TherMaG: Engineering Design of Thermal-Magnetic Generator with Multidisciplinary Design Optimization

Will Hintlian[*], Hanfeng Zhai[†], Mads Peter Berg[‡]

Sibley School of Mechanical and Aerospace Engineering

Applied and Engineering Physics

Cornell University

November 5, 2021

1 Simulation Completion

We completed the simulation and have produced a grand MATLAB script which calls two different COMSOL setups for one given set of design variables. We extract the total cost of the system as described in the previous assignment, simply by multiplying the respective parts the system by their current market prices. We extract the power output by dividing the square of the total magnetic flux difference by the period of a thermal heating/cooling cycle. For a given vector of design variables, we thus run 3 simulations. The first two simulate the magnetic field for different instances of the magnetic permeability, and the last simulates the temperature distribution over time. For computing the efficiency, we can reuse the simulation results for the magnetic fields and this time extract the total exergy change of the system over a thermal cycle. This result is extracted from the same simulation as the heating time, and thus we only need to run 3 separate simulations per function(s) evaluation. We refer to assignment 2 for further details on the physical reasoning. At this point, there are no further issues. As for interesting design points that can be used to initialize optimization algorithms,

[*]Email: wth42@cornell.edu
[†]Email: hz253@cornell.edu
[‡]Email: mpb99@cornell.edu

we did in assignment 2 make a full factorial design vector exploration and identified the design vectors that yielded the best results with respect to each objective function.

For optimizing the cost:
$(y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}) = (0.15, 0.15, 0.05, 0.05, 0.05)$
For optimizing the power output:
$(y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}) = (0.05, 0.45, 0.25, 0.15, 0.15)$
For optimizing the efficiency:
$(y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}) = (0.05, 0.45, 0.25, 0.15, 0.05)$
- where everything is in units of meters.

2 Heuristic Optimization

2.1 Heuristic Algorithm Selection

At the current stage, we mainly focus on one thing that of key interest by the industry: money. For every company, especially startups, core technology is important yet only within the budget limit and provide good profit. Notably, for our problem, the Thermal-Magnetic Generator (TMG) is not widely adopted by the industry in the last few decades largely due to the high cost of Gadolinium, which is the "*active material*" in our settings.

Here, we chose genetic algorithm (GA) for optimizing the TMG geometry to achieve as low cost as possible. GA mimics the Darwinian theory of survival of fittest in nature [1]. The main reason of choosing GA are: (1) GA is simple and easy to understand since it don't require a complex problem formulation for specific problems, which make us easy to grab and use [Ref.]. (2) It is usually faster to solve [Ref.]. (3) GA works well on discrete problems, and deals well with stochastic data [Ref.], which is very important for our implementation. Admittedly, as a heuristic method, GA does not promise a global optimum. However, in our problem, since we are estimating the cost of a product, therefore an generally "good" cost shall be acceptable for the final design [Ref.]. To emphasize, we don't want to sacrifice the product quality or power output of TMG just to reduce cost. Hence, GA can be adopted to achieve such "acceptable result" with simple implementation with a fast solving process.

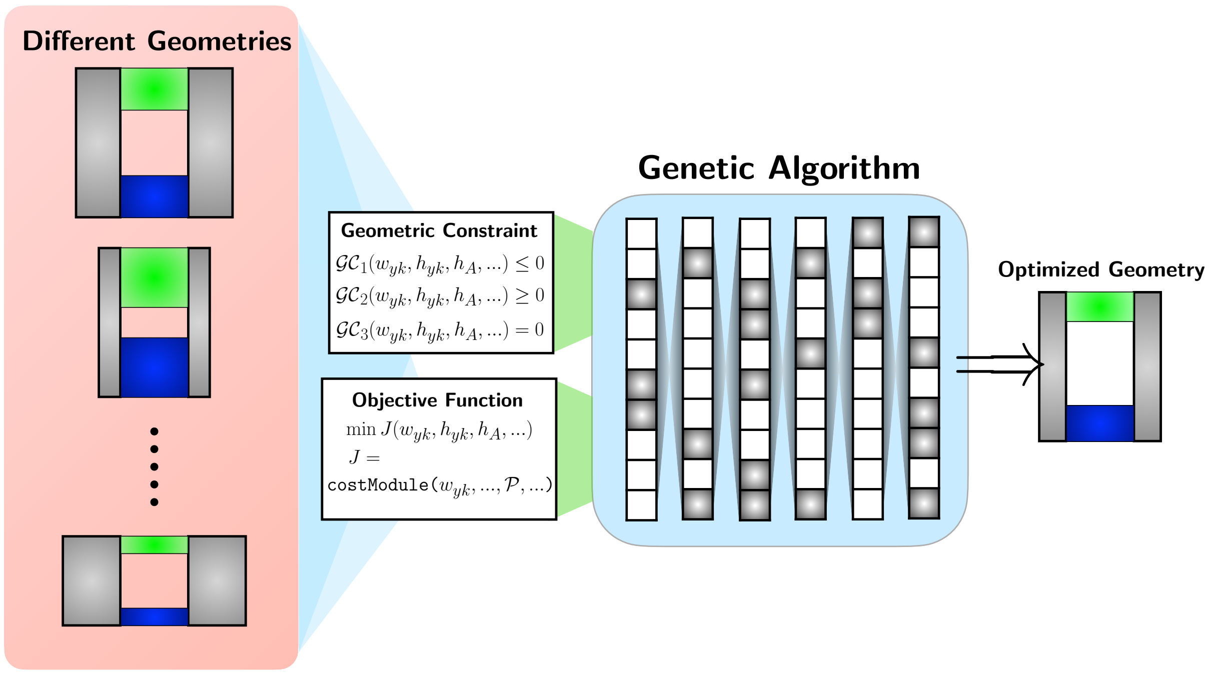
Different Geometries
Genetic Algorithm
Geometric Constraint
$\mathcal{GC}_1(w_{yk}, h_{yk}, h_A, ...) \leq 0$
$\mathcal{GC}_2(w_{yk}, h_{yk}, h_A, ...) \geq 0$
$\mathcal{GC}_3(w_{yk}, h_{yk}, h_A, ...) = 0$
Objective Function
$\min J(w_{yk}, h_{yk}, h_A, ...)$
$J =$
costModule$(w_{yk}, ..., \mathcal{P}, ...)$
Optimized Geometry

Figure 1: Schematic diagram for genetic algorithm optimization.

