PERSONAL NOTES

Design Optimization

Hanfeng Zhai

2021

MAE 5350: HW #1

Hanfeng Zhai[*]

Multidisciplinary Design Optimization

September 20, 2021

Part (a)

Q1. Motivation

Summarize in 5-10 sentences why you decided to take this class. What do you expect to learn? How does this knowledge fit in with your career or research plans?

- Summarize in 5-10 sentences why you decided to take this class. This class attracts me mainly because its name: *Multidisciplinary Design Optimization*. For *Multidisciplinary*: I was involved in many projects are in the intersections of many disciplines: thermal engineering, structural engineering, chemical engineering, bioengineering, etc., and I'm a huge fan of multiphysics system simulation. Usually we solve each modules separately and couple them from specific variables that occurred in both the governing equations. So I'm curious whether the same strategy applies for the *Multidisciplinary* here. Second, I'm excited about the term *Optimization*: data-driven, quasi-static, & physics-informed machine learning methods are extremely powerful and popular terms in my field, and started to be adopted by both academia and the industry. All of these methods are based on different optimization methods to "train" a learning structure. Therefore, the know more details about the optimization methods is the main attraction for me.
- What do you expect to learn? Basic optimization methods that can be applied to many engineering & scientific fields. Currently, MDO already proved its effectiveness in spacecraft engineering, underwater vehicles, aerofoil design, wind turbine system etc. and became very successful in the industry. I believe it can also be applied to micro&nano system, circuits, batteries, lab-on-chip, and many "small" systems!
- How does this knowledge fit in with your career or research plans? I hope to apply what I've learnt in MDO course to my current research of multiscale mechanics, and many potential field of interests

*[redacted] www.hanfengzhai.net
Sibley School of Mechanical and Aerospace Engineering, Cornell University

including microfluidics, lab-on-chip system, batteries, etc. Also, I believe I got more chances to be involved in many industrial projects to view them from the perspective of MDO.

Q2. System decomposition

In Sections 1.1 and 1.2 of *Papalambros and Wilde*, the authors discuss hierarchical levels in system definition and hierarchical system decomposition. To answer the questions below, consider one of the following engineering systems: a wind turbine power system, an Unmanned Aerial Vehicle (UAV), or an underwater vehicle: Woods Hall Oceanographic Institute's Alvin.

1) Describe the system boundary that you would choose in setting up a model for your system. What are the inputs and outputs that cross this system boundary and characterize your system?

- System Chosen: Wind turbine power system.
- System Boundary: The constraints in this system includes ^{1}Basic rule of physics (i.e. Navier-Stokes equation for fluid dynamics, thermodynamics, equilibrium of forces applied to structures), ^{2}Boundaries of local area for setting up the turbines, ^{3}budget, ^{4}the maximum energy that area can support, ^{5}the height limit based on the turbine structure, etc.
- Inputs: The inputs of the system includes ^{1}the number of turbines, ^{2}the geometrical parameters of wind turbine (i.e., volume / size, height, occupation area, etc.), ^{3}The shape of the blade (i.e., angles, area shapes, etc.), ^{4}the materials for turbine, ^{5}the average distance between turbines.
- Outputs: The outputs of the system includes ^{1}the power generation per turbine, ^{2}stress acting on the blade, ^{3}the deformation of the turbine body, ^{4}costs per turbine, ^{5}vibration caused by the turbine working.

2) Propose a component decomposition for your system (use a similar level of detail to that shown in *Papalambros and Wilde* Fig. 1.10).

See Figure 1.

3) Propose an aspect decomposition for your system (use a similar level of detail to that shown in *Papalambros and Wilde* Fig. 1.12).

See Figure 2.

Q3. Simple Unconstrained Optimization/Math Review

When solving the following problem you may use computational tools, but please apply the methods taught in class before doing any computation and show your derivations.

The Rosenbrock function is often used as a simple test problem for optimization algorithms:

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

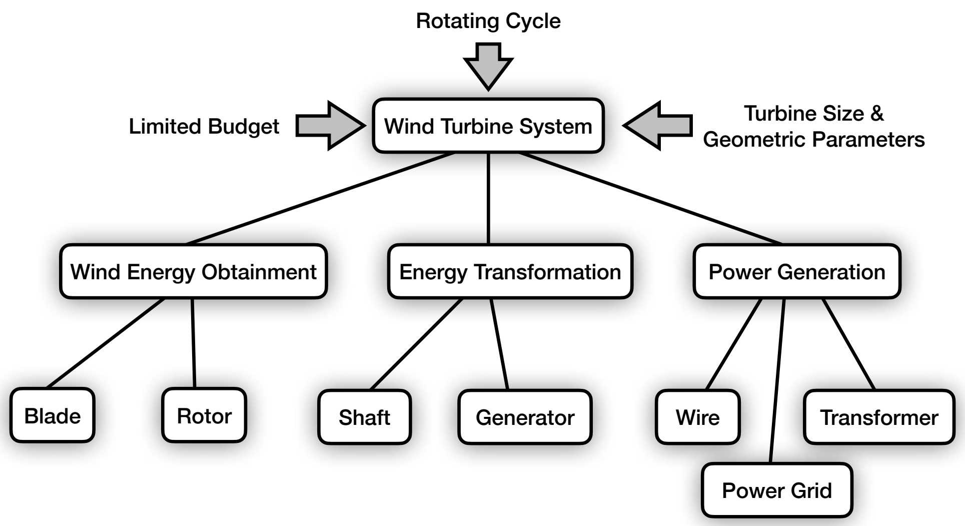


Figure 1: Component decomposition for the wind turbine system (*solution for (2)*).

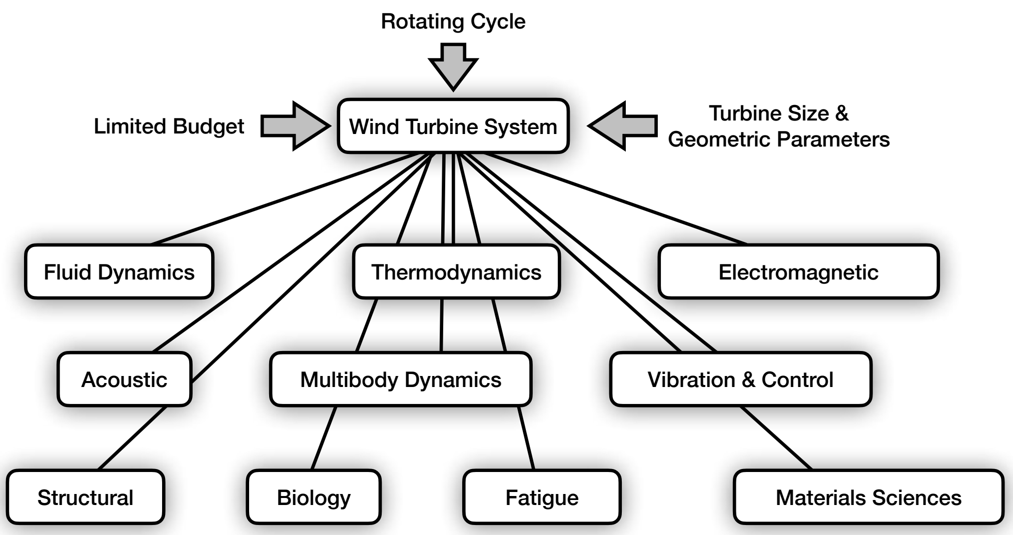


Figure 2: Aspect decomposition for the wind turbine system (*solution for (3)*).

a) Compute the gradient (vector of first derivatives) and Hessian (matrix of second derivatives) of $f(\mathbf{x})$.

Gradient:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} -400x_1x_2 + 400x_1^3 + 2x_1 - 2 \\ 200x_2 - 200x_1^2 \end{bmatrix}$$

Hessian:

$$\mathbf{H}f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} -400x_2 + 1200x_1^2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}$$

b) Show that $\mathbf{x}^* = (1,1)$ is the only local minimizer of this function, and that the Hessian matrix at that point is positive definite.

If we expand the form, we get $f(\mathbf{x}) = 100x_2^2 - 200x_1^2x_2 + 100x_1^4 + 1 - 2x_1 + x_1^2$ If we let $f_{x_1} = 0$ and $f_{x_2} = 0$, we get

$$\begin{cases} -400x_1x_2 + 400x_1^3 + 2x_1 - 2 = 0 \\ 200x_2 - 200x_1^2 = 0 \end{cases}$$

Solving the equation we obtain $x_1 = 1$ and $x_2 = 1$ (because from second row we have $x_2 - x_1^2 = 0$ and from the first row implies both $x_2 - x_1^2 = 0$ and $x_1 - 1 = 0$), thus we can say that $\mathbf{x}^*$ only critical (stationary) point.

Here we obtain the critical point $(1,1)$. Substitute this point into the Hessian $\mathbf{H}f(1,1)$ we get $\begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$. Now we can observe that $D_1 = f_{xx} = 802 > 0$ and $D_2 = \det \mathbf{H} = 400 > 0$, therefore the point $(1,1)$ is a local minimum.

c) Make a contour plot of the objective value of the Rosenbrock function versus the design variables x_1 and x_2 and verify the local minimum graphically.

See Figure 3.

d) Show that the function

$$f(\mathbf{x}) = 8x_1 + 12x_2 + x_1^2 - 2x_2^2$$

has only one stationary point, and that it is neither a maximum or minimum, but a saddle point.

We first need to compute the gradient of the function

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 8 + 2x_1 \\ 12 - 4x_2 \end{bmatrix}$$

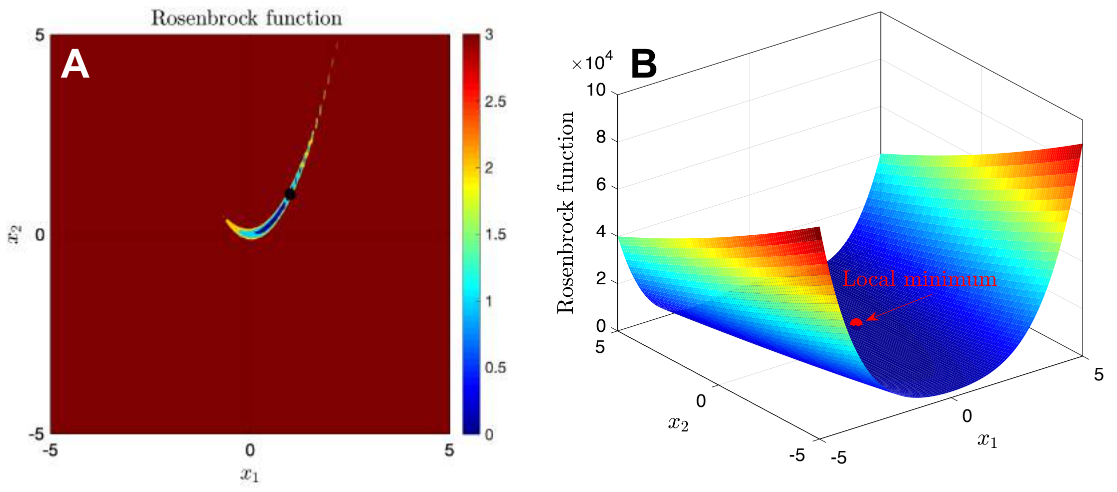


Figure 3: Rosenbrock function contour plot with regards to the coordinates in 2D and 3D. The local minimum point was plotted black in the left sub figure and plotted red in the right sub figure.

