NTNU – Trondheim
Norwegian University of
Science and Technology

Parallel processing of optimization algorithms

A faster way to optimize electric power problems that utilize FEM-simulations

Anne-Siri Borander

Master of Energy and Environmental Engineering
Submission date: June 2014
Supervisor: Robert Nilssen, ELKRAFT

Norwegian University of Science and Technology
Department of Electric Power Engineering

Problem description

When designing an electric machine one wants to find the number of slots, and their placement, that gives the largest flux density in the air gap. Or even more the shape of the magnets to give the highest torque in a given permanent machine. In some cases the problem is even extended by the need of optimizing the size of cooling channels for a fluid to provide the most efficient cooling. For all these cases a FEM-based optimization would be the best choice to get the wanted result. Doing such optimizations is possible today, but except for simple cases this will take a lot of time.

The goal of this thesis is to prove that the above is possible with today's solutions. In other words, to find a way of doing optimizations based on FEM simulations, and further try to parallelize to decrease the execution time. If proven possible the difficulty setting up these simulation and the expected decrease in runtime are expected to be evaluated. The goal can be summarized in the following title:

Parallel processing of optimization algorithms - A faster way to optimize electric power problems that utilize FEM-simulations

It is also expected that the student gives a good description of the procedure and setup of scripts used.

Supervisor:

Prof. Robert Nilssen

Abstract

The aim of this thesis was to find a way of doing optimizations based on FEM-simulations and further to parallelize the computations in order to decrease the execution time. If proven possible the difficulty setting up these simulation and the expected decrease in runtime, would to be evaluated.

Optimizations of magnet field strength and torque utilizing FEM simulations were conducted by using the particle swarm optimization (PSO) algorithm and the Matlab "partial differential equation toolbox" (PDE-toolbox). Two cases were optimized with and without parallel processing, and runtimes were measured for all runs. The aim was to run all simulations 10 times on three different computers. This was achieved to the majority of the simulations, results therefore had a good statistical confidence.

The optimization results were consistent with the theory: the higher number of iterations and particles used in the PSO, the better solution and smaller deviations. The runtime was found to be linear with the product of iterations and particles. This fits the expectations and the theory since the product of iterations and particles equals the total number of FEM calculations done. The FEM-simulation was the most time consuming when executing the code.

The simplest case took up to 7 hours without parallel processing. The same simulation was down to 20 minutes using 12 parallels. The speedup was proportionally alike to the number of cores for the 50/50 simulation in case 1. Case 2 had a lower speedup, but this was also linear. The same tendency was found for sets with fewer particles/iterations, but in these cases deviations were significant.

Setting up the model in the PDE-toolbox from command line was demanding. This may also be due to lack of example cases, even Internet searches turned out empty for similar simulation setups. But worth while when considering the reduced runtime one achieved. To prepare the optimization algorithm for parallel processing was however easy and took very little time.

Sammendrag

Målet med denne masteroppgaven var først og fremst å vise at er mulig å kjøre optimaliseringer basert på elementmetode-beregninger (FEM). Videre skulle parallell prosessering av optimeringsalgoritmen bli brukt for å redusere kjøretiden. Hvis dette viste seg å være mulig skulle vanskelighetsgraden ved å sette opp slike simuleringer og den forventede reduksjonen i kjøretid bli evaluert.

Magnetisk feltstyrke og moment ble optimalisert ved hjelp av elementmetode-beregninger av magnetfeltet. Partikkel-sverm-optimering (partical swarm optimization, PSO) ble brukt sammen med Matlabs "partial differential equation toolbox" (PDE-toolbox) for å kjøre optimaliseringen. Alle simuleringer ble gjort på to forskjellige optimaliseringsproblemer. Begge problemene ble optimert både med og uten bruk av parallellprosessering. Målet var å kjøre alle simuleringer ti ganger og på tre ulike datamaskiner. For de aller fleste simuleringene ble dette også gjort, så de fleste resultatene hadde dermed et godt statistisk grunnlag.

Resultatene fra optimeringene stemte godt overens med teorien; jo flere iterasjoner/partikler, jo bedre ble løsningen og jo mindre ble avviket. Det ble funnet at kjøretiden var en lineær funksjon av produktet av iterasjoner ganget med antall partikler. Partikler ganget med iterasjoner utgjorde antallet FEM-simuleringer simuleringer som ble kjørt, og det var FEM-simuleringene som tok mest av tid når under optimeringene.

Det enkleste av testoppsettene tok opp til 7 timer å optimere uten bruk av parallellprosessering. Den samme optimeringen nede i 20 minutter da 12 paralleller kjørt. For simuleringen med 50/50 (iterasjoner/partikler) var reduksjonen i kjøretid nesten proporsjonalt lik antallet paralleller kjørt for det enkleste testoppsettet. For det andre testoppsettet var sammenhengen også lineær, men reduksjonen i kjøretid var lavere. Tendensen var den samme for settene med færre iterasjoner/partikler, men her fikk man også store avvik.

Det var krevende å få satt opp modellen i PDE-toolbox fra kommandolinja i Matlab. Selv ikke ved grundige internettsøk fant man eksempler på tilsvarende oppsett som det som ble brukt her. Ved å muliggjøre parallell kjøring av FEM-simuleringene i optimeringsalgoritmen ble kjøretiden kraftig redusert, så det var uansett verdt tiden det tok å sette det opp. Når både FEM-simuleringene og optimeringsalgoritmen først fungerte, viste det seg å være lett å endre koden til å kjøre i parallell.

Parallel processing of optimization algorithms

A faster way to optimize electric power problems that utilize FEM-simulations

Student Anne-Siri Borander, Department of Electric Power Engineering
Norwegian University of Science and Technology, Trondheim, Norway
Email: annesibo@alumni.ntnu.no

***Abstract*—The aim of this thesis was to find a way of doing optimizations based on FEM-simulations and further to parallelize the computations in order to decrease the execution time. If proven possible the difficulty setting up these simulation and the expected decrease in runtime, would to be evaluated.**

Optimizations of magnet field strength and torque utilizing FEM simulations were conducted by using the particle swarm optimization (PSO) algorithm and the Matlab "partial differential equation toolbox" (PDE-toolbox). Two cases were optimized with and without parallel processing, and runtimes were measured for all runs. The aim was to run all simulations 10 times on three different computers. This was achieved to the majority of the simulations, results therefore had a good statistical confidence.

The optimization results were consistent with the theory: the higher number of iterations and particles used in the PSO, the better solution and smaller deviations. The runtime was found to be linear with the product of iterations and particles. This fits the expectations and the theory since the product of iterations and particles equals the total number of FEM calculations done. The FEM-simulation was the most time consuming when executing the code.

The simplest case took up to 7 hours without parallel processing. The same simulation was down to 20 minutes using 12 parallels. The speedup was proportionally alike to the number of cores for the 50/50 simulation in case 1. Case 2 had a lower speedup, but this was also linear. The same tendency was found for sets with fewer particles/iterations, but in these cases deviations were significant.

Setting up the model in the PDE-toolbox from command line was demanding. This may also be due to lack of example cases, even Internet searches turned out empty for similar simulation setups. But worth while when considering the reduced runtime one achieved. To prepare the optimization algorithm for parallel processing was however easy and took very little time.

***Index Terms*—Matlab, Parallel processing, techila, PSO, FEM, Optimization**

I. Introduction

When designing an electric machine the aim is to find the number of slots and their placements, which give the largest flux density in the air gap. It is even better to the shape of the magnets to give the highest torque in a given permanent machine in addition. The problem is in some cases extended by the need of optimizing the size of cooling channels for a fluid to provide the most efficient cooling. A FEM-based optimization would be the best choice for all these cases to achieve the wanted result. Such optimizations are possible today, but will take a lot of time apart from in simple cases.

Finite element method simulations (FEM-simulations) are used to solve equations with no universally valid solution. *Optimization algorithms* helps us to find the best set of parameters or the most efficient configuration. *Parallel processing* can be used to speed up independent loops and make things go much faster. Is it possible to utilize the strengths of all these tools at the same time?

The aim of this study was to prove that the above was possible with today's solutions. In other words, try to find a way of running FEM based optimizations, and further try to parallel process the simulation in order to decrease the execution time. If proven possible, the difficulty setting up these simulations, and the decrease in runtime were expected to be evaluated. The aim of this thesis can be summarized in the following title:

Parallel processing of optimization algorithms - A faster way to optimize electric power problems that utilize FEM-simulations

II. Definitions and scope

The results presented here are limited to the use of the Matlab and Comsol software packages. The only focus has been to prove the concept and make the simulations work. In most of the simulations only the runtime of the simulations have been measured. How good the of algorithm itself was has only been discussed briefly.

A list of words, abbreviations and terms used are listed in table I.

TABLE I
Abbreviations and terms

PDE	partial differential equation
FEM	finite element method
PSO	Particle swarm optimization
runtime	time measured from pushing execute to results are returned.
machine	used about electrical machines/motors. PC or computer is consistently used in that intent.
cores	used about computer cores. Not necessarily the same as number of processors.

Italic has been used for Matlab commands or functions and **bold** for variables and matrices throughout the report.

III. Theory

This section is meant to give the reader necessary theory to understand the results presented later in this report. Most equations and tools used in the later chapters, are presented here. The reader is expected to know the basics of magnetic- and electrical fields, and the laws which apply. It is also assumed that the reader is familiar with differential equations, and systems of these.

A. Electromagnetism

The only cases used in this study are electro-magnetical. Maxwell's equations together with the definition of vector potential are all needed to calculate the magnetic field. Only two of Maxwell's equations are relevant [1, p14]:

$$\nabla \times \overrightarrow{H} = \overrightarrow{J} + \frac{\partial \overrightarrow{D}}{\partial t} \tag{1}$$

$$\nabla \cdot \overrightarrow{B} = 0 \tag{2}$$

where the following applies for linear materials

$$\overrightarrow{B} = \mu \overrightarrow{H} \tag{3}$$

The definition of vector potential is [1][p14]:

$$\overrightarrow{B} = \nabla \times \overrightarrow{A} \tag{4}$$

These relations are already implemented in the software used. By defining the permeability (μ) in each domain, the currents ($\overrightarrow{\mathbf{J}}$) and the boundary conditions, the software sets up the needed set of PDEs automatically.

In the plane case (assuming $\nabla \cdot \mathbf{A} = 0$ and that the current density is perpendicular to the plane) the following PDE arise [2]:

$$-\nabla \cdot \left(\frac{1}{\mu} \nabla A \right) = J \tag{5}$$

It is worth noting that in the plane case (2D), the magnetic field density can be calculated from A:

$$\mathbf{B} = \left(\frac{\partial A}{\partial y}, -\frac{\partial A}{\partial x}, 0 \right) \tag{6}$$

The momentarily torque is most certainly of interest in electrical machine cases. The magnetic stress tensor have to be found first to find the torque [3, p197]. For the 2D case the tensor is:

$$\mathbf{T} = \begin{bmatrix} \left(B_x^2 - \frac{1}{2}|B|^2 \right) & B_x B_y \\ B_y B_x & \left(B_y^2 - \frac{1}{2}|B|^2 \right) \end{bmatrix} \tag{7}$$

The force can be found from equation (8) when the stress tensor is known. (Equation (8) is given for the 3D case.)

$$\mathbf{f} = \frac{1}{\mu} \oint_S \mathbf{T} \cdot \mathbf{dS} \tag{8}$$

In equation (8) S is the surface of the entire air gap. Assuming the length in z direction is 1, the surface integral simplifies to a line integral where dS is the outward normal.

When the force is known the torque can be obtained from [4, p6]:

$$\tau = F * r \tag{9}$$

In equation (9) r is the distance between the center of rotation and the point where the force is applied. Equation (9) is valid when the force is perpendicular to r.

The torque in electrical machine can be determined by knowing the magnet field density in the air gap, and the geometry. By adding a virtual circle in the air gap, one can integrate along this line and thus approximate the torque quite well.

B. Finite element method

The finite element method (FEM) is a mathematical method for solving partial differential equations approximately. The basic idea is that a larger problem can be divided into many smaller problems, and the solution for each subproblem can be approximated by i.e. linearization. A complex region is discretized into simple geometric shapes which are called finite elements. Properties, f.i. material properties, are considered over these elements, and given as functions of the unknown in the corners. The approximate solution is obtained by solving these equations for all the corner points, based on the given boundary conditions [5, p1]. It is common to use triangles as these elements, as shown in figure 1. This dividing into triangles and the resulting "net" is often referred to as meshing and mesh respectively.