2.2 Single Objective Heuristic Optimization

The team focused on our cost module for single objective heuristic optimization. This was a logical choice because our cost module is independent of power and efficiency and is fully contained within MATLAB. This will drastically reduce computation time and minimize complexity when setting up the genetic algorithm. The algorithm is reliant on a new objective function adapted from the team's previous work to output cost. The cost objective function is included in Appendix 4. A new geometric constraint function was also adapted from previous work and included in Appendix 4. The main script containing options and the function call to run the genetic algorithm is included in Appendix 4.

After performing an initial setup of the genetic algorithm, the team began to tune the algorithm's parameters to reduce compute time while improving results. First, function and constraint tolerances were adjusted to 1×10^{-6}. We found that using a smaller value increased compute time and number of generations for each algorithm execution, but did not improve results. The team also experimented with different population sizes, but ultimately settled on the default value of 50 recommended by Mathworks for optimizations using five or fewer design variables. Due to our problem's geometric constraints we were unable to adjust mutation rate and relied on Matlab's `mutationadaptfeasible` function. Adjusting `maxgenerations` had no effect on our results as the algorithm generally completed each run in 3-5 generations.

The two settings which have greatest effect on the quality of our results are the crossover ratio and crossover fraction. Crossover ratio adjusts the distance away from the better of two parents at which a child is placed between generations. We found that a value of 1.6 consistently improved the quality of the results and tended to slightly increase the number of generations before convergence. Crossover fraction represents the portion of the next generation created by the crossover of two parents. For our optimization, we found a value of 0.7 yields slightly better and more consistent results.

2.3 Results Analysis

The team set up a script to run the genetic algorithm using our tuned settings 500 times to analyze the results. The script returned the following output:

```
The lowest cost from 500 algorithm runs is 0.3403, corresponding to a design vector of:

    0.0010    0.0020    0.0010    0.0010    0.0010


The algorithm found this to be the optimal result 77 times out of 500. The average optimal
    value is 0.4148
```

Our computationally cheap cost function affords us the luxury of running the genetic algorithm many times to help analyze the results. Because the algorithm found the same optimal result 77 times out of 500, we conclude that we have found the global optimum for cost. The minimized cost function yields a value of \$0.3403.

3 Gradient-based or local derivative-free optimization

3.1 Gradient-based Algorithm Selection

Here, to optimize the cost objective with gradient-based method, we employ the `fmincon` in MATLAB®. `fmincon` is a gradient-based method that is designed to work on problems where the objective and constraint functions are both continuous and have continuous first derivatives [3]. The reason we chose `fmincon` as the optimization methods include: (1) this method is a gradient-based method, agrees with the requirement for Q3; (3) this method is easy to implement, and handy to analyse the results. (3) the method serves a wide range of applications. Here, we employ the `SQP` method and give the analytical form of our problem to the toolbox for optimization evaluation.

The basic of gradient-based optimization is to formulate a Hessian as an optional input. This Hessian is the matrix of second derivatives of the Lagrangian, namely [3]:

$$\nabla^2 L(x,\lambda) = \nabla^2 J(x) + \sum \lambda_i \nabla^2 \mathcal{C}_i(x) + \sum \lambda_i \nabla^2 \mathcal{E}_i(x) \tag{1}$$

where λ_i are the parameters imposed on constraints to formulate the Lagrangian, and $\mathcal{C}_i$ are the inequality constraints, $\mathcal{E}_i$ are the equality constraints. Due to the analytical nature of the cost objective, the `fmincon` can be successfully implemented to TherMaG system.

3.2 Single Objective Gradient-based Optimization

3.2.1 Problem formulation

As stated in our last homework, the cost is a function to the geometric parameters of the TMG. The optimization of the cost can be written in standard form:

$$\begin{aligned} \min_{geometry} \quad & J = \texttt{cost} \\ \text{where } \texttt{cost} = {} & \mathbb{G}\mathcal{A}_{active} + \mathbb{I}\mathcal{A}_{yoke} + \mathbb{N}\mathcal{A}_{mag}, \\ \text{s.t.} \quad h_{yk}(2w_{yk} + w_{gap}) - V_{max} \leq 0 & \quad (\mathcal{C}_1) \\ h_{yk} - L_{max} \leq 0 & \quad (\mathcal{C}_2) \\ 2w_{yk} + w_{gap} - L_{max} \leq 0 & \quad (\mathcal{C}_3) \\ h_A + h_{pm} - h_{yk} \leq 0 & \quad (\mathcal{C}_4) \end{aligned}$$

where $\mathbb{G} = 1.71553 \times 10^5 [\$/m^3]$, $\mathbb{I} = 1.4804 \times 10^3 [\$/m^3]$, $\mathbb{N} = 1.628 \times 10^5 [\$/m^3]$, standing for the price in $[\$/m^3]$ for Gadolinium, Iron, and Neodymium, respectively; and $\mathcal{A}_{active} = h_A w_{gap}$, $\mathcal{A}_{yoke} = 2w_{yk}h_{yk}$, $\mathcal{A}_{mag} = h_{pm}w_{gap}$, stands for the area for active materials, yoke, and magnetic materials, respectively. The upper and lower bounds (`ub` *&* `lb`) of the variables $\mathbb{X} = [h_A, w_{gap}, w_{yk}, h_{yk}, h_{pm}]$ is $[0.05, 0.45]$.