Computing the Hessian of the matrix we obtain

$$\mathbf{H}f(\mathbf{x}) = \begin{bmatrix} 2 & 0 \\ 0 & -4 \end{bmatrix}$$

Let the gradient equals zero we obtain $x_1 = -4$ and $x_2 = 3$, from $\mathbf{H}$ we know that $x_1 = -4$ is the local maximum and $x_2 = 3$ is the local minimum. Computing the determinant of matrix we have $\det \mathbf{H} = -8 < 0$, therefore we say $(-4,\ 3)$ is a saddle point.

To verify this point, we substitute these two points and plot them we obtain figure 4.

From figure 4 we observe that the critical point is the lowest in the x_1 direction and highest for x_2 direction, points verified.

Q4. ***Papalambros and Wilde***

When solving the following problem you may use computational tools, but please apply the methods taught in class before doing any computation and show your derivations.

a) Sketch the function $f(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$ in the interval $-2 < x_i < 2$. Solve for the optimal point $(\mathbf{x}^*)$ and add the point to your plot. Does the result match your intuition?

Let the gradient of the function equals zero we obtain

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 8x_2 + 4(x_1 - x_2)^3 - 1 \\ 8x_1 - 4(x_1 - x_2)^3 + 1 \end{bmatrix} = 0 \tag{1}$$

We can then derive the numerical solution with MATLAB `vpasolve` module, generating the following code:

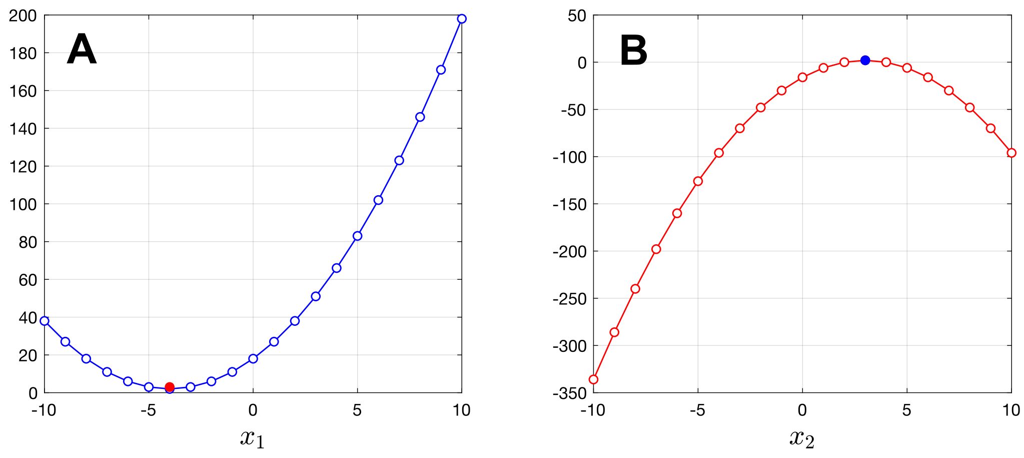


Figure 4: The plot of $f(\mathbf{x})$ on fixed $x_2 = 3$ and $x_1 = -4$.

```
>> syms x1 x2
>> f = @(x1,x2) (x2-x1)^4+8*x1*x2-x1+x2+3;
>> eq1 = diff(f,x1)
    eq1 = 8*x2 + 4*(x1 - x2)^3 - 1
>> eq2 = diff(f,x2)
    eq2 = 8*x1 - 4*(x1 - x2)^3 + 1
% according to equation (1), we know that eq1 - eq2 = 0, of which we derive that x1 =
    -x2, then substitute into Eq. (1)
>> eqn = 8*(-x1) + 4*(x1 + x1)^3 - 1
>> fpasolve(eqn, x1)
ans =

-0.13479721820272227913146897567463
 0.55357993584438379685387442521215
-0.41878271764166151772240544953752
% then we can get the three points' coordinate based on x1 = -x2
```

The function and the three points is sketched as in figure 5. Based on the subfigure **B** and **C** we can see that the optimal point is (0.55357993584438379685387442521215, −0.55357993584438379685387442521215).

b) Sketch $f(x)$ in the same interval $-2 < x_i < 2$ but additionally sketch the constraint $g(\mathbf{x}) = x_1^4 - 2x_2x_1^2 + x_2^2 + x_1^2 - 2x_1 \leq 0$. Solve for the the optimal point ($\mathbf{x}^*$) given this new constraint and add the point to your plot. Is it different from the result you got in part a)? Why? Hint: is $g(x)$ an active or inactive constraint? How can we use this information to help solve the new optimization problem?

As we can observe from figure 6 (especially from subfigure **C**), the inequality constraint $g(\mathbf{x}) < 0$ doesn't include any area where $f(\mathbf{x})$ locate. Therefore, the critical point remains the same, and the constraint is an inactive constraint.

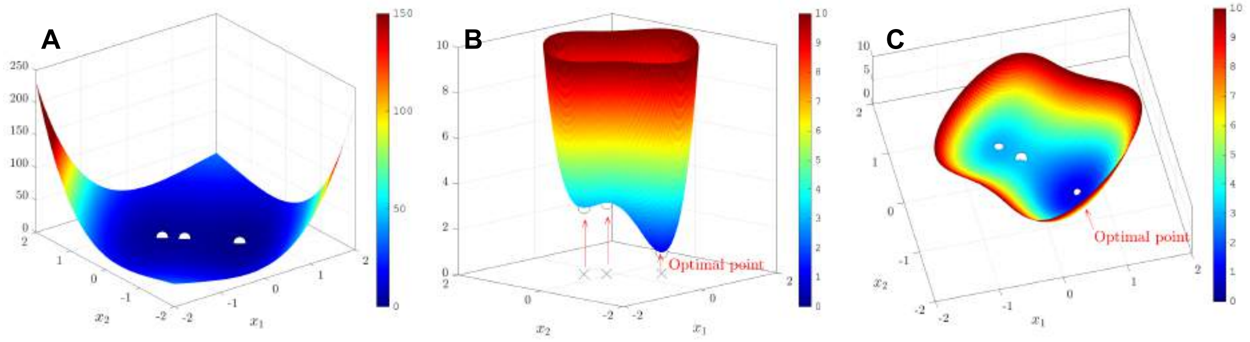


Figure 5: The sketch of function $f(\mathbf{x})$, the three solution points for $\nabla f = 0$, with highlighting the optimal point, in different numerical ranges. **A**. the sketch in the range of $[0, 150]$. **B**. the sketch in the range pf $[0, 10]$. **C**. a different angular view for subfigure **B**.

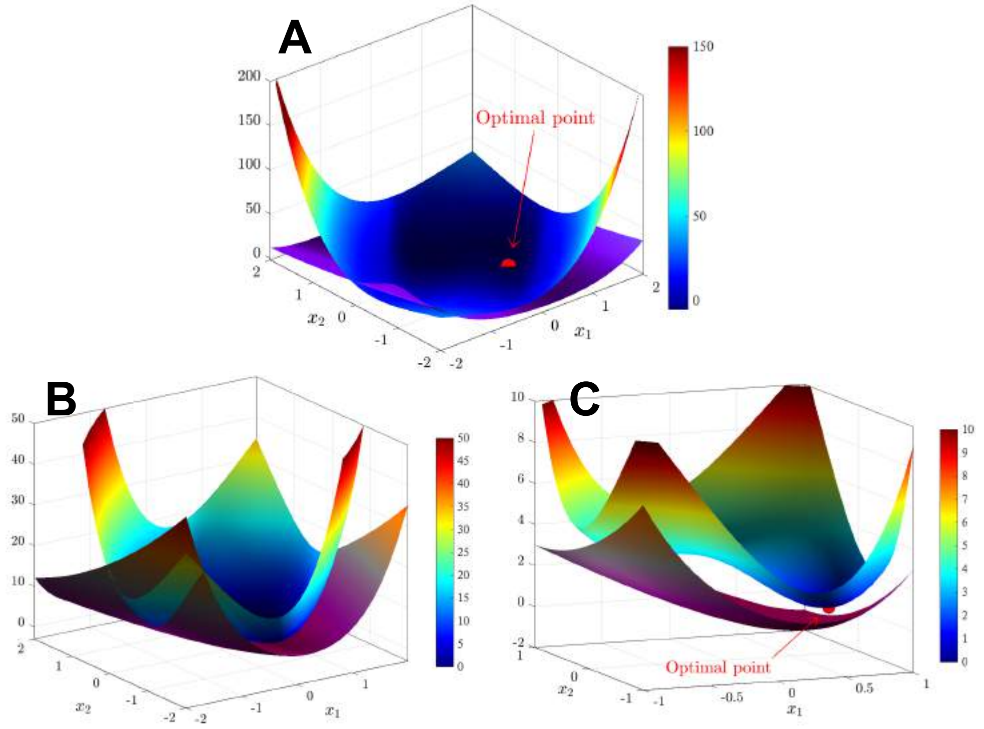


Figure 6: The sketch of the objective function $f(\mathbf{x})$ and inequality constraint $g(\mathbf{x})$ from different angles and different numerical ranges. **A**. the sketch in the range of $[0, 150]$ as can be compared with Figure 5, with the optimal point. **B**. the sketch in $[0, 50]$. **C**. the sketch in $[0, 10]$, highlighting the optimal point.

MAE 5350: HW #2

Multidisciplinary Design Optimization

Hanfeng Zhai

hz253@cornell.edu

Sibley School of Mechanical and Aerospace Engineering, Cornell University

October 9, 2021

Q1. Design of experiments

Recall the airplane design experiment we did in Lecture 4. You can download the results from Canvas. (*The code for Q1 can be downloaded through* https://hanfengzhai.net/data/MDO_A2_Q1.mlx)

(a) Calculate the mean and variance for each experiment (9 experiments).