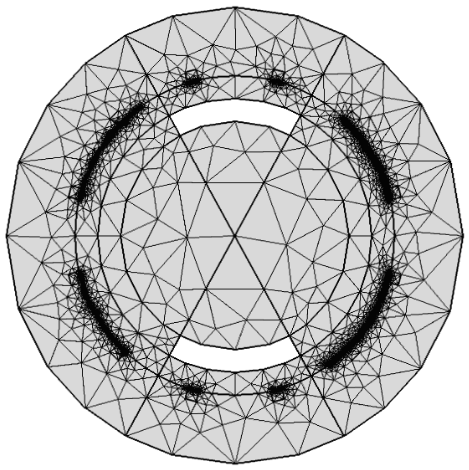


Fig. 1. Example of a mesh

FEM simulations are often used to calculate the magnetic or the electric field in geometries in power electronics. I will not go further in detail about the finite element method here, since the understanding of the underlying mechanism of this is not crucial to understand the results presented. Readers interested are referred to literature on the subject, for instance "Introduction to Finite Elements in Engineering" by Chandrupatla and Belegundu [5].

C. Particle swarm optimization

The particle swarm optimization (PSO) algorithm was first described by J. Kennedy and R. Eberhart back in 1995 [6], but using it in electromagnetics was first done in 2004 by J. Robinson and Y. Rahmat-Samii [7]. They describe the PSO the following way:

"Imagine a swarm of bees in a field. Their goal is to find in the field the location with the highest density of flowers. Without any knowledge of the field a priori, the bees begin in random locations with random velocities looking for flowers.

Each bee can remember the locations that it found the most flowers, and somehow knows the locations where the other bees found an abundance of flowers. Torn between returning to the location where it had personally found the most flowers, or exploring the location reported by others to have the most flowers, the ambivalent bee accelerates in both directions altering its trajectory to fly somewhere between the two points depending on whether nostalgia or social influence dominates its decision.” [7]

All bees will approach the global best field with the highest density of flowers in the end.

The PSO algorithm for optimizing engineering problems works the same way. A solution space for the problem has to be defined. In this study this is done by setting upper and lower limits for all parameters. An expression for, or a value, to optimize with object to is also required. This function or expression denotes the *fitness*. The population of particles (corresponding to the bees) have to be initialized before starting the optimization.

In PSO all particles have a position and a velocity. The velocity is some function of current velocity, personal best and global best. (see figure 2).

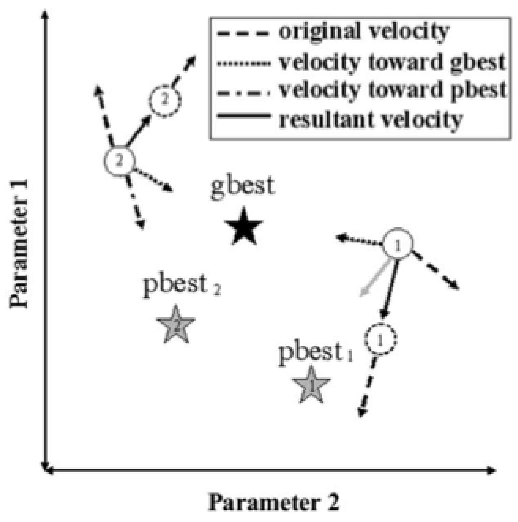


Fig. 2. Visualization of the PSO algorithm [7, fig. 3]

The new velocity can be calculated using equation (10) [7]. c_1=c_2=2 have been proven to work for many applications [8].

$$v_n = w{*}v_n + c_1{*}rand(){*}(p_{best,n} - x_n) + c_2{*}rand(){*}(g_{best,n} - x_n) \quad (10)$$

The position can then be calculated using [7]:

$$x_n = x_n + \Delta t * v_n \quad (11)$$

The last consideration is how to deal with particles that cross the given boundaries. Robinson and Rahmat-Samii [7] describe three ways of dealing with boundary conditions; absorbing walls, reflecting walls and invisible walls. When using absorbing walls the velocity to a particle crossing the border is set to zero. It will then be accelerated back into the solution space in the next iteration. Reflecting walls, does as the description says, just send the particle back with the same velocity into the solution space. Using invisible walls, the velocity is unaffected, but the position is not updated for particles outside the borders. This way these particles inevitable will be accelerated back into the solution space.

An overview of all key terms used when describing PSO is shown in table II.

TABLE II
KEY TERMS IN DESCRIPTION OF PSO (BASED ON A SIMILAR TABLE FOUND IN [7]

Particle	One single individual in the swarm
Position	the particle’s N-dimensional coordinates which represents a solution to the problem.
Population/swarm	The entire collection of particles.
Fitness	A single number representing the goodness of a given solution.
pbest	The location in parameter space of the best fitness returned for a specific agent.
gbest	The location in parameter space of the best fitness returned for the entire swarm.

D. Parallel processing

“Parallel processing is information processing that uses more than one computer processor simultaneously to perform work on a problem. This should not be confused with multitasking, in which many tasks are performed on a single processor by continuously switching between them, a common practice on serial machines.” [9].

Parallel processing is thus a way to speed up the executions of code. Parallel computing have been used in a wide range of engineering applications e.g. optimization problems with great success [10, p4-5]. To enable the process to be run in parallel, the code must consist of portions which can be performed concurrently [10, p85]. A *for* loop will often qualify, as one knows how many times the code shall run. Another important requirement for parallelizing a loop, is that the results have to be independent of the order in which the iterations are done [11, p37]. This implies that in many applications the code is not suited for parallel processing.

There are many types of software that support and utilize parallel processing, among them Matlab. There is a function in the “parallel computing toolbox” in Matlab, *parfor*, which runs the simulation on as many cores as possible/specified [12]. There was a limit on 12 parallels until Matlab version 2013b, but this limit was removed in the newest version 2014a [13]. From an end user perspective, using *parfor* in Matlab is almost identical to the normal *for* loop in the Matlab code.

The expected speed up using parallel processing is not conclusive. It might seem reasonable to assume that if using twice the hardware resources, the execution time will be halved. This is however rarely the case [10, p195]. This can be due to overhead communication in the computer(s), or that some of the parallels take more time than others. The result is that the rest stand idle for some time waiting. A third cause is that one have to renounce to a less effective algorithm in order to make parallelization possible. The theoretical speedup can never exceed the number of parallels [10, p200]. This means that if a given algorithm takes T_s seconds to execute, when parallelizing over n_s cores, the greatest speedup possible is T_s/n_s. The more time the execution of one loop takes relative to the total execution time, the greater the speedup.

10 iterations will, in theory, go equally fast using 12 and 25 cores since (in both cases) every core just have to deal with one iteration. However, with 13 iterations, the 25 parallel computer can (theoretically) be twice as fast. The number of iterations each computer have to handle is referred to as "steps per worker".

Another way of parallelize Matlab code is using the software techila. From a end user perspective techila works the same way as *parfor*, but instead of dividing the jobs onto cores on the current computer, it is divided and distributed to several computers [14].

IV. Description of the simulation experiments

The aim in this work, as stated earlier, was to apply FEM-simulations in optimization algorithms, and then try to decrease runtime using parallel processing. Two models were first set up for the FEM simulations. Both had a changeable geometry. These models were further used for optimization of the geometry that gave the highest magnetic field strength/-torque (respectively for the two cases). The next step was to rewrite the Matlab function to run in parallel using *parfor* and *cloudfor*.

A. Description of test cases

Two different cases were used. The first case was a simple one, to prove the concept. The second was a more complex case, a model of a small electric machine. In the first case the goal was to optimize the magnetic flux density in a certain point in the figure, and the second case the torque was optimized. Both were simulated statically, so there was no function of time or movement in the models.

Case 1 consisted of one magnet and stator with a stator tooth, and can be seen in figure 3. The stator and tooth (indicated in red) were iron with $\mu_r = 4000$. The magnet was removed from the geometry (hence white) and boundary conditions (magnetic flux) were given. The rest in the figure (colored blue) was air with $\mu_r = 1$. All permeability values μ_r was obtained from the Comsol material library. Magnetic insulation around the outer border and magnetic field at $\frac{6.67}{\mu} A/m$ at the two vertical edges of the magnet, was the boundary conditions. All measurements were in centimeters, but this could easily be scaled.

The magnetic flux density at the corner of the tooth (as indicated in figure 3) was the object of optimization, and the goal was to get this as high as possible within the given limits.

Case 2 consisted of a stator and rotor (both in iron), 2 magnets and 12 slots as shown in figure 4. Each slot had a current density of ± 2 and $\pm 4A/mm^2$. The slots had an area of $0.5 * 0.5mm^2$. There were $4A/mm^2$ in the two uppermost slots. The two pairs on each side of the uppermost slots had $2A/mm^2$. The rest were negatively mirrored around the x axis. There were hence three phases, **r**, **s** and **t**, with 120 degrees between each phase. There were $+2A/mm^2$ in phases **r** and **s** and $-4A/mm^2$ in phase **t**. There were copper with $\mu_r = 1$ in the slots. All outer borders had magnetic insulation as in case 1. The magnets were given a magnetic potential equal $1.2 * x$. This meant that the magnetic potential only was dependent on x. Hence the growth in **A**, from left to right, was not perfectly steady, but due to some practical issues (will be discussed later) this had to be sufficient. The result was that the magnet field density was not uniform over the magnet surface, but was steady enough for this purpose.

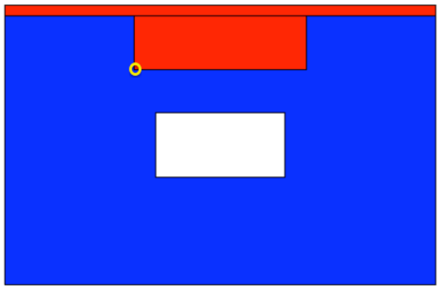


Fig. 3. Test case 1. Blue is air and red is iron, the magnet is shown as white. The yellow circle indicate where the measurements of the B field for the optimization was done.

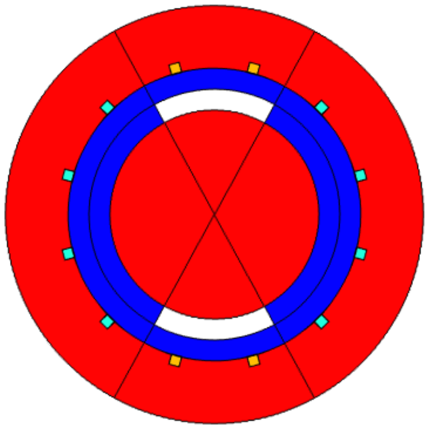


Fig. 4. Test case 2. The red is iron, the blue is magnets and air (both $\mu_r = 1$), and the white is the magnets. The dots are copper with different current density. The orange ones have $\pm 4A/mm^2$ and the green ones have $\pm 2A/mm^2$. The current are positive in the 6 uppermost slots, and negative in hte other 6.

The second case the aim was to get as high torque as possible by changing the magnets' height and width.

B. Setting up the models for FEM simulation

The FEM simulation part was done both by using Comsol and Matlab's PDE-toolbox. First all the geometries had to be drawn, either using GUI or by Matlab script. It was possible to choose a magneto static mode both in Comsol and in the PDE-toolbox which made some corresponding assumptions. (The equation to be solved is the one shown in equation (5) as an example). Magnetic permeability and current density (perpendicular to the xy plane) were given for all subdomains. "Magnetic insulation" were used as border conditions for the outer border. The magnet was also modeled using border conditions.

In order to be able use the models in the optimization they had to be provided as Matlab command line scripts. This was easily done by saving the file as Matlab m-file in Comsol. The only requirement was to couple Comsol and Matlab. This coupling was automatically established by using "COMSOL with MATLAB" from the start menu. There was no export function in the PDE-toolbox, so the Matlab m-function for the models had to be written manually. Appendix B contains a thorough guide for writing the Matlab code for setting up the FEM-simulation in the PDE-toolbox using command line functions.

C. Initial FEM-simulation

An initial simulation was run, both in Comsol and the PDE-toolbox, in order to make sure that the simulations gave the same results and that the models behaved as intended. For the PDE-toolbox the initial simulation was done both using the GUI and command line functions from Matlab.

D. Optimization

The object of optimization in case 1 was the magnetic flux density in the lower, left corner of the stator tooth. The torque was should be optimized for case 2. A reference should be determined for both cases using a parametrical sweep in Comsol. The solution from the optimizations was later compared with this reference solution.

The lower, left corner of the magnet was used as the input for the magnet's placement in case 1. The boundaries of the magnet's placement was set to 0.01 from the walls and bottom of the tooth respectively. This gave a solution space with x between -0.99 and 0.39 and y between -0.79 and -0.11.

In case 2 the heigh and width of the magnets were used as input parameters. The boundaries of the magnets were set so that the height could be between 0.05 and 1.949, width between 0.3 and 15.41. Note that by rounding off to two digits the resulting solution can cross the given limit at 1.949 ($1.949 \approx 1.95$).

PSO was used for the optimization part in this study. PSO was chosen for it's simplicity and and sufficient for this purpose. It was also possible to take advantage of my fellow student Erlend Engevik's work in his specialization project [15] from fall 2013.

The following parameters were used for the PSO:

- c_1=c_2=2
- $\Delta t/dt$=0.2 (if not other specified)

Absorbing walls were used to deal with boundary conditions. The initial population was distributed randomly throughout the solution space.

