NTNU

Norwegian University of
Science and Technology

Design Optimization of Hydropower Generators

Truls Even Hestengen

Master of Energy and Environmental Engineering
Submission date: June 2016
Supervisor: Arne Nysveen, ELKRAFT
Co-supervisor: Erlend Engevik, ELKRAFT

Norwegian University of Science and Technology
Department of Electric Power Engineering

Preface

This Master's thesis is written as the final assignment of the five year long Master of Science program at the department of Electrical Power Engineering at the Norwegian University of Science and Technology. The report is centered around optimization of the synchronous generator where a genetic algorithm has been implemented in MATLAB to minimize the cost of the machine. The final result is intended to be used for education and research purposes.

I want to thank my supervisor, Arne Nysveen, who has been very helpful with clearing up aspects about the assignment and, in particular, the synchronous generator. I also want to thank PhD candidate Erlend Engevik who has made himself available throughout my assignment, whether concering MATLAB syntax or questions about the genetic algorithm. Furthermore, the work done on this assignment would not have been possible without the design calculation program made by former students Aleksander Lundseng and Ivar Vikan.

I want to thank my fellow Master's students at room E320; Anders Espeseth, Andreas Bock and Petter Christiansen. They have made each day of writing a bit more interesting and fun. Lastly, I want to thank my good friends Kailin Jones and Duncan Scott for assisting me with English spelling and grammar.

Date Trondheim May 25, 2016

Signature Tina Hestnes

Abstract

For education and research purposes, a design calculation program for hydropower generators has been made in MATLAB. The user can change design parameters to optimize the generator's cost and efficiency. However, manual optimization is a slow and tedious process and the user might make mistakes which will lead to incorrect results. By implementing an optimization in a computer program, this process will be made faster and more reliable. In this assignment, a genetic algorithm is implemented in MATLAB to optimize the synchronous generator with respect to the total cost of the machine, which includes cost of materials and machine-losses. The purpose of the optimization was to find the lowest machine-cost by varying chosen geometrical parameters. The variables were selected during conversations with supervisor Arne Nysveen and Ph.D. candidate Erlend Engevik.

MATLAB has embedded functions that can perform genetic algorithms when provided with objective function, number of variables and constraints. The design calculation program mentioned has been used actively and made the foundation for the objective function to the implemented optimization. It has been adjusted to meet the demands of objective functions for optimization problems in MATLAB. Furthermore, limits to the machine-design have been implemented as nonlinear constraints. These are gathered in a script under the function name *nonlcons*. For every set of variable values, the algorithm call this script to check if the constraints are met.

To test the implemented optimization, the price of load-dependent losses was increased to determine whether this made the algorithm reduce these losses. Furthermore, the power factor was increased in the ambition this would make the machine cheaper. Lastly, the synchronous reactance was increased. For high values of the synchronous reactance the air gap drops, which in turn led the total magnetization to reduce, and consequently the total cost of the machine decreased also. Ten optimizations were performed for each case in anticipation of variance in results from the genetic algorithm. The results were then gathered in Excel in order to comparatively examine how the algorithm had performed.

Results show that the optimization performs as expected. The load-dependent losses dropped by 8.1% when the cost of these increased from 15000 to 20000 kroner per kW. The cost of constant losses were at 30000 kroner per kW for both cases. They did not change considerably. When the power factor was increased from 0.8 to 1.0, the total cost of the machine dropped by 4.4%. This is because of a reduced magnetization of the machine that reduces the required field current and consequently the copper losses. For the case of a varying synchronous reactance, the magnetization of the machine reduced by 9.3% as the reactance moved from 1.2pu to 4.9pu. Consequently the total machine-cost decreased by 7.6%.

Test results from one optimization to the next also showed large variation between ideal solutions. It is believed that a larger population could bring down this variation. However, the genetic algorithm includes randomly generated mutation which for complex problems necessarily lead to varying results. Furthermore, the ratio between cost of losses and materials, set by the user affects machine-design to a great extent. Increasing the cost of losses while keeping the material costs constant causes the use of copper in the machine to increase significantly. The constraints added ensure that the algorithm designs reliable generators that fulfill demands of the Norwegian grid. In consideration of this impact on machine-design, it is important that the user can adjust these constraints according to the problem at hand.

Sammendrag

For undervisnings- og forskningsformål er et program som utfører designberegninger på vannkraftgeneratorer blitt utarbeidet. Brukeren av programmet kan endre designparametere for å optimalisere generatoren, men må gjøre dette manuelt. Dette er en treg prosess i tillegg til at brukeren kan gjøre feil som vil føre til gale resultater. Ved å implementere optimaliseringen i et dataprogram kan optimaliseringen gjøres raskere og mer pålitelig. I denne oppgaven er en genetisk algoritme implementert i MATLAB for å optimalisere synkrongeneratoren. Optimaliseringen er gjort med tanke på totalkostnad av maskinen hvor denne består av taps- og materialkostnad. Hensikten med optimaliseringen var å finne maskinen med lavest totalkostnad ved å endre valgte geometriske parametere. Variablene ble valgt i sammarbeid med veileder Arne Nysveen og doktorgradsstipendiat Erlend Engevik.

MATLAB har innebygde funksjoner som kan utføre genetiske algoritmer når objektivfunksjon, antall variable og begrensninger er gitt. Det nevnte beregningsprogrammet har blitt brukt aktivt og dannet grunnlaget for objektivfunksjonen som ble implementert. Programmet har blitt endret for å møte krav til objektivfunksjoner i optimaliseringsproblemer i MATLAB. Krav til maskindesign har blitt implementert som ulineære begrensninger i scriptet *nonlcons* med samme funksjonsnavn. For hvert sett av variable kaller algoritmen på denne funksjonen for å sjekke at kravene er møtt.

For å teste optimaliseringen ble prisen på lastavhengige tap økt for å se om algoritmen reduserte disse tapene. Videre ble effektfaktoren økt som burde føre til en lavere totalkostnad av maskinen. Til slutt ble synkronreaktansen økt. Høye verdier av synkronreaktans kommer av et lite luftgap mellom rotor og stator, og fører til at den totale magnetiseringen av maskinen reduseres og dermed en reduksjon av den totale maskinkostnaden. Ti optimaliseringer ble utført for hvert tilfelle da det er usannsynlig at algoritmen gir like svar hver gang. Resultatene ble samlet i Excel og sammenlignet for å kunne avgjøre hvor godt optimaliseringen fungerte.

Resultatene viste at optimaliseringen fungerte som forvenetet. De lastavhengige tapene gikk ned med 8.1% da prisen for disse økte fra 15000 til 20000 kroner per kW. Prisen for de lastuavhengige tapene var 30000 kroner per kW i begge tilfeller og endret seg lite. Da effektfaktoren økte fra 0.8 til 1.0 sank totalkostnaden av maskinen med 4.4%. Dette er på grunn av redusert nødvendig magnetisering av maskinen som fører til lavere behov for feltstrøm og dermed lavere kobbertap i rotor. For tilfellet med endret synkronreaktans sank magnetiseringen av maskinen med 9.3% da synkronreaktansen økte fra 1.2pu til 4.9pu. Dette førte til at maskinkostnaden sank med 7.6%.

Annet enn forventete resultater viste disse også at den beste løsningen fra hver optimalisering varierte stort. Det er sannsynlig at en økning i antall individer i populasjonen kan redusere denne variasjonen. Genetiske algoritmer inkluderer dog tilfeldig genererte mutasjoner som for komplekse problemer nødvendigvis medfører varierende resultater. Videre viste resultatene at forholdet mellom pris på tap og materiale satt av brukeren har stor betydning for maskindesignet. En økning av pris på tap fører til at bruken av kobber i maskinen øker betraktelig. Til slutt er designkravene viktige for å forsikre at algoritmen designer maskiner som møter krav til generatorer koblet til det norske nettet. Det er dog en forutsetning at brukeren er kjent med kravene og kan endre disse slik at de passer for problemet som skal løses da kravene er svært dimensjonerene for maskinen.

NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

NTNU

M A S T E R O P P G A V E

Kandidatens navn : Truls Even Hestengen

Fag : **ELKRAFTTEKNIKK**

Oppgavens tittel (norsk) : Optimalisering av vannkraftgeneratorer

Oppgavens tittel (engelsk) : Design optimization of hydropower generators

Oppgavens tekst:
For education and research purposes, we are developing a design calculation program for hydropower machines. So far, it uses manual optimization such that it is the user that finds the optimum design by changing design parameters. This is time-consuming and may lead to wrong results for inexperienced uses. The program calculates the machine rating and parameters using analytical formulas.

The purpose with this project is to establish a calculation tool that implements optimization algorithms on electrical machines. The objective will be to establish a MATLAB program that can perform a Genetic Algorithm on an optimization problem involving a synchronous machine. With such a tool, one can change a parameter or constraint value from one optimization to the next to get an understanding of how the parameter affects the optimal design.

More specifically the work shall focus on:

- Familiarize with hydropower generator design and optimization practice
- Examination of the existing design tool and selection of optimization parameters
- Evaluation and selection of a suitable genetic optimization algorithm
- Implement the optimization algorithm in the design calculation program
- Test and evaluate the optimization tool on selected cases

Further details of the work to be discussed with the supervisors during the project period.

Oppgaven gitt	: 15. januar 2016
Oppgaven revidert:	: 19. mai 2016
Besvarelsen leveres innen	: 10. juni 2016
Besvarelsen levert	:
Utført ved (institusjon, bedrift)	: Inst. for elkraftteknikk/NTNU
Kandidatens veileder	: Erlend Engevik, NTNU
Faglærer	: Professor Arne Nysveen

Trondheim, 19. mai 2016

Arne Nysveen
faglærer

Table of contents

Figures

Tables

1 Introduction

The first electromagnetic generator was made by Michael Faraday who developed the operating principle in the years of 1831-1832. The generator, called the Faraday Disk, produced a small DC voltage and was very inefficient. The first generator to be used in industry, the Dynamo, was made in 1844. It used electromagnetic induction to produce direct current with commutators. Later, the AC alternator, first two phase and then three phase, made its entry into the market. Today, the synchronous generator is by far the most common machine used for electricity production in the world. Though the main principles remained unaltered, the machine has evolved since its inception and small improvements increasing stability and efficiency continue to be made. For teaching purposes, a MATLAB program that performs the design calculations on synchronous generators from a set of input values has been made at NTNU. The program was made by former students during work on their Master's theses. To get an even better understanding of the complexity of the machine, it is in NTNU's interest to research if the generator design can be optimized with the use of computers. In this context, the following project was suggested: "The purpose of this project is to establish a calculation tool that implements optimization algorithms on electrical machines. The objective is to establish a MATLAB program that can perform a genetic algorithm on an optimization problem involving a synchronous machine. With such a tool, one can change a parameter or constraint value from one optimization to the next to get an understanding of how the parameter affects the optimal design."

Thus, the main objective of the assignment is to develop a MATLAB script that implements a genetic algorithm to optimize the design of the synchronous generator. The design script made by previous students is provided as a modelling tool, however the focus of this project will be directed more towards the optimization of the generator, not the design calculation itself. The process of designing a synchronous generator is very complex. Around 100 parameters have to be decided on, and the design involves both mechanical, electromagnetic and thermal calculations. A detailed understanding is outside the scope of this report.

This assignment will first go through necessary theory on the geometry and design calculations of the synchronous generator, before previous research made by other students is presented. The methodology section will then explain the work done to make the script, and how it will be used to inform the design of the synchronous generator. Results from the various tests are then presented, before being analysed in the discussion section. Lastly, the conclusion will reiterate the general findings of the study.

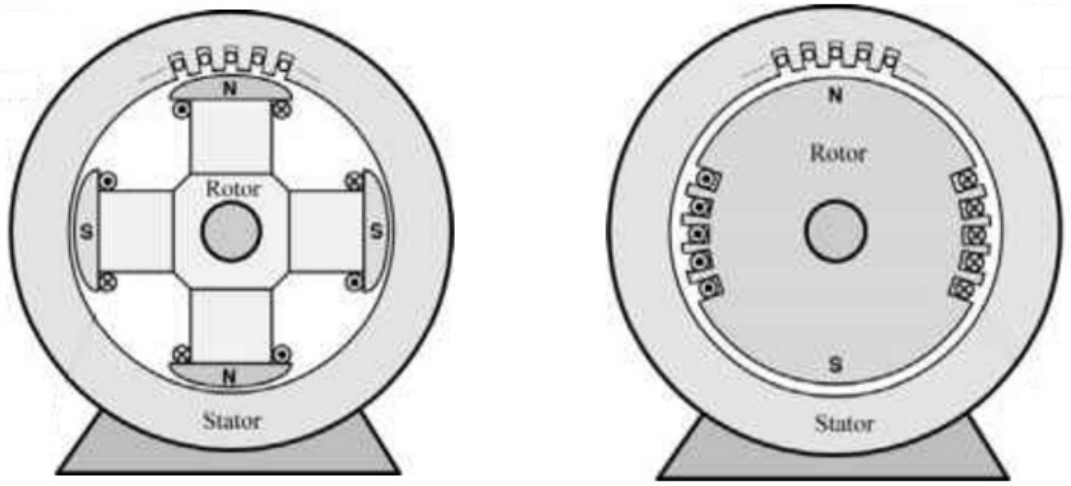
2 Theory

2.1 Synchronous generators

Synchronous generators are preferred in large power plants and produce the majority of the electricity in the world. While the induction generator can only consume reactive power, the synchronous generator has the capability to both produce and consume reactive power. In fact, an induction generator has to be connected to an external source of reactive power to maintain its magnetic field in the stator. As the grid is inductive by nature the ability of a synchronous generator to produce reactive power is a great advantage. However, the induction generators simple construction makes it a good option for windmills and other supplementary power sources connected to an existing power system. As for large hydropower plants, the synchronous generator can therefore be seen as a more advantageous choice.

2.1.1 Construction and design calculations

A synchronous generator consists of a stator and a rotor. The rotor is locked to the synchronous speed of the generator. The generator has a separate field circuit with a DC field current. This current produces a magnetic field that rotates at the same speed as the rotor. The magnetic field induces an AC voltage in the stator winding, in turn producing a lagging current flow when connected to a load. In some machines the rotor magnetic field is produced from permanent magnets, however that will not be discussed further in this report. Synchronous generators have either salient or non-salient poles. A non-salient pole means that the pole is embedded within the surface of the rotor, while a salient pole protrudes radially from the shaft of the rotor. Figure 1 shows the stator and rotor of synchronous generators with salient and non-salient poles. For machines with more than 4 poles, salient poles are the most common. Only these will be discussed further in this report.



(a) Salient poles

(b) Non-salient poles

Figure 1: Rotor and stator in synchronous generators [1]

When designing a synchronous generator, some dimension criteria has to be determined. The customer will decide on the number of generators, their power rating S_n, and power factor pf, also

referred to as $cos(\phi)$. The frequency f is given from the grid, which is 50Hz in Europe. The customer will also decide on an appropriate synchronous speed n_s measured in rounds per minute (rpm). The number of poles N_p can be calculated from Equation 1:

$$N_p = \frac{120f}{n_s} \tag{1}$$

The output voltage is usually not set by the customer. In Europe most generators are connected to a transformer, so it makes little difference for the customer what the output voltage is. Having the output voltage as a free parameter is also very convenient for the generator constructor.

The cost of losses in the machine are of great concern to the customer and therefore important when deciding on the material to be used in the stator core. The customer usually specifies the price of load-dependent and constant losses. It is in their interest to buy a machine with minimal losses while still meeting the dimension criteria. The cost of constant losses are commonly higher than load-dependent losses since the machine will not run at full load at all times.

The synchronous reactance X_d is of significance for the static stability of the generator and an upper limit is often specified by the customer. The transient reactance X'_d and sub transient reactance X''_d are less commonly set by the customer. In addition to demands from the customer, the generator constructor has to meet the formal requirements for rotating electrical machines, as described in IEC 60034-1 [6]. If demands from the customer are less strict than the standard, the standard requirements should be followed rather than demands from the customer. Listed are some of the types of requirements in the IEC 60034-1:

- Thermal tests are to be performed on the generator. This is to ensure that the winding will not overheat, which can lead to a breakdown of the insulation. Limits of temperature increase during testing are given in the standard.
- The harmonics in the generator should be minimized. Limits to the Total Harmonic Distortion (THD) are given in the standard.
- The generator should be able to run normally for thirty seconds with a current 1.5 times the nominal current. It should also withstand a short circuit current of twenty-one-times the nominal current [7].

In addition, the system operator often specifies requirements to the moment of inertia to ensure stability in the grid. This will be discussed further in section 4.5.

2.1.1.1 Voltage and current calculations

When the requirements are set one can start calculating the geometrical parameters of the generator. The inner diameter, D_i, is measured from the shaft center to the stator, including the air gap between rotor and stator. It can be set by the customer or calculated from empirical formulas. The gross iron length, l_b, is also calculated from empirical formulas if it is not given. From D_i and l_b, the utilization factor C can be calculated using:

$$C = \frac{S_n[kVA]}{D_i^2 \cdot l_b \cdot n_s} \tag{2}$$

where S_n is apparant power in kVA. If the gross iron length is not yet known, an empirical formula for the utilization factor freqently used is:

$$C = 0.02 S_n[MVA] + 5.6 \tag{3}$$

Next, the number of stator slot, Q_s, is decided. The stator is often made out of sections of laminated steel put together to form a circle. The circles are placed after each other to the desired length, with air gaps in between for cooling. The gross iron length l_b includes air gaps between the steel plates. A typical section is shown in figure 2.

Figure 2: Lamination in stator [2]

Q_s has to be dividable by the number of phases and by 2 to obtain balance in the machine. Often, the number of slots per pole is calculated using:

$$\frac{Q_s}{N_p} = \frac{F \cdot N}{F \cdot U} \tag{4}$$

where F is the largest common multiplier. N is the number of slots in which a winding repeats itself and the number of slots per pole pair. U is a factor depending on F and N. The number of parallel circuits pnr in the machine can be chosen as a factor of F. If F is 8, then pnr can be 1, 2, 4 or 8. Further, the slot pitch τ_s can be calculated as:

$$\tau_s = \frac{\pi \cdot D_i}{Q_s} \tag{5}$$

The slot pitch is the arc length from one slot to the next and can be divided in slot width b_s and tooth width b_t. Usually, the ratio b_s/b_t is within 0.55-0.75. The pole pitch τ_p is also calculated as:

$$\tau_p = \frac{\pi \cdot D_i}{N_p} \tag{6}$$

An estimated armature loading A_s can be calculated:

$$A_s = \frac{C \cdot 60 \cdot k_m \cdot 10}{\pi^2 \cdot k_f \cdot f_w \cdot \frac{f_j}{1+b_s/b_t} \cdot B_t} [At/cm] \tag{7}$$

where

- k_m is the ratio between maximal and average flux density, often set as 1.515
- k_f is constant equal to 1.11
- f_w is the winding factor, in average around 0.925
- f_j is the ratio between between net and gross iron length, which is usually within 0.80-0.85
- B_t is the maximum tooth flux density when the generator is running at no load

From the armature loading, the nominal current can be calculated:

$$I_n = \frac{A_s \cdot \tau_s \cdot pnr}{2} \tag{8}$$

The nominal voltage is then calculated as:

$$U_n = \frac{S_n}{\sqrt{3} \cdot I_n} \tag{9}$$

2.1.1.2 Stator calculations

Next, the slot dimensions are calculated. A double layer winding with Roebel bars is shown in figure 3a with the geometrical parameters to be decided. The Roebel bar consists of a large number of individual braided strands connected in parallel. This is to improve the current distribution in the stator windings. Furthermore, the strands are intertwined so that they are both at the bottom and the top of the bar during the length of the machine. This is shown in figure 3b. The rotor magnetic field is stronger closer to the rotor. As the top of the bar is closer to the rotor than the bottom, a higher current will flow at the top than the bottom. Intertwining the strands will minimize this effect. For a double layer winding there are two bars in every slot, belonging to two different windings. For a full turn in the machine a winding is placed both at the top and the bottom of the slot for the same reason as the strands are intertwined.

If b_s is already decided from the ratio between b_s and b_t, the slot height h_s has to be adjusted to minimize losses and overheating. From an empirical formula, the allowed armature loading can be calculated from:

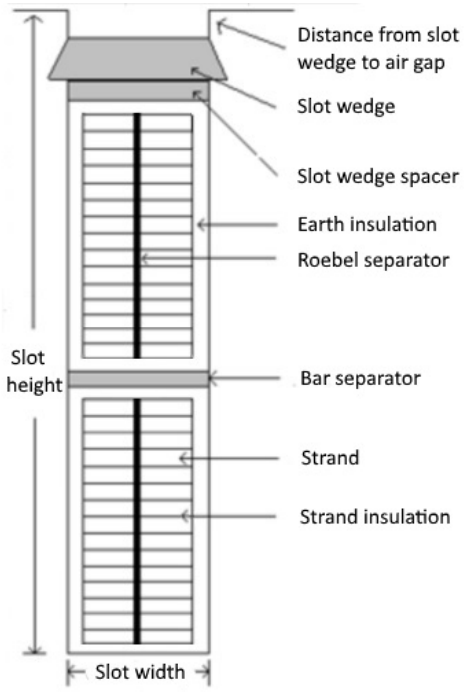
$$\frac{W}{cm^2} = (0.135 - 0.003 \cdot U) \cdot (1 + 0.002(\Delta T - 60)) \tag{10}$$

where U is measured in kV and ΔT is allowed temperature rise in Celsius. The insulation thicknesses are decided from the nominal voltage. The cobber width in each slot can now be calculated from knowing both slot width and insulation thickness. The actual armature loading can be calculated as:

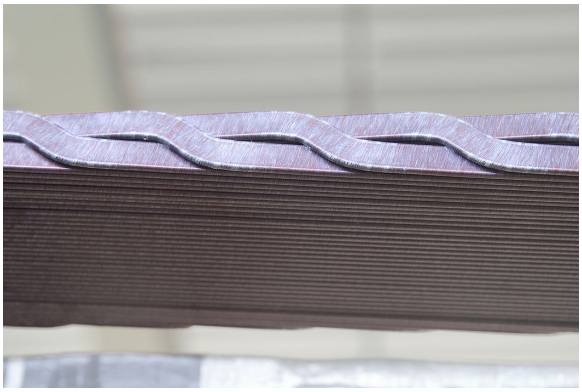
$$\frac{W}{cm^2} = \frac{2.1 \cdot 10^{-6} \cdot I_c^2}{A_{cu,s}(b_s + h_s - h_k)} \tag{11}$$

where

- I_c is the current in each bar, $I_c = I_n/pnr$,
- $A_{cu,s}$ is the cobber area in each bar



(a) Slot parameters [7]



(b) Intertwining of a Roebel bar [8]

Figure 3: Roebel bar

- h_k is the yoke height

Iteration on the slot height is normally performed to find an appropriate value that satisfies the armature loading limit. The current density should also be calculated to check if this is too high. Normally this will be between 2 [A/mm^2] and 5 [A/mm^2].