SOLUTION: The mean value is calculated through

$$\overline{\mathcal{D}}_i = \frac{\sum_i^N \mathcal{D}_i}{N}$$

where $\mathcal{D}$ stands for the distance and N stands for the total numbers of attempts, i is the experiments numbers. Based on such we can calculate the following mean for the nine experiments in Table 1.

The variance is calculated through the absolute minus between the mean values per experiments to the overall mean value, written as $\mathcal{V}_i$:

$$\mathcal{V}_i = \frac{\sum_{i=1}^N (x_i - \overline{x_i})^2}{N}$$

The variance for the nine experiments in Table 2.

(b) Calculate the effect of each design variable setting (12 effects).

$\overline{\mathcal{D}}_1$	$\overline{\mathcal{D}}_2$	$\overline{\mathcal{D}}_3$	$\overline{\mathcal{D}}_4$	$\overline{\mathcal{D}}_5$	$\overline{\mathcal{D}}_6$	$\overline{\mathcal{D}}_7$	$\overline{\mathcal{D}}_8$	$\overline{\mathcal{D}}_9$
19.415	16.43	10.94	15.138	17.207	18.438	21.955	16	14.855

Table 1: The mean value for the nine experiments.

$\overline{\mathcal{V}}_1$	$\overline{\mathcal{V}}_2$	$\overline{\mathcal{V}}_3$	$\overline{\mathcal{V}}_4$	$\overline{\mathcal{V}}_5$	$\overline{\mathcal{V}}_6$	$\overline{\mathcal{V}}_7$	$\overline{\mathcal{V}}_8$	$\overline{\mathcal{V}}_9$
3.2547	22.8058	2.7175	19.0957	45.8479	157.8374	0.0156	7.1289	0.7731

Table 2: The variance for the nine experiments.

$\overline{\mathcal{M}}_{A_1}$	$\overline{\mathcal{M}}_{A_2}$	$\overline{\mathcal{M}}_{A_3}$	$\overline{\mathcal{M}}_{B_1}$	$\overline{\mathcal{M}}_{B_2}$	$\overline{\mathcal{M}}_{B_3}$
-0.9944	-0.0366	0.2075	0.9913	-0.0912	-1.9645
$\overline{\mathcal{M}}_{C_1}$	$\overline{\mathcal{M}}_{C_2}$	$\overline{\mathcal{M}}_{C_3}$	$\overline{\mathcal{M}}_{D_1}$	$\overline{\mathcal{M}}_{D_2}$	$\overline{\mathcal{M}}_{D_3}$
1.6323	-1.1568	-1.0587	0.4505	1.3113	-2.8262

Table 3: The main effect for the twelve variables.

SOLUTION: Calculating the main effect of A_i, named as $\mathcal{M}_{A_i}$:

$$\mathcal{M}_{A_i} = \overline{\mathcal{D}_{(A_i)}} - \overline{\sum_i^N \overline{\mathcal{D}}_i}$$

With such an equation we compute the effects for 12 variables as in Table 3.

(c) What are the design variable settings of the predicted "optimal" airplane?

SOLUTION: From Table 3 comparing $\mathcal{M}_{A_i}$ we can deduce that the optimal design is A_3. Similarly, observing $\mathcal{M}_{B_i}$ we can deduce that the optimal design is B_1; observing $\mathcal{M}_{C_i}$ we can deduce that the optimal design is C_1. And similarly we have D_2 for optimal design.

(d) Assuming that the effects can be added linearly, estimate the range of the predicted optimal airplane.

SOLUTION:

Since the main effects can be added linearly, then the optimal design is $\{A_3,\ B_1,\ C_1,\ D_2\}$. We first calculate the sum main effect by adding the main effects:

$$\sum \overline{\mathcal{M}} = \overline{\mathcal{M}}_{A_3} + \overline{\mathcal{M}}_{B_1} + \overline{\mathcal{M}}_{C_1} + \overline{\mathcal{M}}_{D_2} = 4.1424$$

Then we can calculate the range by adding such to the overall mean

$$\sum \mathcal{M} + \overline{\mathcal{D}} = 20.4963$$

(e) Download the airplane template from Canvas, build the predicted optimal airplane, and fly it 5 times*. Report your results and discuss similarities/differences between the predicted and actual performance.

SOLUTION:

Expts. #	1	2	3	4	5
Dist. (ft)	29.8	25.3	25.9	18.2	20.1

Expts. # / Factor	A	B	C	D
1	A_3	B_1	C_1	D_2
2	A_2	B_1	C_1	D_2
3	A_1	B_1	C_1	D_2
4	A_3	B_2	C_1	D_2
5	A_3	B_3	C_1	D_2
6	A_3	B_1	C_2	D_2
7	A_3	B_1	C_3	D_2
8	A_3	B_1	C_1	D_1
9	A_3	B_1	C_1	D_3

Table 4: The experiment matrix for parameter study.

Most of the experimental distance was larger than the predicted value, three reasons might be accounted for this "inaccuracy": (1) I got better throwing skills (doubt about it); (2) There are errors for my measurement; (3) Errors of measurement by other students.

(f) If we were now to perform a parameter study using the predicted optimal airplane as the baseline, what would the experimental matrix be? Comment on what, if any, new information this new experiment might bring.

Solution:

Compared with the original design table, this table could present us information on when three parameters are fixed at the optimal design, whether the changing of the other parameters may variate the results. It also give us numerous experiments that the original table does not cover. So we may be able to see how each factors variate the final distance more specifically through different experiments.

(g) In one paragraph, discuss the variance results computed in part (a). Explain the possible significance of these results and how they might be used to inform an actual design process.

Solution: In probability theory and statistics, variance is the expectation of the squared deviation of a random variable from its population mean or sample mean. Variance is a measure of dispersion, meaning it is a measure of how far a set of numbers is spread out from their average value [Wikipedia]. Based on such definition, we can deduce that the bigger the variant the higher error of the experiments. Or in other words, when we are approaching the more "accurate" experimental results we should expect to see a smaller variance. Here, from Table 2 we can see that experiments # 2, 4, 5, 6 have a general larger variance, especially for Exp. # 6. Here we can say that the reliability of these experiments are lower than the rest.

Q2. Gradient-Based Optimization: optimal can sizing problem

Consider the simple constrained optimization problem of minimizing the surface area of a cylinder subject to an equality constraint on its volume:

$$\min_{x} f(x_1, x_2) = 2\pi x_1(x_1 + x_2)$$
$$\text{subject to} \quad h(x_1, x_2) = \pi x_1^2 x_2 - V = 0$$

where x_1 is the radius of the cylinder, x_2 is the height of the cylinder, and V is the required volume.

(a) Formulate the Lagrangian function, derive the optimality conditions, and solve the resulting system of equations to determine the dimensions of the minimum-surface-area cylinder that has a volume of 1 liter (1000 cm^3). (We know $V = 0.001 m^3$ (SI Unit))

SOLUTION: According to the definition of a Lagrangian function:

$$L(\mathbf{x}, \lambda) = J(\mathbf{x}) + \sum_{j=1}^{m_1} \lambda_i g_j(\mathbf{x}) + \sum_{k=1}^{m_2} \lambda_{m_1+k} h_k(\mathbf{x})$$

Substituting the given condition to the function L we have:

$$L(\mathbf{x}, \lambda) = 2\pi x_1(x_1 + x_2) + \lambda(\pi x_1^2 x_2 - V) \tag{1}$$

By solving $\nabla_{x_1,x_2,\lambda} L(\mathbf{x}, \lambda) = 0$, we have:

$$\left(\frac{\partial L}{\partial x_1}, \frac{\partial L}{\partial x_2}, \frac{\partial L}{\partial \lambda}\right) = 0 \Longleftrightarrow \begin{cases} 2\pi x_1 + 2\pi(x_1 + x_2) + 2\pi\lambda x_1 x_2 = 0 \\ \lambda\pi x_1^2 + 2\pi x_1 = 0 \\ \pi x_2 x_1^2 - V = 0 \end{cases} \tag{2}$$

Solving the above equations, applying the `vpasolve` function we have[1]:

$$x_1 = 0.054192607013928900874456136482964$$
$$x_2 = 0.10838521402785780174891227296593$$
$$\lambda = -36.905402972880568381936077759178$$

NOTE THAT THE UNIT IN THIS COMPUTATION IS IN **SI UNIT** (*The answer hasn't change since my last version but I switched all the answers to SI Unit to make it more clear and avoid confusion*) I switched the answer from the original cm to m is because Jiayi asked how did I solve the problem and

[1]Codes can be downloaded through https://hanfengzhai.net/data/MDO_A2_Q2.mlx

I explained it to him, and he get confused about my unit, so here I switch all the unit to SI Unit to avoid similar issues.

(b) Repeat the optimization for a constrained volume of 12 US fl oz (355 ml) and compare your dimensions against the standard U.S. beverage can (https://en.wikipedia.org/wiki/Beverage_can) and explain any differences.

SOLUTION: For this problem, an inequality constraint was added as $V \leq 355 \times 10^{-6}[m^3]$, which is written as $V - 0.355 = 0$ in the standard form. Hence the Lagrangian (Eq. 1) should be rewritten in the form:

$$\min_x f(x_1, x_2) = 2\pi x_1(x_1 + x_2)$$

$$\text{subject to} \quad h(x_1, x_2) = \pi x_1^2 x_2 = V, \text{ where V} = 355 \times 10^{-6}$$

$$\rightarrow L(\mathbf{x}, \lambda) = 2\pi x_1(x_1 + x_2) + \lambda(\pi x_1^2 x_2 - 355 \times 10^{-6})$$

With the given setup, we write out the Karush-Kuhn-Tucker (KKT) Conditions:

$$\left(\frac{\partial L}{\partial x_1}, \frac{\partial L}{\partial x_2}, \frac{\partial L}{\partial \lambda}\right) = 0 \Longleftrightarrow \begin{cases} 2\pi x_1 + 2\pi(x_1 + x_2) + 2\pi\lambda x_1 x_2 = 0 \\ \lambda\pi x_1^2 + 2\pi x_1 = 0 \\ \pi x_2 x_1^2 - V = 0 \end{cases} \tag{3}$$

By solving Eq. 3 with `vpasolve` we have:

$$x_1 = 0.038372152480156730624817304791733$$

$$x_2 = 0.076744304960313461249634609583465$$

$$\lambda = -52.121131360411789506046960254401$$

According to the provided reference, *The US standard can is 4.83 in or 12.3 cm high, 2.13 in or 5.41 cm in diameter at the lid, and 2.6 in or 6.60 cm in diameter at the widest point of the body.* So here our optimized results can be identified as a "shorter" and "fatter" version compared with the standard soda can.