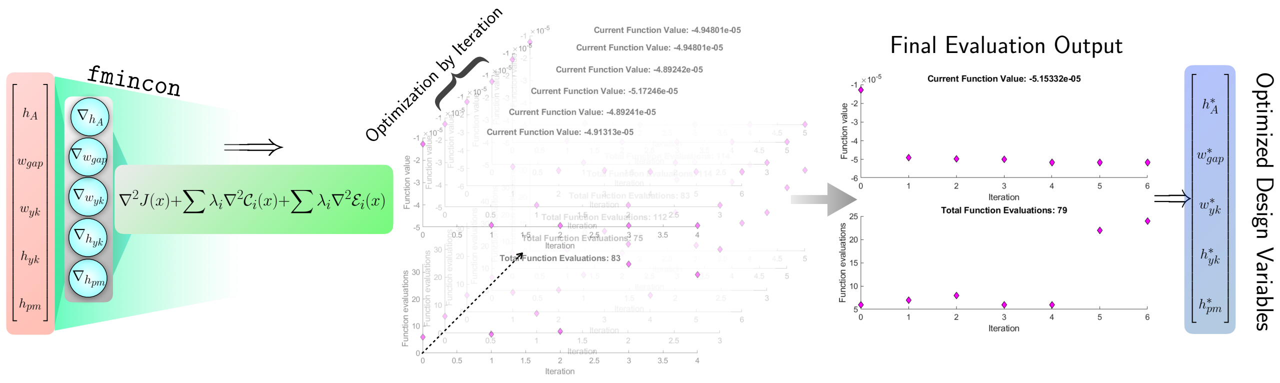
fmincon
Optimization by Iteration
Final Evaluation Output
Optimized Design Variables
$\nabla^2 J(x)+\sum \lambda_i \nabla^2 \mathcal{C}_i(x)+\sum \lambda_i \nabla^2 \mathcal{E}_i(x)$
Current Function Value: -5.15332e-05
Total Function Evaluations: 79
Function value
Function evaluations
Iteration

Figure 2: A schematic diagram representing the gradient-based optimization. Note that the optimization process can be accessed through our supplementary video showing the results of different runs of the optimization algorithm. Due to its stochastic nature, each run will not necessarily produce the same result. This particular video is for the optimization of the efficiency where the initial point was the "good" one.

3.2.2 Algorithm formulation

We therefore formulate the function of objective with `fmincon` on our cost module. A schematic representing our gradient-based optimization is shown in Figure 2. The optimization process can be simplified to:

- Step 1. `System setup`: $\Longrightarrow$ Setting up the running steps, initial design variables (initial guess), specify the optimization methods and related parameters involved (i.e., objective functions, constraints, lower & upper bounds, etc.).
- Step 2. `Optimization in loop`: $\Longrightarrow$ Setting up a `for` loop for running `fmincon` coupled with Livelink®. Ouput the optimized design with saving the iterations diagram.
- Step 3. `Parameters computation`: $\Longrightarrow$ Save the final optimized design and printed it out.

```
clear
clc
close all

tic

numRuns = 1;

% starting point for optimization
x0 = [.05;.1;.05;.05;.05];

```

```
% Set nondefault solver options
options = optimoptions('fmincon','Algorithm','sqp','PlotFcn',{@optimplotfval,
    @optimplotfunccount});

% set upper and lower bounds
lb = [.001, .001, .001, .001, .001];
ub = [.5, .5, .5, .5, .5];

% make anonymous functions
powerFcn = @(power) efficiency_power_modules(power);
efficiencyFcn = @(efficiency) efficiency_power_modules(efficiency);

%outputs = zeros(numRuns,1);
objVals = zeros(numRuns,1);
vectors = zeros(numRuns,5);

% run the optimization many times to see what happens
for i = 1:numRuns

    % Solve
    [solution,objectiveValue,exitflag,output,lambda,grad,hessian] = fmincon(powerFcn,x0
    ,[],[],[],[],lb,ub,@geocon,options;

    outputs(i) = output;
    objVals(i) = objectiveValue;
    vectors(i,:) = solution;

    string = "optim" + num2str(i);
    saveas(gcf,string,'png');

end

[min, index] = min(objVals);

fprintf('The best solution from %d optimization runs is power = %.8f, corresponding to a
    design vector of: \n',numRuns,min);
fprintf('\n');
disp(vectors(index,:));
fprintf('\n');
fprintf('This solution had the following simulation output info: \n');
fprintf('\n');
disp(outputs(index));

toc
```

3.3 Single-objective optimization

Using the script shown above, we move on to do a gradient based optimization of the cost. This is a simple objective function to optimize, and it will be easy to validate that our algorithm is doing a good job. Using an initial design vector given by, $[y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}] = [0.05; 0.1; 0.05; 0.05; 0.05]$, our results are summarized by the following output in MATLAB:

```
The best solution from 30 optimization runs is cost = 0.34027460, corresponding to a design
    vector of:

    0.0010    0.0020    0.0010    0.0010    0.0010



This solution had the following simulation output info:

         iterations: 2
          funcCount: 18
          algorithm: 'sqp'
    constrviolation: 0
           stepsize: 1.4867e-17
       lssteplength: 1
      firstorderopt: 3.7253e-09

Elapsed time is 15.709778 seconds.
```