The next objective is to find the outer diameter. First, the distribution factor K_d is calculated:

$$K_d = \frac{\sin\frac{m\cdot\beta}{2}}{m\cdot\sin\frac{\beta}{2}} \tag{12}$$

where

- m is the number of slots per pole per phase
- β is the angular displacement between slots, $\beta = 180°\cdot N_p/Q_s$

Next, the pitch factor K_p should be chosen to minimize the 5th and 7th harmonics. This is done most efficiently when the coil span is around 0.80-0.85 times the pole pitch. This ratio is called relative pole pitch and will be denoted as y:

$$K_p = \cos(y\cdot\frac{\pi}{2}) \tag{13}$$

In this report, skewing of the coils will not be considered, and so the skewing factor can be omitted when calculating the total winding factor f_w:

$$f_w = K_d\cdot K_p \tag{14}$$

The fundamental flux component Φ_m can now be calculated from:

$$\Phi_m = \frac{E_f}{4\cdot k_f\cdot f_w\cdot f\cdot N_s} \tag{15}$$

where N_s is the number of turns in series per phase and E_f is the induced phase voltage when the generator is running at no load. If not yet decided, the net iron length l_n can now be calculated from first finding the air gap length δ between rotor and stator. An empirical formula often used is:

$$\delta = \gamma\cdot\tau_p\cdot\frac{A_s}{B_\delta} \tag{16}$$

where γ is an empirical factor and B_δ is the maximal flux density in the air gap at no load, from now assumed to be 0.95T. The net iron length is found from:

$$l_n = \frac{\phi_m\cdot k_m}{0.95\cdot\frac{b_t}{\tau_s}\cdot B_t\cdot\tau_p} \tag{17}$$

where the number 0.95 is an iron fill factor. Next, the yoke height h_k can be calculated as:

$$h_k = \frac{\phi_m}{2\cdot 0.95\cdot l_n\cdot B_d} \tag{18}$$

where B_d is the yoke flux. Lastly, the outer diameter can be calculated:

$$D_y = D_i + 2\cdot(h_s + h_k) \tag{19}$$

2.1.1.3 Armature reaction and reactances

When current is flowing in the stator windings, the stator magnetic field will produce armature reaction voltages in the stator. The armature reaction is modelled by the armature reaction reactance X_{ad}. For salient pole machines the armature reaction acts differently for the direct-axis and quadrature-axis. This means that we get different armature reactances for the two axis. It is desirable to adjust the air gap in the machine so that the direct axis synchronous reactance is around 1.1pu. First, the general armature reaction reactance is found. To calculate this, the maximum ampere turns because of armature reactance at rated load is found:

$$F_a = \frac{2.7 \cdot f_w \cdot I_n \cdot N_s}{N_p} \tag{20}$$

Also the magnetic voltage drop in the air gap is calculated:

$$F_\delta = \frac{B_\delta \cdot \delta_{0e}}{1.256 \cdot 10^{-6}} \tag{21}$$

where δ_{0e} is the equivalent air gap length. The equivalent air gap length is found by multiplying the actual air gap length with the total Carter coefficient. The armature reaction reactance for the d- and q-axis can now be calculated as:

$$X_{ad} = \frac{F_a}{F_\delta} \cdot k_d \tag{22}$$

$$X_{aq} = \frac{F_a}{F_\delta} \cdot k_q \tag{23}$$

The second part of the synchronous reactance comes from the leakage reactance in the stator X_l. This reactance consists of four parts:

$$\begin{aligned} X_l = 6.54 \cdot 10^{-7} \cdot \frac{F_a l_b}{\Phi_m f_w^2} \cdot \left(\frac{N_p}{Q_s(3y+1)} \cdot \frac{(h_s + d_{icu} + d_{ij} + 2h_{spk})}{b_u} + \frac{3.6}{N_p}(3y+1)\frac{D_i}{l_b}\right) \\ + 1.1 \cdot \frac{F_a}{F_\delta}\left(\frac{N_p}{Q_s}\right)^2 + \left(\frac{N_p \delta_{me}}{D_i}\right)^2 \cdot \frac{X_{ad}}{2} \end{aligned} \tag{24}$$

where δ_{me} is the average air gap length. The first part is because of leakage flux around the slots in the stator. The second part gives the reactance from leakage flux around the end windings. From slot harmonics in the air gap, so called zigzag reactances, we get the third part. Lastly, the fourth part is because of over harmonics in the air gap or belt leakage reactances.

Finally, the synchronous reactance for the direct axis can be calculated:

$$X_d = X_{ad} + X_l \tag{25}$$

2.1.1.4 Rotor calculations

When the stator calculation are done, one can start calculating the parameters in the rotor. First, the damper windings are designed, which are important for the stability of the generator. Figure 4 shows how the damper windings are placed in the rotor. The damper windings are shorted in both ends so that it forms a cage. When the rotor is pulled into synchronism, voltage will be induced in the windings, leading to a current flow that sets up a magnetic field. This field connects to the stator field producing rotor torque. When the rotor reaches synchronous speed, no voltage will be induced since the rotor has the same speed as the magnetic field from the stator windings. No current will then flow in the damper windings. However, if the load is changing quickly, the rotor speed might drop or increase momentarily. Again current in the damper windings will set up a magnetic field

to pull the rotor back into synchronism. To reduce harmonics, the damper winding span τ_r should not be the same length as the pole pitch. Normal values for τ_r are 0.82 - 0.88 or 1.15 times the pole pitch.

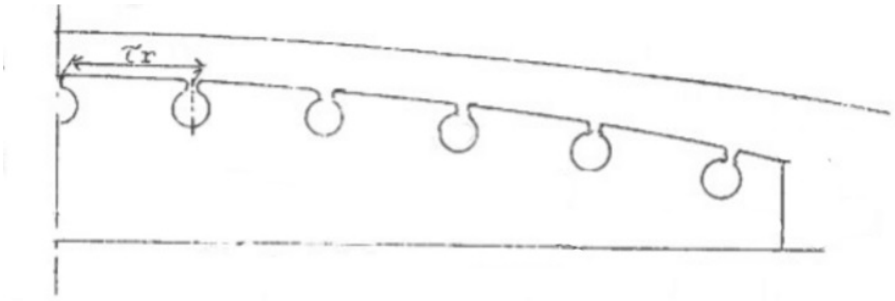


Figure 4: Damper windings [2]

Further, rotor parameters as the pole shoe height and width are decided. The pole shoe height is often set as 5cm, while the width is chosen to fit with the pole pitch.

The magnetic calculations are then done. First, the magnetic voltage drops are calculated. The magnetic voltage drop over the air gap between stator and rotor is found from:

$$U_{m,\delta} = \frac{2B_\delta}{\pi\mu_0} \cdot \delta_{me} \tag{26}$$

where μ_0 is permeability in free space. The magnetic voltage drops in slots and yoke are found by integrating over the magnetic H-field. After finding the voltage drops, one can calculate the leakage inductances in the machine. The total leakage inductance L_σ omitting the skew leakage inductance is:

$$L_\sigma = L_\delta + L_u + L_d + L_w \tag{27}$$

where

- L_δ is the air gap leakage inductance
- L_u is the leakage inductance in the slot
- L_d is the stator tooth leakage inductance
- L_w is the leakage inductance in the end windings

The total magnetization of the machine is then calculated as:

$$\Theta_{mN} = |E_f| \cdot U_{m,tot} \cdot 1.1 \tag{28}$$

where $U_{m,tot}$ is the total magnetic voltage drop and 1.1 is included to account for saturation in the machine. After the magnetic calculations are done, the field winding can be designed. An allowed field loading can be calculated from an empirical formula:

$$\frac{W}{cm^2} = (0.22 + 0.0055\frac{\pi D_i n}{60})[1 + 0.016(\Delta T - 60)] \tag{29}$$

where ΔT is the allowed temperature rise. Some parameters now have to be set:

- end plate height h_{kr}
- insulation thickness between conductors in the field winding b_{if} and between field windings and the pole b_i

- temporary field winding width b_{cuf}
- temporary end plate thickness L_e

An empirical ratio between the average and maximum winding length is given as:

$$\frac{L_{fmd}}{L_{fmx}} = 0.935 \tag{30}$$

From this the winding height can be calculated:

$$h_f[cm] = \sqrt{\frac{2.1{\cdot}10^{-6} \cdot \frac{L_{fmd}}{L_{fmx}}}{\frac{W}{cm^2} \cdot 0.85{\cdot}b_{cuf}} \cdot \Theta_{mN}} \tag{31}$$

The pole core height is then:

$$h_{pk} = h_f + 2h_{kr} \tag{32}$$

and the necessary copper area:

$$A_f n_f = \frac{2.1{\cdot}10^{-6}{\cdot}\Theta_{mN}^2}{\frac{W}{cm^2} h_f} \cdot \frac{L_{fmd}}{L_{fmx}} \tag{33}$$

where A_f is the cross section area of one conductor in the field winding and n_f is the number of conductors. In figure 5 the pole is illustrated. The left figure is a front view of the pole, while the right figure shows the pole from the side. Other than the parameters already mentioned, h_{cuf} is the conductor height, L_{pk} is the pole core length and L_p is the total pole length, including the end plate. One important thing to notice is the windings that are wider than the rest. These are constructed in this manner to increase the surface area of the field winding, which considerably increase the cooling of the rotor and decrease the temperature.

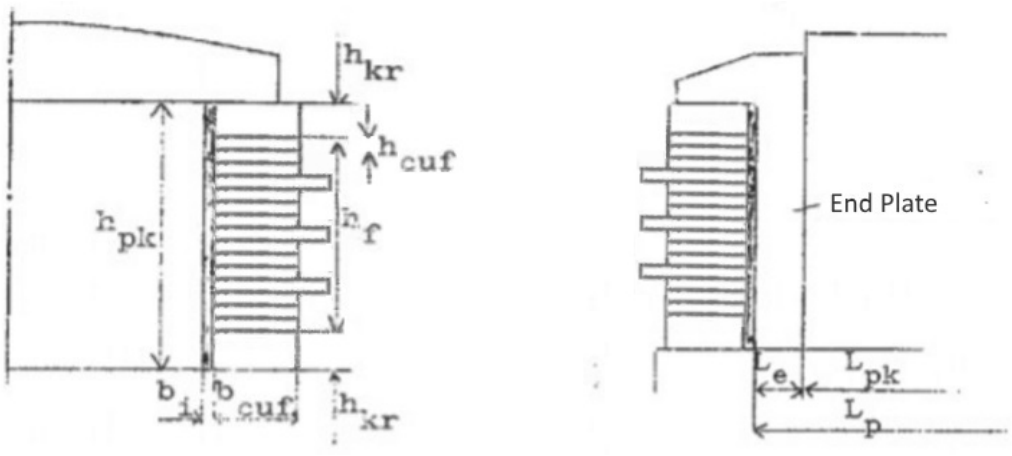


Figure 5: Rotor parameters [2]

To find the number of conductors in the field winding, the formula for field voltage V_f can be rearranged:

$$n_f = \frac{V_f{\cdot}(A_f n_f)}{2.1{\cdot}10^{-6} L_{fmd}{\cdot}N_p{\cdot}\Theta_{mN}} \tag{34}$$

Further, the conductor height is found:

$$h_{cuf} = \frac{h_f - b_{if}(n_f - 1)}{n_f} \tag{35}$$

A new field winding width can now be calculated:

$$b_{cuf} = \frac{A_f n_f}{n_f * h_{cuf}} \cdot 1.03 \tag{36}$$

where the factor 1.03 is added to compensate for rounded corners on the conductors. A_f can now be found:

$$A_f = h_{cuf}[b_{cuf} - (1 - \frac{\pi}{4})h_{cuf}] \tag{37}$$

The actual field loading is then calculated from:

$$\frac{W}{cm^2} = 2.1 \cdot 10^{-6} \frac{\Theta_{mN}^2}{(A_f n_f)h_f} \cdot \frac{L_{fmd}}{L_{fmx}} \tag{38}$$

which should be less than the allowed field loading to prevent the field winding from overheating. Lastly the current density in the field winding is found:

$$S_f = \frac{\Theta_{mN}}{A_f n_f} \tag{39}$$

2.1.1.5 Losses

The machine's geometrical parameters are now set, and the losses can now be estimated. Copper losses are related to heat produced by the current in stator and rotor windings. In the stator windings it should also be accounted for skin effects since AC-current is flowing here. When R_{ac} is the AC-resistance in the stator, the stator copper losses are found from:

$$P_{cu,stator} = 3 \cdot R_{ac} \cdot I_n^2 \tag{40}$$

The field winding only carries DC-current so no skin effect needs to be taken into account:

$$P_{cu,rotor} = R_f \cdot I_f^2 \tag{41}$$

where R_f is the field resistance and I_f the field current.

In addition to copper losses in the rotor and stator there will be stray load losses. These losses consist of amongst others eddy current losses in the copper, able to be calculated fairly accurately. Other components of stray losses however, are harder to estimate. These are losses in the stator teeth due to harmonics in the air-gap field, pole shoe losses, and other minor losses. A calculated estimate shows these additional losses as being between 0.1 and 0.2 percent of the delivered power:

$$P_{add} \approx 0.0015 S_n \cos\phi \tag{42}$$

Next, the iron losses are calculated. For the stator yoke, the iron losses are found from:

$$P_{Fe,y} = k_{Fe,y} \cdot P_{10} \cdot \frac{B_{ys}^2}{1T} \cdot m_{Fe} \tag{43}$$

where

- $k_{Fe,y}$ is a factor implemented to include the effect of saturation in the yoke
- P_{10} is specific loss for steel in [W/kg]
- B_{ys} is yoke flux density

- m_{Fe} is the stator mass excluding teeth and copper

Similarly, iron losses in the stator teeth can be calculated:

$$P_{Fe,t} = k_{Fe,t} \cdot P_{10} \cdot \frac{B_{ts}^2}{1T} \cdot m_{ds} \tag{44}$$

where

- $k_{Fe,t}$ is a factor implemented to include the effect of saturation in the teeth
- B_{ts} is the average tooth flux density
- m_{ds} is the mass of the stator teeth

B_{ts} can be found by calculating the average of the maximum and minimum tooth flux density.

The mechanical losses from moving parts in the machine are now found. These losses are mainly from friction, fans and bearings. The fan and bearing losses are difficult to calculate and should be measured for every machine. The losses can be estimated from an empirical formula:

$$P_{fr} \approx k_v \cdot D^3 \cdot n_s^2 \cdot \sqrt{l_b} \cdot 10^{-5} [kW] \tag{45}$$

where k_v is a machine dependent constant.

The friction losses are meant as losses from friction between the rotor and the medium around. The surface losses can be calculated from:

$$P_{\rho w1} = \frac{1}{32} \cdot k \cdot C_M \cdot \pi \cdot \rho \cdot \Omega^3 \cdot D_r^4 \cdot l_r \tag{46}$$

where

- k is a surface constant that indicate the roughness of the surface
- C_M is moment of inertia constant calculated from measurements and Reynolds number
- ρ is the density of the cooling medium
- Ω is the mechanical speed of the rotor
- D_r is the rotor diameter
- l_r is the rotor length

The losses at the ends can be found from:

$$P_{\rho w2} = \frac{1}{64} \cdot C_M \cdot \rho \cdot \Omega^3 \cdot (D_r^5 - D_{ri}) \tag{47}$$

where D_{ri} is the shaft diameter. The mechanical losses are often added together and called friction and windage losses:

$$P_{fw} = P_{fr} + P_{\rho w1} + P_{\rho w2}$$

If the generator has an external DC-generator to supply the field circuit, the magnetizing losses can be set as 15% of the rotor copper losses. For the case with static magnetization 7% of the rotor copper losses is sufficient.

The total losses can now be added together:

$$P_{tot} = P_{cu} + P_{add} + P_{Fe} + P_{fw} + Pmagn \tag{48}$$

2.2 Optimization problems

The objective of an optimization problem is to find the best possible solution from all feasible solutions. Since the aim often is to decrease the cost of production, most cases of optimization problems are minimization problems, although maximization and minimization are two versions of the same. Generally, we would write optimization problems in this form:

$$\underset{x}{\text{minimize}} \quad f(x)$$

where *f(x)* is the function we want to minimize and x is the variable or a vector of several variables. If a solver that finds the minimum of a problem were to be used on a maximization problem, we can go about easily this by writing the problem as:

$$\underset{x}{\text{minimize}} \quad -f(x)$$

which would maximize $f(x)$.

The function *f(x)* often contains local optima in addition to the global optimal solution. These are less optimal solutions than the global optima. The sample function *peaks* in MATLAB is plotted in figure 6. On inspection one can clearly see there are three local minima. One of these is the global minima and the optimal solution to the solution space if we were to find the minimum value.

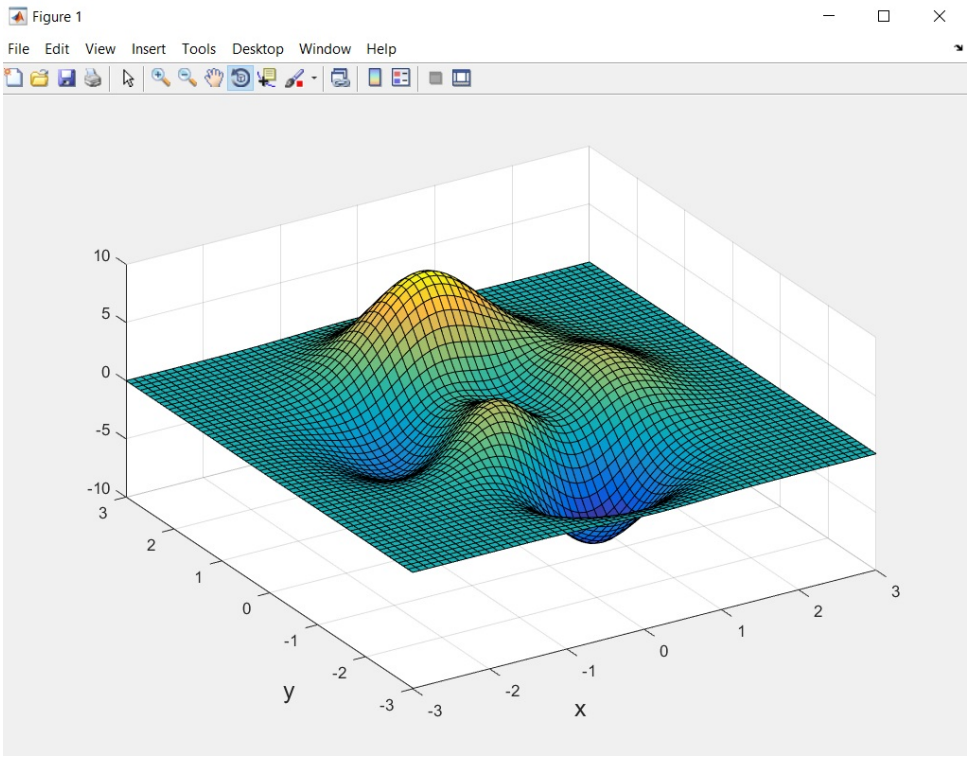


Figure 6: Peaks, MATLAB

2.2.1 Bounds and constraints

Often one are only interested in finding the best solution within a certain area of the solution space. In some cases one often has an idea of where the global optima is located before the algorithm is performed. We can then create bounds in which the algorithm will search inside of. This will decrease the search area and consequently reduce the time needed to find the optimal solution. If one were to find the global minima for the *peaks*-function in figure 6, it would be wise to create bounds to reduce the search area. A lower bound at -3 and an upper bound at 3 for both the x- and y-variable would work well in this case. However, it is important not to create too narrow of bounds that the solution we are looking for is left outside the search area for the algorithm. The lower and upper bound will later be denoted by lb and ub. A common way to specify bounds is to treat lb and ub as vectors of length n, where n is the number of variables. One can then write our bounds as $[lb;ub]$.

In other cases, some of the variables may have other requirements that must be taken into account. For example, if one of the variables in figure 6 was distance or time, then only positive values would make sense in a physical world. To meet this condition we create a lower bound at 0 for that variable. In some cases this will leave out the optimal solution for the unbound problem, but as we are only interested in physically possible solutions, it will give us the best feasible solution. Solutions outside bounds or constraints are called non-feasible solutions.