The Matlab implementation of PSO was based on Erlend Engevik's implementation. The initial population was produced. Then the FEM-simulation was executed to determine the fitness of the current position. This value was returned to Matlab, which compared the results with the current global- and personal bests for each particle, and if necessary updated the entries. The magnet(s) position(s) were updated based on the global- and personal bests and the velocity to each particle. These calculate- and update steps were repeated until the desired number of iterations was reached. The global best magnet position was printed and the best configuration was plotted with the resulting magnetic field and magnet(s) position(s). The complete Matlab code can be found in appendix E.

PSO was tested for different number of iterations and particles. It was first tested with different combinations of 10, 20 and 50 particles/iterations. All simulations were run 10 times and the mean values were used. The deviation were given as a $\pm\%$ value which represented the difference between the mean value and the result with the highest deviation from the mean.

All tests were run using PDE-toolbox. Two sets of runs were in addition executed using Comsol.

Some tests were executed with different tuning of dt, and had a significant effect on the accuracy of the result. These simulations were initially not in the scope of this study, but the results were too interesting to omit.

E. Parallelizing

The optimal magnet placement was found as described above, but another goal for this study was to speed up these optimizations. Two ways of parallelizing were tested, namely the earlier mentioned *parfor* and *cloudfor*. Both these were tested to a good extent with combinations of 10 and 50 (48 for case 2) iterations/particles, and a few runs with even higher numbers of iterations/particles. Also here all simulations were run 10 times.

The only thing which had to be changed in the code for both *parfor* and *cloudfor* was to use actually *parfor* / *cloudfor* instead of a normal Matlab *for* loop. Both needed some additional code to run efficient, but even those extra configurations meant about 10 lines and the optimization was ready to parallel process. It was recommended for *parfor* to start a matlabpool (initialize workers) in advance of running the code. The standard was to use 12 workers or all workers available. If the computer had more than 12 cores that had to be configured. Techila also needed some additional parameters to run efficient. These can be found in appendix C.

Using *parfor*, you can chose how many of the available cores you want to use. The simulations were run on three different computers and using different number of parallels. The aim was to be able to identify some general aspects with parallelizing this type of code. The following computers were used:

- **mac**: My personal computer. Could run maximum 4 parallels on two 2.9GhZ processors.
- **eelk1734**: A terminal server which most of the master students at the institute have access to. Could theoretically run 48 parallels on four 2.2GhZ processors, but had not Matlab 2014a installed.
- **eelk1728**: A terminal server similar to eelk1734. Only a handful of students have access. Had Matlab 2014a so it could run 48 parallels.
- **techila**: Used Computer lab PCs with possibility to run 4 parallels on each. Else wise the specifications were unknown.

The techila solution is new to NTNU so this study was among the first run tests of this kind. There were therefore obviously some startup problems, but once they were solved, implementing and using techila was straight forward. Some tips are included together with the techila parameters in appendix C. Techila was always set up to use as many cores as particles or if exceeded 32 (number of cores I had access to) a multiple of the number of particles when running the simulations. Due to the nature of parallelizing a *for*-loop, the expectations were that the speedup might be proportional to the number of cores.

The outline of the complete data- and work flow are sketched in the figure found in appendix A.

V. Results

A. Initial FEM-simulations

The initial simulations were done in the PDE toolbox, both in the GUI and in the Matlab command line. The command line functions should in theory not be that hard to use, but it took many tries and long time to make them work. The contour plots of the two solutions for case 1 are given in figure 5 and 6. The corresponding plots for case 2 are given later in figure 8 and 9.

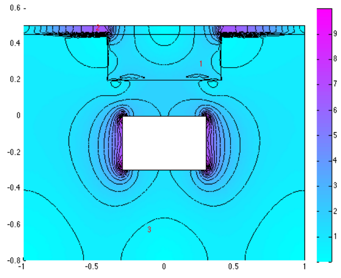


Fig. 5. Counterplot for case 1. Initial simulation of the magnetic field density (B) from the Matlab script using command line functions.

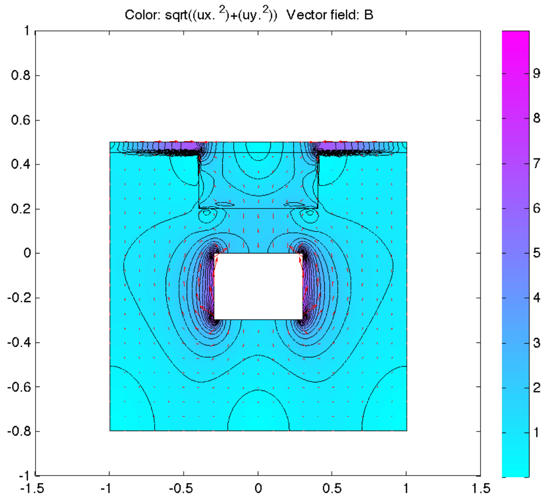


Fig. 6. Counterplot for case 1. Initial simulation of the magnetic field density (B) from the PDE toolbox GUI

When checking the value of the magnet field strength (B-field) in the desired lower, left corner of the machine tooth in case 1, they were 1.388T and 1.747T for the script and the PDE-toolbox respectively.

The same was done in Comsol, which gave the results shown in figure 7(case 1) and in figure 10(case 2). For case 1 the value in the lower, left corner of the machine tooth was 1.518 T in Comsol.

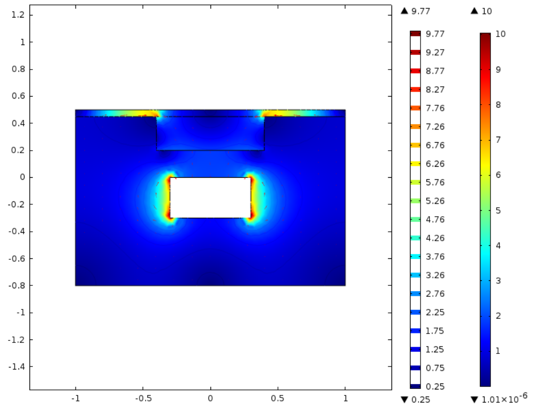


Fig. 7. Counterplot for case 1. Initial simulation of the magnetic field density (B) from Comsol.

There were some difficulties setting the border conditions for the magnet equally in Comsol and the PDE-toolbox, both for case 1 and 2, but the plots and behavior was the same even though the magnitudes measured was different.

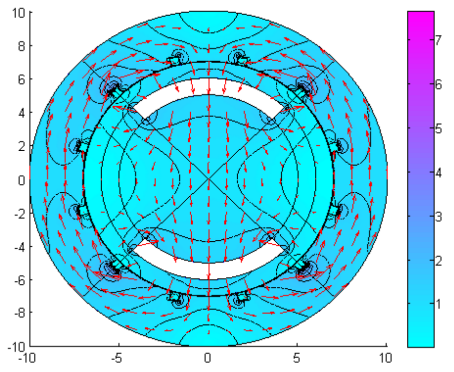


Fig. 8. Counterplot for case 2. Initial simulation of the magnetic field density (B) from the Matlab script using command line functions.

The value of the torque for case 2 from the command line function. The torque from the command line function was $775N * m$ when it was checked. It was not possible to get the torque from the PDE-toolbox GUI. In Comsol the torque was $22N * m$.

There were big differences the B field/torque in the these initial simulations, but the plots were very similar. The torque in case 2 was about 40 times higher using the PDE-toolbox than using Comsol. Because the aim in this project was to optimize B-field (case 1)/torque (case 2) as a function of magnet(s) placement, these differences were accepted since all

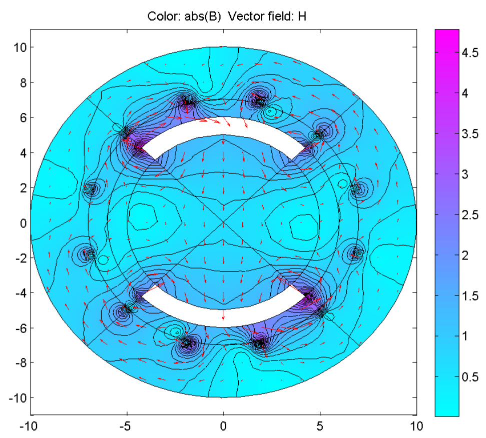


Fig. 9. Counterplot for case 2. Initial simulation of the magnetic field density (B) from the PDE toolbox GUI

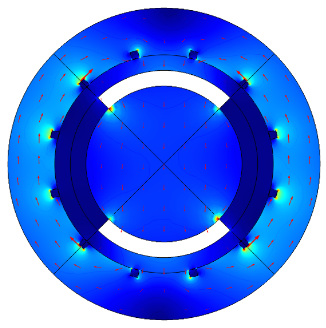


Fig. 10. Counterplot for case 2. Initial simulation of the magnetic field density (B) from Comsol

models behaved the same when tested with different values. Behaving the same, the optimal magnet(s) placement would also be the same.

B. *Optimization case 1*

Several runs were performed to test the PSO algorithm, using different numbers of particles and maximum iterations. These would later be evaluated to see whether or not there was a better set of parameters.

1) Reference solution: A parameter sweep with finer mesh just around the optimum was run to set the reference solution for the case. The result are presented in figure 11. From visual inspection of figure 11, it is clear that the maxima was in the upper, left corner of the figure. The readout showed that the optimal placement of the magnet was with $xmagn = -0.99$ and $ymagn = -0.11$ (variables representing lower, left corner of the magnet). This implies that the maximum is found at the boundaries ($x = -0.99$, $y = -0.11$). The B-field with this magnet placement was $40.13T$. Figure 12 shows what the optimum solution was.

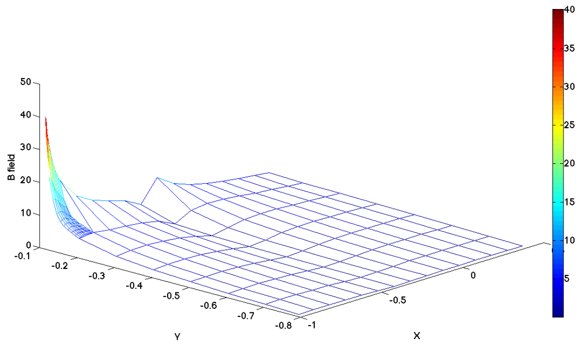


Fig. 11. B field as function of x and y, based on parameter sweep in Comsol

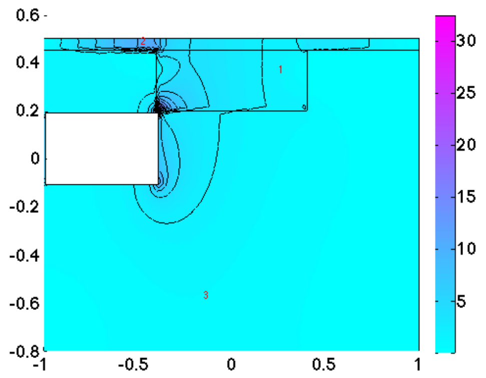


Fig. 12. The optimal solution for case 1

2) PSO solution: The results obtained with PSO and the PDE toolbox are presented in table III. All simulations were run 10 times and the deviation was the difference between the mean and the value with the *largest* deviation given as percent of the mean. There was for some reason a spike at $xmagn = -0.20$ in the script. Neither Comsol or tests with the script indicated that there should be a spike in this position. I did not find the reason, but all evaluations for $xmagn = -0.20$ were escaped from the PSO evaluation.

TABLE III
THE RESULTS FROM OPTIMIZING CASE 1, MEAN VALUES OF 10 RUNS.

Iter. / Part.	10	20	50
10	$x = -0.51 \pm 123\%$ $y = -0.12 \pm 52\%$ $B = 12.20 \pm 112\%$	$-0.66 \pm 50\%$ $-0.11 \pm 0\%$ $17.04 \pm 50\%$	$-0.79 \pm 58.1\%$ $-0.11 \pm 0\%$ $20.48 \pm 58\%$
20	$x = -0.86 \pm 61\%$ $y = -0.11 \pm 1\%$ $B = 22.66 \pm 63\%$	$-0.66 \pm 50\%$ $-0.11 \pm 1\%$ $17.14 \pm 52\%$	
50	$x = -0.92 \pm 64\%$ $y = -0.11 \pm 1\%$ $B = 24.44 \pm 65\%$		$-0.99 \pm 0.1\%$ $-0.11 \pm 1\%$ $26.17 \pm 5\%$

As seen in table III, it is clear that the accuracy increased with more iterations and/or particles. If looking at the numbers in the upper, right cell and the ones in the lower, left cell, it was clear that the solution with more particles was much closer to the reference solution. The solution using 50 particles and

10 iterations does not differ much from the solution obtained with 50 iterations, but the deviation does. The deviation was the same at roughly 60% in both the 50/10 (iterations/particles) and 10/50 runs. A deviation of 60% meant that at least one of the runs gave a position roughly 0.05 higher/lower than the mean. With a total solution space of $1.98 * 0.68$ the deviation of 0.05 was significant. From this it may seem that more particles compared to iterations gave better solutions, and higher number of particles gave smaller deviations.

