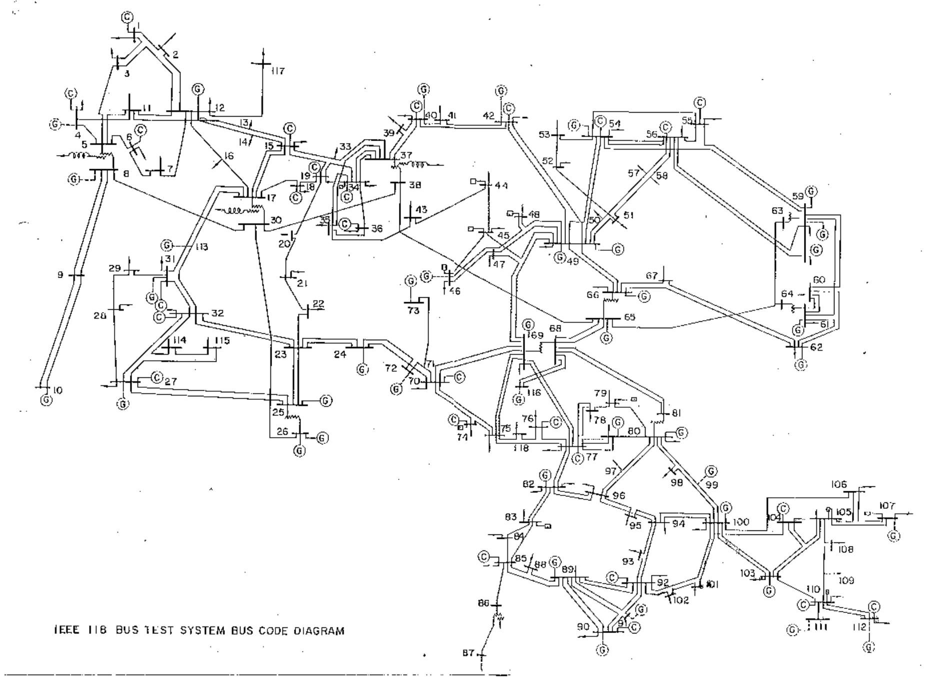

Figure 4.2: Schematic of case118, not the topology [7].

One note has to be made using this existing case118: the branch data are corrected. This is due to a lack of actual values, the power limits (rateA, rateB and rateC) are set to 300 MVA, which is a typical value for cables like those in the case [4].

Besides the existing case, the MATPOWER program also requires the information of the added generator and branch(es). For the generator, the maximum real output power generated is set to 2000 MW, and the maximum and minimum reactive power output to 100 and -20 MVAr respectively. The voltage magnitude setpoint is 1 p.u. The other data is the same as other generators in case118. In order to find the optimal connections, the impact of adding the generator should be large. Therefore, the linear cost coefficient of the generator is set to 10 \$/MWh, which is significantly lower than the other generators. The small quadratic cost component is typical for most cases analysed with MATPOWER.

The branch data is based on the data as follows:

Nominal Voltage	500 kV
R (Ω/km)	0.0128
$x_L = \omega$ L (Ω/km)	0.2454
$b_C = \omega$ C (μ s/km)	96.5

Table 4.2: Model of the resistance (R), reactance (X) and total line charging susceptance (B) for a 500 kV branch [6].

The length of the branch is estimated on 1000 km, which is roughly the distance between Tunisia (North Africa) and Southern Europe. The adaptation of the nominal values to the per-unit system is based on a baseMVA of 100 MVA and a basekV of 500 kV. The format of MATPOWER also requires the MVA rating, which is set to 900 MVA [4]. The ratio, angle, status, angle minimum and angle maximum are the same as the rest of the case branches.

For the interconnecting cables it is difficult to assign corresponding values, this because no information on the topology of the case is present: no distances between buses are known. Therefore the average is calculated from all the cables present in the case, which are then used as the values for the interconnecting cables added to the system.