Some variables may only take integer values. For example, if a firm produces tables and chairs and wants to find the optimal amount of units produced to maximize revenue, only integer values would be functional. The genetic algorithm in MATLAB works well with integer value problems although it has difficulties with simultaneously dealing with integer and equality constraints.

In more complex problems we have other constraints that must be taken into account. These can be linear and nonlinear equalities and inequalities. The linear constraints are written as:

$$\begin{aligned} &A \cdot x \leq b, \\ &Aeq \cdot x = beq \end{aligned}$$

where A is a matrix $m \cdot n$ with m number of linear inequalities and n variables, b is a vector of length m, Aeq is a matrix $p \cdot n$ with p number of linear equalities and beq is a vector of length p. Often, constraints have to be implemented as less than or equal constraints. We can implement a greater than or equal constraint by multiplying with -1 on each side of the inequality. Meaning, if a constraint $A_i \cdot x \geq b_i$ is to be implemented we can rewrite the constraint as $-A_i \cdot x \leq -b_i$. The nonlinear constraints are written as follows:

$$\begin{aligned} &c_i(x) \leq 0, \ i = 1, \ldots, q \\ &ceq_i(x) = 0, \ i = q+1, \ldots, qt \end{aligned}$$

where $c(x)$ is the nonlinear inequalities, $ceq(x)$ is the nonlinear equalities, q is the number of non-linear inequalities and qt is the total number of nonlinear constraints.

We can write the full optimization problem as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & A \cdot x \leq b, \\ & Aeq \cdot x = beq \\ & c_i(x) \leq 0, \ i = 1, \ldots, q \\ & ceq_i(x) = 0, \ i = q+1, \ldots, qt \\ & lb \leq x \leq ub \end{aligned}$$

We call the set of all possible solutions that satisfy all constraints the *feasible region* or a *solution space* to a problem. Figure 7 shows the feasible region of a problem with one variable x and three linear constraints. If we were to maximize y, the best possible solution in the feasible region would be within the circle illustrated.

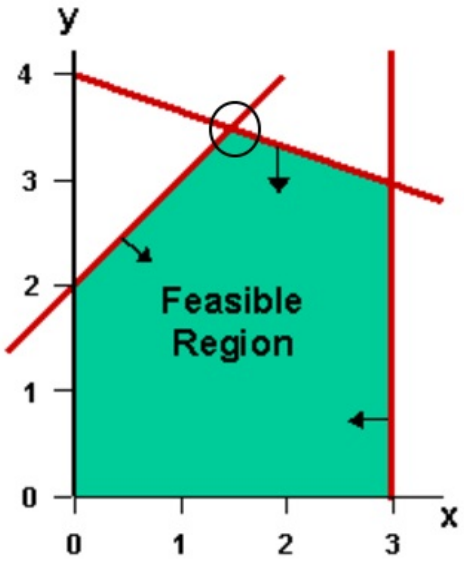


Figure 7: Feasible region for an optimization problem [3]

2.2.2 Solving an optimization problem

The first step to solve an optimization problem is to define the problem with its objective function, bounds and constraints. Secondly, an optimization method should be selected based on the problem type. Sophisticated methods might work better or even be essential on complicated problems, but are likely to be slow on less complicated problems. For each optimization problem the best technique is the one that finds the global optima in the shortest amount of time. In its simplest form an optimization problem could look like this:

$$\underset{x}{\text{minimize}} \quad f(x) = (x-1)^2$$

which would not even require an optimization method to be performed. The optimal solution is clearly $x = 1$ which gives an optima of $f(1) = 0$. For a slightly more complex problem like:

$$\underset{x}{\text{minimize}} \quad g(x) = x^4 - 3x^3 + 1$$

one can find the derivative of the objective function and its zeros. Still there is no need for sophisticated optimization methods. As problems get more complex with several variables and linear and nonlinear constraints, the use of optimizing methods becomes more and more essential. Listed below are some techniques commonly used

- Simplex algorithm
- Combinatorial optimization

- Newton's method
- Local search
- Pattern search
- Evolutionary algorithms
- Simulated annealing
- Particle swarm optimization

Some of these would take too long to do by hand and has to be implemented by a computer to reduce the computational time. Simulated annealing and evolutionary algorithms both perform a high number of function evaluations that make them very difficult to use without a computer. However, it makes them thorough and they are commonly used on problems with non-differentiable objective functions.

Lastly, the optimization technique should be performed on the problem. For a great deal of these techniques, they will not perform properly unless they are adjusted correctly to the problem. For the genetic algorithm, this is explained in detail in section 2.3.2. Sometimes it may be necessary to run the optimization several times as it might not give the same result every time. This is often the case for techniques that use random or semi-random numbers in its algorithm.

2.3 Genetic algorithm

A genetic algorithm is one out of many techniques used to solve optimization problems. Its name comes from Charles Darwin's theory of natural selection, as it involves a crossover, inheritance, selection, and mutation. Genetic algorithms are a part of Evolutionary algorithms that must also follow the principles of: *reproduction*, *natural selection* and *diversity*. From a population of several individuals or function evaluations, a genetic algorithm will pick the most fit individuals to move onto and reproduce the next generation. In a minimization problem, the most fit individuals would have a gene or variable composition that leads to the lowest function evaluation. Meaning, the most fit individual would have the lowest fitness value. Because of the mutation and high number of function evaluations in each generation, the algorithm works well on complex optimization problems.

As in every optimizing method, there is no guarantee that a genetic algorithm will find the global optima. It is wise to run the algorithm several times as it will not give the same answer every time. This comes from the randomly performed mutations. The high number of function evaluations also makes the algorithm slow on simpler problems, where less sophisticated methods are better off. However, the evolutionary process sometimes leads to unimaginable solutions. An example of an unlikely solution found using genetic algorithms is an antenna created by NASA for the ST5 mission in 2006, illustrated in figure 8. The mission was created to identify, develop, build, and test innovative technologies and concepts for use in future missions. NASA stated the ST5-3-10 antenna was 100% compliant with the mission antenna performance requirements, and was confirmed by testing the prototype antenna in an anechoic test chamber at NASA Goddard Space Flight Center [9].

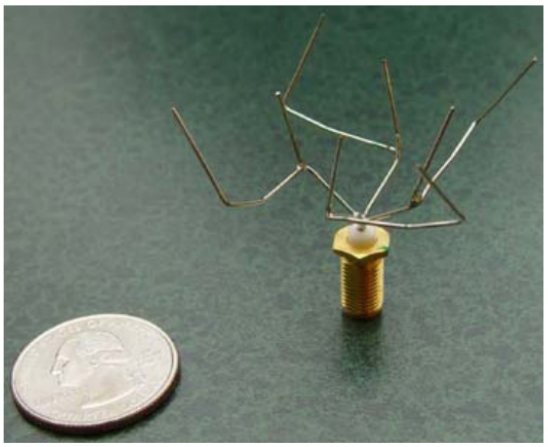


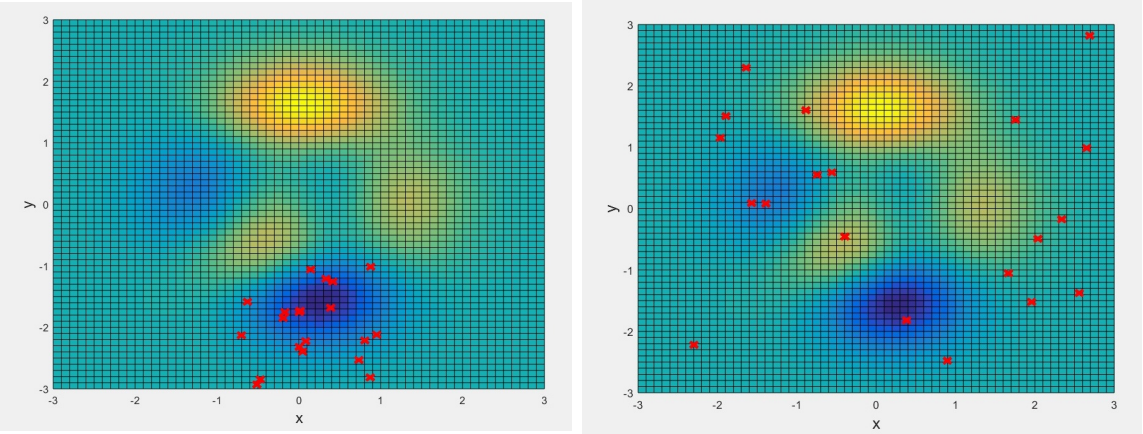
Figure 8: Evolved antenna ST5-3-10 [4]

We call the 3D-plot in figure 6 a *fitness landscape* of a problem, where the fitness is the height of the landscape. The algorithm ranks the individuals by their fitness value, and so a fitness landscape is helpful to visualize the relationship between genotypes and reproductive success. Figure 6 illustrates the fitness landscape of a problem with two variables. For problems with more than two variables, we cannot illustrate the fitness landscape in a simple 3D-plot. The terms fitness landscape and solution space still make sense and are commonly used for problems with more than two variables, although we cannot picture what they would look like. A common problem for optimization techniques is their tendency to get stuck in local optima. This may lead to an early termination of the algorithm, and consequently a less optimal solution. This can also be the case for the genetic algorithms and depends on the fitness landscape.

2.3.1 Initializing

The algorithm begins by randomly selecting a population within the bounds of the problem. It is possible to set an initial population range in MATLAB. This will cause the initial population to take values within a specific range. If we know before the algorithm is performed where the best solution is likely to be, we can possibly decrease the run time by setting an initial population range

around this point. Figure 9 shows the *peaks*-function in a 2D-plot and the first generation in the genetic algorithm. The color shading implies different function values, blue and yellow meaning low and high values as in figure 6. The red dots are individuals that makes up the population in this first generation. In figure 9a, the initial population bound is [-1 -3; 1 -1]. In this case, the initial population is selected randomly within these bounds, while in case (b), the population is randomly selected from the full solution space. The initial population bounds are selected to include the global minima. Clearly, case (a) has more individuals closer to the global optima than case (b).



(a) Initial population range

(b) No initial population bound

Figure 9: Initial population

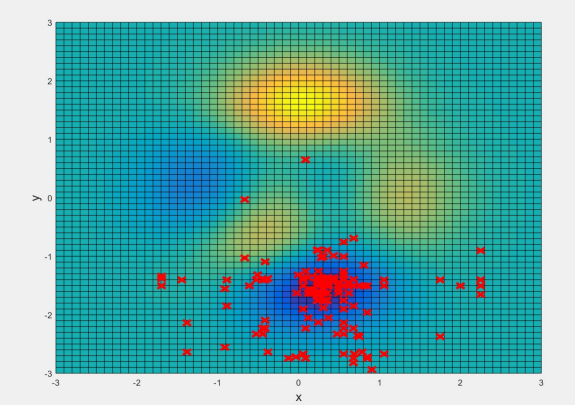
By defining suitable bounds for the initial population we tell the algorithm to evaluate points close to where we think the optimal solution is. This will make the algorithm concentrate more on this area than the rest of the solution space and possibly reduce the number of generations and function evaluations performed. Caution must be used when setting bounds to ensure that the optimal solution is included. Otherwise, the algorithm may terminate in a different place than the optimal solution.

Figure 10 shows the optimization problem after the algorithm has terminated in three different cases. Case (a) begins with an initial population located around the global optimum as in figure 9a. The global optima is found and the algorithm terminates. It is worth noting that in case (a),areas outside the initial population bounds are evaluated as well. The initial population bounds do not apply after initializing, and so the algorithm starts looking for solutions within lb and ub. However, in this case these individuals are not likely to move on to the next generation as there are more fit individuals closer to the global optima.

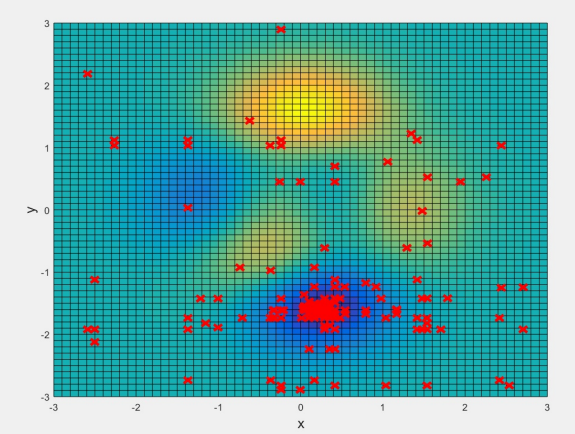
Case (b) has no initial population bounds. Although the algorithm finds the global optima, points from all over the solution space are evaluated. Many of these points are far away from the global optima, and the algorithm spends some time finding the relevant area. The run times for case (a) and (b) were approximately the same, reason being the simplicity of this problem. Despite the lack of initial population bounds, the algorithm very quickly finds where to look for the global optima. For more complex problems, significant time can be saved by defining initial population bounds.

In case (c), the initial population bounds are [-3 1; -1 3]. The algorithm quickly moves out of its start point and towards a local optima. However, it never makes it to the global optima, and instead it get stuck and terminates at the local optima. As the algorithm finds this area of low fitness values, it starts evaluating many points close by, all of which have lower fitness values than the surrounding areas. A lucky mutation could help the algorithm find the global optima, but as the mutation is randomly generated that will not happen every time.

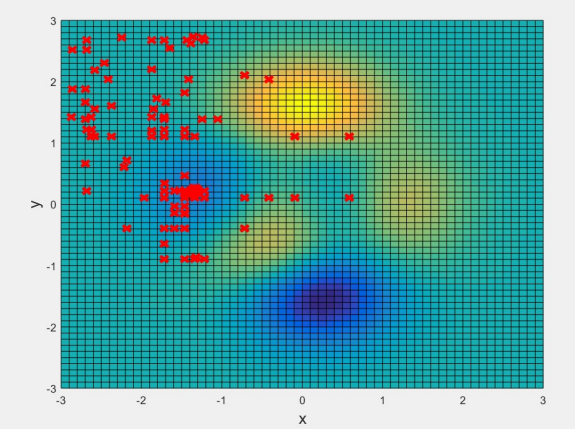
In conclusion, applying well defined initial population bounds can possibly decrease the run time of an optimizing problem. It also increases the algorithm's chances of finding the global optima. It is not given that the algorithm will find the global optima every time. Case (a) has a greater chance of success than case (b). Lastly, poorly placed bounds increases the chances of the algorithm getting stuck in local optimas.



(a) With initial population bounds



(b) No initial population bounds



(c) Poorly placed initial population bounds

Figure 10: Fully performed Genetic Algorithm

2.3.2 Crossover and mutation

After the first generation is produced, the genetic algorithm ranks all the individuals by their fitness. The fitness is usually calculated from the objective function, which is then called fitness function. In a minimization problem the algorithm will rank the fittest individual as the one with the lowest fitness value. The fittest individuals are selected as *elite children*. These will automatically survive to the next round. The number of elite children is called *elite count*. By having at least 1 elite child we make sure that the best fitness value can only decrease from one generation to the next. In other words, the most fit individual will always survive. Setting the elite count to a high number will make the algorithm slow since a greater part of the population survives. The default elite count in MATLAB is 5% of the population size.

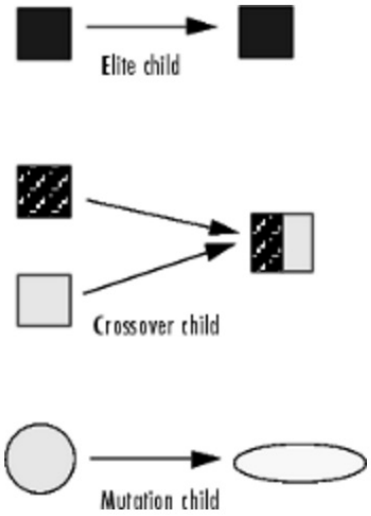
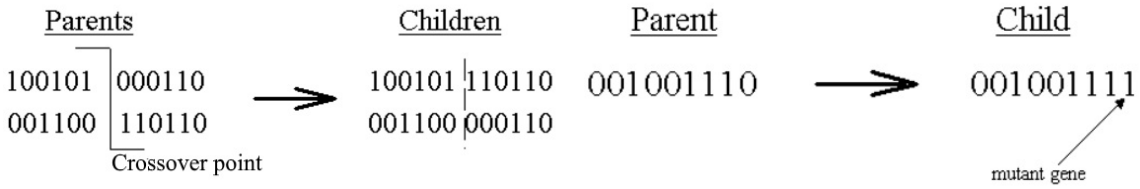


Figure 11: Types of children for the next generation [4]

Next, some individuals are chosen as parents. In most genetic algorithms these parents are chosen random or semi-random with a weighting towards lower fitness values. Pairs of parents are combined to create crossover children. The gene composition of a crossover child will therefore be a mix of both parent's genes. This is illustrated in figure 12a. The crossover point is randomly selected and there can also be more than one. Intuitively, it would be better to only select the fittest individuals to be parents. This would maximize the chance of the crossover children having low fitness values. However, a semi-random selection is better to prevent the genotypes in the population from becoming too similar to each other. Consequently, it helps the algorithm to avoid getting stuck in local optimas. The *crossover fraction* tells us what fraction of the population, except elite children, that are crossover children. The default crossover fraction in MATLAB is 0.8, which is a typical number for genetic algorithms. The remaining children after the elite and crossover children are subtracted from the population will then be mutation children. For example, given a population of twenty-two with an elite count of 2 and a crossover fraction of 0.8. Then, in the next generation, there are two elite children, 0.8*20 = 16 crossover children, and the remaining four are mutation children.



(a) Crossover children

(b) Mutation child

Figure 12: Evolution in the genetic algorithm [5]

The three different types of children are all important to bring the algorithm forward towards the global optimum. Elite children make sure we bring the best known genotypes to the next generation. This secures that the the algorithm never moves backwards to a less optimal solution. Crossovers are important to investigate if a recombination of genes from parents with low fitness value can create even lower fitness values. Mutations adds to the diversity of the population. They can potentially create superior children, but their greatest feature is to reduce the chance of the algorithm terminating at a local optima.

There are several ways to adjust the performance of the genetic algorithm. By increasing the crossover fraction, less mutations are performed, and so we get a less diverse population. If the crossover fraction is set to 1 there will be no mutation whatsoever, and the algorithm is thereby unable to create new genes. Only genes from the initial population can be recombined. With initial population bounds, a crossover fraction of 1 would disable the population from ever leaving those bounds. On the other hand, a crossover fraction of 0 means there are only mutation children besides the elite children. In this case, the algorithm will struggle to carry good genes forward from one generation to the next, causing difficulty with finding a minima. The next figures are reproduced from Mathworks' homepage and are good illustrations of what happens when the crossover fraction is set either too low or too high. The objective function $f(x)$ in question is defined as:

$$f(x_1, x_2, \ldots, x_n) = |x_1| + |x_2| + \ldots + |x_n|$$

with n equal to ten.

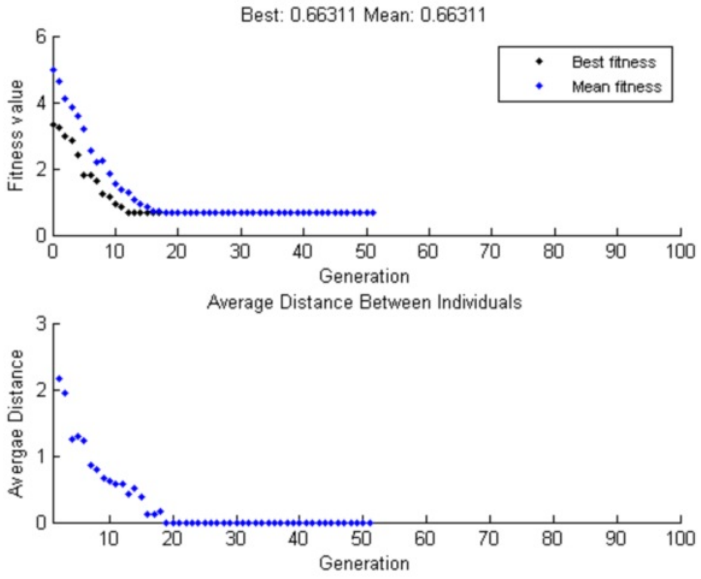


Figure 13: Genetic algorithm when crossover fraction is 1
[10]

In figure 13 the crossover fraction is set to 1. The topmost plot shows fitness values of the population as they change from one generation to the next. Blue represents the mean fitness value of the population while black is the best fitness value. The bottom plot shows the average distance between individuals in each generation and so it expresses the diversity of the population. The algorithm is also run with a crossover fraction of 0.8, although the plots for this case are not given here. With a crossover fraction of 0.8, the best fitness when the algorithm terminated was 0.08. However, in figure 13 the best fitness value is 0.66. The algorithm is not able to generate new genes as there are no mutations, and the diversity of the population is quickly dropping. Already from generation 19 all the individuals in the population have the same genes. This makes the algorithm unable to find the global minima.