C. Solutions using Comsol

The same simulations were then run using Comsol as the FEM-software (on the mac). The results are presented in table IV

TABLE IV
THE RESULTS FROM OPTIMIZING CASE 1 WITH COMSOL, MEAN VALUES OF 10 RUNS.

Iterations / Particles	10	20
10	$x = -0.66 \pm 112\%$ $y = -0.12 \pm 17\%$ $B = 16.85 \pm 72\%$ $t = 388s \pm 2\%$	
20		$x = -0.80 \pm 56\%$ $y = -0.11 \pm 0\%$ $B = -23.46 \pm 57\%$ $t = 1552s \pm 5\%$

Compared to the results presented above, the results using Comsol were similar to the ones using the PDE-toolbox. In these runs, it seemed like the results were a bit closer to the correct solution. However if comparing the runtime with the PDE-toolbox runtimes presented later in section V-F, Comsol takes longer time to execute.

D. Influence of different tuning

For the set of 20 iterations and 10 particles another result was discovered: about half the runs showed the solution to be at the local maxima with $xmagn = -0.33$ instead of the global maxima at $xmagn = -0.99$. This may be because the numbers of iterations/particles were to small and/or because of badly tuning. In order to test this the same simulation were run 3 times with different tuning, $dt = 0.1$, 0.2 and 0.5. All the results can be found in table V. The results were quite different in the three sets. $dt = 0.5$ seemed to be the best in this test. But using this value would in turn give less differences in the optimization results. Which, in this case actually was wanted. It would be easier to evaluate the impact of number of particles/iterations when having larger deviations on the optimization results. All the rest of the simulations were therefore done using $dt = 0.2$.

E. Optimization case 2

Case 2 was then optimized and the corresponding results as in case 1 were found (except that torque and not B-field was the object of optimization). The absolute value of the torque was used as the *fitness* function in PSO. It had no impact whether the motor turned clockwise or anti clockwise.

TABLE V
RESULTS FOR XMAGN WITH DIFFERENT TUNING. 20 ITERATIONS, 10 PARTICLES

dt=0.1	dt=0.2	dt=0.5
-0.33	-0.99	-0.99
-0.33	-0.33	-0.99
-0.33	-0.99	-0.99
-0.77	-0.99	-0.99
-0.99	-0.33	-0.99
-0.99	-0.33	-0.99
-0.33	-0.33	-0.99
-0.99	-0.33	-0.99
-0.99	-0.99	-0.99
-0.34	-0.99	-0.99
mean		
-0.64	-0.66	-0.99

1) *Reference solution:* The optimum was found using comsol in this case as well. This time the optimum was found using a parameter sweep of combinations of width and hight of the magnets.

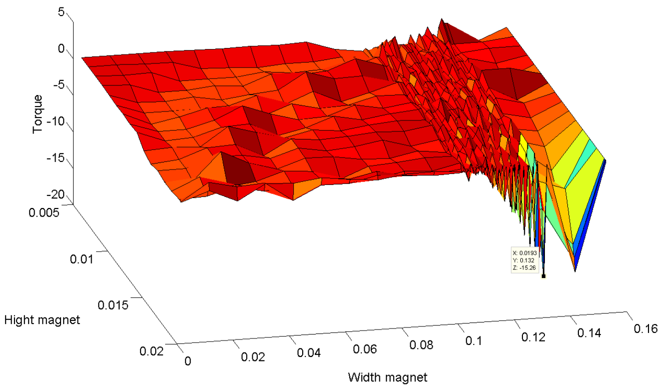


Fig. 13. Torque as function of width and hight of magnet, based on parameter sweep in Comsol. Mark that the model in Comsol is scaled by a factor of 100.

The maximum torque value was $-15.26N*m$ and appeared with height of magnets at 0.0195 (equal 1.95 in the pde-toolbox model) and magnet width at 0.1230 (equal 12.30 in the pde toolbox model). The optimal torque seemed thus to appear outside the set borders, but that was because the border was given with 3 decimals and then the value rounded up.

Figure 14 shows the optimum geometry giving the highest torque.

2) *PSO solution:* The same set of simulations were run for case 2. 48 iterations/particles were used instead of 50, but otherwise the simulations were similar to the ones performed for case 1.

As for case 1, there was rather large deviations ($\sim 60\%$) in the 10/10 run and the solution was far from the optimum. The deviation decreased with increasing number of iterations/particles as for case 1. Unlike in case 1, both 50/10 (iterations/particles) and 10/50 gave approximately the same solution and had the same deviation. So there was no difference on running more iterations or particles in this case. It was actually the 48/10 run that was closest to the reference solution. The deviation between the 48/48 solution and the

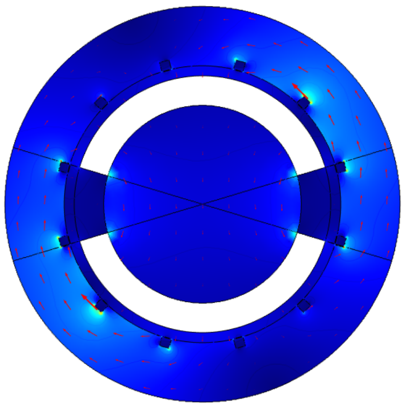


Fig. 14. Solution for highest torque exported from Comsol

TABLE VI
THE RESULTS FROM OPTIMIZING CASE 2, MEAN VALUES OF 10 RUNS. W=WIDTH MAGNET AND H=HIGHT MAGNET.

Iter. / Part.	10	20	48
10	$w = 11.31 \pm 66\%$ $h = 1.87 \pm 32\%$ $\tau = 3370 \pm 60\%$	$12.3 \pm 39\%$ $1.94 \pm 2\%$ $4367 \pm 18\%$	$12.25 \pm 38\%$ $1.94 \pm 2\%$ $4993 \pm 29\%$
20	$w = 12.40 \pm 39\%$ $h = 1.94 \pm 2\%$ $\tau = 3965 \pm 25\%$	$12.81 \pm 2\%$ $1.95 \pm 0.5\%$ $4912 \pm 15\%$	
48	$w = 11.27 \pm 33\%$ $h = 1.94 \pm 3\%$ $\tau = 4197 \pm 27\%$		$12.75 \pm 0.0\%$ $1.95 \pm 0.3\%$ $5641 \pm 7\%$

reference solution was not significant, but it was larger than for the 48/10 run.

F. Parallelizing the optimizations case 1

Four combinations of iterations/particles were ran in the main optimization tests: 10/10, 10/50, 50/10 and 50/50 (iterations/particles). Table VII shows the measured runtime for all these simulations. As for the FEM-simulations all simulations were run 10 times and the mean was used. The deviation still represented the run with the largest deviation from the mean. Some of the simulations taking the longest time were only run a one or a few times for practical reasons. 77% of the simulations were run 10 times, 9% were run 2 times or more so the majority of the results had a good statistical confidence.

The results will be visualized and plotted in the final subsection of this chapter, after presenting all results for both cases.

TABLE VII
MEASURED RUNTIME FOR CASE 1 WITHOUT/WITH PARALLELIZING. MEAN VALUES OF 10 RUNS. NUMBERS IN BRACKETS BEHIND THE COMPUTER NAME IS NUMBER OF WORKERS USED.

	Machine \ iter/par	10/10	50/10	10/50	50/50
for	mac	205s ± 13.2%	1291s ± 6%	1110s ± 5%	8244s[1]
	eelk1734	3090s ± 14%	7607s ± 34%	7312s ± 40%	26134s ± 27%
	eelk1728	458s ± 16%	2334s ± 7%	1911s ± 5%	14491s[1]
parfor	mac (2)	153s ± 22%	986s ± 5%	774s ± 9%	4906s[1]
	eelk1734 (12)	57s ± 9%	286s ± 13%	242s ± 17%	1131s ± 6%
	eelk1728 (12)	49s ± 8%	263s ± 4%	201s ± 5%	1127s ± 6%
	eelk1728 (25)	49s ± 17%	256s ± 18%	146s ± 13%	667s ± 17%
cloudfor	techila on lab PCs	338s ± 10%	814 ± 3%	390 ± 8%	864s ± 10%

It is first worth noting that there were huge differences between computers without parallelizing (the uppermost box in table VII). The runtimes for the runs of 50/10 and 10/50 (iterations/particles) are almost the same for all the simulations. The 50/10 run took some minutes longer time than the other (except for techila). The runtime for techila could so far seem to be a function of the number of particles only. When comparing the runtimes in the uppermost box in table VII (using *for* loop) with the results in the two next boxes it clearly showed that there was much to gain by using parallel processing. Note: the two computers eelk1734 and eelk1728 used about the same time when using *parfor*, but they did not with the normal *for*.

G. Parallelizing the optimizations case 2

The same optimizations were run for case 2. In addition some simulations with a higher number of iterations/particles were run.

All simulations for case 1 were run on three different computers. Unfortunately the eelk1734 server was taken down for reinstallation by the time these simulations were run. Two computers were found to be sufficient for case 2.

TABLE VIII
MEASURED RUNTIME FOR CASE 2 WITHOUT/WITH PARALLELIZING. MEAN VALUES OF 10 RUNS.

	Machine \ iter/par	10/10	48/10	10/48	48/48
for	mac	2258 ± 9%	11102 ± 18%	10337 ± 0.2%[2]	51007s[3]
	eelk1728	3506 ± 9%	17659 ± 2%[4]	16113s[3]	72527s[3]
parfor	mac (2)	1639s ± 10%	8801 ± 11±	8082s ± 6%	37110s[3]
	eelk1728 (12)	458s ± 10%	2301s ± 10%[5]	1799s ± 7%[6]	14795[3]
	eelk1728 (25)	467 ± 10%	2356 ± 13%	1571s ± 7%	7562s ± 18.7%
cloudfor	techila on lab PCs	495s ± 8%	1642 ± 19%	682s ± 6%	2101 ± 2%

As for case 1 it was clear that higher number of iterations/-particles took more time. Parallel processing showed the same tendency of shorter execution time as in case 1. The drop in execution time for the 10/48 run versus the 48/10 run also appeared here.

Two sets of simulations, using 48 parallels and techila, were run in addition to the above. These results are found in table IX. Figure 15 shows the results presented in table IX.

TABLE IX
SOME RUNS WITH HIGHER NUMBER OF ITERATIONS/PARTICLES, ONLY ONE RUN OF EACH. CASE 2.

Machine \ iter/part	1000/5	500/5	5/5	5/500	5/1000
eelk1728, 48 workers	41748s	16306s	196s	7496s	15192s
techila on lab PCs	31831s	16697s	427s	1857s	3348s

Techila performed surprisingly best for most sets in these simulations, except for the ones with few iterations and relatively low number of particles were they performed equally.

[1]Simulation run once
[2]Simulation run 2 times
[3]Simulation run once
[4]Simulation run 5 times
[5]Simulation run 6 times
[6]Simulation run 7 times

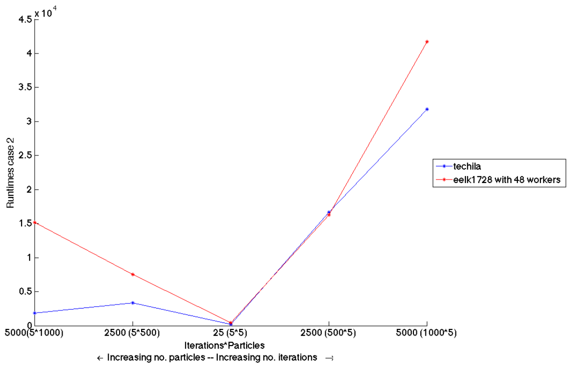


Fig. 15. Runtime for some simulations with techila and eelk1728 using 48 parallels.

This meant that the difference in runtime between techila and a many core computer was dependent on the actual execution time of each iteration.

H. Results from the parallelizing part combined

In this section the results from the parallel processing parts are combined and plotted for visualization of trends and comparison.

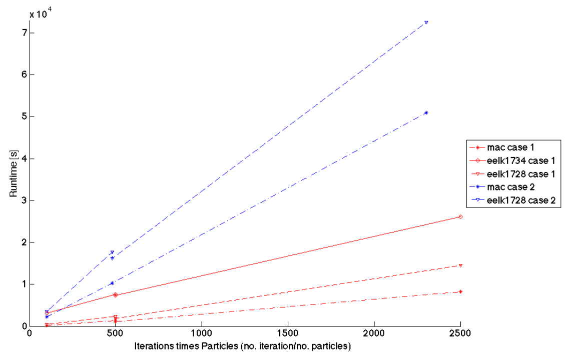


Fig. 16. Runtime as function of iterations and particles using normal for loop (without parallelization). The red ones are case 1 on different computers, and the blue ones are case 2 on the same computers.

The product of iterations and particles is in practice the number of FEM evaluations done during the optimization. Figure 16 shows a linear relationship between number of FEM-evaluations and the runtime. One can see the same tendency in figure 17, but it is not as clear as without parallelization.