4.6 Results

In this result section first the optimum connection of the generator to the network is analyzed, starting with 1 cable and looking at the responses when adding more cables. Second, the cost reduction due to the addition of interconnecting cables to the network is checked. Finally a break-even analysis is done to look at thresholds for when this added generator will turn out economically, or purely evironmentally.

4.6.1 Connecting the generator to the system

Using the brute force method on case118, adding the generator through bus 54 is most optimal. With costs after the added generator of $1.1735 \cdot 10^5$ \$/hour, this offers a cost reduction of $0.12693 \cdot 10^5$ \$/hour. Below is the table with the first 10 connections shown with the corresponding costs and reduction:

Connected bus	Costs \$/hour	Reduction \$/hour
54	$1.1735 \cdot 10^5$	$0.12693 \cdot 10^5$
17	$1.1748 \cdot 10^5$	$0.12558 \cdot 10^5$
56	$1.1766 \cdot 10^5$	$0.12380 \cdot 10^5$
49	$1.1766 \cdot 10^5$	$0.12380 \cdot 10^5$
5	$1.1776 \cdot 10^5$	$0.12280 \cdot 10^5$
61	$1.1778 \cdot 10^5$	$0.12263 \cdot 10^5$
40	$1.1778 \cdot 10^5$	$0.12258 \cdot 10^5$
80	$1.1783 \cdot 10^5$	$0.12211 \cdot 10^5$
37	$1.1783 \cdot 10^5$	$0.12205 \cdot 10^5$
66	$1.1799 \cdot 10^5$	$0.12050 \cdot 10^5$

Table 4.3: First 10 best bus connections with their corresponding costs and cost reduction, using the brute force method: iterating one cable over all buses

When adding more branches from the generator to the system, costs are further reduced as can be seen in figure 4.3. However, the cost reduction becomes smaller with every extra connection.

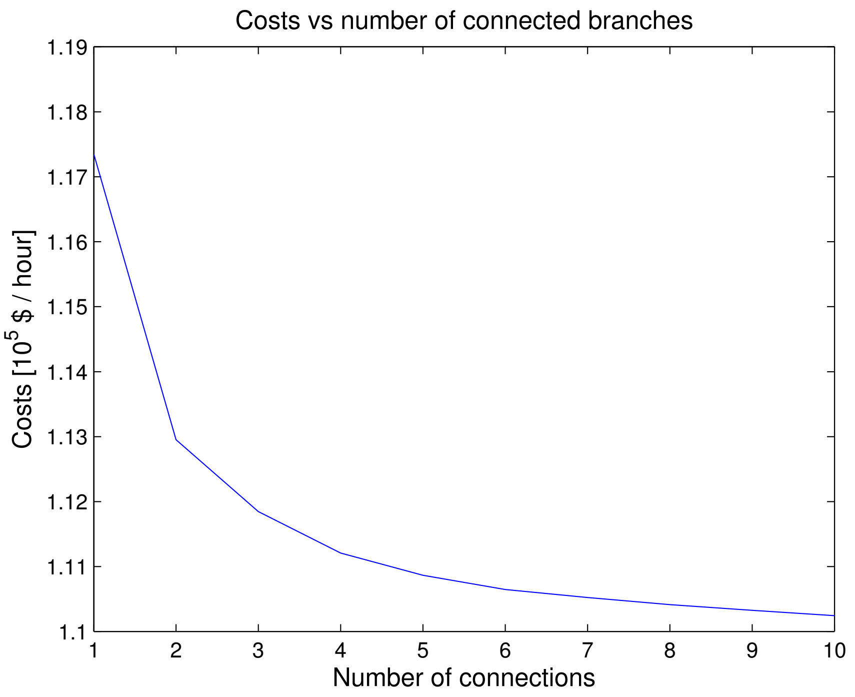

Figure 4.3: Cost reduction due to the addition of branches from the generator to the system.