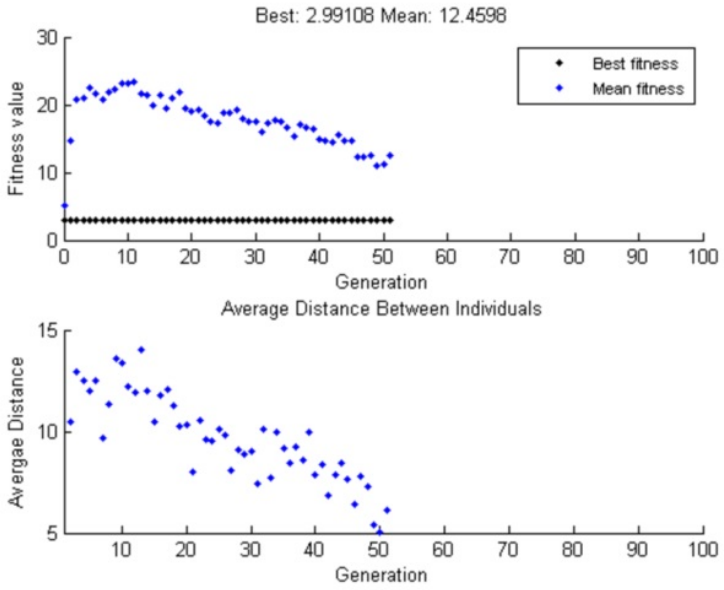


Figure 14: Genetic algorithm when crossover fraction is 0
[10]

The case showed in Figure 14 on the other hand has a crossover fraction of 0, and so all the individuals except the elite children are mutation children. The initial population bounds in this problem is set to -1 and 1 for all the variables, although the problem in general is unbound. This means that after the first generation the variables can take any value. This may lead to high fitness values, while all the fitness values in the first generation have to be 10 or less. This explains why the mean fitness seems to be around 5 in the first generation before it increases drastically. The initial population bounds are chosen to make sure there is a low fitness value in the first generation to illustrate the following: while the best individual moves on to the next generations, there are no improvements after the first generation. The mutations only happens to the other individuals, which improve slowly as can be seen in the top plot. Since we have no crossovers, the improved genes are never recombined with each other to generate better individuals. Other adjustments are also implemented in MATLAB. Among them are *shrink*, which controls how much the average amount of mutations decreases from one generation to the next. An effect of this is seen in the bottom plot, where the average distance between individuals is decreasing. By reducing the amount of mutations performed, the diversity of the next generations decreases.

2.3.3 Final generations and termination

As the algorithms is closing in on a optima, it will by crossovers generate individuals to explore the areas around the best individuals. Generally, the population follows the best individuals and so the diversity of the population is decreasing. In figure 10c the algorithm found a local optima and the population started exploring this optima. Mutations were still performed, but the algorithm was unable to find a better solution. As the individuals became more similar and the mutation level decreased, the algorithm only evaluated points very close to this optima. With such a uniform population and low mutation level the algorithm got stuck in this optima and was unable to find the global optimal solution. However, there is no guarantee that the algorithm will get stuck in the same local optima if run again because of the randomly generated mutations.

Several termination criteria can be set to tell the algorithm when to stop. In MATLAB, the listed stopping criteria are implemented. All of these have default values, though they can be altered to fit the problem at hand.

- **Generations** - maximum number of generations to be performed
- **Time limit** - maximum run time
- **Fitness limit** - a satisfactory fitness value
- **Stall generations** - maximum number of generations performed with a relative change in the fitness value function less than **Function tolerance**
- **Stall time limit** - maximum run time without any improvement of the best fitness value
- **Stall test** - which can be either average or geometric weighed. This generates the stall condition for **Stall generations**.
- **Function tolerance** - satisfactory average change in fitness values

The algorithm terminates when one of these conditions is met. However, in some cases this happens while the best fitness value is still improving, and so the parameters should be adjusted to prevent early terminations.

Whenever a function evaluation is performed the algorithm checks if that value is feasible. However, MATLAB allows small violations of the constraints. The **Nonlinear constrain tolerance**, which is also adjustable, sets an upper bound to violations of the nonlinear constraints. Violations of the linear constraints have to be smaller than the square root of this tolerance for the solution to be feasible.

3 Earlier work

Former students at NTNU completed research on the construction of synchronous generators and their optimization. The work that has been done by these students provides knowledge beneficial to solve the tasks given in this report. Two Master's theses in particular have been used throughout the work of this report. Their most relevant results are presented here.

In 2010, Aleksander Lundseng and Ivar Vikan completed a Master's thesis [7] on generator design for hydropower station upgrade. The underlying motivation for the assignment has its reason in that Europe is facing a period of upgrading and modernization of hydropower generators. This is explained by aging generators or changing needs. The objective of the thesis was to implement the construction of synchronous generators in a data program. The program was to be tested against actual machines or other calculation programs to verify its accuracy. A second part of the thesis was to analyse alternative solutions for generators during modernization of a hydropower plant.

As a preparatory study, Lundseng and Vikan analysed how the synchronous generator is constructed and made a collection of all the calculations that are done to finalize the machine. Their work on the Master's thesis culminated in a MATLAB script called *GenProg* that does the full calculation for the synchronous generator. The inputs to the program are parameters the customer usually provides the manufacturer. Amongst these are power rating and power factor, speed of rotation, number of poles, and constraints on temperature rise and synchronous reactance. The script creates a machine with these start parameters and sends outputs to an Excel spread-sheet. Parameters in the output file include but are not limited to:

- voltage and current ratings
- resistances and reactances
- thermal characteristics and cooling
- magnetic parameters
- geometrical parameters
- losses

Further study concluded that *GenProg* gave accurate results. This was after comparing the outputs with real life values.

The other Master's thesis that has been an important source of information for the work in this report is Erlend Engevik's research on optimal design of tidal power generators from spring 2014 [11]. In this report, genetic algorithms (GA) and particle swarm optimization (PSO) were used to reduce the cost of permanent magnet synchronous generators. While in general this study dealt with tidal power, the understanding of how these techniques perform on generator optimization is relevant both for tidal power and traditional hydropower generators. Engevik's work involved comparing GA and PSO, and hybrid systems in which GA and PSO techniques were combined separately with gradient-based algorithms. The algorithm that performed best and gave the lowest cost was hybrid GA. Going from a pure GA to a hybrid GA led to a cost reduction of 31.2%. Engevik concluded that stochastic algorithms like GA and PSO in general are good at finding the optimal region of a design space, but that a hybrid version is superior at finding the actual optima. Furthermore, the genetic algorithm performed better than the particle swarm optimization.

4 Method

The aim of the work done with this thesis is to implement an optimization technique on a synchronous generator to reduce its cost. Choosing an appropriate technique and a program to perform it were essential to succeed in getting reliable results. Next, it was important to set up the problem so that it reflects the real life problem in a good way. This includes choosing the objective function, optimization variables and constraints. Lastly, to reduce the run time of the optimization, the implementation had to be done as streamlined as possible. The second part of the thesis is to test the optimization by changing appropriate machine parameters.

4.1 Optimization methods

The first decision made when solving the problem in this assignment was that MATLAB was to be used to perform the optimization. From former students Aleksander Lundseng and Ivar Vikan, it already existed a MATLAB script that performed the calculations on a synchronous generator when dimensioning machine-parameters were decided. This made it very convenient to continue with MATLAB and use the script that was made. A second reason for using MATLAB was personal experience with the program. Selecting an unfamiliar program would demand time on learning how to use it, which would give less time to solve the assignment. Thirdly, MATLAB provides a well-working optimization toolbox called *Global Optimization Toolbox* that has been used by former students at NTNU with good results.

As described in the theory section about optimization techniques, there exist a great number of possible optimization techniques. Many of these are implemented in the optimization toolbox in MATLAB. MATLAB only needs information about the problem to be solved and what type of technique to be used. Erlend Engevik compared genetic algorithms with particle swarm optimization in his Master's thesis [11] and found that genetic algorithms worked better when the optimal design of tidal power generators were to be found. It is through discussions with Engevik that the genetic algorithm was chosen as optimization technique. No research was intended to be done on comparing different optimization techniques. This concludes the process in deciding implementation program and optimization technique and was to a great extent determined before the assignment was handed out.

4.2 GenProg

GenProg had in Aleksander Lundseng's and Ivar Vikan's research given good results when comparing the output values with real life machines. It was decided that *GenProg* was to be used in the work with this assignment because it did the design calculations thoroughly. Considerable amounts of time and effort were spent on becoming familiar with the script. This was important to be able to use *GenProg* for something it was not originally meant to do. The script is trying to design a functional machine based on iterations and empirical formulas from a set of inputs that are extracted from an Excel spread-sheet. Figure 15 shows a screen shot of how the required values are listed in Excel. These are the values that *GenProg* needs to run correctly. If one of these is left out the script might crash during the calculations.

2	Generator specifications			
3				
4	Required values:			
5	Apparent power	Sn	105	MVA
6	Power factor	Cosphi	0,9	
7	Speed of rotation	ns	375	rpm
8	Number of poles	Np	12	poles
9	Runaway speed	nr	612,459	rpm
10	Maximum temperatur rise	dTmx	95	
11	Moment of inertia	M	0	tm^2
12	Generator maximum voltage	Vmx	15	kV
13	Maximum value of synchronous reactance	xd	1,2	pu
14	Maximum value of transient reactance	xd1	0,4	pu
15	Minimum value of subtransient reactance	xd2	0,15	pu
16	Maximum tooth flux density	Btmx	1,7	T
17	Maximum pole core flux density	Bpmx	1,6	T
18	Maximum yoke flux density	Bymx	1,3	T
19	Specify ratio	bsdbt	0,735	
20	Core section length	bcs	0,04	m
21	Cooling duct length	bv	0,006	m
22	Filling factor (iron core)	kFe	0,95	
23	Current density in stator winding	Ss	3	A/mm^2
24	Height of one strand i the statorbar	hcus	2,12	mm
25	Required feild voltage	Vf	400	V
26	Current density in rotor winding	Sf	3	A/mm^2
27	Negative sequence voltage	Vnmx	20	%
28	Skewving (in number of slots)	s	0	spor

Figure 15: Required input values, *GenProg*

The actual values given in the figure are strictly illustrative and can be altered to fit the machine that the user wants to examine. Other than the required values, the Excel spread-sheet include *Optional values*, *Slot dimensions* and *Pole dimensions*. These are all parameters that the user can set if he wants, and so they are all practically optional parameters. If any of these are specified, *GenProg* will treat that parameter as a constant. If not specified, the script will calculate an appropriate value for the parameter based on iteration and empiricism.

GenProg first extracts all the specified values from Excel. Then, in the following order, the script calculates

- inner diameter and iron length
- number of slots
- nominal voltage, armature loading and nominal current
- stator parameters
- rotor parameters
- leakage inductances

- losses
- necessary cooling
- moment of inertia and total weight

Lastly, *GenProg* writes the calculated data to an Excel file called *Output* and terminates.

4.3 Modifying GenProg

It quickly became clear when reading through *GenProg* that it needed to be altered to be useful in an optimization problem. MATLAB spends considerably amounts of time reading and writing to Excel, which caused the run time to be around ten seconds. Keeping in mind that solving an optimization problem involves calling on the objective function a high number of times, it was important to reduce the run time of *GenProg*. The first change was therefore to comment out the writing to Excel. This does not change the script in any way except stopping it from creating an output file. Also, the reading from Excel was commented out. Though this creates an obvious problem. The script now has no information about the machine in question and will crash during calculations. To solve this, instead of reading from Excel the values were simply declared at the beginning of the script.

Secondly, two-way communication between the user and the program had to be removed. For example, *GenProg* calculates possible Q_s-values write these numbers to an Excel file. It then waits for the user to type the desired number of slots. This would not work for an optimization technique because it would be too time consuming. As discussed before, the run time has to be low because of the high number of objective function evaluations. This was first omitted by giving the number of slots a value at the beginning of the script. Later, as explained in section 4.5 this was again changed.

When starting to modify *GenProg*, the focus was on making the script as efficient as possible. After the changes mentioned above were made, the script still gave the exact same calculated data as before, but the run time had now reduced to well under a second. To keep *GenProg* unchanged, the code was copied and pasted into a new script called *Genkalk* as it was clear that more changes had to be done.

4.4 Objective function

When setting up an optimization problem, it is common to start with the objective function. In this case, the objective function to be minimized is the cost of the generator. Further, the cost can be divided into two parts, the cost of materials and the cost of losses when the machine is running. The material costs depend on the price of the materials [kr/kg] and the amount of materials [kg] needed. For the synchronous generator in question, the rotor is made from steel, the stator is made from laminated steel and the windings are made out of copper. The prices of these materials varies with manufacturer, diameter and amount of details. In Engevik's Master's thesis the cost of laminations was €4/kg based on price of M235-50A for approximated 30 pieces of 2m diameter winch laminations. The price of copper wire was €11/kg based on quotation from Dahrentråd on a 5mm*2mm wire with MV insulation and the cost of steel was €6/kg. These prices were adapted for the work with this assignment. To get the cost of the machine in Norwegian currency, the exchange rate between Euro and Norwegian krone was noted March 7th. 2016 at 9.3583. Both the price of materials and the exchange rate are easily changed in the MATLAB script. *Genkalk* calculates the weight of all the major components in the machine, which are the stator core, stator winding, rotor core and rotor winding.

The second part are the prices of losses. These will vary widely with machine size and kWh-price. However, the machine only runs at full load parts of the time, and so the the constant losses should

be more expencive than the load-dependent losses. An estimated price of constant losses is 30000 kroner per kW. These are the iron losses, no load losses in the rotor, the friction and windage losses and the magnetizing losses in the magnetizing machine. The load-dependent losses are the copper losses in rotor and stator and the additional losses. These losses can be estimated to be around 20000 kroner per kW. The following objective function was implemented in MATLAB:

$$C_{tot} = C_{loaddep} \cdot P_{loaddep} + C_{const} \cdot P_{const} + C_{steel} \cdot G_{rotor} + C_{lam} \cdot G_{stator} + C_{copper} \cdot G_{wire}$$

where

- $C_{loaddep}$ is the price of load-dependent losses in kroner per kW
- $P_{loaddep}$ is the load-dependent losses in kW
- C_{const} is the price of constant dependent losses in kroner per kW
- P_{const} is the constant losses in kW
- C_{steel} is the price steel in kroner per kg
- G_{rotor} is the rotor weight in kg
- C_{lam} is the price laminations in kroner per kg
- G_{stator} is the stator weight in kg
- C_{copper} is the price of copper wire in kroner per kg
- G_{wire} is the combined wire weight in kg

4.5 Selecting variables

To optimize the synchronous generator, some parameters have to vary while others are kept constant. It is of interest to analyse how the cost of the machine vary with some parameters while the defining machine parameters stay the same. The machine should for example not change its power rating when a change is done. Other parameters that are set to be constant are the power factor, the pole number and the synchronous speed. These are values that the customer would decide and they to great extent define how the machine performs. We want to find the cheapest machine that still provides what the customer expects. During conversations with Arne Nysveen and Erlend Engevik some parameters were discussed as potential variables. These were the number of slots, inner diameter and length, slot dimensions and pole core dimensions. The diameter and length of the machine are both important parameters as they determine the size of the machine. With a constant power rating it was desirable to examine how the algorithm dimensioned the machine so that the cost was minimized. It was also interesting to look at how the slots adjusted to get the cheapest generator. For a high number of slots, the slot width becomes narrower. This again leads to a higher current density in the slot and possibly overheating. As an option the slot height can be increased, which would decrease the current density again. However, this increases the number of strands stacked on top of each other that in turn increases the AC-resistance in the stator windings. Number of slots, slot width and slot height are closely linked together and was therefore chosen as variables. Lastly, the pole core height and pole core width were added to the list of variables. The pole shoe height is often set to 5cm as a well working value based on empiricism. The pole core dimensions were therefore selected as variables instead.

Genkalk was now transformed from a script to a function by declaring the function *Genkalk*.

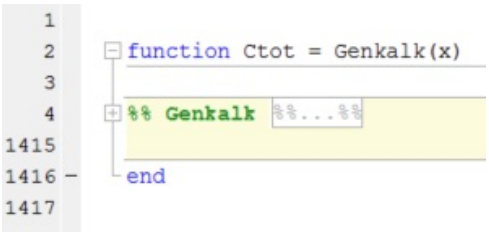


Figure 16: *Genkalk* collapsed

Here, most of the script is collapsed to illustrate the function declaration. *Genkalk* takes in an input **x** that is a vector containing all the variables. For every function call to *Genkalk* with a set of variables the function returns the machine-cost. The genetic algorithm picks a set of variables and evaluates that set by how it performs when *Genkalk* sends its output back to the algorithm. It is therefore important that when a set of variables is sent to *Genkalk*, the script gives one and only one output back to the algorithm. Furthermore, *Genkalk* should give the same output every time it is called on with the same input. However, the same output may be produced from more than one set of inputs.

The variables were now examined further. First, the number of poles Q_s can only take integer values. Furthermore, it has to be divisible by 3 for the phase balance to be achieved and by 2 for symmetry. Instead of keeping the number of slots as a variable, $Q_s/6$ was selected as the variable and called Q_{ss}. Q_{ss} can also only take integer values. However, it can take all positive integer values, while Q_s only could take every 6th integer number. Integer variables are easily implemented for the genetic algorithm in MATLAB. The second variable, the inner diameter D_i, is continous and was therefore kept as it was. This was also the case for the slot width and pole core width. The iron length on the other hand can not take all possible values. The stator is made from laminated plates with cooling ducts in between, both with a specific width. This implies that the gross iron length has to satisfy Equation 49:

$$l_b = (n_v + 1)b_{cs} + n_v \cdot b_v \tag{49}$$

where n_v is the number of cooling ducts, b_{sc} is the length of the laminated plate and b_v is the cooling duct length. n_v was selected as variable instead of the gross iron length. As for Q_s, n_v can only take integer values. Similarly, the slot height and pole core height both need to provide space for respectively the stator and rotor windings. The stator winding consists of a number of strands stacked on top of each other as shown in figure 3a in the theory section. The slot height h_s needs to be large enough to fit the Roebel bars:

$$h_s = n_{dl} \cdot h_{cus} + n_{dl} \cdot d_{icu} + h_m + h_{spk} + h_{gls} + h_{ds} + 4d_{ij} \tag{50}$$

ndl was chosen as variable instead of the slot height and can only take integer values. In the same way, the rotor winding is made up of a number of conductors as can be seen in figure 5. The conductor height h_{cuf} was adopted from the work of Aleksander Lundseng and Ivar Vikan that had given a value of 1.249mm. The pole core height can be calculated as:

$$h_{pk} = n_f \cdot h_{cuf} + (n_f - 1)b_{if} + 2b_i + 2h_{kr} \tag{51}$$

and n_f was chosen as the variable.

In addition to the variables mentioned, two more were added to the list. In *GenProg*, one of the required inputs was the current density in the rotor. To achieve this current density, *GenProg* was designed to iterate over the field winding height and pole core height. This led the magnetic calculations to be performed within a while-loop, changing the pole core height for each iteration.

Then when the pole core height was set, a new while-loop calculated the necessary copper area and the width of the field winding. This structure with while-loops makes the script slower and should be avoided if possible when optimizing. To eliminate the while-loop, the field winding width was introduced as a variable. Lastly, during testing it seemed as the synchronous reactance was limited by the script in some way. A high X_d makes the machine less stable, but is cheaper to design. However, it did not take as high values as expected. The synchronous reactance is closely linked to the air gap and increases when the air gap length decreases. In *Genprog* the minimum air gap length δ_0 is either extracted from the excel spread-sheet as an input or calculated from empirical formulas. During the transformation to *Genkalk*, the empirical formula were chosen to calculate δ_0. This empirical approximation limited the air gap length and made it impossible for the synchronous reactance to vary freely. As it was an option to extract δ_0 from the excel spread-sheet directly, δ_0 should also be able to be selected as a variable and not be calculated from these formulas. Later testing verified that the synchronous reactance increased and varied freely when the air gap was a variable.

In total nine variables were selected with Q_{ss}, n_v, n_{dl} and n_f being integer variables:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} Q_{ss} \\ D_i \\ n_v \\ b_u \\ n_{dl} \\ b_{pk} \\ n_f \\ b_{cuf} \\ \delta_0 \end{bmatrix}$$

Genkalk was now tested against *GenProg* and gave as expected similar results. From an output Excel file from *GenProg*, the eight variables were copied and sent to *Genkalk* as input. *Genkalk* gave almost the exact same machine-design as *GenProg*, which was what *Genkalk* was intended to do. One while-loop was left as it was in *Genkalk*. The iteration over necessary cooling air flow was performed at the end of the script and did not affect the geometry of the machine in any way. For each iteration the air flow was increased until the temperature rise for all the different nodes in the machine were within the maximum temperature bounds. The calculations were performed to be sure that the cooling air flow prevented the machine from overheating. As there are no costs related to temperature rise in the machines in this assignment, it is satisfactory to know that the machine will not overheat. Although there are costs related to losses when current is flowing through a resistance and generates heat, these are already accounted for.

4.6 Bounds, constraints and penalty

The dimensioning parameters of the generator were now set. Although the intention is to optimize the generator, some parameters need to be set to initialize the optimization. Some of the most important parameters are listed below. A full list can be found in the appendix.

- $S_n = 50\text{MVA}$
- $\cos\phi = 0.9$
- $N_p = 16$
- $n_s = 375\text{rpm}$

which are typical values for a medium sized hydropower generator.

A lower and upper bound were now set for all the variables. To be sure that the optimal solution were within these bounds, they were set fairly wide. In *GenProg* the lower and upper limit of number of slots is the circumference divided by 0.12 and 0.04 respectively. An inner diameter between 3 and 5 meters is expected for a hydropower synchronous generator at 50MVA. For a generator with an inner diameter of 3m the bounds become 79 and 236 slots, while with a 5m inner diameter they are 131 and 393 slots. The bounds for Q_s were set to 78 and 390 that led to a lower bound at 13 and an upper bound at 65 for Q_{ss}. For the inner diameter a lower bound of 2m was decided to be sure not to leave the optimal solution outside the bounds. The peripheral speed on the rotor increases with an increasing diameter, and the construction becomes difficult if the speed is too high. A common limit on the peripheral speed is 150m/s for salient pole synchronous generator. They are designed to withstand stress at runaway speed, which differs with different turbines. For a Pelton turbine the runaway speed is around 1.85pu while it can be from 2.0pu to 2.2pu for a Francis turbine [12], though these numbers differ from different sources and machine sizes. A runaway speed of 1.8pu was chosen. From the runaway speed and maximal peripheral speed, the maximum inner diameter can be calculated:

$$D_{i,max} = \frac{150 \cdot 60}{\pi \cdot n_r} \tag{52}$$

With n_r equal to 1.8pu = 675pu the maximum inner diameter becomes 4.244m. This was chosen as the upper limit for the inner diameter. For the rest of the variables, bounds were set from trial and error.