The initial design vector here was arbitrary, but the optimal result is exactly the same as the one found by our genetic algorithm in the previous section. If we instead use the "good" guess, given by the vector, $[y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}] = [0.15; 0.15; 0.05; 0.05; 0.05]$ pulled from the effects table in our Design of Experiments from Assignment 2, the results are as follows:

```
The best solution from 30 optimization runs is cost = 0.34027460, corresponding to a design
    vector of:

    0.0010    0.0020    0.0010    0.0010    0.0010



This solution had the following simulation output info:

         iterations: 2
          funcCount: 18
          algorithm: 'sqp'
    constrviolation: 0
           stepsize: 1.8527e-18
       lssteplength: 1
    firstorderopt: 7.4506e-09

Elapsed time is 13.554918 seconds.
```

These results make a lot of sense. The total cost is a simple function to compute and a gradient based optimization algorithm should quickly be able to figure out that having smaller dimensions will generally decrease the overall cost. There are also constraints on the dimensions, so it is not trivial to figure out which configuration exactly, is the best. As seen however, it did turn out that with both initial guesses, the gradient based method moved our design vector into and along a constraint boundary such that the final point was the same, namely the design vector, $[0.0010; 0.0020; 0.0010; 0.0010; 0.0010]$. Additionally, we see that the elapsed time for the optimization with a "good" starting point is approximately 14% less than with an arbitrary initial guess, which also makes sense. The team is pleased that both the heuristic and gradient based optimization methods yielded identical results in minimizing cost.

If we try doing the same for the efficiency or the power, the situation is different. Different initial guesses will lead to different optima. It was not required for the project, but we here demonstrate doing the same procedure with the efficiency as the objective function. To achieve this result, the team wrote a function evaluation script which couples Matlab with COMSOL to automatically run simulations with design variables dictated by the optimization algorithm. Let us start with an arbitrary design vector again given by, $[y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}] = [0.05; 0.1; 0.05; 0.05; 0.05]$. The following output is then generated[1]:

```
The best solution from 30 optimization runs is efficiency = -0.00002090, corresponding to a
    design vector of:

    0.0438    0.1274    0.0969    0.0276    0.0124


This solution had the following simulation output info:

         iterations: 9
          funcCount: 136
          algorithm: 'sqp'
    constrviolation: 0
           stepsize: 9.6833e-07
       lssteplength: 3.2199e-05
```

[1]Note here that the efficiency should be multiplied by -1, as the problem is formulated as one of minimization of the negative efficiency

```
      firstorderopt: 0.2493
       bestfeasible: [1x1 struct]

Elapsed time is 29333.725125 seconds.
```

The computed optimal design vector is given by, [0.0438; 0.1274; 0.0969; 0.0276; 0.0124]. Note that efficiency is negative. This is because the optimization minimizes the objective function, so we multiplied the result of our efficiency calculation by -1 before passing the value into the optimization function. In this way, the most negative value represents the highest efficiency. Trying again with the "good" guess, this time given by, $[y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}]$=[0.05; 0.45; 0.25; 0.15; 0.05], the output becomes:

```
The best solution from 30 optimization runs is efficiency = -0.00008848, corresponding to a
    design vector of:

    0.0759    0.5000    0.4300    0.0662    0.0976


This solution had the following simulation output info:

         iterations: 7
          funcCount: 147
          algorithm: 'sqp'
    constrviolation: 0
           stepsize: 8.7723e-07
       lssteplength: 1.8562e-06
      firstorderopt: 2.3595
       bestfeasible: [1x1 struct]

Elapsed time is 28687.589087 seconds.
```

As seen, the optimal design is here supposed to be given by, [0.0759; 0.5000; 0.4300; 0.0662; 0.0976]. There clearly is a significant difference. The two different design vectors do have considerable differences in the ratios between individual design variables, and going as as to say that different "design niches" have been found, might not be completely unjustifiable. It is however clear that the design based on the "good" guess outperforms the one based on the arbitrary one substantially, which seems like a valuable experience to have. For the particular case of the optimization of the efficiency, let us show how the algorithm performed across iterations graphically. For the arbitrary guess, we refer to figure 3. For the "good" guess, we refer to figure 4.

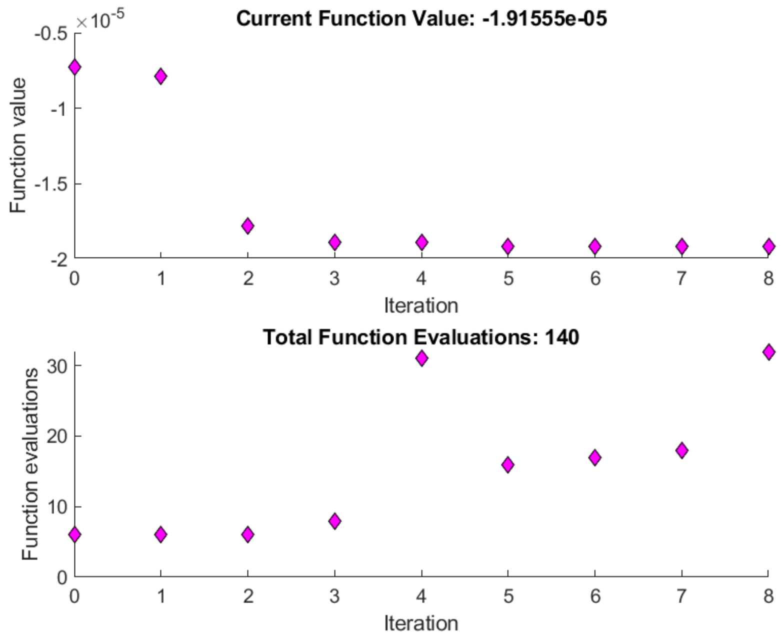

Figure 3: Plot showing the function evaluation at each iteration (top) and the number of function evaluations for the given iteration (bottom) in the case where we do gradient based optimization of the efficiency based on the aforementioned arbitrary design vector as the initial guess. The design space seems ”flat” in the beginning, but eventually, some ”hyper-geometric” edge is being reached and the value of the function drops rapidly