(c) Formulate the can optimization problem as a geometric program in standard form.

SOLUTION: Write such a problem into geometric form:

$$\min_x 2\pi x_1^2 + x_1 x_2$$

$$\text{subject to} \quad 2.8169 \times 10^3 \pi x_1^2 x_2 = 1$$

(d) Use Python/MATLAB/R/Julia to independently solve the problem numerically using either the cvxpy library for Python, the CVX library for MATLAB, CVXR for R or Convex.jl in Julia*. Does the solution agree with your analytical solution found in part (a)?

SOLUTION: We construct the following code based on the course materials:

```
cvx_begin gp
    variables x1 x2 lambda
    minimize( 2*pi*x1*(x1 + x2) )
    subject to
        pi*x1*x2 == 355e-6
cvx_end
```

And we obtain the following results:

$$x_1 = 8.4782 \times 10^{-21}$$
$$x_2 = 1.3328 \times 10^{16}$$
$$\lambda = 1$$

Which is very different from what we obtain in (a) with gradient based methods.

Q3. Gradient-Based Optimization: Newton's method

Consider the function

$$f(x_1, x_2) = \frac{1}{2}(x_1^2 - x_2)^2 + \frac{1}{2}(1 - x_1)^2$$

(a) At what point does f attain a minimum?

SOLUTION: Solving $\nabla_{x_1} f = 0$ & $\nabla_{x_2} f = 0$, we have $x_1 = x_2 = 1$.

(b) Perform (by hand) one iteration of Newton's method using the starting point:$x_1 = 2, x_2 = 2$.

SOLUTION: Based on Newton's method, the search direction can be computed as

$$\mathbf{S} = -[\mathbf{H}(x^0)]^{-1} \nabla J(x^0)$$

Here, $J = f(\mathbf{x})$, and $x^0 = (2, 2)$. So we first compute the gradient and Hessian:

$$\nabla J = \begin{bmatrix} x_1 - 2x_1(-x_1^2 + x_2) - 1 \\ -x_1^2 + x_2 \end{bmatrix} \text{ \& } \mathbf{H} = \begin{bmatrix} 6x_1^2 - 2x_2 + 1 & -2x_1 \\ -2x_1 & 1 \end{bmatrix}$$

For the first step, substituting the first coordinate into the direction $\mathbf{S}$ we have

$$\mathbf{S}(2, 2) = -[\mathbf{H}(2, 2)]^{-1} \nabla J(2, 2) = [-0.2, 1.2]^T$$

Q4. Gradient-Based Optimization: constraint qualifications

Consider the problem

$$\min_x f(x_1, x_2) = x_1^2 + x_2^2$$

$$\text{subject to } h(x_1, x_2) = x_2^2 - (x_1 - 1)^3 = 0$$

(a) Formulate the Lagrangian function and derive the KKT optimality conditions. Can you solve the resulting system of equations to determine the optimal solution? Explain why this method might fail for this problem.

Solution: We first formulate the Lagrangian function:

$$L(\mathbf{x}, \lambda) = J(\mathbf{x}) + \sum_{j=1}^{m_1} \lambda_i g_j(\mathbf{x}) + \sum_{k=1}^{m_2} \lambda_{m_1+k} h_k(\mathbf{x})$$

Substitute $f = J$ and the equality constraint h we have:

$$L(\mathbf{x}, \lambda) = x_1^2 + x_2^2 + \lambda \left(x_2^2 - (x_1 - 1)^3 \right)$$

By solving the Lagrangian we have

$$\left(\frac{\partial L}{\partial x_1}, \frac{\partial L}{\partial x_2}, \frac{\partial L}{\partial \lambda} \right) = 0 \Longleftrightarrow \begin{cases} 2\,x_1 - 3\,\lambda\,(x_1 - 1)^2 = 0 \\ 2\,x_2 + 2\,\lambda\,x_2 = 0 \\ {x_2}^2 - (x_1 - 1)^3 = 0 \end{cases} \tag{4}$$

We then derive the KKT condition: if there exists a feasible optimal point $\mathbf{x}^*$,

$$\lambda_i g_j(\mathbf{x}^*) = 0,\ j = 1, ..., m_i, \lambda_i \geq 0$$

$$\begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} + \lambda_i \begin{bmatrix} -3\,(x_1 - 1)^2 \\ 2\,x_2 \end{bmatrix} = 0$$

Try solving this equation with MATLAB Built-in function `vpasolve` we generate the following codes:

```
>> syms x1 x2 lambda
>> L = x1^2 + x2^2 + lambda*(x2^2 - (x1 - 1)^3);lagran = [diff(L,x1); diff(L,x2); diff(L,lambda)]
>> eq1 = lagran(1); eq2 = lagran(2); eq3 = lagran(3)
>> [x1sol , x2sol , lambdasol] = vpasolve([eq1,eq2,eq3],[x1,x2,lambda])
x1sol =
0.66666666666666666666666666666667 - 0.74535599249992989880305788957709i
0.66666666666666666666666666666667 + 0.74535599249992989880305788957709i
0.66666666666666666666666666666667 + 0.74535599249992989880305788957709i
```

```
0.66666666666666666666666666666667 - 0.74535599249992989880305788957709i

x2sol =
- 0.72898887936316829444036282580464 - 0.11360575564930422273334997589706i
- 0.72898887936316829444036282580464 + 0.11360575564930422273334997589706i
  0.72898887936316829444036282580464 - 0.11360575564930422273334997589706i
  0.72898887936316829444036282580464 + 0.11360575564930422273334997589706i

lambdasol =
-1.0
-1.0
-1.0
-1.0
```

From the results we can then deduce that the system cannot be solved. Based on the *Nunemacher, 2003* paper we know that the reason why the Lagrangian multiplier fails when the constraints' geometry is not smooth or the critical point is neglected (”*But to be assured that the method succeeds, we must know that the geometry is right—that is, the set defined by $g(x, y) = k$ is a smooth curve in the plane*”)

(b) Solve this problem using the exterior penalty method discussed in L8, using a quadratic penalty function. (Hint: derive an expression for the solution as a function of the penalty parameter ρ, and then take the limit as $\rho \to +\infty$)

Solution: we first formulate a pseudo-objective Φ_Q with the Quadratic Penalty Function for exterior penalty method:

$$\Phi_Q(\mathbf{x}, \rho_p) = x_1^2 + x_2^2 + \rho_p \left(x_2^2 - (x_1 - 1)^3\right)^2$$

We then first calculate the gradient of the pseudo-objective function:

$$\nabla\Phi_Q = \begin{pmatrix} 2\,x_1 + 6\,\rho_p \left({\left(x_1-1\right)}^3 - {x_2}^2\right){\left(x_1-1\right)}^2 \\ 2\,x_2 - 4\,\rho_p\,x_2 \left({\left(x_1-1\right)}^3 - {x_2}^2\right) \end{pmatrix}$$

and also the Hessian:

$$\mathbf{H}\Phi_Q =$$
$$\begin{pmatrix} 18\,\rho_p\,{\left(x_1-1\right)}^4 + 6\,\rho\left(2\,x_1-2\right)\left({\left(x_1-1\right)}^3 - {x_2}^2\right) + 2 & -12\,\rho_p\,x_2\,{\left(x_1-1\right)}^2 \\ -12\,\rho_p\,x_2\,{\left(x_1-1\right)}^2 & 8\,\rho_p\,{x_2}^2 - 4\,\rho_p\left({\left(x_1-1\right)}^3 - {x_2}^2\right) + 2 \end{pmatrix}$$

Here, by analyzing the second term of the gradient $\nabla\Phi_Q$ we can rewrite it into the form:

$$2x_2\left(1 - 4\rho_p((x_1-1)^3 - x_2^2)\right)$$

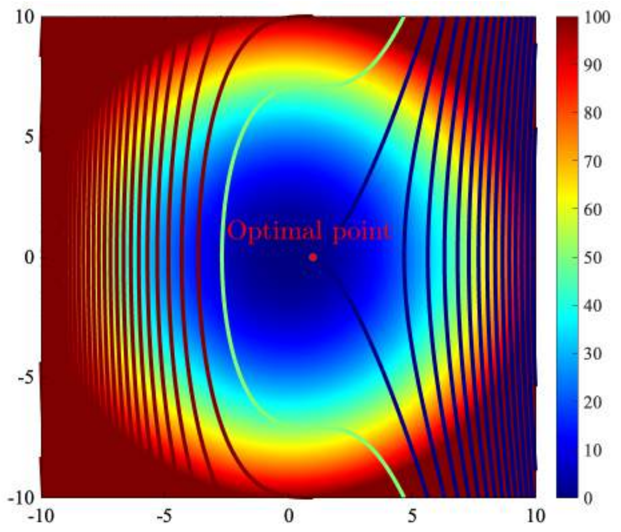


Figure 1: The sketch for the objective and constraint.

And we can deduce that if we let $x_2 = 0$ agrees for $\nabla\Phi_Q = 0$; therefore the first term of the gradient can be written as $2x_1 + 6\rho_p(x_1 - 1)^5 = 0$, taking $\rho_p \to \infty$ we get $x_1 = 1$.

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

(c) Plot the contours of the objective function and the equality constraint. Explain how the plot relates to your answers in (a) and (b).

The codes for running the program are shown as belows:

```
x1 = -10:0.1:10;x2 = x1;
[x1,x2] = meshgrid(x1,x2);
f =x1.^2 + x2.^2;
h  = x2.^2 - (x1 - 1).^3;
x1 = -10:0.1:10;x2 = x1;
contour(x1,x2,f);hold on
contour(x1,x2,h)
```

The figure containing the objective and the constraint are shown as in the following figure, where our estimated optimal point are plotted in the red dot.