The next step was to include constraints associated with maximum current density, synchronous reactance and flux density. It was clear that at least some of these constraints were nonlinear. For the genetic algorithm in MATLAB to deal with nonlinear constraints, it has to be provided with a function that tells the algorithm if a set of variables obeys the nonlinear constraints or not. The function and script *nonlcons* was created to keep track of the nonlinear constraints. Figure 17 shows the function declaration of *nonlcons*. Most of the script is collapsed.

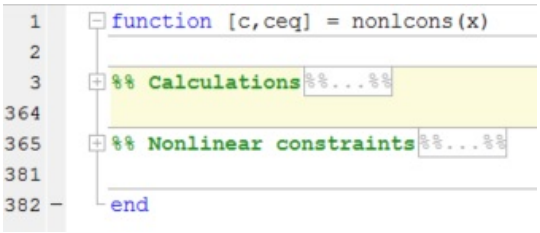


Figure 17: *nonlcons* collapsed

As can be seen, the function takes in the variable x and returns an array with two components, c and c_{eq}. When the genetic algorithm runs, it will perform a high number of function evaluations. For every set of variables, the algorithm also checks if it obeys the nonlinear constraints. So the input x is the set of variables discussed in the previous section. The output c is a vector where each component represents a nonlinear constraint. The constraints have to be based on less than statements. For example, if the following constraints were to be implemented:

$$x_1 x_2 < 50$$
$$2x_1^2 - 3x_2 x_3 > 15$$

then they would be rewritten like:

$$x_1 x_2 - 50 < 0$$
$$-(2x_1^2 - 3x_2 x_3 - 15) < 0$$

and represented in the script as:

$$c(1) = x_1 x_2 - 50$$
$$c(2) = -(2x_1^2 - 3x_2 x_3 - 15)$$

If all the components in c are negative for a set of variables, the constraints are satisfied. The vector c_{eq} contains the nonlinear equalities. However, the genetic algorithm does not handle both integer value variables and nonlinear equalities. Therefore, it has to be set as an empty vector in *nonlcons*.

The nonlinear constraints would be very complicated and chaotic if they were to be expressed only by the inputs x_1-x_9. There are a high number of calculations done going from the inputs to values as reactances and flux density. Furthermore, the script *nonlcons* does not have access to the parameters in *Genkalk*. Consequently the code in *Genkalk* was copied to *nonlcons*. After deciding what constants were needed, the excess code was removed to improve efficiency. More efficient ways of accessing these parameters were not examined.

The constraints could now be implemented. First, the synchronous reactance X_d in a synchronous generator should be limited to maintain static stability for the generator. A common upper limit for the synchronous reactance for hydropower generators is 1.2pu. Additionally, the transient reactance should not be above 0.4pu. The sub transient reactance is important for limiting the short circuit currents and to prevent failures from spreading to other parts of the power system. A lower limit of 0.15pu was implemented for the sub transient reactance.

The current density needed to be limited to prevent overheating in both stator and rotor. *GenProg* itself makes sure that sufficient cooling is present for each machine. However, run time can be reduced by ruling out some of the machines with high current density before *GenProg* is called. Only an upper limit was set for the current density in rotor and stator. Common values are from 2 to 5A/mm^2. An upper limit of 5A/mm^2 was implemented for the stator and rotor current density S_s and S_f.

To avoid too much magnetic saturation in the machine that causes high losses, constraints were implemented in *nonlcons*. High flux densities also cause noise that should be limited. Norm values give guidelines for the flux density in different parts of the machine. The magnetic flux density is higher in the stator teeth than in the stator yoke because the area that the magnetic flux spread over is smaller in the teeth than in the yoke. Similarly, the flux density is higher in the pole core than in the rotor yoke. The constraints set were therefore lower for the rotor and stator yoke than in the pole core and stator teeth. For the bottom parts of the stator teeth the maximum allowable flux density set was 1.7T. The flux density is largest closer to the air gap at the bottom parts of the stator teeth because of leakage flux. In the same way the flux density is higher at the bottom parts of the pole core. Here, a constraint of 1.6T was set. For the rotor yoke the flux density is assumed constant. A limit of 1.3T was set for the rotor yoke. The stator yoke flux density is in *Genkalk* calculated from $Bymx$, which is the maximum yoke flux density. This parameter is set as 1.3T, and consequently the stator yoke flux density will always be 1.3T. Hence no constraint for the stator yoke flux density is needed.

For large generators connected to the grid, the system operator often specifies restrictions to its moment of inertia. This is to maintain enough swing mass in the power system, which ensure system stability. Statnett, the Norwegian system operator says in FIKS 2012 [13] that the ratio between the time constants for the moment of inertia T_M and the water way T_w should be more than 4. T_w is usually set to 1s. T_M is calculated as two times the inertia constant H:

$$H = \frac{1}{2} \frac{J\omega_m^2}{S_n[kVA]} \tag{53}$$

where J is the moment of inertia and ω_m is the angular velocity in mechanical radians per second.

Although this is not an absolute requirement but more of a guideline, a constraint was implemented to make sure that $T_M/T_w > 4$.

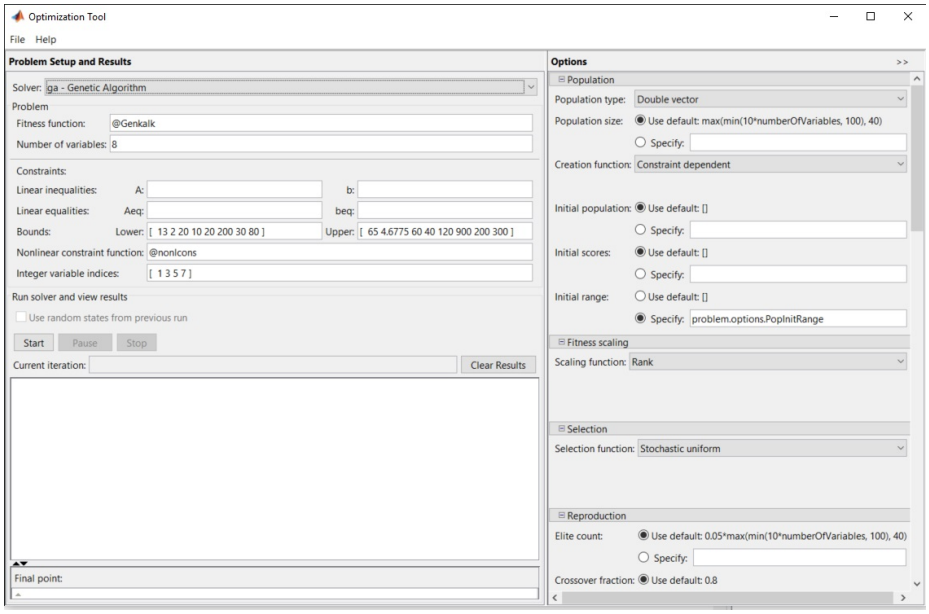


Figure 18: Optimization application, MATLAB

The optimization was continuously tested in the optimization application in MATLAB. The screen shot illustrated in figure 18 shows the application. The fitness function is provided along with the number of variables, variable bounds, nonlinear constraint function and what variables are integer variables. If not specified, the algorithm options are set to default values. During testing, some machine constructions that the algorithm suggested were not physically possible or the algorithm would crash during the calculations. Occasionally, machine-designs resulted in an imaginary machine-cost, which the genetic algorithm could not deal with, reason being that MATLAB tried calculating the logarithm of negative numbers. It proved that the genetic algorithm would sometimes suggest variable sets that made some geometrical parameters negative. Often, the slot width b_u would be larger than the slot pitch τ_s, and so the slot tooth width b_d became negative. To prevent this a nonlinear constraint saying that the slot width had to be less than 0.8 times the coil pitch was implemented. Furthermore, a constraint had to be implemented to make sure that adjacent poles with field windings were not overlapping. The parameter *polklaring* was calculated as the space between the bottommost part of adjacent field windings, and a constraint saying that this parameter had to be greater than 0.002m was implemented in *nonlcons*.

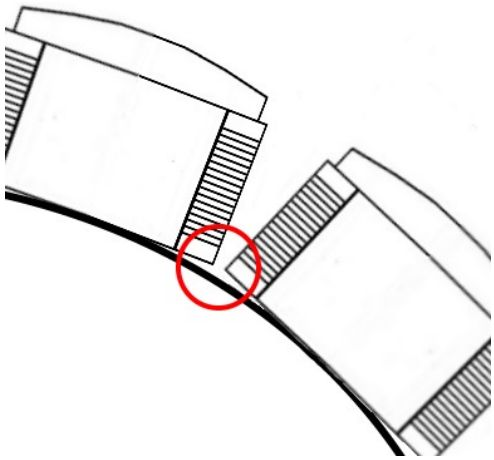


Figure 19: Spacing between adjacent poles

taumn is the clearance between adjacent poles and should also be greater than zero. It is very unlikely that *taumn* is negative if *polklaring* is greater than zero. However, a constraint saying that *taumn* have to be greater than 0.002m was implemented to be sure this will not happen. In total 12 nonlinear constraints were implemented. They are listed below:

- $X_d < 1.2\text{pu}$
- $X_{dt} < 0.4\text{pu}$
- $X_{dtt} > 0.15\text{pu}$
- $S_s < 5\text{A/mm}^2$
- $S_f < 5\text{A/mm}^2$
- $B_{dmax} < 1.7\text{T}$
- $B_{drmx} < 1.6\text{T}$
- $B_{yr} < 1.3\text{T}$
- $\frac{T_M}{T_w} > 4$
- $bu < 0.8 \cdot \tau_s$
- $polklaring > 0.002\text{m}$
- $taumn > 0.002\text{m}$

They all have to satisfied for the algorithm to certify a set of variables as feasible. However, small violations of the constraints are by default allowed. The *Constraint tolerance* is an upper limit of how big these violations can be. The default value of the constraint tolerance is 0.001.

Lastly, the cooling air flow had to be limited. During testing, some machines ended up with a very high cooling need. A common limit for the cooling air flow speed v_{im} is around 16m/s. Setting this as a constraint was avoided because of the iterative process of calculating the cooling air flow. The iteration would for some variable sets get stuck because the cooling need was simply too high. Another solutions was to implement a penalty for high air flow speed values. A common way to limit the solution space for the genetic algorithm is to penalize the fitness function whenever something undesirable occurs. In this case, it was desirable to increase the cost of the machine when the cooling

air flow speed became more than 16m/s. The code section in figure 20 shows how the penalty was implemented. It was placed in the while-loop that calculates the cooling need.

```
695 -     while aqth1 == 0
696           %%%...%%
707           %%
708 -         if vim > 16
709 -             penalty = 1000*log10(vim-15);
710 -             if vim > 30
711 -                 aqth1 = 1;
712 -             end
713 -         end
714           %%%...%%
1126      %%
1127 -   end
```

Figure 20: Code section that implements a penalty for high cooling air flow speed

If v_{im} becomes more than 16m/s, a penalty value is calculated. The iteration will continue until either the cooling need is meet or v_{im} becomes more than 30m/s. The binary variable $aqth1$ is then set to 1 instead of 0 that tells MATLAB to exit the while-loop. Lastly, the penalty value is added to the objective function. The penalty value is calculated to greatly penalize the objective function as soon as v_{im} becomes more than 16m/s. A logarithmic function is used to adjust down the penalization of high cooling air flow speeds. This makes the plotting of function evaluations easier.

Furthermore, it was not enough to makes sure that *Genkalk* would not crash from imaginary numbers. *nonlcons* contains a lot of the same code as *Genkalk* and would crash similarly by the reasoning made earlier. An *if*-statement was implemented saying that only when certain conditions were met, the calculations of the constraint parameters would be done. If not, the calculations would not be performed and the nonlinear constraints were set as not satisfied.

4.7 Performing the algorithm

A fourth script *cost* was made to contain the information about the optimization problem and to start the genetic algorithm. This script creates a struct called *problem* with all the information MATLAB needs to perform the optimization. A struct or a structure array is a data type that groups data were each field can contain any type of data. Besides setting the fitness function, number of variables, bounds and constraints, the optimization options are also set here. The options are set with the *optimoptions*-function in MATLAB and are collected in a field called *options* in the structure *problem*. *options* is itself a structure with different fields for all the option settings. The following code example shows how the options are specified in *cost*. The first statement sets the crossover fraction to 0.6. To not cancel out option settings that may already be set, *option* is added to the statement before setting the crossover fraction. The second statement tells MATLAB to plot the best fitness values for each generation. It also makes MATLAB display the iteration. This include but is not limited to the generation number, function call count, the best and mean fitness value and the number of stall generations.

```
problem.options = optimoptions(problem.options,'CrossoverFraction',0.6);
problem.options = optimoptions(problem.options,'PlotFcn',{@gaplotbestf},'Display','iter');
```

Figure 21: Examples on how to implement option settings

The options are structured in groups. Some of these are listed below, although the full list can be found on-line at Mathworks' homepage [14].

- plot options
- population options
- fitness scaling options
- reproduction options
- mutation options
- crossover options
- stopping criteria options

Initial population bounds were specified in the option settings to reduce the run time of the optimization. The bounds were set after running the optimization a considerable number of times to be sure that the optimal solution were within these bounds. *cost* is fully given in appendix B.

4.8 Testing of the implementation

Besides implementing the optimization, it was desirable to test and analyse how changes in the constant parameters affected the optimal solution. This is a good way of examining the correlation between a parameter and the machine-design. First, the relationship between cost of losses and the optimal solution were to be analysed. Normally the load-dependent losses are less costly than than the constant losses. To figure out what happens to the generator design when these two are equally expencive, a test was designed. The optimization were to be run ten times for three different cases. In case 1, the price of constant losses was 30000 kroner per kW, while the price load-dependent losses was 15000 kroner per kW. For case 2, the constant losses cost 30000 kroner per kW, while the load-dependent losses cost 20000 kroner per kW. In case 3 both prices were going to be 25000 kroner per kW. The losses in these three cases were then to be compared. The test should reveal if a change in the price changes the composition of these losses.

The second test to be performed was to change the power factor $\cos\phi$. A typical value for the power factor is 0.8. In the test the optimization would be run ten times with the power factor at 0.8, 0.9 and finally 1.0. The requirement that synchronous generators have to be able to generate reactive power makes the generator more expencive. Changing the power factor from 0.8 to 1.0 should result in the optimization algorithm finding cheaper machines, Furthermore, it is desirable to examine if the geometrical parameters changes. Lastly, the optimization were to be performed without the constraint for moment of inertia to see if the algorithm suggested even cheaper designs. The power factor would then be 1.0.

Thirdly, the synchronous reactance X_d is limited in *nonlcons* to be under 1.2pu. This is to maintain the machine stability. Often, this requirement is not set for smaller machines. They can then be made cheaper and smaller because the air gap can be smaller, however it also makes the machine more unstable. It is desirable to test if the machine-design changes when this requirement changes. Ten optimizations each were to be performed with $X_d < 1.2$, $X_d < 2.0pu$ and with X_d unlimited.

The transient and sub transient reactance were not to be constrained in the cases of $X_d < 2.0pu$ and when X_d was unlimited.

For all the tests, a normal case scenario is included. The normal case is when all constraints are included, the power factor is 0.9 and the cost of load-dependent and constant losses is respectively 20000 kroner per kW and 30000 kroner per kW.

5 Results

The main part of this project was to develop a MATLAB script that implements a genetic algorithm to optimize the design of the synchronous generator. In total three scripts were made. How these scripts were implemented was explained thoroughly in the methodology section. A closer look at how they relate to each other is gone through here. Figure 22 shows a flowchart of how the optimization works.

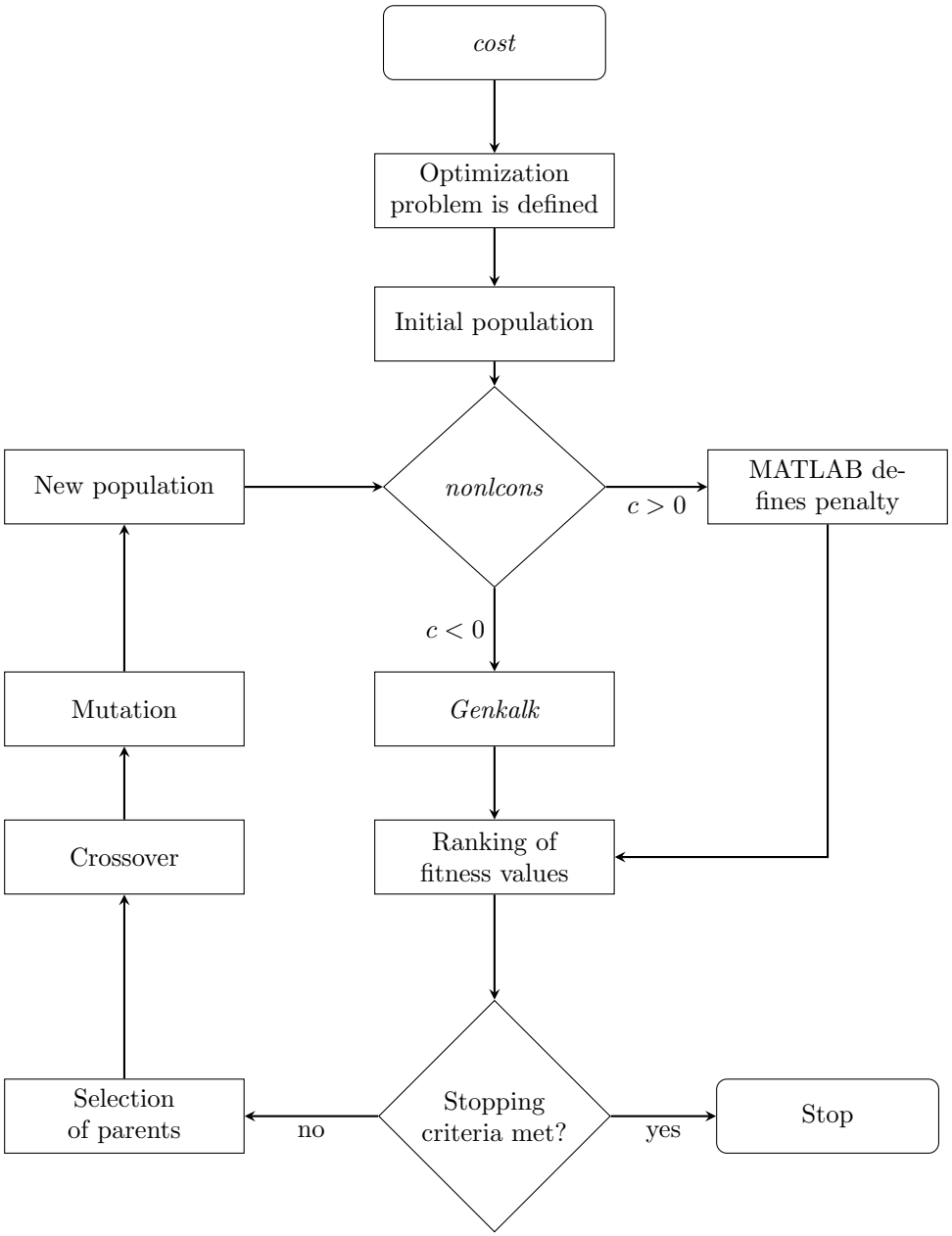


Figure 22: Flowchart of the implemented genetic algorithm

Cost contains the information about the optimization problem and performs the genetic algorithm on the problem. After the problem has been defined in *cost*, the script calls on the embedded MATLAB-function *ga*. It randomly selects the initial population from within the initial population bounds. Every individual is then checked up against the nonlinear constraints in *nonlcons*. If any of components in the output c is larger than the constraint tolerance, MATLAB penalize that individual. The individuals that satisfy the constraints are evaluated in *Genkalk*. All the individuals are then ranked after their fitness value. If one of the stopping criteria explained in section 2.3.3 are met, the algorithm terminates. Otherwise, parents are selected from the current generation. A new population is then created through crossovers and mutations. The algorithm will continue until one of the stopping criteria is met or the user terminates it manually.

5.1 Changing the price of losses

Table 1 and 2 show the results from the first test where the prices of losses were varied. The given values are average values from ten runs with standard deviation in parentheses.

Table 1: Losses

$C_{loaddep}$	C_{const}	$P_{loaddep}$ [kW]	P_{const} [kW]	P_{tot}[kw]
15000	30000	345.36 (7.2)	338.55 (7.9)	683.91 (14.4)
20000	30000	317.52 (8.3)	339.07 (8.15)	656.59 (15.9)
25000	25000	304.52 (6.5)	343.59 (10.1)	648.10 (15.2)

The losses are given in table 1. For the first case, $P_{loaddep}$ is 8.7% higher than for case 2, and 13% higher than for case 3. The standard deviation is 7.2kW for the first case, 8.3kW for case 2 and 6.5kW for the last case. This is respectively 2.1%, 2.6% and 2.1% of the average values. Similarly, the constant losses in case 1 are 0.15% lower than in case 2 and 1.5% lower than in case 3. Case 1 has the highest total losses at 683.91kW, being 4.2% higher than in case 2 and 5.5% than in case 3. The standard deviations are respectively 2.1%, 2.4% and 2.3% for case 1, 2 and 3. The full results are given in appendix C.