There is a drop between the second point and the third point marked on the graphs in both figure 16 and 17. The point are given in the same order as in the tables VIII and VII for all graphs, which means that the 2nd point is 50(48)/10 (iterations/particles) and the 3rd is 10/50(48). The set of more iterations and fewer particles are hence slower than the simulation with few iterations and many particles, even though the total number of FEM executions were the same.

The two figures 18 and 19 shows the speedup as a function of number of cores for the two sets with 50/48 particles and both 10 and 50 iterations.

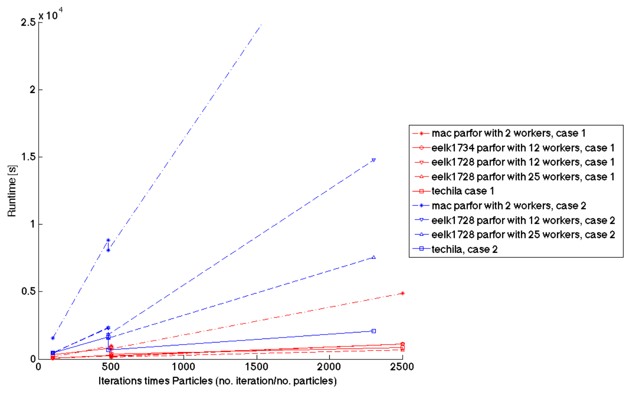


Fig. 17. Runtime as function of iterations times particles, this time for the parallelized runs. The red ones are case 1 on different computers, and the blue ones are case 2 on the same computers.

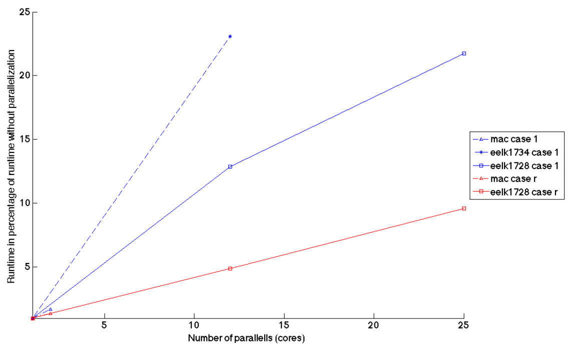


Fig. 18. Speedup for the 50/50 case as function of cores. The two graphs for mac (case 1 and 2) were identical and hence only one of them is seen in the plot.

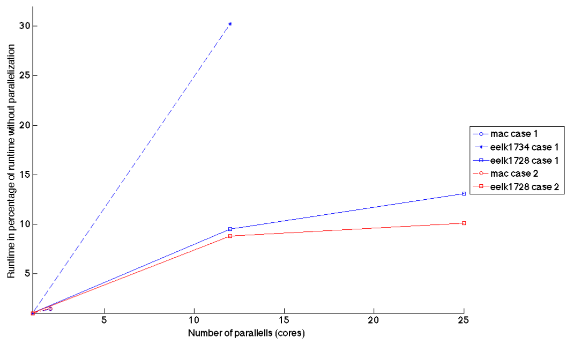


Fig. 19. Speedup for the 10 iteration and 50 particles case as function of cores. The two graphs for mac were also here (case 1 and 2) identical and hence only one of them is seen in the plot.

VI. Discussion

A. The FEM simulations

Both results from the the initial FEM simulations and from the optimizations were as expected, and matched the physical understanding of the cases. The magnitude of the torque in the PDE-toolbox simulation was about 400 times higher than the Comsol simulation in case 2. Which was suspected to be due to some extension problem (i.e. cm vs. m). The models behaved the same way, and from the optimization results the optimal height and width of magnets were the same (the PSO solution matched the reference solution). The difference in the B field in case 1 was in comparison small. If the aim had been to find and evaluate the B-field/torque, this would be a great uncertainty. But since only the placement of the magnet(s) was of interest in this study thus the Comsol solution could be used as reference for the magnet(s) placement anyway. It was not put much effort into finding the reason for this difference since the models behaved the same way.

There were problems when using trigonometric functions in the boundary conditions of the magnets in case 2. The function for the magnetic field at the boundary therefore ended up of just being a function of x instead of a function of both x and y. As for the deviation in the magnitudes of the B-field and torque, this was found to be bearable. It might have affected the optimal solution, but were the same for all simulations of case 2. This means that they were valid anyway.

B. The difficulty setting up the FEM simulations

The PDE-toolbox GUI was used to set up the model in both case 1 and 2. This was the easiest, but also simplest of the three ways to set up the FEM simulation. Setting up the model from command line was, in contrast, as difficult as the GUI was easy. Especially setting up the boundary conditions from script was challenging. I ascribe this partly to the lack of examples, even not thorough Internet searches turned up empty when looking for similar examples. To my knowledge there is little documentation of setting up PDE-toolbox models from scripts, especially in optimization applications. Setting up the model in Comsol was straight forward, having used Comsol in other studies in advance. If not, this might also be challenging. A note with the workarounds found, and a guide is attached in appendix B.

It may cause problems setting up the model in the PDE-toolbox, if one wants to set up a similar case without removing the magnet from the geometry

C. PSO results

The optimization results matched the expectations quite well. As did the simulation results match the reference solution. For both cases the PSO simulation worked and gave expected results.

Even though it was not the aim of this study to evaluate how to run the PSO for the most accurate solution, some work was done on this as well. The reason was that it was expected to be some relation between iterations/particles and accuracy. It would be interesting to see if that matched the relation between runtime and iterations/particles. Erlend Engevik found in his specialization project that it seemed to be better with more iterations than more particles [15]. The results presented here do not entirely match. In the case 1 it was in fact the opposite. More particles gave a better solution than more iterations. In case 2, the results were almost the same with more iterations and more particles. Most simulations done however had a relative small number of both particles and iteration. It must also be taken into account that there was a local optimum present and that as few as 10-50 particles can be “trapped” there even with a large number of iterations.

As mentioned earlier the result was affected by dt to a great extent. Almost all results were equal to the correct solution when using $dt = 0.5$, whereas about half the runs hit the local maxima when using $dt = 0.1$. $dt = 0.2$ was used partly because it made it easier to compare different sets of runs and the quality of the solution. Setting $dt = 0.5$ made the algorithm quite aggressive, which may have causes the swarm to gather around one location too early (before all the solution space is checked) and hence the tendency to hit a local maxima would increase for some problems.

It is important to emphasize that it was exactly the same implementation that was used in all cases. The quality of the results were therefore the same regardless whether *for*, *parfor* or *cloudfor* were used in the optimization.

D. The difficulty setting up PSO

The PSO implementation used in this study, was based on Erlend Engevik’s earlier implementation of PSO. Setting up the PSO, was therefore not difficult since only a minimal of editing was needed to make it work for this purpose. Most optimization algorithms are much more complex, but many of them are pre-implemented in software. PSO has not to my knowledge been pre-implemented in any software. Matlab is not shipped with the PSO, but has other algorithms such as the *genetic algorithm* which is one of the alternatives to PSO. If using an algorithm already implemented in the software used, the difficulties of setting up the optimization drops dramatically, especially in situations with no access to earlier implementations of the algorithm.

E. Runtime of optimization without parallelization

The optimizations based on FEM simulations without parallelization took a long time. The longest simulations for the two cases used about 7 and 13 hours respectively. There was also big differences between the different computers, f.i. for case 1 the mac used about 2.5 hours for the 50/50 simulation, while eelk1734 used 7 hours. The mac was definitely fastest on all simulations without parallelizing (as one can see from tables VII and VIII). This was because the mac has higher frequency on the CPU than the two other computers.

It was clear that the runtime increased linearly with increasing number of iterations and particles as shown in figure 16 and 17 (presented in the previous section). The product of iterations times particles is equal to the number of FEM simulation calls executed. This linearity implicates that every FEM evaluation took the same time, which seems logical.

It seemed like the set with more iterations were a bit slower than the same set with more particles for all runs. All graphs went down from the point with 48 iterations/10 particles to the ones with the opposite as shown in figure 16 and 17. This fall is suspected to be because more iterations which means that the loop comparing and updating the personal- and global bests have to be executed far more times, which again takes more time.

There are always many sources to errors when measuring runtime: load on the computer, other users, faults in the code, and how well the code is written for mention some. All the computers used have simultaneously run other programs and/or simulations while running the simulations for this study. This will necessarily have an impact the runtime measured. The computer eelk1734 got twice the theoretical speedup in case 1 (will be discussed in the next section). This is because the reference runs without parallel computing was slow. The server could have been heavily loaded at the time and hence the execution took an extraordinary long time. Since the rest of the results (except the ones considering speedup for eelk1734) are consistent, I am certain that the results found here, and the tendencies detected are correct. The sources to errors discussed here are the same for the next section about runtime using *parfor*. Overall the deviations in the measured runtimes were quite small. Only the discussed eelk1734 had deviation above 10%. The large deviations in the runtimes measured for eelk1734 (highest at 40%) supports the suspicion that there was heavy load and that the computer did not have a steady performance.

F. Decreased runtime using parfor

The correlation between total number of FEM evaluation and the runtime discussed above, also seemed to be valid when using *parfor*. It will therefore not be discussed further here.

The speedup using *parfor* cannot theoretically exceed the increase in number of cores. Which means that for the 12 workers computers, the best possible runtime will be one twelfth of the runtime without parallelization. For the computer eelk1734 this speedup was exceeded to a good extent, with 12 parallels the speedup was almost 30 times in the 10/50 case. For eelk1734 there were similar tendencies in the other simulations. This must have been due to heavily loading on the server, as discussed above.

I see in retrospect that the sets of particles/iterations chosen was not the best suited to the number of cores on the available servers. The numbers of iterations/particles should have been a multiple of the number of workers in order to get even better results. The correlation between runtime and number of cores would then have been clearer for the parallelized simulations. It can be deduced from a theoretical point of view that the best speedup is dependent on the number of steps per worker (using PSO this equals the number of particles). This applies both to *parfor* and *cloudfor*.

The speedup was proportionally alike to the number of cores for the 50/50 simulation in case 1. Case 2 had a lower speedup, but this was also linear. The same tendency was found for sets with fewer particles/iterations, but in these cases deviations were significant. So it would be expected that even more cores would decrease the runtime even more.

It was not possible to parallelize using Comsol, as mentioned in section IV-D. The optimization ran just fine using the *for* loop and Comsol, but when trying to use *parfor* errors erupted. I could not find any other explanation than it was some licensing issue. The hope was to find a solution at some stage, but unfortunately it was not.

G. Decreased runtime using cloudfor

Techila is a cloud solution that use idle lab PCs to execute the parallels. Since it have to compile the code and distribute it to vacant computers, it used more time initializing for every *cloudfor* (In PSO this means for every iteration) than *parfor*. Therefore *cloudfor* is even more dependent on the runtime of the FEM simulation. In advance it was expected that techila would not be good unless the number of particles were high, there was only a few iterations and the FEM simulation took quite some time.

For case 1 each FEM simulations took so little time that the difference to the simulations without parallelization was not noticeable until the 50/50 run, techila was then 10 times faster than the mac. Techila was at least 10 times faster compared to eelk1734 for all simulations run for case 1, but as discussed earlier it must have had quite heavy load. For case 2, techila was 5-25 times faster (depending on the number of FEM-simulations) than the simulations without parallel processing. Techila hence performed much better than expected.

When discussing the possibility of using comsol to the FEM-simulations, techila would require comsol installed on all nodes. If comsol is installed on all nodes, there should be no problem with the licensing as the one discussed above, since only one comsol instance will be active per worker. This limits however techila to only use one of the available workers per node (only one of the parallels on each lab PC).

H. Comparison of parfor and cloudfor

The resulting runtime for techila ended up between the runtimes on the mac with 2 workers and the runtimes of the servers with 12 workers in most cases. However, techila seems to have access to faster CPUs since it almost managed to compensate for the time used initializing, compilation and communication even for small numbers of iterations/particles. Techila is also more flexible when concerning number of cores/computers. If for example techila is extended with 4 cores, it will mean one more idle lab PC. For a 48 cores server to extend with 4 cores is both costly and difficult.

The trend identified in this study shown that simulations where the FEM-simulation took as little time as in case 1, techila was mostly slower than *parfor* using 12 or more workers. The longer time every FEM-simulation took the better techila performed, and for case 2 techila performed overall best. For number of parallels exceeding the number of cores on the supercomputer (in this study 48 or above) techila is expected to perform even better. However I had only access to 32 parallels on techila so I did not manage to test it in this project.

For large numbers of iterations and only a few particles techila was expected to be very much slower than the supercomputer (using 25/48 workers), but the runtime was in the same range. In the opposite case techila was superior, for 1000 particles and 5 iterations it was almost 20 times faster even though it had 16 workers less than the supercomputer. This clearly shows that for some cases, where every iteration takes a lot of time and the number of iterations are low, techila would be preferred.

I. The difficulty parallelizing

It was surprisingly easy to go from the non-parallelized *for* loop to *parfor* or *cloudfor*. It took less than 1 hour work to change to *parfor*. Techila was a new solution, even to the IT department, so setting it up and making the simulation run required several trials, and many errors had to be solved. Once getting the configuration right, both in my- and the server's end, all simulations ran without any problems at all. Next time setting up techila will be as easy as *parfor* was this time.