Appendix. Data & Code

Code for Q1

The following code used for Q1 is written in MATLAB R2021a.

```
>> %% --------------Code for Q1 - (a)--------------
>> %% Calculating the mean value
>> D_1 = mean(data(1,2:5)); D_2 = mean(data(2,2:7)); D_3 = mean(data(3,2:5)); D_4 = mean(data(4,2:7));
    D_5 = mean(data(5,2:5)); D_6 = mean(data(6,2:5)); D_7 = mean(data(7,2:3)); D_8 = mean(data(8,2:3));
     D_9 = mean(data(9,2:5));
>> D_matrix = [D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9]
D_matrix =
       19.415   16.43   10.94   15.138  17.207  18.438   21.955   16  14.855
>> % Calculate the overall mean value
>> D_overall = mean(D_matrix)
D_overall =
       16.709
>> %% Calculate the variance
>> V_1 = D_1 - D_overall; V_2 = D_2 - D_overall; V_3 = D_3 - D_overall; V_4 = D_4 - D_overall; V_5 =
    D_5 - D_overall; V_6 = D_6 - D_overall; V_7 = D_7 - D_overall; V_8 = D_8 - D_overall; V_9 = D_9 -
    D_overall;
>> V_matrix = [V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9]
V_matrix =
       2.7063   -0.2787  -5.7687  -1.5704 0.4988  1.7288  5.2463   -0.7087  -1.8537
```

```
>> %% --------------Code for Q1 - (b)--------------
>> M_A_1 = mean([D_1,D_2,D_3]) - D_overall; M_A_2 = mean([D_4,D_5,D_6]) - D_overall; M_A_3 = mean([D_7,
    D_8,D_9]) - D_overall;
>> M_A = [M_A_1, M_A_2, M_A_3];
>> M_B_1 = mean([D_1,D_4,D_7]) - D_overall; M_B_2 = mean([D_2,D_5,D_8]) - D_overall; M_B_3 = mean([D_3,
    D_6,D_9]) - D_overall;
>> M_B = [M_B_1, M_B_2, M_B_3];
>> M_C_1 = mean([D_1,D_6,D_8]) - D_overall; M_C_2 = mean([D_2,D_4,D_9]) - D_overall; M_C_3 = mean([D_3,
    D_5,D_7]) - D_overall;
>> M_C = [M_C_1, M_C_2, M_C_3];
>> M_all = [M_A; M_B; M_C]
M_all =
      -1.1137       0.21907       0.89463
       2.1274      -0.16287       -1.9645
       1.2421       -1.2343    -0.0078704
```

MAE 5350: HW #3

Multidisciplinary Design Optimization

Hanfeng Zhai[*]

Sibley School of Mechanical and Aerospace Engineering,

Cornell University

November 3, 2021

Q1. Penalty Methods continued

Consider the problem

$$\begin{aligned} \min \;\; & x + y \\ \text{subject to} \;\; & x^2 + y^2 = 2 \end{aligned} \tag{1}$$

(a) Apply KKT conditions to solve the problem

Solution: We first rewrite the problem in standard form:

$$\begin{aligned} \min \;\; & x + y \\ \text{subject to} \;\; & x^2 + y^2 - 2 = 0 \end{aligned} \tag{2}$$

We first write out the pseudo objective:

$$(x + y) + \lambda(x^2 + y^2 - 2) = 0 \tag{3}$$

By applying the KKT condition:

$$\begin{aligned} 1 + 2x\lambda &= 0 \\ 1 + 2y\lambda &= 0 \\ \lambda(x^2 + y^2 - 2) &= 0 \\ \lambda &\geq 0 \end{aligned} \tag{4}$$

[*]Email: hz253@cornell.edu

By solving this problem with `vpasolve` in MATLAB® and generate the following code in an `.mlx` file to directly output the LaTeX font results:

```
syms x y lambda
eqn1 = 1+ 2*lambda*x == 0; eqn2 = 1+2*lambda*y == 0; eqn3 = lambda*(x^2 + y^2 - 2) == 0;
eqns = [eqn1,eqn2,eqn3];
vars = [x y lambda];
answer = vpasolve(eqns, vars);
[(answer.x),(answer.y),(answer.lambda)]
```

The solutions are:

$$\begin{pmatrix} 1.0 & 1.0 & -0.5 \\ -1.0 & -1.0 & 0.5 \end{pmatrix} \tag{5}$$

Since we already know $\lambda \geq 0$, therefore the solutions are obtained

$$x = y = -1; \; \lambda = 0.5 \tag{6}$$

(b) Formulate a quadratic penalty function for this problem. Derive the corresponding optimality conditions as a function of the penalty parameter ρ.

Solution: We first write out the Quadratic Penalty Function:

$$\Phi_Q(\mathbf{x}, \rho_p) = (x + y) + \rho_p(x^2 + y^2 - 2)^2 \tag{7}$$

We then first calculate the gradient of the pseudo-objective function:

$$\nabla\Phi_Q = \begin{pmatrix} 1 + 4\,\rho_p\,x\,(x^2 + y^2 - 2) \\ 1 + 4\,\rho_p\,y\,(x^2 + y^2 - 2) \\ (x^2 + y^2 - 2)^2 \end{pmatrix} = 0 \tag{8}$$

To obtain the optimal point we need $\nabla\Phi_Q = 0$, analyzing the three terms, we chose exterior penalty function method to let $\rho_p \to +\infty$ we know that $(x^3 + xy^2 - 2) = (x^2y + y^3 - 2) = (x^2 + y^2 - 2)^2 = 0$, in such case we deduce that $x = y = \pm 1$.

(c) Plot the contours of your quadratic penalty pseudo-objective for the case of $\rho = 0.5$ and $\rho = 5$. Graphically determine the optimal solution in each case and comment.

From Figure 1 we can deduce from the scheme boundary as the black dotted lines from sub figures **A** and **B** as ρ increases the cruve is approximating the point $(-1, -1)$. Therefore we can obtain that optimal point is $x = y = -1$.

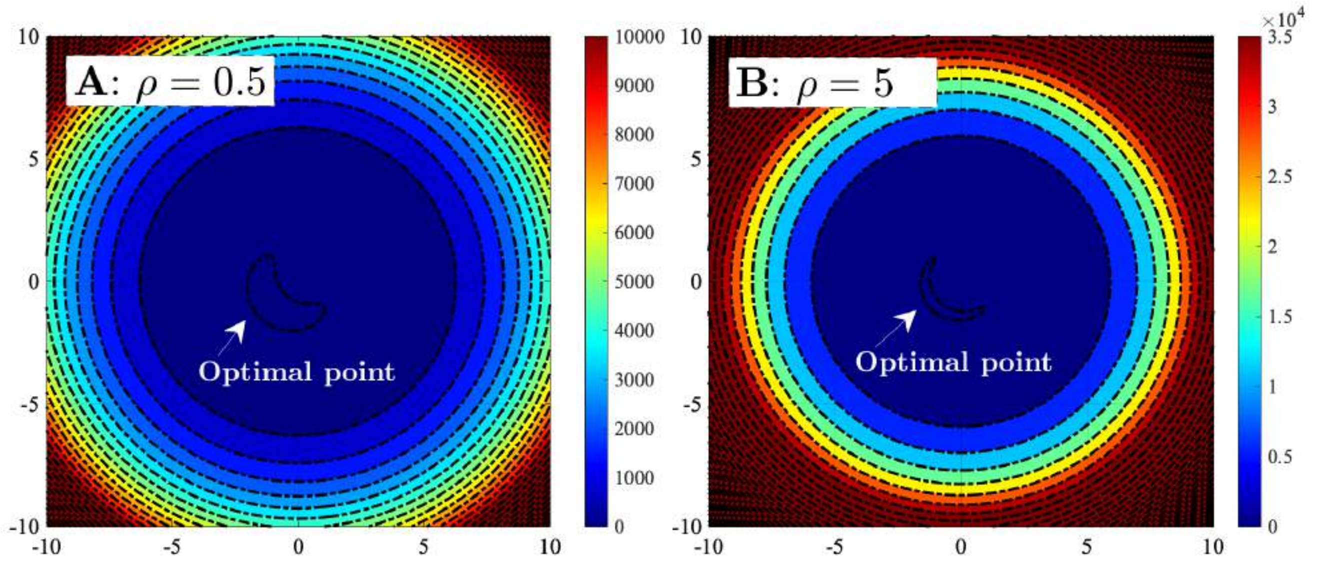
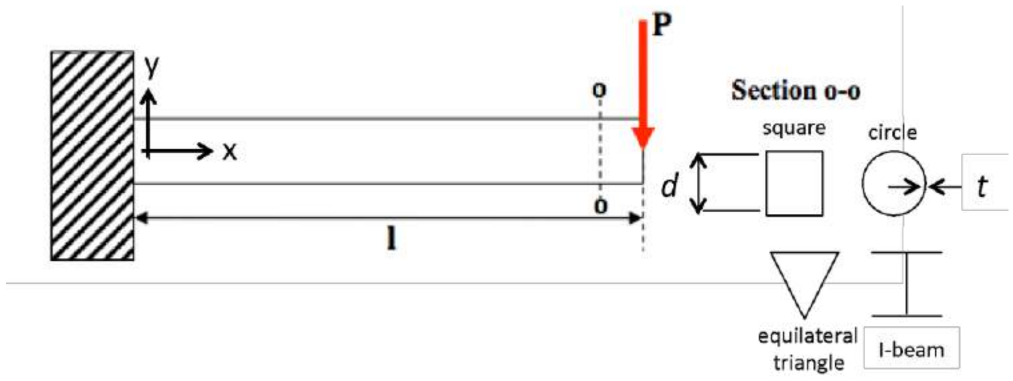


Figure 1: The contour plot for the two quadratic penalty pseudo-objective functions, when **A**. $\rho = 0.5$. **B**. $\rho = 5$.

Q2. Simulated Annealing

A cantilever beam of uniform cross-section with linear dimension d and thickness t (see figure below) has to be designed for minimum mass. The beam is of fixed length $l = 0.3m$ and carries a tip load of $P = 1kN$ at the free end. There are four different geometries available for the beam cross-section: square, circle, equilateral triangle, and I-beam. Finally, the beam can be made of steel, aluminum, or titanium.