We also tried doing this for the total power output. Note that again, maximum power is represented by the most negative result. Using as the initial guess the same arbitrary one as in the previous two cases, we get the output,

```
The best solution from 30 optimization runs is power = -0.00003466, corresponding to a
    design vector of:

    0.1503    0.1788    0.1375    0.0413    0.1994


This solution had the following simulation output info:

         iterations: 6
          funcCount: 91
          algorithm: 'sqp'
    constrviolation: 2.7756e-17
           stepsize: 1.3562e-06
       lssteplength: 7.7310e-06
      firstorderopt: 0.0427

Elapsed time is 46578.355250 seconds.
```

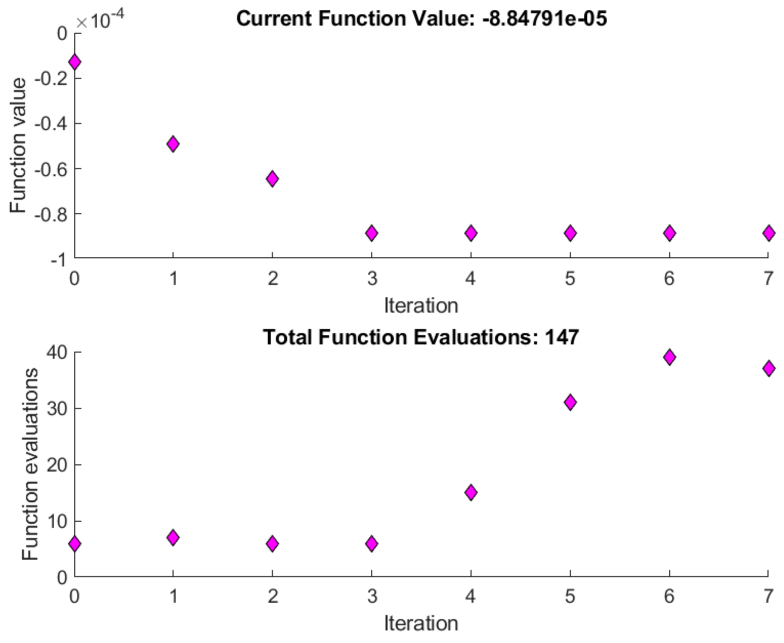

Figure 4: Plot showing the function evaluation at each iteration (top) and the number of function evaluations for the given iteration (bottom) in the case where we do gradient based optimization of the efficiency based on the aforementioned ”good” design vector as the initial guess. The descent towards a better function evaluation seems to begin ”right away” here, whereas there was a ”flatter” region to first be traversed in the case of the arbitrary guess (figure 3).

Using instead the ”good” guess, which for power optimization is given by[2], $[y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}] = [0.05; 0.45; 0.25; 0.15; 0.15]$;, the following output is generated

```
The best solution from 30 optimization runs is power = -0.00002438, corresponding to a
    design vector of:

    0.0500    0.4500    0.2500    0.1500    0.1500



This solution had the following simulation output info:

         iterations: 1
          funcCount: 34
          algorithm: 'sqp'
    constrviolation: 0
           stepsize: 1.1156e-06
       lssteplength: 4.5999e-05
      firstorderopt: 0.0187

Elapsed time is 21144.058804 seconds.
```

[2]Again, the minus sign can be ignored

Interestingly, the arbitrary guess outperforms the qualified one! This would appear strange, but is not at all unbelievable. Our physical model is extremely complicated at an initial guess being stuck at a sub optimal solution is to be expected. In this case, it simply turned out that one initial guess - although in itself a better one - was more prone to getting caught in this sub-optimal ”trap”[3]. This is of course the big danger and shortcoming of gradient based methods and would possibly not have become an issue for a heuristic method. This seems like an even more valuable lesson to have learned.

3.4 Sensitivity Analysis

To apply sensitivity analysis, thanks to the analytical nature of the cost module, we employ the MATLAB symbolic toolbox to derive the effects of each variables, i.e., $\nabla_{h_A} J$, $\nabla_{w_{gap}} J$, $\nabla_{w_{yk}} J$, $\nabla_{h_{yk}} J$, $\nabla_{h_{pm}} J$; and the effects of each constraints (derivatives of the multiplier λ_i), i.e., $\nabla_{\lambda_i} J$ on the cost objective.

```
clc; clear; close all

syms Gadolinium Iron Neodymium
syms h_A w_gap w_yk h_yk h_pm

magnetArea = h_pm*w_gap;
activeMaterialArea = h_A*w_gap; % in reality, this would not be a solid mass
yokeArea = 2*w_yk*h_yk;

J = Gadolinium*activeMaterialArea + Iron*yokeArea + Neodymium*magnetArea;

J_hA = diff(J,h_A);
J_wgap = diff(J,w_gap);
J_wyk = diff(J,w_yk);
J_hyk = diff(J,h_yk);
J_hpm = diff(J,h_pm);

[J_hA; J_wgap; J_wyk; J_hyk; J_hpm]
```

Running the previous code we generate:

```
               Gadolinium*w_gap
Gadolinium*h_A + Neodymium*h_pm
                    2*Iron*h_yk
                    2*Iron*w_yk
```