J. The overall difficulty of setting up the test cases

The only real difficulty met when setting up the optimization and parallelizing, was the set up of the model in the PDE-toolbox. Much time was also used to get the model- and PSO parameters right, but that is another matter. Going from normal *for* loop to *parfor*/*cloudfor* was, as discussed above easy. The time used setting up the simulations was in my opinion worth while the reduced runtime.

Comsol was much easier to set up than the PDE-toolbox (from command line). Comsol is also a more advanced FEM-simulation software. Concerning speedup the PDE-toolbox performed best in the few tests done here.

K. Means for further speedup

From the above discussion, more cores will give an even higher speedup (given that the increase in number of cores make less steps per worker). A quite fine mesh was used in these simulations, which slowed down the FEM-simulations. A fine mesh gives a more accurate result, but it may not always be necessary.

No stop criteria was used in the PSO algorithm, which could speed up the optimization. A stop criteria could stop execution when a given accuracy on the solution was obtained, even though the number of iterations not jet was reached. Then the PSO would not use more iterations than necessary to get the required accuracy, and hence finish faster.

There have been little focus on optimize the PSO implementation. There might also be time saved by making the implementation more efficient.

L. Other optimization algorithms

The results found in this study indicates that there is much to gain using parallel processing. As long as the code have for loop(s) which are able to run for itself, parallel processing can be utilized. In the PSO code used here, the loop was split up to be able to parallelize. There is the same opportunity in many other optimization algorithms. Erlend Engevik did some testing utilizing parallelization, both applied to PSO and the genetic algorithms, and found that the genetic algorithm was both better and faster than PSO in his simulation cases [16].

VII. Conclusions

The aim of this thesis was to prove that it is possible to utilize FEM-simulations in optimizations of electric power problems. It was expected that parallel processing would be required to be able to run these simulations within reasonable time. This was proven right. Even a small and simple model has shown to use up to 7 hours without parallelizing. The same simulation was done in about 18 minutes when using terminal servers with 12 cores. The simulation was done in about 10 minutes when using 25 parallels. This is actually a speedup just below the theoretical maximum. This also indicate that more cores can be utilized for larger problems. A more complex model was also tested, and then the speedup was lower from 13 hours down to less than one hour. In both cases the speedup was linearly dependent on the number of parallels. All simulations were run 10 times and the deviation was less than 20% for all computers except for one, but it is suspected that computer was heavily loaded at the time the first simulations were run.

Initially both Comsol and the PDE-toolbox were tested for the FEM-simulations, mostly the PDE-toolbox was used due to difficulties running Comsol in parallel. Comsol was by far the easiest of the two to set up from the Matlab command line, but again, it proved to be a bit slower in the few tests computed. The boundary conditions proved to be very difficult to set up in the PDE-toolbox, but that may be due to lack of examples. Setting up the rest of the model was also difficult. Relative to the speedups already discussed it was worth the effort.

The cloud solution *techila* was also tested and gave similar results as the 48 cores machine. Techila performed much better than expected, and may be the best choice for parallelization in the future, even for a quite small number of iterations. Techila can also utilize idle lab PCs which is a great advantage over using supercomputers. It can easily be scaled up to as many parallels as you wish without necessarily the costs of a supercomputer able to run the same number of parallels.

In this study a new way of optimizing power electric problems have been tested, and I think such optimizations will be seen more to in the future.

VIII. Future work

Other optimization algorithms applied the cases used in this study, is a natural continuation of this work. Also will optimizations using Comsol and parallelized, give a good contribution to the results found. Comsol also has an optimization toolbox which can be tested and evaluated against a Matlab-based optimization. Off course testing whether tendencies found in this study also applies to other cases will be important in order to verify the results found.

Acknowledgment

I want to thank my supervisor professor Robert Nilssen, who has helped me all the way. John Floan and the super-computing community at the Norwegian University of Science and Technology have provided resources and answered all my questions, a huge thank to them. I also want to thank all my family who proofread and reviewed this article. At last I want to thank my fellow student Erlend Engevik to be allowed to use his code and for great cooperation, which have improved the results of both our master theses.

REFERENCES

[1] J. Skar, "Maxwells likninger (norwegian)," In the compendium of "TFE4120 Elektromagnetisme" by "tapir akademiske forlag", 2011.

[2] MathWorks. (2014) Documentation center, pde toolbox, magnetostatics. [Online]. Available: http://www.mathworks.se/help/pde/ug/magnetostatics.html

[3] D. A. Lowther and P. P. Silvester, *Computer-Aided Design in Magnetics*. Springer-Verlag, 1986.

[4] S. J. Chapman, *Electric Machinery Fundamentals*. McGraw-Hill, 2012.

[5] T. R. Chandrupatla and A. D. Belegundu, *Introduction to Finite Elements in Engeneering*. Prentice Hall, 2002.

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, Nov 1995, pp. 1942–1948 vol.4.

[7] J. Robinson and Y. Rahmat-Samii, "Particle swarm optimization in electromagnetics," *Antennas and Propagation, IEEE Transactions on*, vol. 52, no. 2, pp. 397–407, Feb 2004.

[8] R. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, 2001, pp. 81–86 vol. 1.

[9] E. Rasmussen. (2002) Parallel processing. [Online]. Available: Encyclopedia.com:http://www.encyclopedia.com/doc/1G2-3401200250.html

[10] G. K. Ananth Grama, Anshul Gupta and V. Kumar, *Introduction to Parallel Computing*, 2nd ed. Addison-Wesley, 2003.

[11] R. W. Schonkwiler and L. Lefton, *An Introduction to Parallel and Vector Scientific Computing*. Cambridge University Press, 2006.

[12] Mathworks. R2014a documentation, parallel computing toolbox. [Online]. Available: http://www.mathworks.se/help/distcomp/parfor.html

[13] Mathworks, "R2014a documentation, parallel computing toolbox release notes." [Online]. Available: http://www.mathworks.se/help/distcomp/release-notes.html#R2014a

[14] T. Technologies, *End-user guide, Techila with Matlab*, Techila Technologies, March 2013. [Online]. Available: http://www.techila.fi/wp-content/uploads/2013/08/Techila-with-MATLAB.pdf

[15] E. L. Engevik, "Comparative study of stochatic optimization techniniques for electrical machine design," specialization project, Norwegian University of Science and Technology (NTNU), 2013.

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

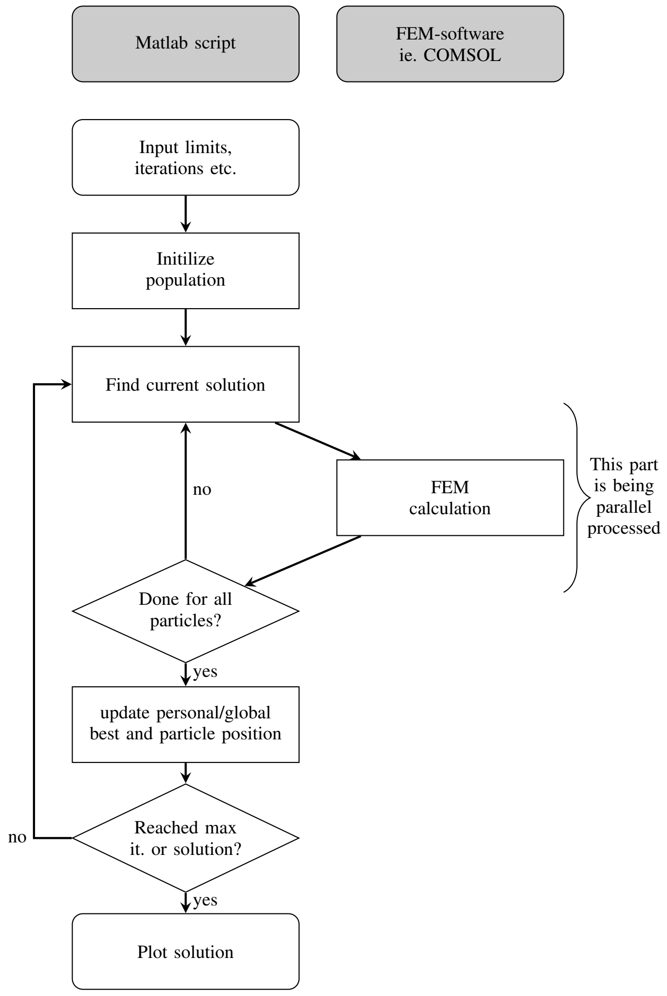
[17] Mathworks, "R2014a documentation, partial differential equation toolbox." [Online]. Available: http://www.mathworks.se/help/pde/index.html

[18] Mathworks. "R2014a documentation, partial differential equation toolbox, decsg." [Online]. Available: http://www.mathworks.se/help/pde/ug/decsg.html

[19] Mathworks, "R2014a documentation, partial differential equation toolbox, assemb." [Online]. Available: http://www.mathworks.se/help/pde/ug/assemb.html

[20] Hpc wiki, ntnu hpc group. [Online]. Available: https://www.hpc.ntnu.no/display/hpc/Techila+Distributed+Computing+Solution

APPENDIX A
FLOW CHART



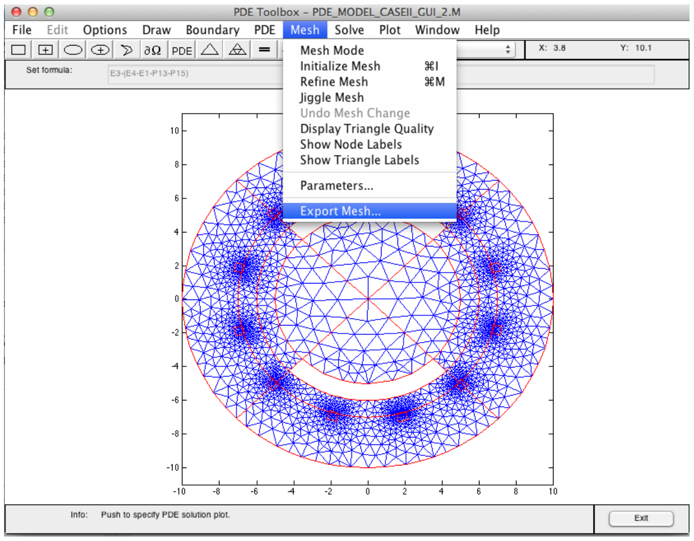
Matlab script
FEM-software ie. COMSOL
Input limits, iterations etc.
Initilize population
Find current solution
FEM calculation
This part is being parallel processed
no
Done for all particles?
yes
update personal/global best and particle position
Reached max it. or solution?
no
yes
Plot solution

Fig. 20. Overall flow chart

APPENDIX B
NOTE ABOUT HOW TO SETUP MODEL FOR PDE TOOLBOX FROM COMMAND LINE

This note contains all about setting up a magneto static model for the PDE-toolbox found during the work with this study. In addition I recommend using Matlab's online documentation [17]. Since all work in this master thesis have been in 2D, all description below are for the 2D case. Also note that for all parameters there are different formats and ways of obtaining them that are approved, but I will only describe the method I have used.

Table X shows an overview of the different variables. By actually setting up the model using the GUI, all these variables can be exported to the Matlab workspace. Figure 21 shows how to export the mesh, all other variables are similar. From the experience from the work with this thesis it is recommended to set up the simulation in the GUI before making the script. That way one gain two big advantages: firstly it is possible to copy in most variables in the beginning and generate them by scripting later, and secondly you can use them as reference for your own. If one of the matrices/-variables are unchanged and then the imported from the GUI, it can be used in the simulation.



Set formula: E3-(E4-E1-P13-P15)

Info: Push to specify PDE solution plot.

Fig. 21. Screenshot explaining how to export mesh in the GUI

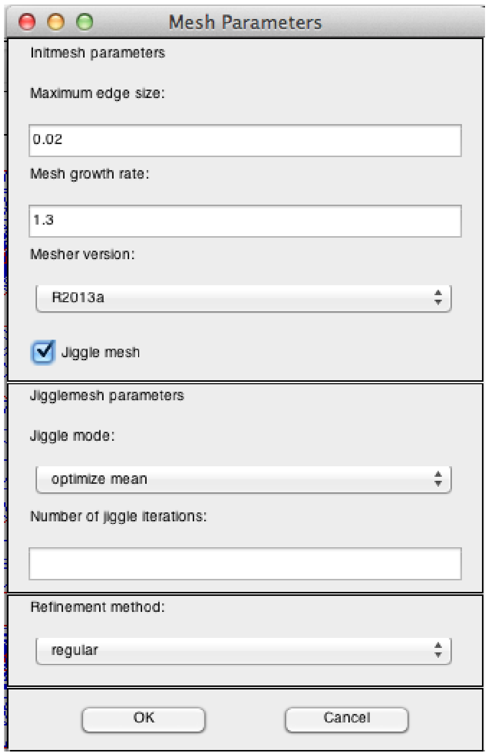
TABLE X
TABLE OF PDE TOOLBOX VARIABLES

g	the geometry matrix
b	the boundary condition matrix. Has one column for each column in g.
p	one of the three mesh matrices. Containing the position of all nodes
e	second of the mesh matrices. Containing information of all geometry edges (note! not all triangle edges)
t	third of the mesh matrices. Contains all triangle numbers and indices to all three corner nodes
f	information about the current density (J)
c	material properties
a	seems to be zero for all magnetostatic problems
u	the solution vector. Contains the vector potential (A) in all nodes

Below I will go through all steps required to make a model from a matlab script. **Bold** will be used for matlab variables/-matrices and *italic* will be used for matlab commands.