There are two explicit constraints given for this design problem:



(1) Maximum bending stress in the beam has to be less than 90% of the yield stress of the material chosen:

$$\sigma_{\max,x} = \frac{Ply_s}{I_s} \leq 0.9S_m$$

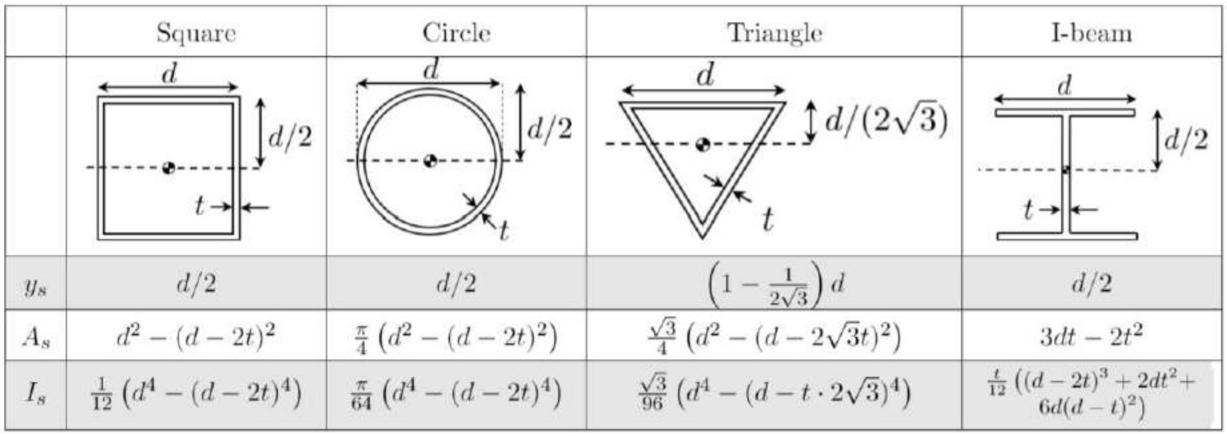
where $\sigma_{\max,x}$ is the normal stress in the x-direction, y_s is the maximum distance from the shape centroid to the top or bottom edge, I_s is the cross-sectional moment of inertia of the shape chosen, and S_m is the yield stress of the material.

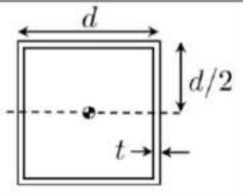
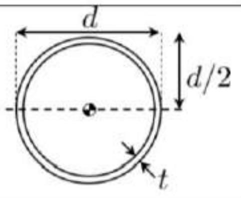
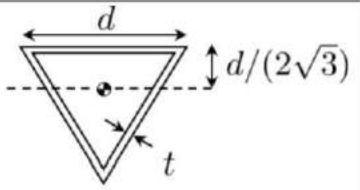
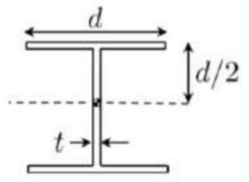
(2) Maximum tip deflection of the beam has to be less than 3mm:

$$\delta = \frac{Pl^3}{3E_m I_s} \leq 0.003$$

where E_m is the Young's modulus of the material chosen and I_s the same as in (1).

The table below summarizes key relationships for each of the different beam sections:



	Square	Circle	Triangle	I-beam
				
y_s	$d/2$	$d/2$	$\left(1-\frac{1}{2\sqrt{3}}\right)d$	$d/2$
A_s	$d^2-(d-2t)^2$	$\frac{\pi}{4}\left(d^2-(d-2t)^2\right)$	$\frac{\sqrt{3}}{4}\left(d^2-(d-2\sqrt{3}t)^2\right)$	$3dt-2t^2$
I_s	$\frac{1}{12}\left(d^4-(d-2t)^4\right)$	$\frac{\pi}{64}\left(d^4-(d-2t)^4\right)$	$\frac{\sqrt{3}}{96}\left(d^4-(d-t\cdot 2\sqrt{3})^4\right)$	$\frac{t}{12}\left((d-2t)^3+2dt^2+6d(d-t)^2\right)$

The table below summarizes key material properties of steel, aluminum, and titanium:

Material	**Young's modulus E_m [GPa]**	**Yield stress S_m [MPa]**	**Density ρ_m [kg/m3]**
Steel	200	300	7600
Aluminum	75	200	2700
Titanium	120	800	4400

(a) Find a constraint relating d and t as a function of the shape to upper-bound t, and a constraint relating d and l to upper-bound d. Finally, use engineering judgment to lower-bound t. Formulate the full optimization problem in standard form including the bounding constraints on t and d.

Solution: First, the constraints of yield stress and tip deflection can be simplified to:

$$\frac{y_s}{I_s} \leq 0.9\frac{S_m}{Pl}$$
$$I_s \geq \frac{1000Pl^3}{9E_m}$$

where y_s and I_s are related to t and d.

According to the given table, we can write $y_s = y_s(d)$, $A_s = A_s(t,d)$, and $I_s = I_s(t,d)$. Therefore we assume there exists functions f and g that $t = f(y_s, A_s, I_s)$ and $d = g(y_s, A_s, I_s)$.

The mass of the beam can be written as $m = \rho_m V_m$, where ρ_m is adapted from the given table.

The volume V_m can be calculated from $V_m = A_s y_s$, where for the three different beam sections:

$$\begin{aligned}
\text{square}: V_m &= (d^2 - (d-2\,t)^2)l \\
\text{circle}: V_m &= \frac{l\pi\left(d^2 - (d-2\,t)^2\right)}{4} \\
\text{triangle}: V_m &= -\frac{l\sqrt{3}\left(\left(d-2\sqrt{3}\,t\right)^2 - d^2\right)}{4} \\
\mathrm{I-beam}: V_m &= l(3\,d\,t - 2\,t^2)
\end{aligned} \tag{9}$$

We can then write out the problem in standard form

$$\begin{aligned}
\min \ & \rho_m V_m \\
\text{s.t.} \ & \frac{Ply_s}{I_s} \le 0.9 S_m \\
& \frac{Pl^3}{3E_m I_s} \le 0.003
\end{aligned} \tag{10}$$

Also, considering the lower and upper bounds, we reconsider the schematic figures given in the instructions. We know that the geometry cannot violate basic physical sense; therefore we can write out the relations between t and d as for the upper bounds for t:

$$\begin{aligned}
\textbf{Upper bounds for t}: \quad & \text{For square}: t \le \frac{d}{2} \\
& \text{For circle}: t \le \frac{d}{2} \\
& \text{For triangle}: t \le \frac{d}{2\sqrt{3}} \\
& \text{For I-beam}: t \le \frac{d}{2}
\end{aligned}$$

For the upper bounds for d, we know that with the definition of a cantilever beam, the width cannot exceeds its length, therefore:

$$\textbf{Upper bounds for d}: \quad d \le l$$

And for real-world applications, a beam usually has a thickness of 200 - 300 mm [Ref.]. Therefore we set the lower bound of t as 100 mm.

(b) Solve this constrained optimization problem using the Simulated Annealing (SA) algorithm.

Solution: To solve this problem, we first need to formulate the objective function (evaluation function in SA), nominated as `objectiveBeam.m` (As shown in the following codes). Here, we impose the constraints as in the objective function (evaluation function), by using the absolute penalty function method: the pseudo objective can be written as $\mathcal{J} = J + \rho_p(|\texttt{constraint1}| + |\texttt{constraint2}| + ...)$, where $\mathcal{J}$ is the pseudo objective and J is the original objective function, ρ_p is the multiplier for the constraints[1]. However, it should be noted that this methods does not guarantee all the constraints will be perfectly satisfied, since there are multiple constraints and is we multiplying them to the same ρ_p there might be some solutions violating the constraints[2]. In this objective function, the input is taken as a vector containing four components of the design variables: t, d, materials and shapes. The parameters corresponding to the four input variables are given in the form of `if` loops. And the eventual objective is the `mass`, which also contain the constraints as we just mentioned.

```matlab
function mass = objectiveBeam(x0)
Input = x0;
%% Input the vars.
t = Input(1); d = Input(2); material = Input(3); shape = Input(4);
%% optimization constants
P = 1e3; l = .3;
%% judge loop for materials
if material == 1 % steel
    Em = 200e9;
  S_m = 300e6;
  rho_m = 7600;
elseif material == 2 % alum
  Em = 75e9;
  S_m = 200e6;
  rho_m = 2700;
elseif material == 3 % titanium
  Em = 120e9;
  S_m = 800e6;
  rho_m = 4400;
end
%% judge loop for shape
if shape == 1 % square
  y_s = d./2;
  A_s = d.^2 - (d - 2.*t).^2;
  I_s = (1/12).*(d.^4 - (d - 2.*t).^4);
elseif shape == 2 % circle
  y_s = d./2;
  A_s = (pi./4).*(d.^2 - (d - 2.*t).^2);
  I_s = (pi./64).*(d.^4 - (d - 2.*t).^4);
elseif shape == 3 % triangle
  y_s = (1 - 1./(2.*sqrt(3))).*d;
```

[1] In this way we impose the constraints to the evaluation function for simulated annealing.
[2] This will be further discussed in the next few sub questions.

```
  A_s = (sqrt(3)./4).*(d.^2 - (d - 2.*sqrt(3).*t).^2);
  I_s = (sqrt(3)./96).*(d.^4 - (d - 2.*sqrt(3).*t).^4);
else % I-beam
  y_s = d./2;
  A_s = 3.*d.*t - 2.*t.^2;
  I_s = (t./12).*((d - 2.*t).^3 + 2.*d.*t.^2 ...
            + 6.*d.*(d - t).^2);
end

%% judge if constraints are violated

sigma_max = (P.*l.*y_s)./(I_s);
delta = (P.*l.^3)./(3.*Em.*I_s);

rho_p = 1e7; % penalty function --> this value is very important!!

P = rho_p * abs(sigma_max-0.9*S_m) + rho_p * abs(delta-0.003) + rho_p * abs(2*t - d) + rho_p * abs
    (d - l);
mass=rho_m*A_s*l + P;

% relation of mass to t & d
end
```

And then we can write out the perturbation function, nominated as perturbBeam.m: taking similar strategy as the objective (evaluation) function, we first input the design variables in a vector form, then we use the MATLAB built-in `randi` function to randomly generate a number between 1 to 4 to judege which inout design variables to perturb. Then if the variables are shape or materials, we perturb them by randomly generate a new number within the design range. If the perturbed design variables are t or d, we first perturb either t or d by randomly generate a real number within the given range ([0, 1] in our problem), and link the other with the geometric inner constraint as we given in Q1:

```
function [xp]=perturbBeam(x0)
Input = x0;
t = Input(1); d = Input(2); material = Input(3); shape = Input(4);
%
num = randi([1 4]);
%%
    if num == 3
        matrl = randi([1 3]);
        xp = [t d matrl shape];
    end

    if num == 4
        shap = randi([1 4]);
        xp = [t d material shap];
    end
%%
xp_t=x0(1);
xp_d=x0(2);
xlb=0;
xub=1;
%
    if num == 1
%         indx=round(rand(1)+1);
       dx=(xub-xlb)*rand(1)+xlb;
       xp_t=dx;
       if shape == 1 %square
           xp_d = 2*xp_t - dx;
       elseif shape == 2 %circle
           xp_d = 2*xp_t - dx;
       elseif shape == 3 %triangle
           xp_d = 2*sqrt(3)*xp_t - dx;
       elseif shape == 4 %I-beam
           xp_d = 2*xp_t - dx;
       end
       xp = [xp_t xp_d material shape];
    end

    if num == 2
%         indx=round(rand(1)+1);
       dx=(xub-xlb)*rand(1)+xlb;
       xp_d=dx;
       if shape == 1 %square
           xp_t = xp_d/2 + dx;
       elseif shape == 2 %circle
           xp_t = xp_d/2 + dx;
       elseif shape == 3 %triangle
           xp_t = xp_d/(2*sqrt(3)) + dx;
       elseif shape == 4 %I-beam
           xp_t = xp_d/2 + dx;
       end
       xp = [xp_t xp_d material shape];
    end

end
```