Table 2: Geometrical parameters

$C_{loaddep}$	C_{const}	Q_s	D_i [m]	l_b [m]	b_u [mm]	h_s [mm]	b_{pk} [mm]	h_{pk} [mm]
15000	30000	178.2 (14.21)	3.70 (0.08)	1.26 (0.09)	22.4 (2.23)	108 (4.12)	363 (11.1)	185 (11.96)
20000	30000	183.6 (12.06)	3.61 (0.12)	1.42 (0.20)	22.0 (0.84)	117 (5.72)	345 (29.0)	191 (7.28)
25000	25000	181.8 (16.11)	3.67 (0.12)	1.36 (0.17)	22.4 (1.45)	130 (4.91)	355 (20.4)	193 (10.71)

Table 2 shows the inner diameter, gross iron length, slot width and height and pole core width and height for the three cases. These values represent most of the geometrical parameters that were selected as variables. As in the previous table, these values are also average values from ten optimizations with standard deviation in parentheses.

5.2 Changing the power factor

For the second test, the power factor $\cos\phi$ was varied. It was set to 0.8 in case 1, 0.9 in case 2 and 1.0 in case 3. The constraint for the moment of inertia was removed for case 4 (*), where the power factor was 1.0.

Table 3: Cost of materials and losses

$\cos\phi$	C_{loss}	C_{lam}	C_{steel}	C_{wire}	C_{tot}
0.8	16840 (309)	1175 (36)	2406 (61)	1446 (82)	21866 (217)
0.9	16523 (397)	1152 (34)	2374 (68)	1625 (151)	21673 (212)
1.0	15687 (210)	1085 (61)	2286 (114)	1839 (87)	20907 (76)
1.0 (*)	15535 (269)	1015 (47)	1953 (43)	1457 (113)	19960 (191)

Table 3 shows the cost of losses, laminations, steel and wire in addition to the total cost of the machine for the four cases. In table 4, some of the major geometrical parameters are shown. In both tables, the values given are the average values from ten optimizations with the standard deviation in parentheses. The cost of losses is 6.8% lower for case 3 than case 1 and 5.1% lower than for case 2. Similarly, the cost of laminations and steel are lowest for the case with $\cos\phi$ equal to 1.0. However, case 3 has the highest wire cost, being 27.2% higher than in case 1 and 13.2% higher than in case 2. The total costs are again lowest in case 3, and at 20907 kNOK (thousand Norwegian kroner), it is 4.4% lower than for case 1 and 3.5% lower than for case 2. For the geometrical parameters the slot height is 8.7% higher for case 3 than for case 1 while the standard deviations are 3.9% and 6.3% respectively for case 1 and 3. The results are similar for the pole core height where in case 3 it is 229mm on average, 28% higher than i case 1. The pole core width on the other hand is largest for case 1, 9.7% higher than in case 3. Here, the standard deviations are 16.9mm and 13.8mm for case 1 and 3, or 4.7% and 4.2% of the average values. Case 4 has the lowest total cost at 19960 kNOK or 4.5% lower than in case 3. In fact, the loss cost and all the material costs are lower in case 4 than 3. Of the geometrical parameters, the diameter has dropped from 3.61m to 3.32m, a decrease of 8.0%. The slot height has increased to 131.3mm, whereas both the pole core height and width has decreased.

Table 4: Geometrical parameters

$\cos\phi$	Q_s	D_i [m]	l_b [m]	b_u [mm]	h_s [mm]	b_{pk} [mm]	h_{pk} [mm]
0.8	174.6 (11.83)	3.65 (0.08)	1.37 (0.11)	21.9 (1.95)	115 (4.49)	363 (16.9)	178 (10.15)
0.9	183.6 (12.06)	3.61 (0.12)	1.42 (0.20)	22.0 (0.80)	118 (5.72)	345 (29.0)	191 (7.28)
1.0	187.8 (21.97)	3.61 (0.09)	1.35 (0.15)	22.7 (1.52)	125 (7.83)	331 (13.8)	229 (11.07)
1.0 (*)	171 (14.76)	3.32 (0.14)	1.38 (0.17)	21.9 (1.98)	131.3 (5.49)	313 (19.2)	180.3 (13.49)

5.3 Changing the constraint for synchronous reactance

In the third test, the constraint for the synchronous reactance was changed. In case 1, the upper limit is at its normal value at 1.2pu. For case 2, X_d is limited to 2.0pu while it is unconstrained in case 3. For case 2 and 3 the transient and sub transient reactance are not constrained. Again, the tables show the average values of ten optimizations with the standard deviation in parentheses. In addition to the costs, X_d and the nominal current I_n are included in table 5. The synchronous reactance is for case 1 and 2 pushed towards the limit as can be seen in the table. For the unconstrained case, X_d ends up at 4.9pu with a standard deviation of 0.2pu. This case also has the largest nominal current at 5509A, 22.8% more than in case 1 and 8.1% more than in case 2. The costs are lowest for case 3, except the wire cost, which for case 3 is 1785 kNOK, 9.8% higher than for case 1. The total cost of the machine is 21673 kNOK, 20565 kNOK and 20021 kNOK for case 1, 2 and 3. Here the standard deviations are 0.98%, 0.53% and 0.30% of the average values.

Table 5: Cost of materials and losses

Constraint	X_d	C_{loss}	C_{lam}	C_{steel}	C_{wire}	C_{tot}
$X_d < 1.2pu$	1.2 (0.00)	16523 (397)	1152 (34)	2374 (68)	1625 (151)	21673 (212)
$X_d < 2.0pu$	2.0 (0.00)	15642 (274)	1043 (39)	2196 (94)	1684 (74)	20565 (110)
Unconstrained	4.9 (0.20)	15071 (211)	996 (25)	2169 (72)	1785 (97)	20021 (60)

Table 6 again shows the geometrical parameters representing the selected variables. The slot width for case 3 is 15% higher than for case 1 with the standard deviation being 0.8mm and 1.54mm or 3.6% and 6.1% of the average values for case 1 and 3 respectively. The pole core height is 14% higher in case 3 than in case 1 and 5.3% higher than for case 2. Here the standard deviations are 7.3mm, 12.8mm and 7.3mm for case 1, 2 and 3. In percentage of the average values this equal 3.8%, 6.2% and 3.3%.

Table 6: Geometrical parameters

Constraint	Q_s	D_i [m]	l_b [m]	b_u [mm]	h_s [mm]	b_{pk} [mm]	h_{pk} [mm]
$X_d < 1.2pu$	183.6 (12.06)	3.61 (0.12)	1.42 (0.20)	22.0 (0.80)	118 (5.72)	345 (29.0)	191 (7.28)
$X_d < 2.0pu$	184.2 (17.60)	3.67 (0.07)	1.24 (0.11)	23.0 (1.28)	126 (4.33)	350 (14.3)	207 (12.76)
Unconstrained	175.2 (11.00)	3.65 (0.07)	1.24 (0.11)	25.3 (1.54)	124 (6.14)	349 (17.8)	218 (7.26)

In table 7, the magnetic voltage drop over the air gap $U_{m,\delta}$ along with the total voltage drop $U_{m,tot}$ and total magnetization of the machine Θ_{mN} are given. Also field and stator current with their current densities are shown. The magnetic voltage drop over the air gap reduces with an increased synchronous reactance. In the unconstrained case $U_{m,\delta}$ is only 27.4% of the equivalent value in case 1, while the total magnetization drops with 9.3% from case 1 to case 3. Both the current density in the rotor and stator circuit reduces when X_d increases. However, the nominal current is increasing at the same time.

Table 7: Magnetization and current

Constraint	$U_{m,\delta}$	$U_{m,tot}$	Θ_{mN} [At]	I_f [A]	S_f [A/mm^2]	I_n [A]	S_s [A/mm^2]
$X_d < 1.2pu$	16082 (977)	17018 (1001)	31404 (1851)	289 (24.1)	1.90 (0.15)	4485 (488)	2.90 (0.15)
$X_d < 2.0pu$	10758 (639)	11690 (660)	29994 (1702)	253 (30.5)	1.65 (0.12)	5095 (406)	2.82 (0.08)
Unconstrained	4402 (320)	5234 (329)	28335 (1269)	224 (16.9)	1.48 (0.11)	5509 (280)	2.74 (0.07)

6 Discussion

The main purpose of this assignment was to implement an optimization technique for the synchronous generator using geometrical parameters as variables. Further, it was desirable to test the optimization by varying some major machine parameters or constraints. This was important to examine the implemented optimization and to research if it gave reliable results. The discussion part of this assignment is therefore going to answer both how well the results can be trusted as well as what they mean.

There are several reasons why MATLAB was selected as optimization program. First, the script *GenProg* was written in MATLAB-code and was essential for the work in this assignment. Secondly, knowledge about MATLAB made it a convenient program to use instead of getting to know a new program. Lastly, the work of Erlend Engevik on optimization of tidal power generators was done in MATLAB and had shown good results. MathWorks provides sufficient information on-line on how optimization problems can be implemented. This has been important during the work on this assignment. Information is easily accessible and well described. In addition to explaining syntax and options for the implementations, more thorough information on how the optimization algorithms works is given. This has made MATLAB a suitable choice for this assignment. Furthermore, the optimization application makes the implementation as convenient as it can get, only demanding a fitness function and the number of variables. As for any program the debugging can be tedious and challenging. Although MATLAB uses embedded functions for optimization problems, knowledge about what these functions do is not crucial when debugging. On the other hand, important requirements to the objective function are that:

- all inputs give real outputs.
- the output is a number and not a vector, matrix etc..

Furthermore, when a nonlinear function is provided:

- the output vectors c and c_{eq} need to be defined.
- c should not change size for different inputs.
- c_{eq} have to be empty when integer value variables are present.

It is essential that *Genkalk* work as intended and do the machine calculation correctly. During the Master's theses of Aleksander Lundseng and Ivar Vikan, *GenProg* was tested against real life machines and gave satisfactory results. Still, errors have been found and corrected as a consequence of the work on this assignment. Errors can still be present in the script and this should be kept in mind when concluding from the results. A second source of misleading results is the implementation of variables in *Genkalk*. Some of the variables selected are possible to calculate from empirical formulas. This means that there are two possible ways of selecting some of the parameters for the same machine. However, this is also the case in *GenProg* where the input Excel file contains both required and optional values. If not specified, the optional parameters are calculated from empirical formulas. Results have shown no sign that the implementation of variables has caused invalid results.

The genetic algorithm was chosen based mostly on the work of Erlend Engevik that had shown a superior performance to the particle swarm optimization. Both techniques are suitable for the problem at hand, reason being the complexity of the problem with nonlinear constraints and integer value variables. The simulations showed that the solutions to the optimization varied from each simulation. For the normal case scenario, the best set of variable values from ten optimizations are shown, and are plotted as a percentage of the average value in figure 23. It is clear that the best solution for each optimization varies significantly. Variable 3 is the number of cooling ducts and

varies most in this plot with the highest value being 32.9% larger than the average value and 66.7% larger than the lowest value.

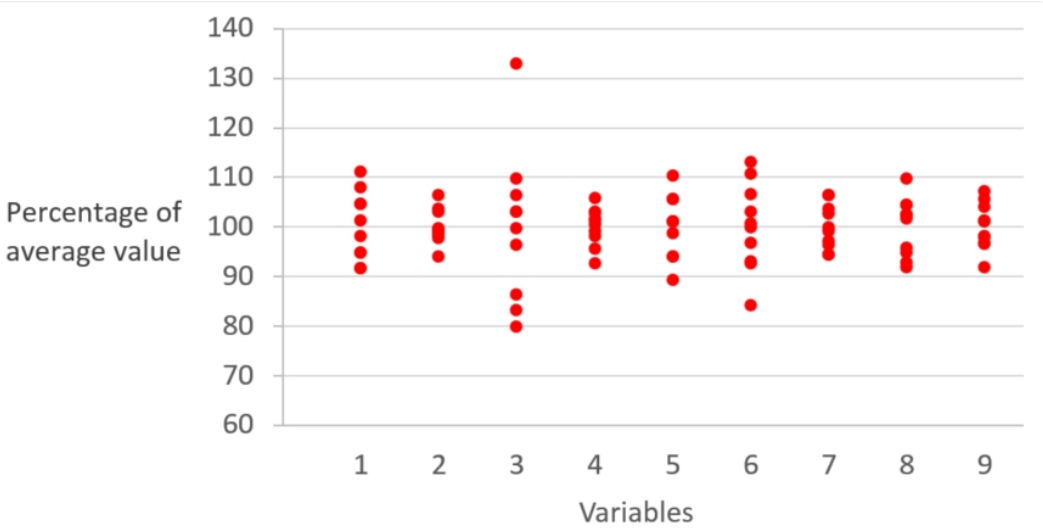


Figure 23: Optimal set of variable values from ten optimization for the normal case scenario

The varying results can be explained by the population being too low, which does not allow the algorithm to search the whole solution space. Increasing the population size will presumably give less diversified results, but will significantly increase the run time of the optimization. Also, to further narrow down the result spread when dealing with genetic algorithms, a hybrid solver is often implemented. However, in this case hybrid solvers can not be used because the variable set includes integral value variables. Although the algorithm gives varying results, the aim of the optimization is not to find the same result every time, but to find low fitness values. By running the optimization several times, the chances of finding the optimal solution increases. The genetic algorithm is in fact often used to find several different solutions and then try them out in practice to see which one actually performs the best. This was the case for NASA when using the genetic algorithm to design antennas for the ST5-mission. The antenna in figure 8 was just one out of many designs that the algorithm suggested. Therefore, for complex problems, varying results are not indicative of unreliable results. However, the high standard deviation indicate that only performing ten optimizations for each case may not be enough.

In the first test the cost of load-dependent and constant losses were varied. The test results showed a significant decrease in the load-dependent losses when the price of the these was increased, which was expected. Figure 24 shows load-dependent and constant losses for the three cases, case 1 having the lowest price for load-dependent losses at 15000 kroner per kW. The values plotted are the average values of the ten optimizations performed for each case. The standard deviation is included as error bars. The decrease in load-dependent losses is clear, while the increase in constant losses moving from case 1 to 3 is minimal. in fact, this increase can not be ruled out as a random variation due to the relatively high standard deviation.

The significant decrease in load-dependent losses while the constant losses stay fairly constant can be explained by their ability to change. The load-dependent losses consists of copper losses in stator, full load copper losses in rotor and the additional losses. The copper losses are able to change largely by adjusting the nominal current and the copper area in the machine. Figure 31 and 32 in appendix C shows the full results from test 1. The DC copper losses in the stator winding decreases drastically

from case 1 to case 3. It is calculated as three times the DC copper losses in each phase, which in turn is calculated as the DC resistance in each phase, R_{DC}, times the nominal current squared. It is clear that this component of the load-dependent losses can vary largely when the nominal current changes.

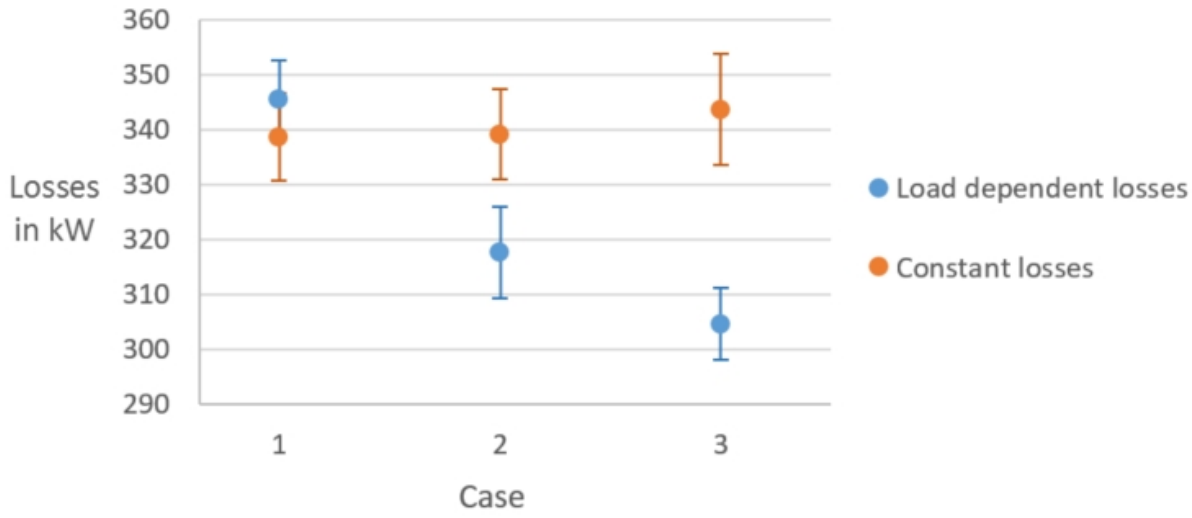


Figure 24: load-dependent and constant losses in when changing the price of losses

In addition, the slot height increases from case 1 to case 3 as can be seen in table 2, which in turn increases the copper area in the stator windings and by this decreases the DC-resistance. There is also a smaller decrease in the full load copper losses in the rotor windings. This coincides well with the increase of the pole core height, making more space for the field winding which reduces the current density and thus the copper losses. However, the standard deviation for both the pole core height and the full load rotor copper losses is high and indicate that more tests should be performed to conclude that this trend is in fact representative for what happens when the price of losses is changed. The constant losses on the other hand consists of magnetizing losses, iron losses, no load copper losses in rotor and friction and windage losses, which do not have the same ability to change. The magnetizing losses are small, around 5kw, and close to constant for all tests done. The iron losses are proportional to the weight of lamination in the stator. It also varies with the magnetic flux density in the stator, but that will stay close to constant. The iron losses increases as the load-dependent losses become more expensive, which implies that the amount of lamination in the machine is increasing. On the other hand, the no load copper losses in the rotor is decreasing at the same time. These losses are proportional to the field current and follows the same trend as the full load rotor copper losses as can be seen in figure 31. The decrease of the no load copper losses to some degree compensate for the increase of iron losses. Thus there are two reasons why the constant losses do not change as much as the load-dependent losses. First, the constant losses do not have the same ability to change as the load-dependent losses. Secondly, the no load copper losses in the rotor decreases at the same time as the iron losses increases. The friction and windage losses stay fairly constant for all cases.

Furthermore, the total losses are larger for case 1 than for case 2 and 3. As the price of losses changes, how the algorithm priorities to minimize the losses will also change. The change from case 1 to case 2 makes this very clear. The constant losses are equally expencive for both cases while the

load-dependent losses are 15000 kroner per kW in case 1 and 20000 kroner per kW in case 2. Since the cost of losses increases from case 1 to case 2, the algorithm will try to decrease the total losses at expense of the total cost of materials. This also explains some of the reason why the iron losses increases from case 1 to case 2. As the loss costs increases, the algorithm tries to minimize the losses by increasing the copper area in the machine and reducing current density. Similarly, the weight of the stator core is allowed to increase because the cost of materials now are relatively lower compared to the cost of losses. There is also an increase of the iron length from case 1 to case 2, which increase the machine weight including the weight of lamination. The standard deviation for the iron length is too large to say if this a qualitative increase. Nevertheless, as the amount of lamination increases, the iron losses increases too.

The second test was intended to reveal the relationship between power factor and machine-design. From the results it was clear that cheaper machines could be designed when lowering power factor requirements, as can be seen in figure 25.

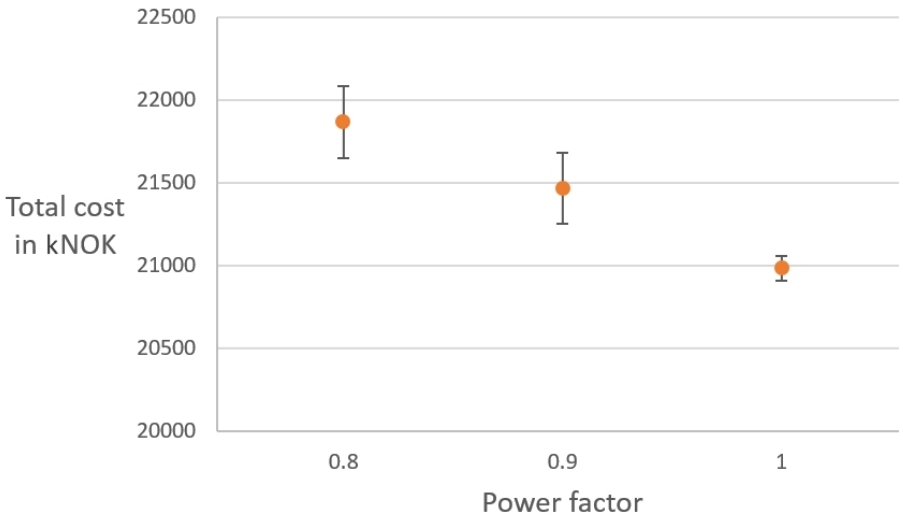


Figure 25: Total cost when changing the power factor

Again, the standard deviation is given by the error bars in the figure. The decrease in total cost is related to the decrease of the cost of losses shown in figure 26. As a consequence of a higher power factor, the total magnetization in the machine can be reduced that in turn demands a lower field current. This leads to a lower current density and consequently reduced rotor copper losses.