1) Geometry: I found it easiest to make the geometry by first making a so called **gdm** matrix and then use *decsg* to make **g**. In gdm there is a coloumn for each drawn figure. The top row contains a number the tells what kind of figure it is, the next rows containes properties as corner points, center and radius according to [18][paragraph "Geometry Description Matrix"], and last zeros until the column length equals that of the largest of the vectors. For examples see lines about 35-55 in appendix G. *decsg* also requires an expression for the geometry, for examples see line 15 in appendix F and line 56 in appendix G.

2) Meshing: This is quite straight forward. I have just used *initmesh* and the only things I have specified are **Hmax** and **MesherVersion** (see line 19 in appendix F). I recommend taking a look at the specifications in the GUI and find what settings that are used there.



Initmesh parameters

Maximum edge size: 0.02

Mesh growth rate: 1.3

Mesher version: R2013a

[✓ Jiggle mesh]

Jigglemesh parameters

Jiggle mode: optimize mean

Number of jiggle iterations:

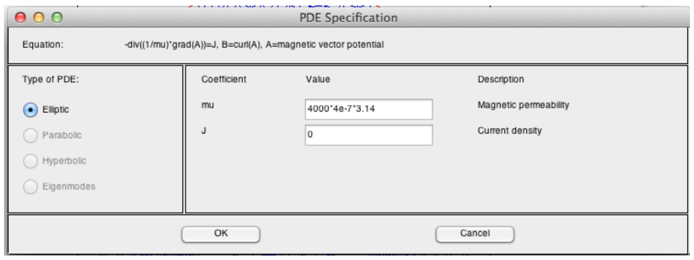
Refinement method: regular

Fig. 22. Screenshot of mesh settings in the GUI

The command *initmesh* returns the three meshing matrices; **p**, **e** and **t**. If ever referring to a point/boundaries in the geometry (f.i. for setting up boundary conditions) it is important to understand the relations between these matrices and the solution. **p** is the easiest; it contains x and y coordinates to all nodes in the geometry. **e** contains all edges in the geometry, so all triangle edges that lies "on top" of the "drawn" geometry has a column in **e**. The first two rows in **e** contains the indices to the starting and stopping node, these indices are equal to the indices to the row i **p**. Rows 3 and 4 in **e** contains parameter values at the starting and stopping point. Row 5 has the boundary segment number, which is the one needed to specify boundary conditions. Last rows 6 and 7 has the subdomain numbers to both sides of the edge. Choose "Show Edge Labels" and "Show Subdomain Labels" from the "Boundary"-menu in the toolbox and what subdomain numbers and boundary segment number is will hopefully be easy understandable. The last meshing matrix **t** contains the indices to the three corner nodes in all triangles in row 1-3, the triangle numbers in the 4th row.

3) PDE specifications: The specifications are given by the three variables **f**, **c** and **a**. I used the same format as if exported from the GUI. That means a string with the value of **f**, **c** and **a** as text divided by exclamation mark (!). If having one entry for each geometric figure, in the same order as in the geometry matrix **gdm**, I got it working. If comparing to the GUI (see screenshot in figure 23) **f** equals J, **c** equals mu and a is zero. See line 25-27 in appendix F and line 115-139 in appendix G for examples.

4) Boundary conditions: First of all, boundary conditions can only be specified for borders between geometry and



Equation: -div((1/mu)*grad(A))=J, B=curl(A), A=magnetic vector potential

Type of PDE: [✓ Elliptic] Parabolic Hyperbolic Eigenmodes

Coefficient	Value	Description
mu	4000*4e-7*3.14	Magnetic permeability
J	0	Current density

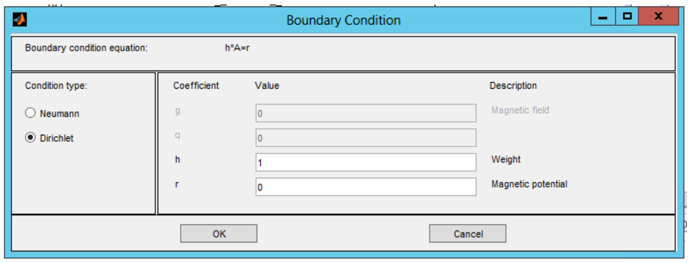
Fig. 23. Setting up PDE specifications in the GUI

surroundings. Go to "Boundary mode" in the GUI to see which boundaries you can specify conditions to in the geometry in question. One can choose between Dirichlet and Neumann boundary conditions (or both). Equation (12) describes the Dirichlet boundary conditions and Equation (13) describes the Neumann boundary conditions.

$$h * A = r \tag{12}$$

$$\frac{n}{\mu} * \nabla A + q * A = g \tag{13}$$

In the first equation **h** is explained as "weight" and **r** as "magnetic potential" in the GUI (see figure 24). Using Neumann as conditions type **g** is "Magnetic field" and **q** is a constant. Note that **q**, **g**, **h** and **r** is expressed in strings.



Boundary condition equation: h*A=r

Condition type: Neumann [✓ Dirichlet]

Coefficient	Value	Description
g	0	Magnetic field
q	0	
h	1	Weight
r	0	Magnetic potential

Fig. 24. Setting up PDE specifications in the GUI

The **b** matrix was the one most difficult to understand. For each column in the decomposed geometry matrix **g** there is a column in **b**. There is one column for all distinct edge labels (which is the same label as in the 5th row in e). I've concluded for myself that every column have to correspond to one line segment (the line between two points, either corners and/or a point where two drawn lines cross) in the drawn geometry (as can be seen in "Draw mode" in the GUI). I have not found anywhere this is said, but my interpretation have seemed to work.

The first row in **b** has the number of PDEs, this can be set to zero for edge segments that is not borders. For a 2D case there is usually only one PDE. The second row contains the number of Dirichlet conditions. The GUI only accepts one boundary condition (either Dirichlet or Neumann), but using the command line there is no limit as far as I have found. The next rows contains the ***lengths*** of the strings containing **q**, **g**, **h** and **r** respectively. The next rows contain **q**, **g**, **h** and **r** character by character according to the above number of characters.

So if you all in all have one PDE, one set of Diriclet boundary condition and one set of Neumann condition and the border conditions can be expressed as the following $1 * A = 2 * x$ and $\frac{n}{\mu} * \nabla A + (y + x) * A = 100$, then **q**='y+x', **g**='1045', **h**='11' and **r**='x'.

$$\begin{bmatrix}
\cdot & 1 & \text{(number of PDEs)} & \cdot \\
\cdot & 1 & \text{(one Dirichlet condition)} & \cdot \\
\cdot & 3 & \text{(number of chars in q)} & \cdot \\
\cdot & 4 & \text{(number of chars in g)} & \cdot \\
\cdot & 2 & \text{(number of chars in h)} & \cdot \\
\cdot & 1 & \text{(number of chars in r)} & \cdot \\
\cdot & 'y' & \text{(1st char in q)} & \cdot \\
\cdot & '+' & \text{(2nd char in q)} & \cdot \\
\cdot & 'x' & \text{(3rd char in q)} & \cdot \\
\cdot & '1' & \text{(1st char in g)} & \cdot \\
\cdot & '0' & \text{(2nd char in g)} & \cdot \\
\cdot & '4' & \text{(3rd char in g)} & \cdot \\
\cdot & '5' & \text{(4th char in g)} & \cdot \\
\cdot & '1' & \text{(1st char in h)} & \cdot \\
\cdot & '1' & \text{(2nd char in h)} & \cdot \\
\cdot & 'x' & \text{(1st char in r)} & \cdot
\end{bmatrix} \tag{14}$$

The length of the vector is determined by the longest column. All values below those specifying the boundary condition is zero. See lines 93-113 in appendix G for examples. More about the **b** matrix can be found at Mathwork's webpage [19].

5) Solution: The solution **u** is magnetic potential in all nodes, corresponding to **p**. The pde is solved by using the Matlab command *assempde*. Useful functions when the solution are obtained is *pdegrad* that evaluates the gradient in triangle midpoint and *pdeprtni* that interpolates from triangle midpoint to nodes. Note that the B-field can be found from the gradient of u (for a 2D case) from equation (6). In code: B=[uy,-ux]

6) Problems: During this work I have encountered quite some problems. The first one was how to get only one index from each query in the process finding the labels for the boundary condition. The solution I found that worked was to multiply all x and y values with 1e4, round off and then scale back. This way I could get an accuracy of four digits, but one difference in the 15th decimal did not matter. For values around zeros I used absolute value < 0.01 instead. This approach seemed to work with other problems as well, as with geometry problems. In the end I had an unsolved problem with geometry/meshing, but there seemed to be no system about it and the simulations ran most times so this is still unsolved.

Appendix C
Note about setting up techila and configure code to use cloudfor

Techila is quite straight forward to both setup and use.

7) Installation: In the HPC wiki [20] there are information about how to sproceed to set up techila at NTNU. After doing the installation it is important to add the new techila folder to your Matlab path.

8) Techila parameters: All simulations done in this thesis is done by using the following set of parameters in the code:

```
numCores=32;
  stpw=ceil(popSize/numCores);

cloudfor mm=1:popSize
  %cf:stepsperworker=stpw
  %cf:peach LocalCompile=false
  %cf:peach RemoteCompile=true
  %cf:peach CompilationPlatforms= {{'Windows',
      'amd64', 'mcc_64'}}
      [loop content]
  cloudend
```

It can be confusing that the techila parameters looks like comments, but the rule is that all lines preceding *cloudfor* that starts with *%cf:* is techila parameters. The **stepsperworker** says how many iteration each core should do. I had maximum 32 cores available, so for instance if I should simulate 64 particles **stepsperworker** would be 2. The next three lines tells techila that I want remotely compilation and that I want to run the code on Windows machines. At NTNU, windows compilation automatically means using idle computer lab PCs to execute the code.

9) Problems running code: If you don't manage to run your code, try put it into the /lib/Matlab/ folder (in the techila folder), that solved it for me. Also try to delete all files created automatically by techila during execution (techila_tmp-folder, techila_*.mat for instance).

Appendix D
Note with information about resources at NTNU

These are some of the resources I've located at NTNU during the work with this thesis. I have not been in contact or used all, but it is included in the hope to help other students at NTNU working with the same topic. This was written in may 2014, and I do not guarantee the correctness of any of it.

TABLE XI
Resources at NTNU

The techila solution	Cloud software for parallel computing. Tested in this report, and can be used together with, amongst others, Matlab. It is possible to run hundreds of parallels using techila at NTNU, but in that case one need to get into an contract with the the section for Scientific Data Processing.
48 cores computers	The department of Electric Power Engineering have three such machines. eelk1734 and eelk1728 are two of these. Anders Gytri and Kurt Salmi administrate these machines. At least to eelk1728 supervisor has to request access on behalf of the student.
100 cores computers with 1TB memory	There is one such computer at NTNU, I am not sure where but I guess it is at IDI. Heard about it from Anders Gytri.
Powerful graphical work stations	There are four of these placed in F312 that belong to the department of Electric Power Engineering. Supervisor has to request access on behalf of the student.
The section for Scientific Data Processing	Part of the IT department at NTNU. In this project John Floan from this department has helped me a lot.
Full license Comsol	There is a full license on eelk1728. It is also possible to require a license, supervisor has to file this request.