[3]Actually, the initial guess was pretty much ”inside the trap from the beginning”, which see from it conveging right away (only one iteration)! Perhaps the tolerance could have been tuned to avoid this, but it is likely that the results would have been pretty close to the same

```
                Neodymium*w_gap
```

Therefore we can write out all the effects in analytic forms:

$$\begin{aligned}
\nabla_{w_{yk}} J &= 2\mathbb{I} h_{yk} \quad (\mathtt{E}_1)\\
\nabla_{h_{yk}} J &= 2\mathbb{I} w_{yk} \quad (\mathtt{E}_2)\\
\nabla_{h_{pm}} J &= \mathbb{N} w_{gap} \quad (\mathtt{E}_3)\\
\nabla_{h_A} J &= \mathbb{G} w_{gap} \quad (\mathtt{E}_4)\\
\nabla_{w_{gap}} J &= \mathbb{G} h_A + \mathbb{N} h_{pm} \quad (\mathtt{E}_5)
\end{aligned} \tag{2}$$

To estimate the effect of changing constraints, we can compute the Lagrangian multipliers

$$\begin{aligned}
\lambda_1 &= \frac{\partial J}{\partial \mathcal{C}_1} = \frac{\partial(\mathbb{G}\mathcal{A}_{active} + \mathbb{I}\mathcal{A}_{yoke} + \mathbb{N}\mathcal{A}_{mag})}{\partial(h_{yk}(2w_{yk} + w_{gap}) - V_{max})} \quad (\mathtt{E}_6)\\
\lambda_2 &= \frac{\partial J}{\partial \mathcal{C}_2} = \frac{\partial(\mathbb{G}\mathcal{A}_{active} + \mathbb{I}\mathcal{A}_{yoke} + \mathbb{N}\mathcal{A}_{mag})}{\partial(h_{yk} - L_{max})} \quad (\mathtt{E}_7)\\
\lambda_3 &= \frac{\partial J}{\partial \mathcal{C}_3} = \frac{\partial(\mathbb{G}\mathcal{A}_{active} + \mathbb{I}\mathcal{A}_{yoke} + \mathbb{N}\mathcal{A}_{mag})}{\partial(2w_{yk} + w_{gap} - L_{max})} \quad (\mathtt{E}_8)\\
\lambda_4 &= \frac{\partial J}{\partial \mathcal{C}_4} = \frac{\partial(\mathbb{G}\mathcal{A}_{active} + \mathbb{I}\mathcal{A}_{yoke} + \mathbb{N}\mathcal{A}_{mag})}{\partial(h_A + h_{pm} - h_{yk})} \quad (\mathtt{E}_9)
\end{aligned}$$

We can also do a estimate on the effects on the changing parameters:

$$\begin{aligned}
\frac{\partial J}{\partial \mathbb{G}} &= h_A w_{gap} \quad (\mathtt{E}_{10})\\
\frac{\partial J}{\partial \mathbb{I}} &= 2w_{yk} h_{yk} \quad (\mathtt{E}_{11})\\
\frac{\partial J}{\partial \mathbb{N}} &= h_{pm} w_{gap} \quad (\mathtt{E}_{12})
\end{aligned} \tag{3}$$

Theoretically, to calculate from $\mathtt{E}_6$ to $\mathtt{E}_9$ we need to apply the chain rule, i.e.,

$$\begin{aligned}
\lambda_1 &= \frac{\partial J}{\partial \mathcal{C}_1} = \frac{\partial J}{\partial w_{yk}}\frac{\partial w_{yk}}{\partial \mathcal{C}_1} + \frac{\partial J}{\partial h_{yk}}\frac{\partial h_{yk}}{\partial \mathcal{C}_1} + \frac{\partial J}{\partial w_{gap}}\frac{\partial w_{gap}}{\partial \mathcal{C}_1}\\
\lambda_2 &= \frac{\partial J}{\partial \mathcal{C}_2} = \frac{\partial J}{\partial h_{yk}}\frac{\partial h_{yk}}{\partial \mathcal{C}_2}\\
\lambda_3 &= \frac{\partial J}{\partial \mathcal{C}_3} = \frac{\partial J}{\partial w_{yk}}\frac{\partial w_{yk}}{\partial \mathcal{C}_3} + \frac{\partial J}{\partial w_{gap}}\frac{\partial w_{gap}}{\partial \mathcal{C}_3}\\
\lambda_4 &= \frac{\partial J}{\partial \mathcal{C}_4} = \frac{\partial J}{\partial h_{pm}}\frac{\partial h_{pm}}{\partial \mathcal{C}_4} + \frac{\partial J}{\partial h_{yk}}\frac{\partial h_{yk}}{\partial \mathcal{C}_4} + \frac{\partial J}{\partial h_{\mathcal{A}}}\frac{\partial h_{\mathcal{A}}}{\partial \mathcal{C}_4}
\end{aligned} \tag{4}$$

As shown in Equation (4), calculating the main effects of the four constraints requires complicated mathematical derivations, which are not what we desired. Thence we figured out another way to estimate which constraint is the active one and which are not: the original gradient-based optimization was ran by many times through cancelling each constraints and thence we can deduce the active constraint.

By cancelling $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_4$, the optimization output are

```
The best solution from 3 optimization runs is cost = 0.34027460, corresponding to a design
    vector of:

    0.0010    0.0020    0.0010    0.0010    0.0010


This solution had the following simulation output info:

         iterations: 2
          funcCount: 18
          algorithm: 'sqp'
    constrviolation: 0
           stepsize: 1.4867e-17
       lssteplength: 1
      firstorderopt: 3.7253e-09
       bestfeasible: [1x1 struct]

Elapsed time is 1.025314 seconds.
```

The optimization results are same as our previous results, indicating that both $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_4$, are inactive constraints.