From table 3, it should be noted that the cost of copper wire is rising with an increased power factor while the cost of lamination and steel decrease. The material costs are proportional to the weight, and so an increased cost indicate an equally increased mass. As the total magnetization in the machine decreases, the flux density in rotor and stator will decrease. Thus, the cross section area of rotor and stator can be reduced to lower the cost of materials. This will increase the flux densities again, although they are limited by the nonlinear constraints. Since the machine weight is reduced, the moment of inertia drops as well which in some cases causes the constraint $T_M/T_w > 4$ not to be met. The algorithm therefore compensate by adding more copper to the machine. This is an unfortunate solution as copper is expensive. For a real life problem, a better solution would be to make the rotor ring larger.

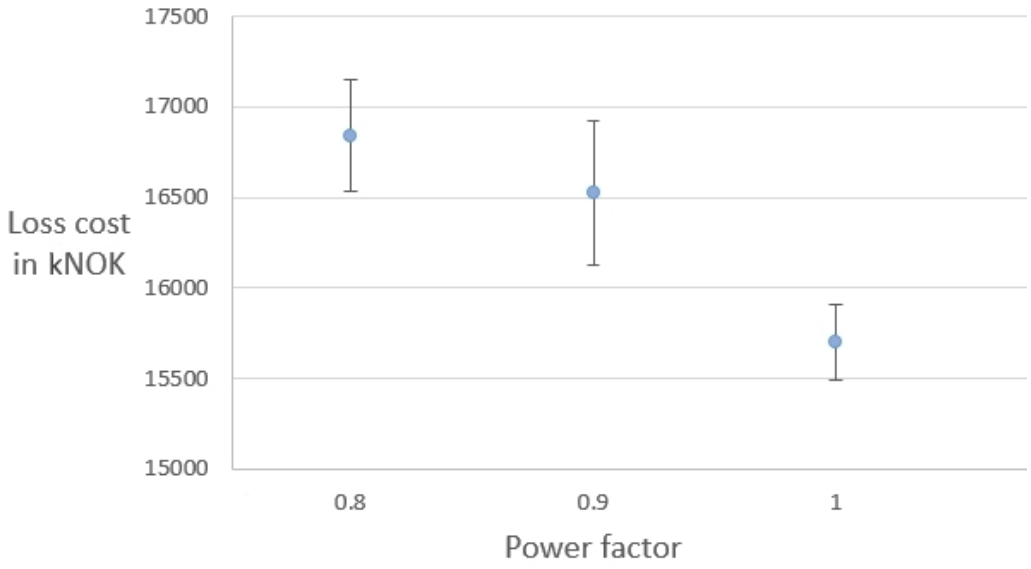


Figure 26: Cost of losses when changing the power factor

A first step to improve the optimization would be to implement two prices for steel in the rotor. One price for steel used in the poles, and a lower price for steel in the rotor ring. In case 4, this constraint was removed. As a consequence, the algorithm reduces the diameter to design a more compact machine as can be seen in table 4. As in case 3, the use of lamination and rotor steel are lower than in case 1 and 2. However, copper is no longer used to compensate for the reduced moment of inertia. Instead the machine becomes smaller and cheaper than in the other cases. The slot height increases to keep the stator copper losses low. With a smaller diameter there is less space for slots in the stator, so the algorithm compensate by adding copper to each slot, and at the same time lowering the number of slots. Similarly the pole core height and width decreases with the reduced diameter.

Lastly, the third test was constructed to examine how the requirement of the synchronous reactance defines the machine. Testing showed that there is a significant relationship between the synchronous reactance and the total cost of the machine as can be seen in figure 27. The synchronous reactance is given in per unit value. Of the component of the total cost, the only one to increase when X_d increased was the copper cost. The synchronous reactance is closely linked to the air gap between rotor and stator. When X_d decreases, this is because the air gap becomes smaller. A smaller air gap reduces the leakage flux and the magnetic voltage drop over the air gap, and consequently the total magnetization of the machine drops. This is illustrated in figure 28. Similarly to test 2, the reduced magnetization led to a lower flux density that again made the algorithm reduce the use of lamination and steel. The earlier prediction that a smaller air gap lead to a smaller machine does not seem to be the case here. Table 6 states that the inner diameter stays fairly constant when X_d is varied. The iron length on the other hand reduces from 1.42m to 1.24m going from X_d equal to 1.2pu to X_d equal to 2.0pu, though the length does not have the same impact on the moment of inertia as the diameter. Since the constraint related to the moment of inertia is active, the copper area in rotor and stator increases and the diameter is not allowed to shrink as it was predicted to.

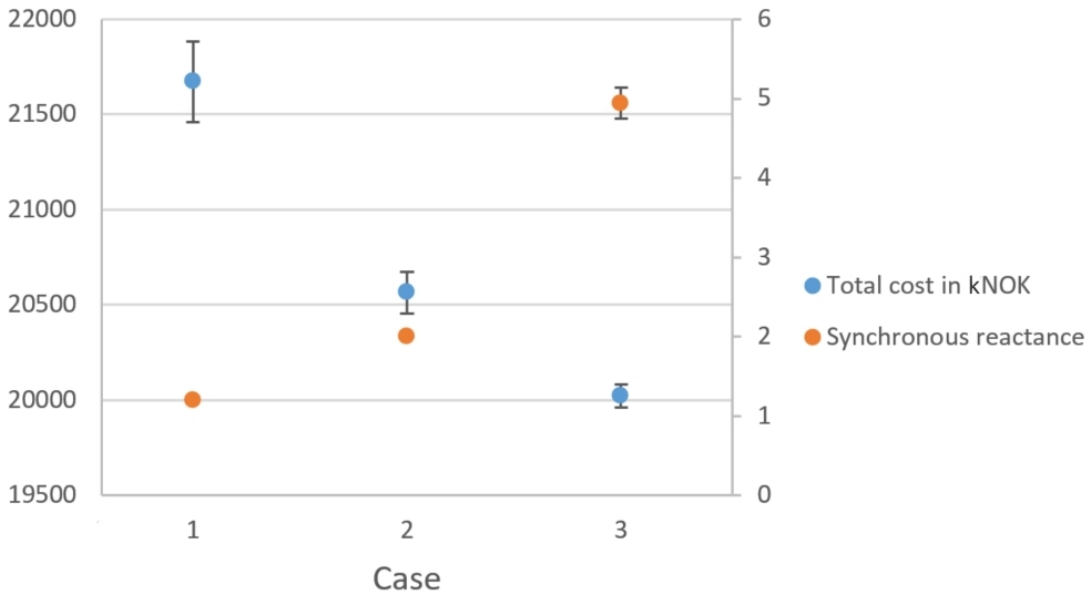


Figure 27: Total cost when the synchronous reactance increases

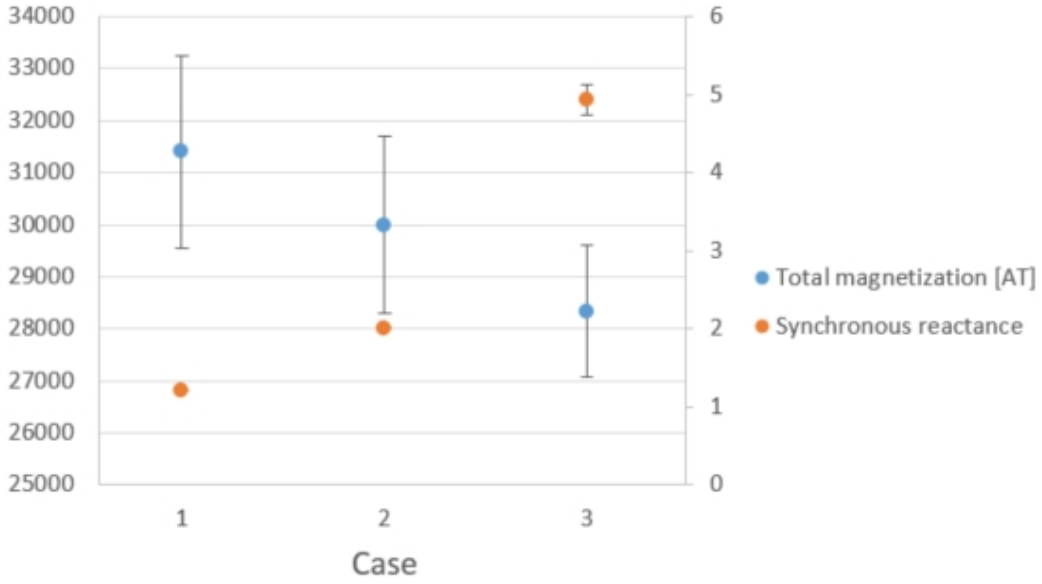


Figure 28: Total magnetization when the synchronous reactance increases

7 Conclusion

Performing genetic algorithms in MATLAB is well established because of embedded functions and the optimization application which simplifies their implementation. For the purpose of this assignment, the MATLAB-script *GenProg* was important in creating the objective function to be minimized. The script was rewritten to meet MATLAB's demands for objective functions, and to minimize the optimization run time. To include requirements of the machine-design the script *nonlcons* contains nonlinear constraints that corresponds to the requirements. Lastly, a final script *cost* was designed to contain optimization information and options, and to begin the algorithm. Default values were used for parameters as crossover fraction and population size. Results show that the optimal solution varies for each simulation. It is believed that increasing the population size will significantly reduce the result diversification. However, this will also lead to increased run time. It is not possible to include a hybrid solver because MATLAB cannot simultaneously deal with integer value variables and hybrid solvers. Although the results are diverse this is not necessarily indicative of a malfunctioning optimization, rather more likely indicates that the problem is very complex.

The testing of the optimization gave results as expected. Increasing the cost of load-dependent losses led to their decrease, however the material cost became greater. By changing the power factor from 0.8 to 1.0 the total cost of the machine went down 4.4%, which is a result of lower required magnetization. This was also the case when increasing the synchronous reactance. The results show that the algorithm has the ability to adjust the machine-design and find other solutions when parameters are altered by the user. However, more than ten optimizations should be performed for each case in order to provide results more credibility.

The genetic algorithm is a fitting optimization technique for synchronous generators due to proven performance on problems with high complexity and integer value variables. It is important, however, to adjust optimization options to fit the problem at hand. Doing so reduces the run time and increases the chances of the algorithm finding the best possible solution. In particular, the population size must be large enough to enable the algorithm to search the whole solution space. Limiting this size will ensure the run time is not increased too much. The user of the implemented optimization should also be aware of some parameters and constraints that affect the machine-design greatly. The ratio between price of losses and price of materials impacts the optimal solution to a large degree. Increasing the price of losses leads to a significant increase of copper in the machine. The constraints implemented limit machine-design, and can be omitted if necessary. They are chosen to ensure reliable machines that fulfill requirements of generators in the Norwegian grid. However, the requirement related to the moment of inertia is a guideline suggested by the Norwegian system operator Statnett SF. This constraint to a great extent decides the machine-design, and particularly affects the inner diameter.

7.1 Further work

To further test the validity of the optimization, suggested designs should be compared with real life generators. This would help verify that the algorithm is able to create functional machines. Comparisons may also reveal if there are considerably flaws to the design. Additional constraints may then have to be implemented. Furthermore, there may still be errors in the machine calculations performed in *Genkalk*. One way of testing this could be to send input values to *Genkalk* that represent a real life machine. The full machine-design can be viewed in the output Excel spread-sheet, which should then match those of the real life machine in question.

The prices of materials and losses are crucial for the machine-design. One step toward improving optimization would be to implement several prices on materials used in different parts of the machine.

In particular should the price of steel used in poles and rotor ring be different. Lastly, optimization options should be adjusted as this has not been in examined thoroughly in this assignment.

References

[1] "AC generator," visited 04.04.16. [Online]. Available: http://www.freeenergyplanet.biz/renewable-energy-systems/synchronous-machines-design-of-synchronous-machines.html

[2] P. E. Westgaard and D. O. W. Andersen, *Dimensjoneringseksempel for synkronmaskin.* NTH, 1965.

[3] "Feasible region," visited 29.03.16. [Online]. Available: http://fisher.osu.edu/~croxton.4/tutorial/Geometry/graphing3.html

[4] "Genetic algorithm," visited 11.03.16. [Online]. Available: https://en.wikipedia.org/wiki/Genetic_algorithm

[5] "Genetic algorithms," visited 30.03.16. [Online]. Available: http://www.lucifer.com/~david/thesis/survey.html

[6] "IEC 60034-1," visited 03.03.16. [Online]. Available: http://www.nostop.cn/1com1net/webeditor/UploadFile/200922717728626.pdf

[7] A. Lundseng and I. Vikan, "Upgrade and optimization of hydro power generators," Master's thesis, Norwegian University of Science and Technology, 2010.

[8] "Stator windings," visited 20.04.16. [Online]. Available: https://en.partzsch.de/stator-windings

[9] "ST5 mission," visited 01.04.16. [Online]. Available: http://www.nasa.gov/mission_pages/st-5/main/index.html

[10] "Vary crossover and mutations," visited 30.03.16. [Online]. Available: http://se.mathworks.com/help/gads/vary-mutation-and-crossover.html

[11] E. L. Engevik, "Optimal design of tidal power generator using stochastic optimization techniques," Master's thesis, Norwegian University of Science and Technology, 2014.

[12] "Construction of synchronous machines," visited 14.04.16. [Online]. Available: http://www.slideshare.net/Anu71/synchronous-machines-35168491

[13] "Funksjonskrav i kraftsystemet," visited 19.04.16. [Online]. Available: http://www.statnett.no/Global/Dokumenter/Kraftsystemet/Systemansvar/FIKS%202012.pdf

[14] "Genetic algorithm options," visited 18.04.16. [Online]. Available: http://se.mathworks.com/help/gads/genetic-algorithm-options.html

Appendix A, Generator specifications

Table 8: Generator specifications

Main input data		
Power rating	S_n	50MVA
Power factor	$\cos\phi$	0.9
Synchronous speed	n_s	375rpm
Number of poles	N_p	16
Runaway speed	n_r	612.459rpm
Maximum temperature rise	dTmx	89K
Stator input data		
Lamination length	b_{cs}	40mm
Cooling duct length	b_v	6mm
Number of parallel circuits	pnr	2
Strand height	h_{cus}	2.12mm
Height of slot wedge	h_{spk}	6mm
Height of bar seperator	h_m	7mm
Height of slot wedge spacer	h_{hgls}	2mm
Distance from slot wedge to air gap	h_{ds}	1mm
Roebel separator width	d_{rs}	0.5mm
Strand insulation	d_{icu}	0.1mm
Rotor input data		
Slot opening, damper windings	b_D	3mm
end plate height	h_{kr}	7mm
Insulation between field winding and pole	b_i	4mm
Field winding insulation	b_{if}	0.3mm
Pole shoe height	h_{ps}	50mm
Pole tooth height	h_{pt}	28mm

Appendix B, MATLAB-scripts

```
%% Genetic algorithm to solve optimization problem on synchronous generator

%% Formulation of pptimization problem for Genetic Algorithm

problem.solver = 'ga';                                  % Setter GA som optimaliseringsteknikk
problem.fitnessfcn = @Genkalk;                          % Setter objektivfunksjon
problem.nvars = 8;                                      % Setter antall variable
problem.lb = [13 2     20 10  20 200  30  80];          % Setter øvre grense for variablene
problem.ub = [65 4.2441 60 40 120 900 200 300];         % Setter øvre grense for variablene
problem.nonlcon = @nonlcons;                            % Setter funksjon for sjekk av ulineære betingelser
problem.intcon = [1,3,5,7];                             % Setter heltallsvariable

%% Options

problem.options = optimoptions('ga');                   % Velger at "options-struct" skal inneholde verdier for GA
problem.options = optimoptions(problem.options,'PlotFcn',{@gaplotbestf},... % Plotter det beste individet i hver generation,
    'Display','iter');                                  % skriver ut iterasjonen til kommandovinduet
problem.options = optimoptions(problem.options,'InitialPopulationRange'... % Setter grenser for den første generasjonen
    ,[20 3 25 15 30 280 70 110; 40 4.2 45 25 55 450 120 140]);
problem.options = optimoptions(problem.options,'CrossoverFraction',0.8); % Setter crossoverfraction, default-verdi er 0,8

%% Run with Genetic Algorithm

[x,f,exitflag,output,population,scores] = ga(problem);  % Setter i gang optimaliseringen, lagrer verdiene som er angitt
```

Figure 29: *cost*

```
1    function [c,ceq] = nonlcons(x)
2
3    %% Calculations%%...%%
388
389
390  %% Nonlinear constraints
391
392 -      c(1) = bu - tauu*0.8;              % Sjekker at bu < tauu*0.8
393 -      c(2) = -(taumn-0.002);             % Sjekker at taumn > 0. Hvis ikke vil enkelte verdier bli imaginære
394 -      c(3) = Ss*1e-6 - 5;                % Sjekker at strømtettheten i stator er mindre enn 5A/mm^2
395 -      c(4) = Sf*1e-6 - 5;                % Sjekker at strømtettheten i rotor er mindre enn 5A/mm^2
396 -      c(5) = -10*(polklaring-0.002);     % Sjekker at polklaringen er positiv
397 -      c(6) = Bdmax - 1.7;                % Sjekker at flukstettheten i statortenner er mindre enn 1.7T
398 -      c(7) = Bdrmx - 1.6;                % Sjekker at flukstettheten i polkjernen er mindre enn 1.6T
399 -      c(8) = Byr - 1.3;                  % Sjekker at flukstettheten i rotoråket er mindre enn 1.3T
400 -      c(9) = Xdpu - 1.2;                 % Sjekker at synkronreaktansen er mindre enn 1.2pu
401 -      c(10) = Xdt - 0.4;                 % Sjekker at transient synkronreaktans er mindre enn 0.4pu
402 -      c(11) = -(Xdtt - 0.15);            % Sjekker at subtransient synkronreaktans er større enn 0.15pu
403 -      c(12) = -(T_M/T_w-4);              % Sjekker at veiledende krav til treghetsmoment fra Statnett er fulgt
404
405 -      ceq = [];                          % Må stå som tom matrise når heltallsvariable benyttes
406 -  end
```

Figure 30: *nonlcons* collapsed

Appendix C, Optimization results

Pris	Hva måles	Pcusackw	Pcusdckw	Prflkw	Paddkw	Pmagnkw	PFekw	Prnlkw	Pfwkw	Plastavkw	Plastuavkw	Ptot
15000/30000	1	15.6676302	153.965552	36.5138292	129.819595	5.58045759	111.780958	43.2069935	170.326331	335.966606	330.894741	666.861346
	2	14.3860178	160.880692	39.1504233	129.850759	5.9798673	106.74317	46.2762524	173.775132	344.267892	332.774421	677.042314
	3	18.2963675	153.712914	44.5738802	129.739746	6.82172401	105.849275	52.87932	176.983165	346.322908	342.533485	688.856393
	4	16.6191287	166.954216	48.2875212	129.779556	7.37291367	101.471178	57.0398169	181.33102	361.640421	347.214929	708.85535
	5	15.3681861	157.269982	39.0753852	129.838415	5.97305955	105.667318	46.2540369	173.131982	341.551968	331.026396	672.578364
	6	16.3705333	161.087089	43.3319532	130.063939	6.61868915	101.429867	51.220749	193.899876	350.853514	353.169182	704.022696
	7	15.7553851	150.476016	38.291629	129.908746	5.85207527	105.646349	45.3094462	170.92832	334.431776	327.73619	662.167966
	8	16.5641612	158.525446	40.0698258	129.988875	6.12147385	103.282711	47.3798007	187.83907	345.148309	344.623056	689.771365
	9	15.5569204	159.525185	40.8879784	129.862809	6.24586132	103.246382	48.3386118	176.495993	345.832893	334.326848	680.159741
	10	16.2662127	154.833444	46.6104982	129.865192	7.11813206	100.082495	55.0771027	178.895844	347.575347	341.173574	688.748921
	snitt	16.0850543	157.723054	41.6792924	129.871763	6.36842538	104.51997	49.298213	178.360673	345.359163	338.547282	683.906446
20000/30000	1	19.7380954	135.733612	40.6564942	129.808937	6.21035721	114.559036	48.0628945	180.911578	325.937138	349.743866	675.681004
	2	17.492353	122.291073	39.1519208	129.9599	5.98877044	115.894882	46.4019426	170.582616	308.895247	338.86821	647.763457
	3	19.2647881	140.666198	40.9573495	130.028862	6.25746602	107.837529	48.4350222	192.987792	330.917198	355.517809	686.435007
	4	15.6807541	133.589034	38.2137065	130.148928	5.84009266	110.12274	45.2161886	175.332044	317.632423	336.511064	654.143487
	5	17.0470002	131.036402	36.2194168	129.907312	5.53478916	114.144294	42.8489998	171.371795	314.210131	333.899878	648.110009
	6	13.2572726	125.844985	35.2010269	130.236353	5.3794422	118.033635	41.6481475	163.112168	304.539638	328.173393	632.71303
	7	16.6253129	136.950817	37.5256977	129.828747	5.73278806	114.864021	44.3712745	170.680717	320.930575	335.648801	656.579376
	8	15.2650149	130.896821	37.7397867	130.03687	5.76622966	112.834513	44.6349227	169.069436	313.938493	332.305102	646.243595
	9	17.9187617	137.735318	42.3221594	129.89445	6.46549534	108.523263	50.0420597	180.825594	327.870689	345.856412	673.727101
	10	17.1746027	127.720157	35.6085066	129.864545	5.44214723	118.739675	42.1364539	167.846122	310.367811	334.164398	644.532209
	Snitt	16.9463956	132.246442	38.3596065	129.971491	5.8617578	113.555359	45.3797906	174.271986	317.523934	339.068893	656.592828
25000/25000	1	16.5328027	109.531733	32.4050898	130.060555	4.95417083	124.852024	38.3687791	161.108389	288.53018	329.283363	617.813543
	2	20.6978842	116.020368	37.8279017	130.007268	5.78126004	116.350571	44.7615275	183.354374	304.553422	350.247733	654.801155
	3	17.6284636	116.244999	35.0267423	129.958486	5.35247849	123.018375	41.4372362	169.075178	298.858691	338.883268	637.741959
	4	17.7349875	119.632307	39.4379089	130.076088	6.02508353	116.843853	46.6347129	180.168602	306.881292	349.672251	656.553543
	5	21.1928687	120.204367	38.3422132	129.769118	5.85971121	118.567316	45.367947	170.943328	309.508567	340.738302	650.246869
	6	20.4392849	117.482495	37.6300566	130.094751	5.75100886	116.031293	44.5272129	186.757277	305.646588	353.066791	658.713378
	7	20.5556361	115.672043	37.7376627	130.185773	5.76711433	117.940553	44.6496849	193.650834	304.151115	362.008187	666.159301
	8	20.2893763	118.536435	36.0148217	130.165254	5.50295637	117.527118	42.5988408	196.548238	305.005887	362.177153	667.183039
	9	19.7829966	122.743908	41.5011275	130.146462	6.34257447	112.482094	49.1070792	191.334578	314.174495	359.266325	673.44082
	10	18.3885764	118.390921	41.086046	130.020461	6.29487238	115.235339	48.8407023	177.767804	307.886004	348.138718	656.024722
	Snitt	19.3242877	117.445958	37.700957	130.048422	5.76312305	117.884854	44.6293723	181.07086	304.519624	343.585086	648.10471