The figure above (4.3)is generated by using the brute foce method (iterating over all possible buses) to obtain the optimal bus connection. The first branch from the new generator is then pinned on the found bus, following with again the brute force iteration to find the optimal bus for the second branch, and so on (fixed node iteration method). In the figure shown a total of 10 branches have been connected from the generator to the network, in the following order:

Number of connections	Successive branch connected to bus number	Costs \$/hour
1	54	$1.1735 \cdot 10^5$
2	17	$1.1295 \cdot 10^5$
3	105	$1.1185 \cdot 10^5$
4	40	$1.1121 \cdot 10^5$
5	75	$1.1087 \cdot 10^5$
6	12	$1.1065 \cdot 10^5$
7	112	$1.1052 \cdot 10^5$
8	32	$1.1042 \cdot 10^5$
9	56	$1.1033 \cdot 10^5$
10	76	$1.1025 \cdot 10^5$

Table 4.4: First 10 optimal branch connections with their corresponding bus and costs. As can be seen adding an extra branch reduces the costs.

It can be observed that in fact the first branch in table 4.4 corresponds to the first one found in table 4.3 with the corresponding costs, as would be expected. The second found connection also corresponds to the second best option with the brute force technique (using 1 cable). Obiously now both costs are no longer the same since in table 4.3 only 1 cable is connected to bus 17, where as in the table 4.4 a cable is connected to 54 as well as 17. Looking at the third bus connection, it is observed that this one not even appears in the best 10 buses with the single connection. This can be understood since connecting a generator this size to the network changes the entire system, resulting in other optimum connection buses.

Since MATPOWER minimalizes the costs by adjusting the available generators output, adding more branches causes an extra parameter for minimization. Explaining the continued reduction in the costs when more branches are connected. To amplify this theory the next figure shows the result when the added generator is connected to the entire network (each bus in case118)

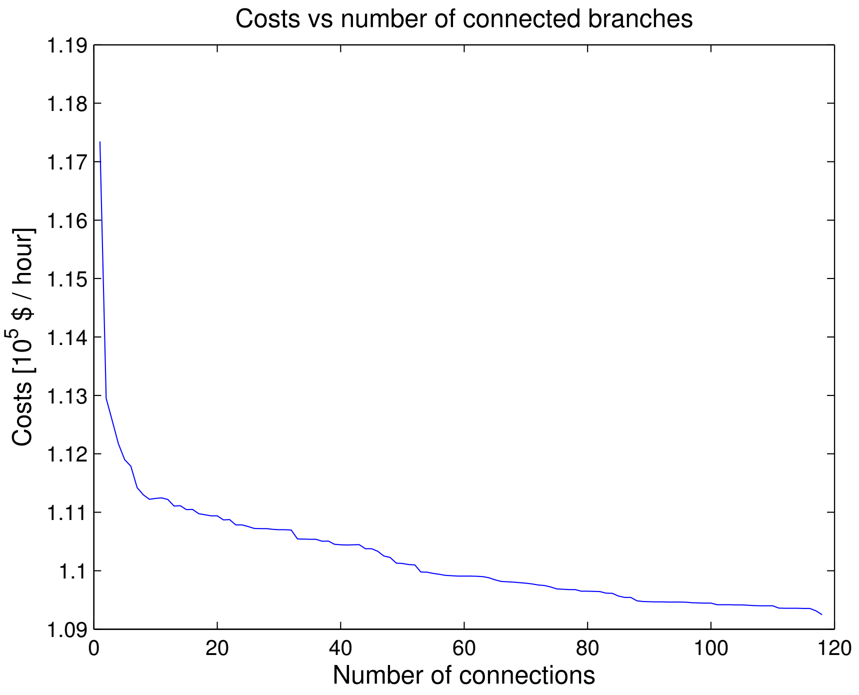

Figure 4.4: Added generator is connected to all buses in the network. A continud decrease in operating costs is observed.

Note that in generating this figure a different order of adding the branch to the next bus is used as in figure 4.3. Here the sorted list of optimal buses is used optained by the brute force iteration with 1 cable, where the first 10 buses are shown in table 4.3. The reason for this is simple, a huge reduction of computation time compared to the fixed node technique.