By cancelling $\mathcal{C}_3$, the generated optimization output are

```
The best solution from 3 optimization runs is cost = 0.33731380, corresponding to a design
    vector of:

   1.0e-03 *

    1.0000    1.0000    1.0000    1.0000    1.0000


This solution had the following simulation output info:

         iterations: 2
          funcCount: 18
          algorithm: 'sqp'
    constrviolation: 0
```

```
          stepsize: 7.9967e-18
      lssteplength: 1
     firstorderopt: 7.4506e-09
      bestfeasible: [1x1 struct]

Elapsed time is 1.062669 seconds.
```

This indicate that $\mathcal{C}_3$ is the active constraint, and the action of cancelling $\mathcal{C}_3$ is to relax the active constraint. And we already know the new optimized design and original constraint optimized design are $[0.0010, 0.0020, 0.0010, 0.0010, 0.0010]$, $[0.0010, 0.0010, 0.0010, 0.0010, 0.0010]$, respectively. With the constraint problem figured out, we can continue with the effect (sensitivity analysis) computation for each design variables and parameters.

Based on the theoretical equations {Equation (2) *&* Equation (3)}, we can therefore code all the previous effects ($\mathtt{E}_i$) (numerical) in MATLAB, formulating a new function `sensitivity`:

```
function [Ematrix] = sensitivity(Input)

close all
clc


x = Input;
w_yk = x(1); h_yk = x(2); h_pm = x(3); h_A = x(4); w_gap = x(5);

%% Define the constants

Gadolinium = 1.71553e5; %$/m^3
Iron = 1.4804e3; %$/m^3
Neodymium = 1.628e5; %$/m^3
V_max = .125;  L_max = .5;

%% Effects of the design variables

J_hA =  Gadolinium*w_gap; %E1
J_wgap = Gadolinium*h_A + Neodymium*h_pm; %E2
J_wyk = 2*Iron*h_yk; %E3
J_hyk = 2*Iron*w_yk; %E4
J_hpm = Neodymium*w_gap; %E5

magnetArea = h_pm*w_gap;
activeMaterialArea = h_A*w_gap; % in reality , this  would  not be a solid  mass8
yokeArea = 2*w_yk*h_yk;

J = Gadolinium*activeMaterialArea + Iron*yokeArea + Neodymium*magnetArea;
```

```
%% Constraints

c1 = h_yk*(2*w_yk + w_gap) - V_max;
c2 = h_yk - L_max;
c3 = 2*w_yk + w_gap - L_max;
c4 = h_A + h_pm - h_yk;

%% Effects of parameters

J_G = h_A*w_gap;
J_I = 2*w_yk*h_yk;
J_N = h_pm*w_gap;

% We cancel this part due to the complex nature of the problem
% E6 = diff(J,c1);
% E7 = diff(J,c2);
% E8 = diff(J,c3);
% E9 = diff(J,c4);


%% Main sensitivity analysis: calculate E_i (Effects)

E1 = J_wyk;
E2 = J_hyk;
E3 = J_hpm;
E4 = J_hA;
E5 = J_wgap;
E10 = J_G;
E11 = J_I;
E12 = J_N;

%% Obtain the eventual values
Ematrix = [E1; E2; E3; E4; E5; E10; E11; E12];
categories = categorical({'Yoke Width','Yoke Height','Permanent Magnet Height','Active
    Material Height','Gap Width','c1 = h_yk.*(2.*w_yk + w_gap) - V_max','c2 = h_yk - L_max',
    'c3 = 2.*w_yk + w_gap - L_max','c4 = h_A + h_pm - h_yk','Gadolinium','Iron','Neodymium'
    });
figure
hold on
bar(categories,Ematrix) %visualize the value
title('Main Effect of Design Variables');
figure
hold on
labels = {'Yoke Width','Yoke Height','Permanent Magnet Height','Active Material Height','Gap
     Width','c1 = h_yk.*(2.*w_yk + w_gap) - V_max','c2 = h_yk - L_max','c3 = 2.*w_yk + w_gap
     - L_max','c4 = h_A + h_pm - h_yk','Gadolinium','Iron','Neodymium'};
```

```
pie(Ematrix,labels) %visualize the weight (or effects) of each design vars. --> this one
    makes more sense
title('Main Effect of Design Variables');
end
```

By giving a random initial guess (input design variables vector), and running the command `test = sensitivity([.001; .002; .001; .001; .001]);` we can therefore generate the 8 main effects for 5 design variables[4] $\{y_{wk}, h_{yk}, h_{pm}, h_A, w_{gap}\}$ and 3 parameters $\{\mathbb{G}, \mathbb{I}, \mathbb{N}\}$:

```
E1 =
5.9216
E2 =
2.9608
E3 =
162.8000
E4 =
171.5530
E5 =
334.3530
E10 =
1.0000e-06
E11 =
4.0000e-06
E12 =
1.0000e-06
```

By visualizing the results we could generate Figure 5, from which we could conclude that for this specific initial design variable the Gap Width (w_{gap}) seem to be the driver in this problem. And this results can be said partly satisfy our expectations, since here we only optimize the geometry to minimize cost, hence any main height or width of the TMG body may be contribute largely to the cost optimization.