And to execute the main file, we input the following commands to run the main file with our evaluation and perturbation function. We

```
>> xo = [0.0500     0.2000     3.0000     1.0000];
>> file_eval = 'objectiveBeam';
>> file_perturb = 'perturbBeam';
```

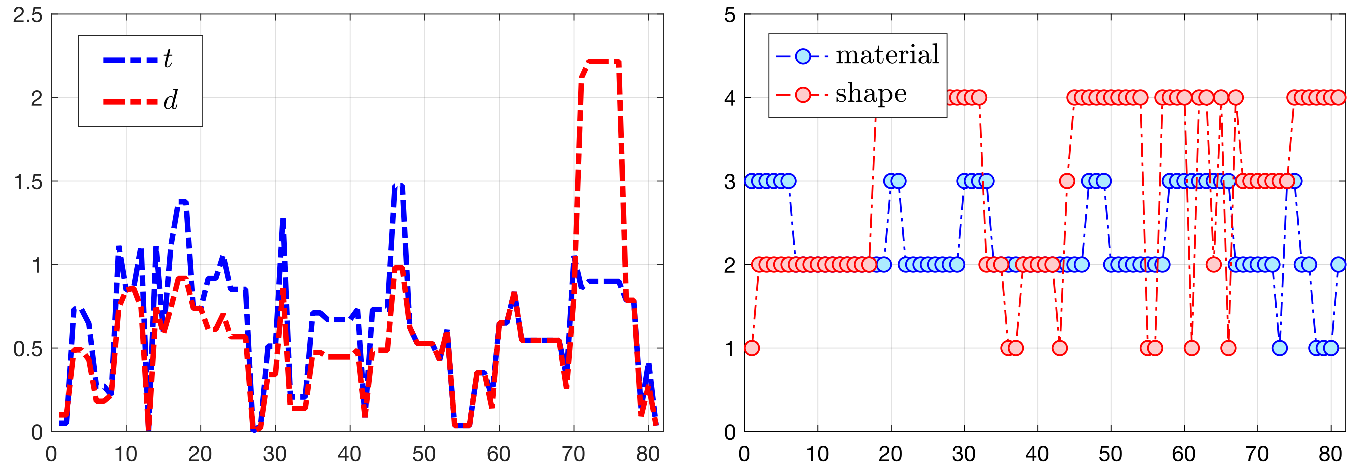


Figure 2: The changing history of the four design variables $[t, d, materials, shape]$ with the simulated annealing iterations.

```
>> [xbest,Ebest,xhist]=SA(xo,file_eval,file_perturb);
```

And by then we generate the eventual output with a history diagram as shown in Figure 2. The eventual optimal point is $[0.1044,\ 0.2572,\ 2.0000,\ 3.0000]$.

(c) Describe the optimal beam design in terms of geometry and material choice.

Solution: Based on our results, we know that for the optimized design, the thickness t and length d are: $t = d = 0.356$, in unit $[m]$. And the materials and shape obeys material $= 2$, which is Aluminum, and shape is I-beam. Obviously this violates our constraints on the geometric relation between t and d. This is because that the four constraints are imposed through Absolute Penalty Function methods, which does not guarantee all the constraints are perfectly satisfied. We thence increase the value of ρ_p to 10 times of its original value (manually), and the optimal point is manually changed as shown in Table 1.

(d) Explore how you can "tune" the SA algorithm (e.g., cooling/annealing schedule, initial guess, stopping criteria) and report your findings on their impact on the solution quality and computational time in a concise format.

Solution: Now, there are a couple of things that we can change in the SA algorithm: (1) the multiplier ρ_p we used in the Absolute Value Penalty Function (we used 1×10^7 in the original code). (2) the cooling methods (we used exponential cooling in the original code). (3) the initial point (we used $(0.05, 0.1, 3, 1)$ in the original design).

For the first parameters, we switched the value of ρ_p for a couple of values and generated the following results as in Table 1. From the results we first know that the *Absolute Value Penalty Function* penalty methods does not satisfies that the constraints are strongly enforced to optimiza-

ρ_p value	Final Design
10^3	(0.0522 0.1287 2.0000 3.0000)
10^4	(0.0500 0.1000 2.0000 1.0000)
10^5	(0.2035 0.5015 2.0000 3.0000)
10^6	(0.0486 0.1198 2.0000 3.0000)
10^7	(0.2435 0.2435 2.0000 4.0000)
10^8	(0.1844 0.1844 2.0000 4.0000)
10^9	(0.1495 0.1495 2.0000 4.0000)
10^{10}	(0.1230 0.3031 2.0000 3.0000)
10^{11}	(0.0500 0.1000 2.0000 4.0000)
10^{12}	(1.3878 0.9252 2.0000 1.0000)
10^{13}	(0.0703 0.1732 2.0000 3.0000)
10^{14}	(1.4589 0.9726 2.0000 1.0000)
10^{15}	(0.5949 1.4658 2.0000 3.0000)

Table 1: Different optimal points align with different ρ_p values.

Methods	Final Design
Exponential cooling	(0.0690 0.1700 2.0000 3.0000)
Linear cooling	(0.1199 0.1199 2.0000 4.0000)

Table 2: Different optimal points corresponding to the SA different cooling methods.

tion problems. Therefore some points in the optimal does not perfectly satisfy all the constraints. From Table 1 we can deduce the optimized materials and shape is $(2, 3)$, which are Aluminum and Triangle.

(2) We can also tune the cooling methods. Based on Table 1 we think $\rho_p = 10^{11}$ would be a good fit for the SA algorithm. The switched results are shown in Table 2.

Estimating the final design of the two cooling methods we can deduce that for Linear cooling the final mass is 11.6446 (here numerical value is sufficient to deduce optimality) and the exponential cooling mass is 8.4654. Hence we can deduce exponential cooling is the more preferred method (only for this case). Note that due to the constraints are imposed through the *Absolute Value Penalty Function* method, and since there are many constraints in this problem. We weight each constraints the same, thence there are some solutions (optimal point) does not perfectly satisfy the constraints.

(3) Later on, by tuning initial guesses, we generate Table 3 to show how different initial points generate different optimal points. Here, our strategy is we randomly give an initial point, and let the SA find the optimized point, and we take the optimal point as the initial again until the optimal point are no longer renewed.

(e) Based on your findings, discuss which SA tuning parameters most affect the solution quality and computation time.

Solution:

Initial Point	Final Design
(0.0100 0.2100 3.0000 1.0000)	(0.0100 0.2100 2.0000 3.0000)
(0.0100 0.2100 2.0000 3.0000)	(0.0745 0.0745 2.0000 4.0000)
(0.0050 0.1360 3.0000 4.0000)	(0.1441 0.3552 2.0000 3.0000)
(0.1441 0.3552 2.0000 3.0000)	(0.0649 0.0649 2.0000 4.0000)
(0.0500 0.2100 1.0000 1.0000)	(0.1504 0.3707 2.0000 3.0000)
(0.0160 0.2990 3.0000 4.0000)	(1.1067 0.7378 2.0000 1.0000)
(1.1067 0.7378 2.0000 1.0000)	(0.0330 0.0812 2.0000 3.0000)
(0.0330 0.0812 2.0000 3.0000)	(0.0317 0.0317 2.0000 4.0000)

Table 3: Different optimal points corresponds to different initial points.

Cooling method	CPU time
Exponential cooling	6.0984e+03
Linear cooling	6.5275e+03

Table 4: The results of the CPU time corresponding to different cooling methods.

Solution quality: From the above tables, it can be detected that cooling schedules (methods) strongly variate the results of the optimal point with same initial point and conditions.

CPU time: For computation time, to quantitatively describe the difference, we create the following tables: we first consider changing the cooling methods, and then we fix the method and change the initial points to see how the CPU computation time variate.

Based on the observations from Tables 4, 5, 6; we can deduce that initial points has very little influence on the CPU time, and for different ρ_p values, the change by the power of 10 only variate CPU time by the scale of 10^{-2}; but for different cooling schedules (methods), the CPU time are strongly variate by different methods. We therefore deduce that cooling methods playing a more significant role in both solution quality and computation time.

(f) What is your final suggested design?

Solution: As proposed and explained many times previously, due to the methods we chose and the paralleled multiple constraints, the final optimal points (suggested design may not perfectly satisfy all the constraints). Yet due to many attempts in the previous tables we can still pick a "acceptable" choice of design based on our general engineering background for the problem (a general "lightweight design" of the beam that can take the loading and satisfies the constraints can be accepted).

Initial point	CPU time
(0.0100 0.2100 3.0000 1.0000)	6.9235e+03
(0.0200 0.2800 3.0000 4.0000)	6.9407e+03
(0.0160 0.1200 1.0000 1.0000)	6.9614e+03
(0.0136 0.2300 1.0000 3.0000)	6.9740e+03
(0.0310 0.2970 2.0000 1.0000)	6.9875e+03

Table 5: The results of the CPU time corresponding to different initial points.

ρ_p value	CPU time
10^{10}	7.0713e+03
10^{20}	7.0820e+03
10^{30}	7.0879e+03
10^{40}	7.0937e+03
10^{50}	7.1016e+03
10^{60}	7.1794e+03
10^{70}	7.1848e+03
10^{80}	7.1905e+03
10^{90}	7.1951e+03

Table 6: The results of the CPU time corresponding to different ρ_p values.

During the attempts to optimize the design problem, we notice a design in Table 4 (the red marked one) presents a generally "good" design as we explained previously: (1) It satisfies all the engineering constraints. (2) The choice of material and shape agrees with most of the results as displayed in Tables 3, 4, 5, 6. (3) It already go through three optimization processes to this point. We therefore pick it as the final design. Hence, the final suggested design is thickness $t = 0.033$, in meters, cross section diameters $d = 0.0812$, in unit meters, shape as circle and materials as Titanium. **Based on these parameters and variables, the optimized mass is 1.9280kg.**

For this problem, I discussed with Mads and Gabrielle, and also asked Prof. Maha Haji for help. I also helped my teammates after I generated the results.