Appendix E
PSO code

```
function res = PSO_function(popSize,imax, LB, UB)
% popSize = number of particles
% imax = number of iterations
% LB = lower boundary (in 2D LB=[xmin ymin])
% UB = upper boundary (in 2D UB=[xmax ymax])

% starting timer
tic

% initilizing of variables
c1 = 2;
c2 = 2;
nvar = 2; %dimention of the problem
dt = 0.2; %tuning, how "aggressive" the algorithm
     is
fitness = zeros(popSize,1);

% creates the initial population
x = PSO_initialPop(popSize,nvar,LB,UB);

% finding the result for the initial population
% this loop was later changed to parfor/cloudfor
for i = 1 : popSize
  fitness(i,1) = pde_model_commandline(x(i,1),x(i
      ,2));
end

% initilation more variables
pb = fitness; %personal best for all particles
xPb = x; %the corresponding x,y placement for pb
v = zeros(popSize,length(LB)); %current velocity
[gb,iGB]=max(fitness); %starting global best (gb)
xGb = x(iGB,:); %starting global best position

% doing the iterations
for i =1:imax
  [w = PSO_inertialWeight(i,imax)

  % finding current solution for all particles
  % this loop was later changed to parfor/
      cloudfor
  for j = 1 : popSize
    fitness(j,1)=pde_model_commandline(x(j,1),x(j
        ,2));
  end

  % compare current to personal- and global bests
  for j = 1 : popSize
    % compares present position to personal best
    if fitness(j,1) > pb(j,1)
      pb(j,1) = fitness(j,1); %updates p. best
          value
      xPb(j,:) = x(j,:); %updates p. best
          position
    end
    % compares current postition to the global
        best
    if fitness(j,1) > gb
      gb = fitness(j,1); %updates global best
          value
      xGb = x(j,:); %updates gb position
    end
  end

  % updating velocity and position
  for  k = 1 : popSize
    v(k,:) = w*v(k,:) + c1*rand*(xPb(k,:) - x(k
        ,:)) +  c2*rand*(xGb - x(k,:)); %new
        velocity
    x(k,:) = x(k,:) + dt*v(k,:); %new position

    % dealing with particles outside boundries
    for m = 1 : length(LB)
      if (x(k,m) < LB(m))
        x(k,m) = LB(m);
        v(k,m) = 0;
      elseif (x(k,m) > UB(m))
        x(k,m) = UB(m);
        v(k,m) = 0;
      end
    end
  end
end

% stopping timer
time=toc;

% plotting final solution
pde_model_commandline(xGb(1),xGb(2),i,1);

% returning runtime and the global best values
res=[time gb xGb(1) xGb(2)]

end
```

Appendix F
PDE toolbox model case 1

```
function Bfield = pde_model_commandline(xmagn,
    ymagn, i, ifPlot)

% Material parameters
mu_0=4*pi*1e-7; %Permeability vakuum
mu_iron=4000*mu_0; %Permeability iron

% Geometry description, defining the different
    objects
area = [3 4 -1 1 1 -1 0.5 0.5 -0.8 -0.8];
iron_1 = [3 4 -1 1 1 -1 0.5 0.5 0.45 0.45];
iron_2 = [3 4 -0.4 0.4 0.4 -0.4 0.45 0.45 0.2
    0.2];
magnet = [3 4 xmagn xmagn+0.6 xmagn+0.6 xmagn
    ymagn+0.3 ymagn+0.3 ymagn ymagn];

%making the geometry matrix and plots it
gdm = [area; iron_1; iron_2; magnet]';
g = decsg(gdm, 'R1-R4+R3+R2', ['R1'; 'R2'; 'R3';
    'R4']');

% Mesh generation
hmax=0.02;
[p,e,t]=initmesh(g,'Hmax',hmax,'MesherVersion','
    R2013a');

% Boundary conditions (b was exported from gui)
b=[0 0 0 1 1 1 1 ... 41 0 0 0];

% PDE coefficients
f='0!0!0';
c=strcat(num2str(1/mu_iron),'!',num2str(1/mu_iron
    ),'!',num2str(1/mu_0));
a='0.0!0.0!0.0';

%Solving PDE
u = assempde(b, p, e, t, c, a, f); %u is magnetic
     potential

%finding gradient to u at triangle MIDPOINT
[ux,uy] = pdegrad(p,t,u);

%Making the B-field at triangle midpoint
B=[uy;-ux];

%compute gradient of A in NODES
ux_node=pdeprtni(p,t,ux);
uy_node=pdeprtni(p,t,uy);

```

```
%absolute value of B is the length of vector at
    NODES
absB=sqrt((ux_node.^2)+(uy_node.^2));

%Plotting solutions
if ifPlot==1
    %plot geometri og contour-plot
    figure(1)
    set(gcf,'name','Contour plot of B-field')
    hold on
    pdegplot(g, 'subdomainLabels','on');
    pdeplot(p,e,t,'xydata',absB,'contour','on','
        levels',15);
    hold off
    [more plotting]
end

%finding solution in (-0.4,0.2)
index=intersect(find(p(2,:)==0.2),(find(p(1,:)
    ==-0.4))); %index in solution vector

if isempty(index)
    Bfield=0;
elseif xmagn==-0.2
    Bfield=0;
else
    Bfield=absB(index);
end
end
```

Appendix G
PDE Toolbox Model Case 2

```
function Torque = pde_model_CaseII_commandline_2(
    hmagn, wmagn, i, ifPlot)

%parameters for the geometry
r_rotor=5; %radius rotor
dr_airgap=2; %width airgap
dr_stator=3; %width stator
width_slot=4; %in degrees
depth_slot=0.5;
ant_slots=12; %number of slots
%radius for torque-calculation circle
r_circle_torque=r_rotor+dr_airgap-0.03;
%at what degrees the slots are placed
grader_slot=[15 45 ... 345];

%the currents in the three phases r, s and t
curr_r=2e3;
curr_s=4e3;
curr_t=2e3;

%input boundary conditions
q_neumann='0'; %q, Neumann
g_neumann='0'; %g, Neumann
h_dirichlet='1'; %h, Dirichlet
r_dirichlet_magnet='1.20000*x'; %r, Dirichlet
r_dirichlet_outer='0'; %r, Dirichlet

% Material parameters
mu_0=4*pi*1e-7; %Permeability vakuum
mu_iron=4000*mu_0; %Permeability iron
mu_copper=mu_0; %Permeability copper

% Geometry description:

% Defining all the circles
area_rotor = [4 0 0 r_rotor r_rotor 0 0 0 0 0 0
    0];
[similar for the other circles]

% Defining all the slots
slots=zeros(12,ant_slots);
slots(1,:)=2;
slots(2,:)=4;
slots(3,:)=cosd(grader_slot-(width_slot/2))*(
    r_rotor+dr_airgap-0.001);
slots(7,:)=sind(grader_slot-(width_slot/2))*(
    r_rotor+dr_airgap-0.001);
[row 4-6,8-10 contains x,y for rest of the
    corners]

% The polynoms to make the magnet widht
%angle in rad from x-axis to magnet "start"
ang_magn=((pi*r_rotor-wmagn)/r_rotor)/2;
y=sin(ang_magn)*(r_rotor+dr_airgap+dr_stator);
x=cos(ang_magn)*(r_rotor+dr_airgap+dr_stator);
area_neg_tri = [2 5 0 -x -10 -10 -x 0 y y -y -y];
area_pos_tri = [2 5 0 x 10 10 x 0 y y -y -y];

% Making the geometry matrix
gdm = [area_rotor' area_air' area_stator'
    circle_torque' slots area_magnet'
    area_neg_tri' area_pos_tri'];
[g, bt] = decsg(gdm, '(E3+P1+P2+P3+P4+P5+P6+P7+P8
    +P9+S1+S2+S3+E2+E5)-(E4-E1-T1-T2)',['E1'; 'E2
    '; [labels corresponding to gdm] ; 'T2']');

% Mesh generation:
hmax=0.2;
[p,e,t]=initmesh(g,'Hmax',hmax,'MesherVersion','
    R2013a');

%Get edge labels for outer circle and magnet
%find index in p to a known point at all outer
    boundaries
index_outer_1=intersect(find(p(2,:)==10),(find(
    abs(p(1,:))<0.01)));
[dping the same for p(1,:) and -10]

%find index in p to a known point at all long
    sides of magnets
index_bottommagn_bottom=intersect(find(p(2,:)==-
    r_rotor),(find(p(1,:)<0.001)));
index_bottommagn_top=intersect(find(round(p(2,:)
    *100000)==-(round((r_rotor+hmagn)*100000))),(
    find(abs(p(1,:))<0.01)));
[same for top magnet]

%find index in p to a known point at all long
    sides of magnets by finding index to center
    and recognicing that the short sides always
    has same boundary number plus/minus 1 as the
    four that meats in the center
index_center=intersect(find(p(2,:)==0),(find(p
    (1,:)==0)));

%finding boundary labels to the edges by finding
    the index in the first or second row of the e
     matrix. the label is in row 5.
%labels outer circle
label_outer_1=e(5,(find(e(1,:)==index_outer_1)));
label_outer_2=e(5,(find(e(1,:)==index_outer_1)));
[similar for index_outer_2, ..._3 and ..._4]

%labels long sides magnets (must have two since
    the long sides divide into two segments when
    long)
label_topmagn_bottom=e(5,(find(e(1,:)==
    index_topmagn_bottom)));
label_topmagn_bottom_2=e(5,(find(e(2,:)==
    index_topmagn_bottom)));
[similar for the rest of the indices]

%labels short sides magnets
label_center1=e(5,(find(e(1,:)==index_center)));
label_center2=e(5,(find(e(2,:)==index_center)));

label_topmagn_3=label_center1(1)+1;
[similar for the 3 others]

```

```
%making boundary condition matrix b (doc pdebound
    )
ne = size(g,2); % number of edges

N=1;%number of PDEs
M=1;%number of Dirichlet conditions

%assume only Dirichlet boundary contitions
b=zeros((6+numel(q_neumann)+numel(g_neumann)+
    numel(h_dirichlet)+numel(r_dirichlet_magnet))
    ,ne);
for k = 1:ne
    switch k
        %setting boundary conditions for outer
            circle
        case {[any of labels for outer circle]}
           b(:,k)=[1;1;1;1;1;1;'0';'0';'1';'0'
               ;0;0;0;0;0;0;0;0];
        %setting boundary conditions for magnets
        case {[any of labels for magnets]}
            b(:,k)=[1;1;1;1;1;length_r;'0';'0';'1
                ';r_dirichlet_magnet(1);...;
                r_dirichlet_magnet(9)];
        %setting the rest of the edges to having
            none
        otherwise
            b(:,k)=[0;1;1;1;1;1;'0';'0';'0';'0'
                ;0;0;0;0;0;0;0;0];
    end
end

% Setting up PDE coefficients

%same order as in gdm
current=[0 0 0 0 curr_t curr_t ... -curr_r 0 0
    0];
f_temp=bt*current';

%material constants
iron=1/mu_iron;
air=1/mu_0;
copper=1/mu_copper;

%same order as in gdm
material=[iron air iron ... iron];
c_temp=bt*material';

%making the f and c variables. Did not find a
    better way to do this.
f=num2str(f_temp(1));
c=num2str(c_temp(1));

for i=2:length(f_temp)
    f=strcat(f,'!',num2str(f_temp(i)));
    c=strcat(c,'!',num2str(c_temp(i)));
end

a=0; %seems to be 0 for all magnetostatic
    problems

%Solving PDE (u is magnetic potential)
u = assempde(b, p, e, t, c, a, f);

%find gradient to u at triangle MIDPOINT
[ux,uy] = pdegrad(p,t,u);

%compute gradient of A in NODES
ux_node=pdeprtni(p,t,ux);
uy_node=pdeprtni(p,t,uy);

%Making the B-field at triangle midpoint ref[1]
B=[uy;-ux];

%absolute value of B at node points
absB=sqrt((ux_node.^2)+(uy_node.^2));

%Plotting solutions
if ifPlot==1
    %plot geometri og contour-plot
    figure(1)
    set(gcf,'name','Contour plot of B-field')
    hold on
    pdegplot(g);
    pdeplot(p,e,t,'xydata',absB,'contour','on','
        levels',20);
    hold off

    [more plots]
end

% Calsulating magnet area
TotalMagnetArea=(((hmagn+r_rotor)^2*((pi/2)*
    ang_magn))-(r_rotor)^2*((pi/2)*ang_magn))*2;

%finds all boundary labels to the "torque circle"
index_temp=intersect(find(round(p(1,:)*100)==
    round(0*100)),find(round(p(2,:)*100)==round(
    r_circle_torque*100)));
label_circle_torque_1=e(5,(find(e(1,:)==
    index_temp)));
label_circle_torque_1x=e(5,(find(e(2,:)==
    index_temp)));
[similar in the three other directions as well]

%finds index i p to all nodes at the "torque
    circle"
node_points_circle_torque=[];
for k = 1:length(e(1,:))
    switch e(5,k)
        case {label_circle_torque_1 ...
            label_circle_torque_4x}
            node_points_circle_torque=[
                node_points_circle_torque e(1,k)
                ];
    end
end

%calculation of tensor and force
sum_f=0;
for n=(node_points_circle_torque);
    %tensor
    T=[(Bx(n)^2-0.5*absB(n)^2) Bx(n)*By(n); By(n)
        *Bx(n) (By(n)^2-0.5*absB(n)^2)];
    %force in each point
    f=(1/mu_0)*T*[(p(1,n)/r_circle_torque);(p(2,n
        )/r_circle_torque)];
    %force causing axial torque
    df=sqrt((dot(tS,f)*[nS(2);-nS(1)](1))^2+(dot(
        tS,f)*[nS(2);-nS(1)](2))^2);
    %summing up force contributions
    sum_f=sum_f+df;
end
%multiplying with average length between nodes
F=sum_f*((r_circle_torque*2*pi)/length(
    node_points_circle_torque));
Torque=F*r_circle_torque*1e-4; %due to scaling
end
```