[4]- at this particular point, it should be noted

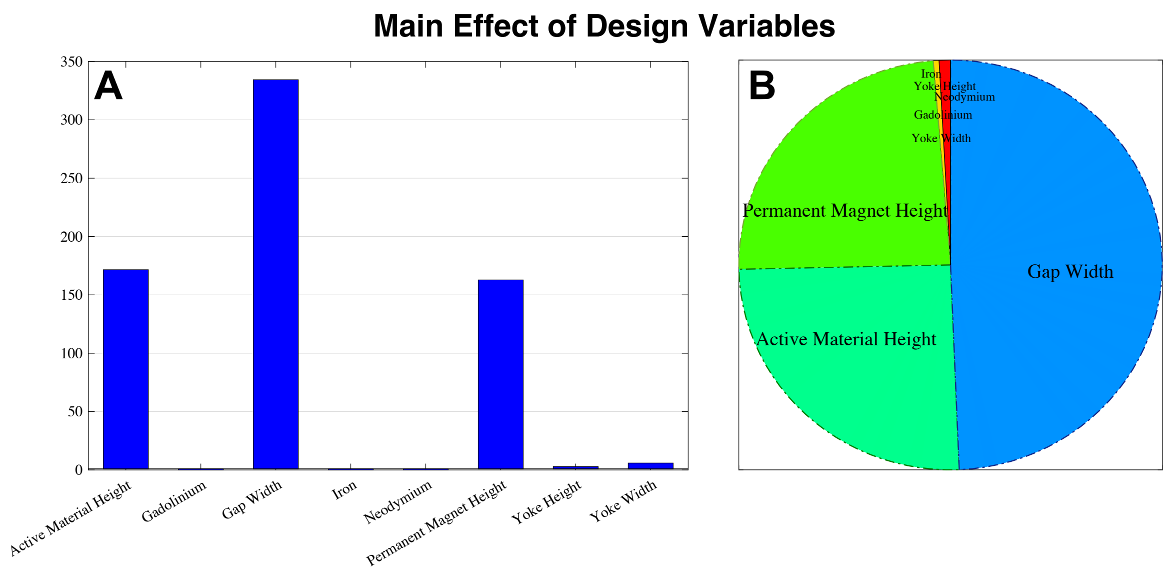

Figure 5: The graph representing each effects for design variables and parameters. Subfigure **A** is the bar plot for effects and **B** is the π plot.

4 Appendix

objee.m: Function returning cost for an input design vector.

```
function f = objee(x)
% x = [h_A; w_gap; w_yk; h_yk; h_pm];

    Gadolinium = 1.71553e5; % $/m^3
    Iron = 1.4804e3; % $/m^3
    Neodymium = 1.628e5; % $/m^3

    w_yk = x(1);
    h_yk = x(2);
    h_pm = x(3);
    h_A = x(4);
    w_gap = x(5);

    magnetArea = h_pm*w_gap;
    activeMaterialArea = h_A*w_gap; % in reality, this would not be a solid mass
    yokeArea = 2*w_yk*h_yk;
    f = Gadolinium*activeMaterialArea + Iron*yokeArea + Neodymium*magnetArea;
end
```

geocon.m: Function handling geometric constraints for the genetic algorithm optimization.

```
function [c,ceq] = geocon(x)

    V_max = .125;
    L_max = .5;

    w_yk = x(1);
    h_yk = x(2);
    h_pm = x(3);
    h_A = x(4);
    w_gap = x(5);

    c(1) = h_yk*(2*w_yk + w_gap) - V_max;
    c(2) = h_yk - L_max;
    c(3) = (2*w_yk + w_gap) - L_max;
    c(4) = (h_A + h_pm) - h_yk;

    ceq = [];
end
```

TherMaG_GA.m: Script setting up and calling the genetic algorithm to optimize cost and return results.

```
clear
clc

% number of times to run the algorithm
numRuns = 500;

% set upper and lower bounds
lb = [.001, .001, .001, .001, .001];
ub = [.5, .5, .5, .5, .5];

% set optimization options
options = optimoptions('ga', ...
        'CrossoverFcn',{@crossoverheuristic,1.6},'Display',...
        'iter', ...
        'FunctionTolerance', 1e-6, ...
        'PopulationSize', 50, ...
        'CrossoverFraction', 0.7,...
        'MaxGenerations', 2000,...
        'ConstraintTolerance', 1e-6,...
        'MutationFcn',{@mutationadaptfeasible});

% initialize vars for storing results
objectives = zeros(numRuns,1);
```

```
dVectors = zeros(numRuns ,5);

% run the GA a few times to find the best result
for i = 1:numRuns

    [solution,objectiveValue] = ga(@objee,5,[],[],[],[],lb,ub,@geocon,[],options);

    objectives(i) = objectiveValue;
    dVectors(i,:) = solution;

end

% round the final objectives to aid judgement of the algorithm consistency
objectives = round(objectives,4);
[min, index] = min(objectives);
count = sum(objectives == min);
avgObjective = mean(objectives);

fprintf('\n');
fprintf('The lowest cost from %d algorithm runs is %.4f, corresponding to a design vector of
    : \n', numRuns, min);
fprintf('\n');
disp(dVectors(index,:));
fprintf('\n');
fprintf('The algorithm found this to be the optimal result %d times out of %d. The average
    optimal value is %.4f\n', count, numRuns, avgObjective);
```

References

[1] Katoch, S., Chauhan, S.S. & Kumar, V. A review on genetic algorithm: past, present, and future. Multimed Tools Appl 80, 8091–8126 (2021).

[2] Tabak, Daniel; Kuo, Benjamin C. (1971). Optimal Control by Mathematical Programming. Englewood Cliffs, NJ: Prentice-Hall. pp. 19–20. ISBN 0-13-638106-5.

[3] fmincon Documentation. Mathworks, Inc. URL: https://www.mathworks.com/help/optim/ug/fmincon.html#busp5fq-7