Obviously, it is not realistic to connect the generator to all the existing buses. Geographically, this would be impractical, and the costs of construction and maintenance would rise substantially. At a certain point the extra investment of another cable is expected to be economically undesirable, even though it reduces the overall cost/hour calculated by MATPOWER. A scenario is chosen where 4 connections are made to the generator, thereby reducing the costs as far as can be seen in figure 4.3. This is due to a lack of information about the construction and maintenance costs of cables. Investiging these costs is beyond the scope of this paper. The connections will be made to the top 4 buses found with the fixed node technique, which are 54, 17, 105 and 40.

4.6.2 Interconnecting buses within the existing network

With the network connected to the generator, optimization concerning interconnecting branches (within the network itself) is analysed.

Minimizing the computation time, only the top 10 nodes found by fixed node (see table 4.4), are interconnected. From all these possible configurations the top 10 is listed below, with the corresponding costs and the reduction due to the interconnection (compared with the generator connected with 4 branches):

Node from	Node to	Costs $/hour	Reduction $/hour
40	76	$1.1108 \cdot 10^5$	
105	76	$1.1108 \cdot 10^5$	
40	75	$1.1110 \cdot 10^5$	
105	75	$1.1110 \cdot 10^5$	
54	76	$1.1112 \cdot 10^5$	
105	112	$1.1113 \cdot 10^5$	
17	76	$1.1113 \cdot 10^5$	
40	12	$1.1113 \cdot 10^5$	
56	76	$1.1114 \cdot 10^5$	
105	12	$1.1114 \cdot 10^5$	

Table 4.5: First 10 optimal interconnection branches looking at the optimum 10 nodes found in table 4.4, with the corresponding costs and reduction compared to the network with the generator connected (with 4 branches).

As can be observed the reduction is minimal, only about 0.11% compared to merely the generator connected. As mentioned before, investigation on the breakeven point for an investment in an extra cable vs costs reduction per hour, is beyond the scope of this paper.

To compare the outcome of the assumption used above (to minimize computation time), all possible interconnections have been calculated (merely by brute force technique) in the following table:

Node from	Node to	Costs $/hour	Reduction $/hour
10	76	$1.1106 \cdot 10^5$	
10	118	$1.1106 \cdot 10^5$	
66	76	$1.1107 \cdot 10^5$	
10	74	$1.1107 \cdot 10^5$	
66	118	$1.1107 \cdot 10^5$	
25	76	$1.1107 \cdot 10^5$	
69	76	$1.1107 \cdot 10^5$	
25	118	$1.1107 \cdot 10^5$	
76	89	$1.1108 \cdot 10^5$	
69	118	$1.1108 \cdot 10^5$	

Table 4.6: First 10 optimal interconnecting branches iterating over all buses.

Comparing both tables a couple of things can be observed. At first only a small difference is seen between both optimum interconnections (only an improvement of 0.018%). Another observation can be made that the top 10 nodes (for table 4.5) do not even enter the overall top 10 list (table 4.6). To be exact they can be found just outside the top list, as can be understood due to the simular operation costs. This implies that the assumption used to minimize computation time does not reveal the optimum results. However the variations are small, these could be of any significance when the number of buses in the network increases.

4.6.3 Break-even analysis

A break-even analysis for the situation of adding the 2000 MW generator to case118 with four cables to buses 54, 17, 105 and 40 is shown in 4.5 below:

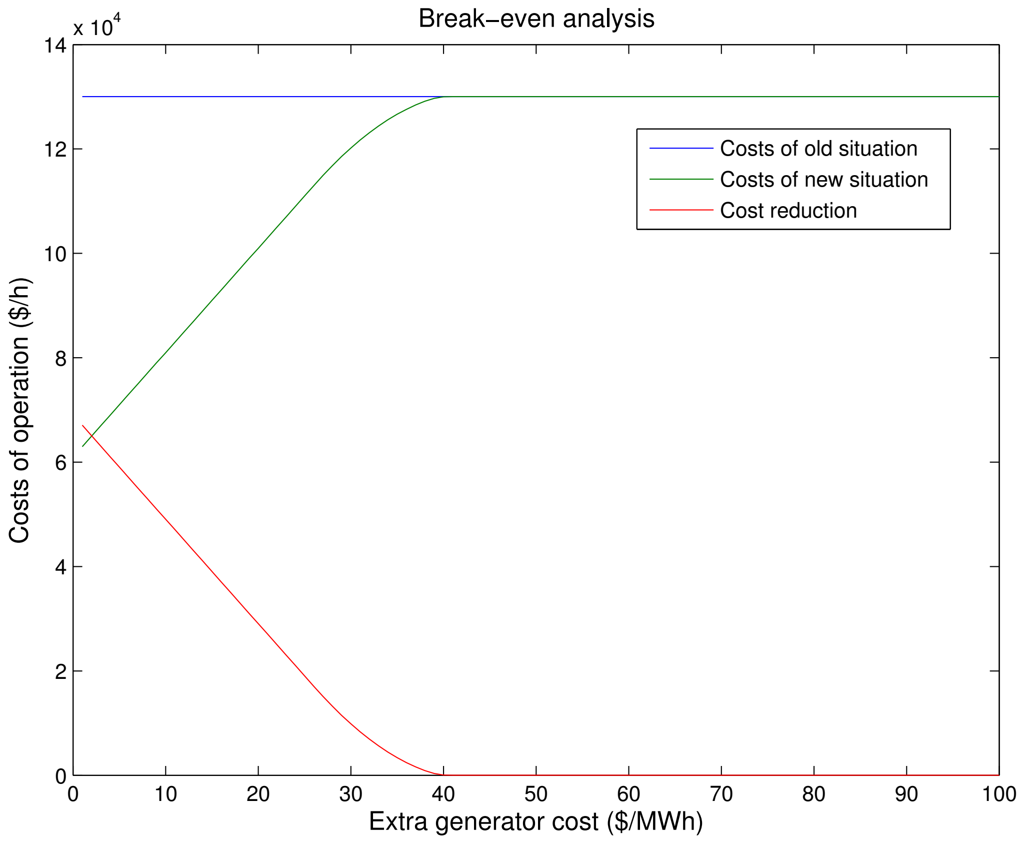

Figure 4.5: Break-even analysis of adding the 2000 MW generator to case118 with three cables to buses 54, 17 and 105.

In this analysis, only the linear cost coefficcient of the extra generator is taken into account, and it is varied from 0 to 100 $/MWh. The costs of the old situation are plotted in blue. As can be seen in figure 4.5, for low costs of the extra generator, the cost reduction is reasonably big. As much power as possible is now generated at the new generator. As this generator becomes more expensive, the costs converge to the old situation: the optimal power flow algorithm now generates no power at the new generator as it is too expensive, and the situation is the same as before.

This analysis shows that, in the scenario of case118 with the extra generator as specified, the latter is able to lower the total costs of operation if its own costs of generation are below 38 $/MWh. Note that in a break-even analysis like this, an assumption is made that lowering the total costs of operation is the goal of adding the generator. If this is not the goal, for example lowering the use of fossil fuels is, the costs of operation are allowed to be higher to achieve this.

Therefore, a scenario is considered where the generator is connected to the same four optimal buses, but the output is set to a *fixed output level* of 2000 MW. The same linear cost model is used, and the results can be seen in figure 4.6 below:

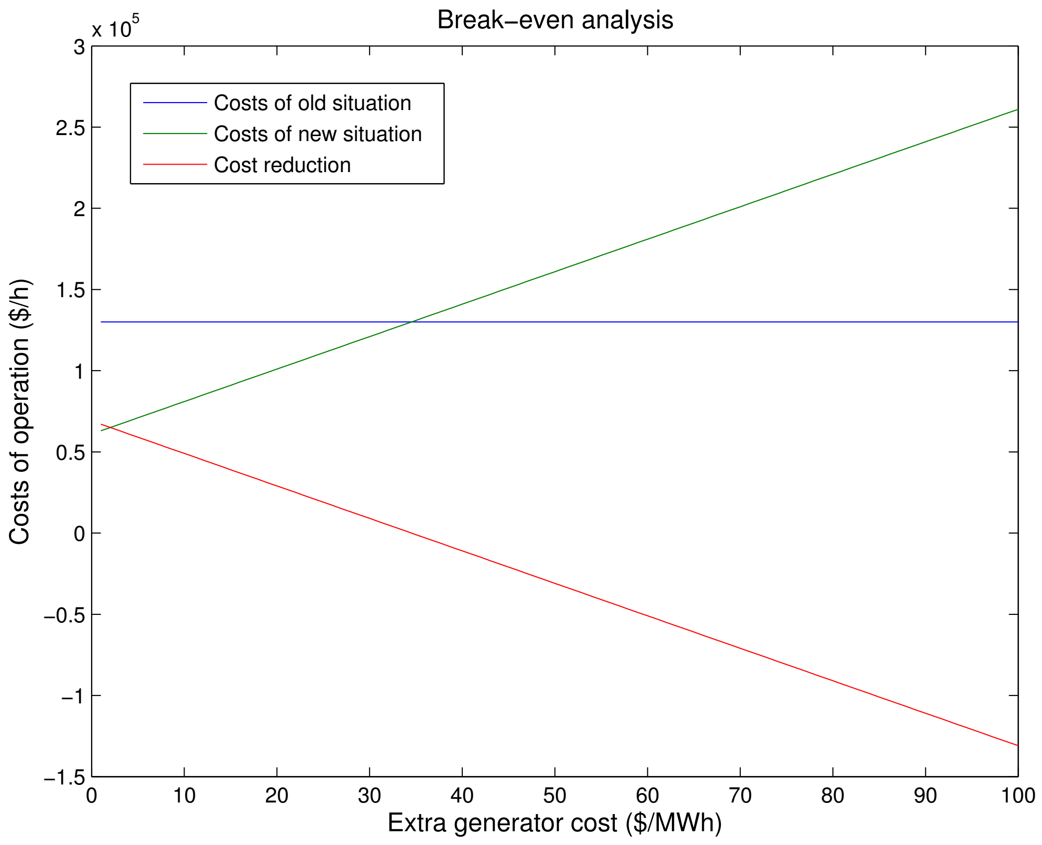

Figure 4.6: Break-even analysis of adding the 2000 MW generator to case118 with three cables to buses 54, 17 and 105.

Naturally, the break-even point is the same. However, the total costs now keep rising after the costs of the new generator pass the break-even point, and the cost reduction will become negative. Reading the costs from this figure merely demonstrates a method of weighing the benefits and costs of a using such a generator in possible situations.

Chapter 5

Conclusion and recommendations

When an extra generator of this size (2000 MW) is connected to any existing network, the whole system changes. Starting with a brute force iteration technique, a first impression of suitable optimum connections are obtained. This impression however does not lead to correct predictions for chosing successive buses when adding more branches (from the generator to the network).

It has been found, starting from the brute force analysis, that only for the first 2 optimum connections (in the case of 118 buses) correct predictions could be made. After that the system has changed as such that the most optimum third connection could not be predicted anymore, see section 4.6.1. This is highly unfortunate since with these predictions, computation time for finding the optimum connection buses could have been reduced enormously. Especially when systems with thousands of buses, for example the European network, are examined. This analysis cannot be used directly for other cases.

Next, a trend of decreasing costs has been observed when more and more branches (from the generator to the network) are added to the system. It is found that the operation costs per hour keep decreasing with every added generator branch, as can be seen in figure 4.4. This reduction can be clarified by looking at the optimising algorithm. Here all available parameters are tweaked until a most optimal configuration has been established. Adding more and more cables directly implies more and more parameters (of freedom), therefore a lower optimum can be found. At a certain point the investment for building another cable (probabaly a long distant one) cannot compete with the reduction the the operation costs. To be able to find this break-even point, more knowledge of the manufacting costs of these branches same as the distances between buses in the network, is required.