Figure 31: Changing cost of losses, losses

Pris	Hva måles	Ctot	Qss	Di	nv	bu	ndl	bpk	nf	bcuf	delta0
15000/30000	1	19971.5801	33	3.59898871	29	19.249206	40	354.483494	119	110.257107	0.02016054
	2	20018.9322	31	3.63891862	28	20.8341388	37	357.493623	114	112.690417	0.02092497
	3	20129.1311	29	3.71891712	25	21.5189891	42	375.940865	101	118.408986	0.02176935
	4	20366.1543	31	3.76779438	24	20.737887	40	378.501864	95	123.615682	0.0229067
	5	19922.3373	29	3.63490647	28	22.5128911	37	357.546275	111	116.4522	0.02102695
	6	20573.2675	33	3.84341835	24	21.1401348	40	373.133225	100	132.318047	0.02359759
	7	19814.7446	27	3.5851499	30	25.365064	36	341.590811	109	119.253617	0.02090581
	8	20316.0687	30	3.78521766	25	22.3896767	39	371.824734	108	124.809853	0.02231328
	9	19999.5205	29	3.67498479	27	22.9615817	37	357.07551	106	121.592752	0.02177662
	10	20088.1823	25	3.70985355	26	27.1998886	37	358.720547	94	127.695908	0.02265495
	snitt	20119.9919	29.7	3.69581495	26.6	22.3909458	38.5	362.631095	105.7	120.709457	0.02180368
20000/30000	1	21921.38	28	3.74324117	25	21.0682014	47	390.666244	112	110.532945	0.02047427
	2	21548.22	32	3.56601476	31	21.6283662	45	333.755326	105	123.248381	0.02117033
	3	22157.13	30	3.83805225	24	22.122939	45	382.185912	106	125.649586	0.02237556
	4	21659.22	33	3.57880083	32	22.6356659	40	319.860593	103	132.037425	0.02205227
	5	21484.56	29	3.58791767	30	22.6929449	42	347.537706	116	115.123488	0.02019659
	6	21616.51	34	3.39042871	40	22.3119551	38	290.533581	109	125.758461	0.02051342
	7	21573.44	31	3.60121663	29	20.3978873	43	355.697978	116	111.742597	0.02016812
	8	21503.75	32	3.52443901	33	22.3811786	40	320.934322	108	123.322193	0.02113567
	9	21801.66	28	3.72169605	26	23.2932634	43	367.620708	103	122.31934	0.02173168
	10	21465.97	29	3.54887016	31	21.8100197	43	344.846508	113	114.023477	0.01916639
	Snitt	21673.1851	30.6	3.61006772	30.1	22.0342421	42.6	345.363888	109.1	120.375789	0.02089843
25000/25000	1	21240.05	29	3.40335544	38	24.0811271	43	307.36246	119	116.47205	0.01884687
	2	21556.61	26	3.71685433	27	25.1772433	48	369.783704	112	118.302489	0.02070232
	3	21382.78	32	3.54025691	32	20.6526184	47	341.455848	119	112.478053	0.01969465
	4	21630.62	34	3.65965615	29	20.5966471	48	345.176611	105	125.918991	0.02177004
	5	21360.16	26	3.62117163	28	23.4244875	48	365.437897	113	110.471167	0.01966538
	6	21678.17	30	3.73720832	27	22.7968656	49	362.490667	110	124.439328	0.02147045
	7	21896.85	33	3.79815942	26	20.9052528	52	368.165286	109	127.87408	0.02191082
	8	21924.22	29	3.83642585	25	22.4618391	50	385.238454	119	119.244402	0.02108536
	9	21895.67	33	3.78462764	26	21.5537175	49	361.807847	100	132.180098	0.02284626
	10	21547.58	31	3.6450835	29	22.4224567	47	344.289866	100	127.370136	0.02178203
	Snitt	21611.2715	30.3	3.67427992	28.7	22.4072255	48.1	355.120864	110.6	121.475079	0.02097742

Figure 32: Changing cost of losses, geometrical parameters

cos(phi)		Ptot	In	Un	Ctap	Clam	Csteel	Cwirestator	Cwirerotor	Ctot
0,8	1	648713.083	4417.79393	6534.37302	16357.8804	1174.49885	2431.18539	601.187446	975.97434	21540.7265
	2	671828.819	5208.66389	5542.21083	16909.7792	1158.81473	2355.21377	541.123548	793.111258	21758.0425
	3	671263.418	4156.28082	6945.51565	16933.0724	1159.98602	2380.06018	568.110019	905.2636	21946.4922
	4	674415.169	4712.95268	6125.14392	17032.7784	1176.24932	2388.46884	548.541041	839.028679	21985.0663
	5	644595.49	4761.8871	6062.20031	16249.4159	1189.07026	2472.42479	583.489502	984.036751	21478.4372
	6	661615.576	4180.89802	6904.62032	16674.3463	1275.78783	2552.42534	475.651137	872.427205	21850.6378
	7	668260.601	4719.22056	6117.00875	16886.7815	1139.78725	2378.49612	600.10523	914.606054	21919.7762
	8	680403.761	4736.61142	6094.54965	17223.5774	1163.17763	2397.14814	562.797241	873.865468	22220.5659
	9	682148.189	4420.02143	6531.07998	17209.4654	1153.72492	2350.70923	556.438774	825.930654	22096.269
	10	669853.833	5175.59316	5577.62417	16924.4728	1155.96	2349.0722	591.80069	844.521635	21865.8274
0,9	Snitt	667309.794	4648.9923	6243.43266	16840.157	1174.70568	2405.5204	562.924463	882.876564	21866.1841
	1	675681.004	5053.13348	5712.79456	17011.0587	1155.58459	2337.16052	578.724282	838.856185	21921.3843
	2	647763.457	4215.53316	6847.89144	16343.9512	1168.47371	2353.52215	688.962936	993.307851	21548.2179
	3	686435.007	5122.95458	5634.9345	17283.8782	1096.05743	2281.55516	606.302798	889.34011	22157.1337
	4	654143.487	4192.36112	6885.74114	16447.9804	1122.63406	2347.19422	673.674345	1067.74099	21659.224
	5	648110.009	4647.77087	6211.04488	16301.199	1158.11475	2406.18294	617.814185	1001.25404	21484.5649
	6	632713.03	3545.58947	8141.80933	15935.9945	1201.02907	2516.00844	721.064491	1242.41404	21616.5106
	7	656579.376	4320.39645	6681.6816	16488.0755	1162.61832	2389.65948	582.411597	950.676102	21573.441
	8	646243.595	4127.69806	6993.61074	16247.9229	1147.80417	2388.96632	657.919252	1061.13665	21503.7493
	9	673727.101	5141.19622	5614.941	16933.1061	1103.63962	2279.56937	611.202303	874.139485	21801.6569
	10	644532.209	4482.98252	6439.35446	16232.2882	1200.20134	2440.48825	606.915214	986.075481	21465.9684
1,0	Snitt	656592.828	4484.96159	6516.38036	16522.5455	1151.61571	2374.03068	634.49914	990.494094	21673.1851
	1	642280.017	5242.41697	5506.52755	16044.4283	990.053881	2109.12703	762.922414	1082.76802	20989.2996
	2	627410.997	4951.30176	5830.28765	15716.3128	1134.31475	2330.97258	627.280556	1056.94812	20865.8288
	3	615087.336	4382.05781	6587.66148	15450.5485	1201.54181	2498.74297	639.764202	1160.52375	20951.1212
	4	637898.145	4272.27658	6756.93929	15949.2126	1020.30194	2146.63055	779.122038	1145.1122	21040.3794
	5	619250.27	4962.05205	5817.65632	15476.9657	1092.9606	2337.67773	676.776034	1196.38825	20780.7683
	6	629327.392	4750.45701	6076.78659	15742.7039	1061.38176	2254.43211	711.756597	1105.74074	20876.0151
	7	629009.561	5096.37676	5664.32091	15747.8395	1113.61806	2295.08669	640.778736	1060.09096	20857.4139
	8	612032.861	4559.36404	6331.47807	15328.6292	1138.5141	2421.79649	691.574856	1274.80619	20855.3208
	9	631369.09	5040.51651	5727.09432	15779.0205	1029.66242	2193.44668	740.549117	1122.84236	20865.521
	10	629198.064	4007.68841	7203.0334	15736.6596	1068.8339	2267.72446	738.856995	1171.68503	20983.76
	Snitt	627286.373	4726.45079	6150.17856	15697.232	1085.11832	2285.56373	700.938154	1137.69056	20906.5428

Figure 33: Changing power factor, costs

cos(phi)		Qss	Di	nv	bu	ndl	bpk	nf	bcuf	delta0
0,8	1	29	3.52530492	33	23.2934787	38	332.187584	100	121.989127	0.01946301
	2	26	3.63411883	28	23.0799773	41	373.895645	97	112.888856	0.01925487
	3	33	3.66236998	29	20.2057235	41	355.201539	99	123.338331	0.02035323
	4	29	3.7062611	27	20.943926	43	378.638633	102	115.770968	0.01958329
	5	27	3.51770275	33	24.2308289	38	337.006317	106	116.321271	0.018676
	6	30	3.56549633	30	17.9418827	44	376.653207	118	98.6668544	0.01703819
	7	29	3.68230015	29	23.6465363	39	353.856548	97	126.826346	0.02063315
	8	30	3.78886195	26	21.216831	43	383.382966	103	121.37364	0.02023492
	9	31	3.71354909	27	20.3706186	42	374.103107	96	120.801892	0.02038286
	10	27	3.67795954	28	23.8454489	42	362.912451	94	123.553173	0.01999801
0,9	Snitt	29.1	3.64739246	29	21.8775252	41.1	362.7838	101.2	118.153046	0.01956175
	1	28	3.74324117	25	21.0682014	47	390.666244	112	110.532945	0.02047427
	2	32	3.56601476	31	21.6283662	45	333.755326	105	123.248381	0.02117033
	3	30	3.83805225	24	22.122939	45	382.185912	106	125.649586	0.02237556
	4	33	3.57880083	32	22.6356659	40	319.860593	103	132.037425	0.02205227
	5	29	3.58791767	30	22.6929449	42	347.537706	116	115.123488	0.02019659
	6	34	3.39042871	40	22.3119551	38	290.533581	109	125.758461	0.02051342
	7	31	3.60121663	29	20.3978873	43	355.697978	116	111.742597	0.02016812
	8	32	3.52443901	33	22.3811786	40	320.934322	108	123.322193	0.02113567
	9	28	3.72169605	26	23.2932634	43	367.620708	103	122.31934	0.02173168
	10	29	3.54887016	31	21.8100197	43	344.846508	113	114.023477	0.01916639
1,0	Snitt	30.6	3.61006772	30.1	22.0342421	42.6	345.363888	109.1	120.375789	0.02089843
	1	32	3.74350113	24	23.4141717	49	343.788711	134	123.8785	0.02756939
	2	28	3.60009597	28	22.5742531	45	339.443089	127	116.582633	0.02088747
	3	28	3.45387795	35	23.8940513	39	309.594136	120	117.529041	0.01901285
	4	38	3.68631895	26	20.3442951	50	330.870627	133	126.096839	0.0274332
	5	29	3.57185727	29	23.7488632	44	327.832253	144	114.583402	0.02235133
	6	32	3.63831386	27	22.2693879	47	342.818101	138	115.002137	0.02434795
	7	28	3.62665221	27	22.7847515	46	342.201327	129	117.659044	0.02164142
	8	29	3.47117468	34	24.6906475	40	301.493077	130	121.74654	0.02100065
	9	31	3.66979455	26	23.4630376	47	337.229173	137	120.296671	0.02548126
	10	38	3.6140828	28	19.5873961	49	334.687697	144	114.517539	0.02524315
	Snitt	31.3	3.60756694	28.4	22.6770855	45.6	330.995819	133.6	118.789235	0.02349687

Figure 34: Changing, geometrical parameters

Xd		Ptot	In	Ctap	Clam	Csteel	Cwirestator	Cwirerotor	Ctot	Xd	delta0
< 1,2	1	675681.006	5053.13348	17011.0588	1155.58459	2337.16052	578.724281	838.856186	21921.3844	1.20078786	0.02047427
	2	647763.459	4215.53316	16343.9513	1168.47371	2353.52215	688.962936	993.307851	21548.2179	1.19886432	0.02117033
	3	686435.006	5122.95458	17283.8782	1096.05743	2281.55516	606.302798	889.34011	22157.1337	1.20094644	0.02237556
	4	654143.486	4192.36112	16447.9803	1122.63406	2347.19422	673.674345	1067.74099	21659.224	1.20096159	0.02205227
	5	648110.009	4647.77087	16301.199	1158.11475	2406.18294	617.814185	1001.25404	21484.5649	1.20096496	0.02019659
	6	632713.03	3545.58947	15935.9945	1201.02907	2516.00844	721.064491	1242.41404	21616.5106	1.20093839	0.02051342
	7	656579.376	4320.39645	16488.0755	1162.61832	2389.65948	582.411597	950.676102	21573.441	1.20095423	0.02016812
	8	646243.595	4127.69806	16247.9229	1147.80417	2388.96632	657.919252	1061.13665	21503.7493	1.20095727	0.02113567
	9	673727.1	5141.19622	16933.1061	1103.63962	2279.56937	611.202303	874.139485	21801.6569	1.20053442	0.02173168
	10	644532.21	4482.98252	16232.2882	1200.20134	2440.48825	606.915214	986.075481	21465.9685	1.20092589	0.01916639
	snitt	656592.828	4484.96159	16522.5455	1151.61571	2374.03068	634.49914	990.494094	21673.1851	1.20068353	0.02089843
<2	1	634449.012	4989.0095	15740.5967	1021.30883	2147.35337	695.080954	976.311183	20580.651	2.00053409	0.01427106
	2	629665.927	5526.87733	15625.6546	1036.47867	2191.86147	673.885988	971.620185	20499.5009	1.99896123	0.0134907
	3	628389.044	5340.73395	15592.4251	1023.31709	2166.39635	710.051965	1013.15725	20505.3478	2.00086493	0.01397911
	4	619710.713	5071.86221	15405.3092	1091.34738	2294.89498	643.141909	1036.97753	20471.671	1.99002559	0.01250982
	5	630046.295	5534.02076	15658.993	1042.41008	2194.00593	695.504125	941.697061	20532.6102	2.00040093	0.01352954
	6	644156.658	5615.69448	15945.4152	1020.79428	2134.0031	639.624666	902.776553	20642.6138	1.9991462	0.01376557
	7	608126.676	4368.60122	15156.2184	1120.24188	2363.28504	690.882517	1159.34408	20489.9719	2.00088983	0.01221515
	8	616300.486	5183.62071	15330.4625	1074.92241	2311.61654	683.659512	1029.27015	20429.9311	2.00075817	0.01274907
	9	648927.408	4730.24578	16091.2399	989.26935	2054.42705	738.790352	934.205263	20807.9319	1.99997092	0.01554347
	10	640635.611	4584.84541	15880.0043	1007.08069	2101.51032	713.637131	985.868958	20688.1014	2.00001498	0.0149646
	snitt	630040.783	5094.55114	15642.6319	1042.71707	2195.93541	688.425912	995.122822	20564.8331	1.99915669	0.01370181
ingen begrensning	1	599854.315	5408.4106	14766.7578	1008.91418	2257.50553	746.483431	1165.02669	19944.6877	4.62155106	0.00530459
	2	613049.754	5770.88644	15078.8977	994.693016	2142.14709	746.548774	1028.29321	19990.5798	5.08149934	0.00498142
	3	598759.436	5000.04021	14745.4601	1051.31769	2313.36917	710.303865	1208.51929	20028.9702	5.33562512	0.00418863
	4	614762.109	5681.14589	15087.5721	982.224082	2182.17325	709.839742	1020.97669	19982.7858	5.06615603	0.00495196
	5	617861.48	5246.62763	15180.0431	967.432254	2102.00564	763.334415	1032.24944	20045.0649	4.67412366	0.00589286
	6	622790.676	5459.72486	15295.4727	963.540415	2076.69559	774.379023	970.886208	20080.974	4.95515087	0.00553231
	7	629196.589	5751.05291	15454.5463	1000.21807	2136.25815	685.824598	877.475357	20154.3224	5.00663183	0.00505689
	8	604380.833	5945.11672	14886.0757	1015.37075	2228.51718	735.872252	1081.24982	19947.0857	4.88000797	0.0048573
	9	614728.103	5213.95529	15105.7326	977.499386	2101.05085	761.680157	1082.65623	20028.6192	4.83447628	0.00559544
	10	614471.342	5615.93699	15104.8955	999.18606	2154.66338	718.010124	1034.16084	20010.9159	4.93798464	0.00509999
	snitt	612985.464	5509.28975	15070.5454	996.03959	2169.43858	735.227638	1050.14938	20021.4006	4.93932068	0.00514614

Figure 35: Changing constraint for synchronous reactance, costs

Xd		Qss	Di	nv	bu	ndl	bpk	nf	bcuf	delta0
< 1,2	1	28	3.74324117	25	21.0682014	47	390.666244	112	110.532945	0.02047427
	2	32	3.56601476	31	21.6283662	45	333.755326	105	123.248381	0.02117033
	3	30	3.83805225	24	22.122939	45	382.185912	106	125.649586	0.02237556
	4	33	3.57880083	32	22.6356659	40	319.860593	103	132.037424	0.02205227
	5	29	3.58791767	30	22.6929449	42	347.537706	116	115.123488	0.02019659
	6	34	3.39042871	40	22.3119551	38	290.533581	109	125.758461	0.02051342
	7	31	3.60121662	29	20.3978873	43	355.697978	116	111.742597	0.02016812
	8	32	3.52443901	33	22.3811786	40	320.934322	108	123.322192	0.02113567
	9	28	3.72169605	26	23.2932634	43	367.620708	103	122.31934	0.02173168
	10	29	3.54887016	31	21.8100197	43	344.846508	113	114.023477	0.01916639
	snitt	30.6	3.61006772	30.1	22.0342421	42.6	345.363888	109.1	120.375789	0.02089843
<2	1	32	3.69898668	25	22.3008253	47	351.608814	117	124.146099	0.01427106
	2	28	3.68586402	25	24.189374	46	359.165195	123	117.78978	0.0134907
	3	30	3.66897189	26	24.20454	46	344.447693	118	124.968971	0.01397911
	4	29	3.6141546	27	22.5342105	46	351.828512	132	112.502189	0.01250982
	5	28	3.69261269	25	24.3733201	47	366.063769	120	116.67191	0.01352954
	6	29	3.7473983	23	22.6113935	48	371.867949	119	118.210869	0.01376557
	7	31	3.50723578	32	22.9057313	42	317.336038	122	122.206897	0.01221515
	8	28	3.59885371	28	24.5565064	44	353.830171	131	109.97547	0.01274907
	9	36	3.75678103	24	21.1545532	49	343.411416	103	137.18233	0.01554347
	10	36	3.71096723	25	20.7617323	48	341.430009	110	132.882827	0.0149646
	snitt	30.7	3.66818259	26	22.9592187	46.3	350.098957	119.5	121.653734	0.01370181
ingen begrensning	1	28	3.56338701	29	27.0412402	42	331.759585	136	117.568288	0.00530459
	2	28	3.67236407	25	25.828493	47	351.560425	126	121.671819	0.00498142
	3	29	3.51211323	31	25.5170523	41	318.434758	132	120.526212	0.00418863
	4	28	3.64915038	26	26.3373427	43	356.463863	128	116.307293	0.00495196
	5	32	3.68653941	25	23.9740046	47	348.738763	123	124.937531	0.00589286
	6	31	3.71974362	24	24.4044245	48	358.574483	118	125.000302	0.00553231
	7	29	3.73886729	23	23.5176364	49	389.021539	124	110.175873	0.00505689
	8	26	3.59508288	28	28.5006757	43	340.290155	126	119.778561	0.0048573
	9	32	3.67524474	25	23.5614347	48	338.719382	126	128.15469	0.00559544
	10	29	3.66852444	25	24.6718418	47	353.048814	129	119.697187	0.00509999
	snitt	29.2	3.64810171	26.1	25.3354146	45.5	348.661177	126.8	120.381776	0.00514614

Figure 36: Changing constraint for synchronous reactance, geometrical parameters