Furthermore the influence of adding interconnecting branches, within the network itself, have been examined. Here only a slight reduction of merely 0.11 %, compared to the network with the generator connected with 4 branches (see section 4.6.2), was found. Again an assumption for saving computation time has been tested. Here the top ten list, found with the fixed node iteration method (see section 4.4), is assumed to give the most optimum interconnection. Unfortunately again this prediction has been found to not predict the most optimum configuration, however the difference is only small (about 0.018% 4.6.2). Overall the reduction of interconnecting branches are of such small size that their investment is probabaly not realistic.

An important note has to be made concerning the usage of branches in this analysis:
As has been stated frequently, no real information is present for the topology of the network. Modeling the connecting branches from the generator to the network, with the correct data is therefore realistically not achievable. Resulting in the fact the branch data is not changing with every connection. Physically, this means that the same branch is used for coupling the generator to the network. Making no distinction for different positions of the connection, say southern Europe or northern Europe.
Same holds for the usage of interconnecting branches. Here an overall average branch from the case is used.

Translating the results and conclusions to cases with an increased number of buses, no consistent theory can be drawn. Assuming that a system does not change significantly with an extra generator (of this size), leads to non optimized connections in the network with 118 buses. For now it is not known how a larger system would react. With 118 buses, the first 2 optimization branches have been predicted correctly, meaning that the system does not change significantly after the first connection. This analysis has to be repeated with a (slightly) bigger case in order to predict how the size of a system influences the responce, when a generator is attached. If in these cases (with increased number of buses), the number of correctly predicted connections increases. Predictions using the brute force method could lead to sufficient branch connections, meaning a huge decrease in computation time. With sufficient branch connections the breakeven point is meant where the investment of an extra connection does not compete with the reduction in operation costs.

However from this research no such conclusions can be made. Another aspect that could lead to a direct decrease in computation time, is a precisely known topology of the system. Then immedate nodes can be disregarded from the iteration process, when their distances are unrealistic.

From the costs breakeven analysis is concluded that in order for the generator to be economically desirable, the operation costs have to be lower then 38 \$/MWh (in the case with 118 branches). If economics is not the only criteria, a certain trade-off can be made using the plots shown in 4.6.3.

It has been found that the MATPOWER program is an easy accesable package for computing electrical power systems. Also codes has been manufactured in order to generate all the outcomes of this paper, which are not bound to a special case. They can therefore be reused and updated as desired (taking the note concerning the usage of branches into account!). A recommendation for further users is to verify how larger cases react to the added generator. If better predictions are the outcome, a huge amount of computation time can be speared using the brute force technique with a single cable translated to the number of breakeven connections. Compared to the fixed node iteration method for all the breakeven connections.

Bibliography

[1] Electrical power system essentials
Authors: P. Schavemaker & L. Van der Sluis
John Wiley & Sons 2008

[2] A Case Study in the Future Challenges in Electricity Grid Infrastructure
Authors: van den Akker et al.
March 12, 2012

[3] MATPOWER 4.1 User's Manual
Authors: Zimmerman & Murillo-Snchez
December 14, 2011

[4] NKT Cables 2014: High Voltage Cable Systems
`http://www.nktcables.com/support/download/catalogues-and-brochures/high-voltage-and-offshore/~/media/Files/NktCables/download%20files/com/HighVolt_e_200309.ashx`
viewed 28 April 2014

[5] Load Flow Computations in Hybrid Transmission - Distribution Power Systems
Authors: L.Wobbes & D. Lahaye
Februari 13, 2012

[6] AC Transmission in Power system stability and control
Author: P. Kundur
McGraw-hill 1994

[7] University of Washington: Electrical Engineering
`http://www.ee.washington.edu/research/pstca/`
viewed 29 April 2014