After a short discussion with Mads on Sunday, I found out that my methods may not be the best methods since enforcing the four constraints simultaneously to the objective may be the reason that some solutions violates the constraints, as I stated and explained previously. Therefore I think it is important to add this part to my Q2 as a patch to my individual part.

If we take all the methods same as previously, yet only changing the strategy that applying four constraints simultaneously to the evaluation, to only enforce the given two constraints in the instructions, and take the quadratic penalty function instead of absolute penalty function, the evaluation function writes:

```
function mass = objectiveBeam(x0)
Input = x0;
%% Input the vars.
t = Input(1); d = Input(2); material = Input(3); shape = Input(4);
%% optimization constants
P = 1e3; l = .3;
%% judge loop for materials
if material == 1 % steel
    Em = 200e9;
```

```
  S_m = 300e6;
  rho_m = 7600;
elseif material == 2 % alum
  Em = 75e9;
  S_m = 200e6;
  rho_m = 2700;
elseif material == 3 % titanium
  Em = 120e9;
  S_m = 800e6;
  rho_m = 4400;
end
%% judge loop for shape
if shape == 1 % square
  y_s = d./2;
  A_s = d.^2 - (d - 2.*t).^2;
  I_s = (1/12).*(d.^4 - (d - 2.*t).^4);
elseif shape == 2 % circle
  y_s = d./2;
  A_s = (pi./4).*(d.^2 - (d - 2.*t).^2);
  I_s = (pi./64).*(d.^4 - (d - 2.*t).^4);
elseif shape == 3 % triangle
  y_s = (1 - 1./(2.*sqrt(3))).*d;
  A_s = (sqrt(3)./4).*(d.^2 - (d - 2.*sqrt(3).*t).^2);
  I_s = (sqrt(3)./96).*(d.^4 - (d - 2.*sqrt(3).*t).^4);
else % I-beam
  y_s = d./2;
  A_s = 3.*d.*t - 2.*t.^2;
  I_s = (t./12).*((d - 2.*t).^3 + 2.*d.*t.^2 ...
            + 6.*d.*(d - t).^2);
end

%% judge if constraints are violated

sigma_max = (P.*l.*y_s)./(I_s);
delta = (P.*l.^3)./(3.*Em.*I_s);

rho_p = 1e50; % penalty function --> this value is very important!!

P = rho_p * max([0, sigma_max-0.9*S_m])^2 + rho_p * max([0, delta-0.003])^2;
% P = rho_p * abs(sigma_max-0.9*S_m) + rho_p * abs(delta-0.003) + rho_p * abs(2*t - d) + rho_p *
    abs(d - l);
mass=rho_m*A_s*l + P;

% relation of mass to t & d
end
```

And we also do a slight change for the constraint function, by enforcing different perturbation for variables t and d:

```
function [xp]=perturbBeam(x0)
Input = x0;
l = .3;
t = Input(1); d = Input(2); material = Input(3); shape = Input(4);
%
num = randi([1 4]);
%%
    if num == 3
        matrl = randi([1 3]);
        xp = [t d matrl shape];
    end

    if num == 4
        shap = randi([1 4]);
        xp = [t d material shap];
    end
%%
xp_t=x0(1);
xp_d=x0(2);
xlb=0;
xub=1;
%


    if num == 1
        xp = [t d material shape];
        if shape==3
            xub=d/(2*sqrt(3));
            xlb=0.01;
        else
            xub=d/2;
            xlb=0.01;
        end
        indx=round(1);
        dx=(xub-xlb)*rand(1)+xlb;
        xp(indx)=dx;
    end
    if num == 2
        xp = [t d material shape];
        if shape==3
            xub=l;
            xlb=t*(2*sqrt(3));
        else
            xub=l;
            xlb=t*2;
        end
        indx=round(2);
        dx=(xub-xlb)*rand(1)+xlb;
        xp(indx)=dx;
    end
end
```

In this way as many tries the output won't violate the constraints:

```
...
Counter: 158 Temp: 591.3979 P(dE)= 0.99141
Counter: 158 Temp: 591.3979 Accepted inferior configuration (uphill)
Counter: 158 Temp: 591.3979 Need to reach equilibrium at this temperature
Counter: 158 Temp: 591.3979 Perturbing configuration

xp =

    0.0238    0.1268    1.0000    2.0000

Counter: 159 Temp: 591.3979 P(dE)= 0.99387
Counter: 159 Temp: 591.3979 Accepted inferior configuration (uphill)
Counter: 159 Temp: 591.3979 System nearly frozen

tt =

  230.8100

Counter: 159 Temp: 532.2581 System frozen, SA ended
Best configuration:

xbest =

    0.0197    0.0815    2.0000    3.0000

Lowest System Energy: 2.2659
```

As stated, this is a patch or supplementary to my original solution in Q2, and I hope this can provide a reference.

Q3. Heuristic Optimization (PSO or GA)

Problem B – Genetic Algorithm

Consider the problem

$$\min f(x_1, x_2, x_3) = |x_1|^{0.8} + 5\sin(x_1^3) + |x_2|^{0.8} + 5\sin(x_2^3) + |x_3|^{0.8} + 5\sin(x_3^3)$$

$$\text{subject to} \quad -5 \le x_i \le 5, \ \text{for } i = 1, 2, 3$$

(a) Use the Genetic Algorithm to solve this problem.

Solution: We apply the Optimization Toolbox in MATLAB®:

We first define the objective function in MATLAB, nominated as `objectiveFcn`:

```
function f = objectiveFcn(x,a)
a = 0.8;
f = abs(x(1)).^a + 5*sin(x(1) .^3) + abs(x(2)).^a + 5*sin(x(2) .^3) + abs(x(3)).^a + 5*sin(x(3)
    .^3);
end
```

And running the genetic algorithm optimization by calling the MATLAB built-in function `ga`, by recalling the previously defined objective function `objectiveFcn`, first select the `nonlinear` class for `Objective`, and define the lower and upper bounds `lb = -5` *&* `ub = 5`:

```
clear; clc
a = 0.8;
% Pass fixed parameters to objfun
objfun5 = @(x)objectiveFcn(x,a);

% Set nondefault solver 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});
bo = 3;
% Solve
[solution,objectiveValue] = ga(objfun5,bo,[],[],[],[],repmat(-5,bo,1),...
    repmat(5,bo,1),[],[],options);
% Clear variables
clearvars objfun5 options5
% Display the results
solution
objectiveValue
```

The results of the optimization is shown in Figure 3. The optimal point is obtained as $x_1 = x_2 = -1.1527$ & $x_3 = 1.6745$, as shown in sub figure **D** in Figure 3.

(b) Explore various "tuning" parameters (e.g., number of generations, population size, mutation rate) and report your findings on their impact on the solution quality and computational time in a concise format.

Solution: Based on the instructions and the functionality of the MATLAB `ga` function, the

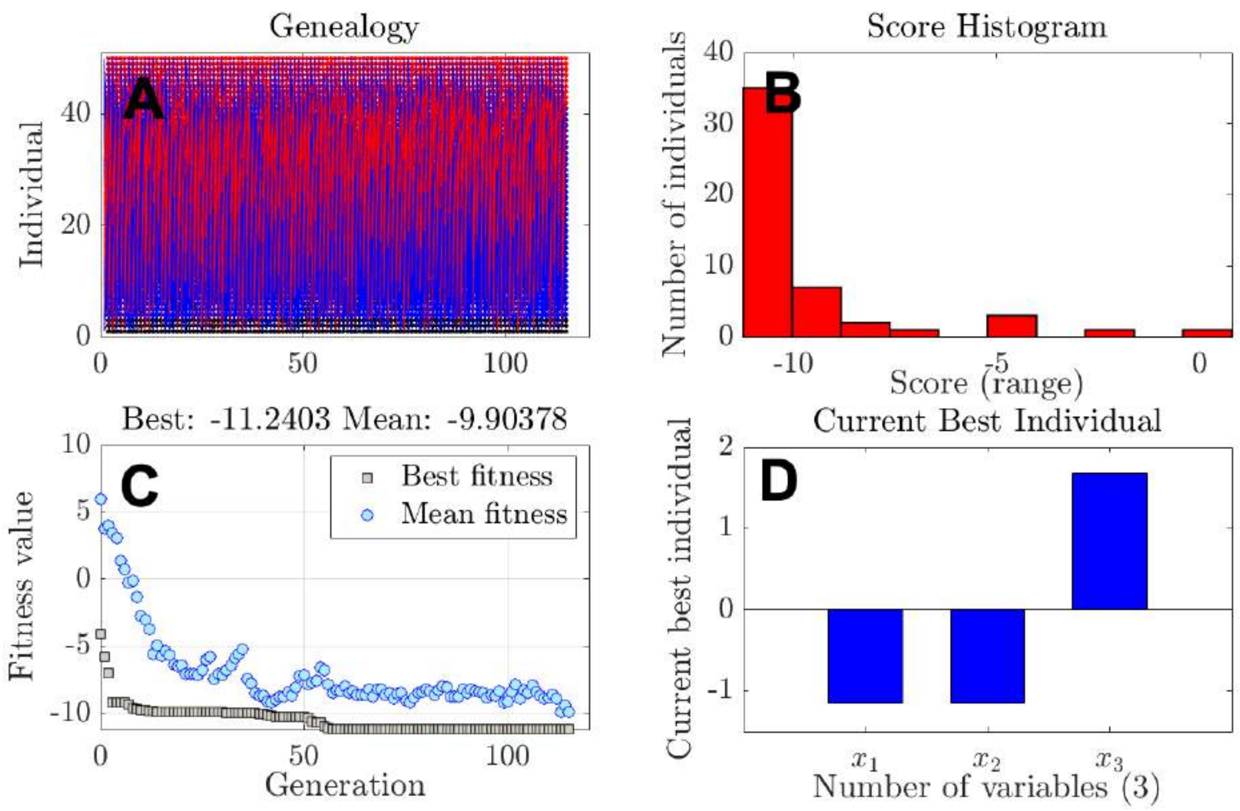


Figure 3: Calculation results of optimization based on genetic algorithm.

Function Tolerance	CPU time	Solution	Objective
10^{-3}	1.3907e+04	(-1.1527 -1.1527 -1.1528)	-11.6273
10^{-4}	1.3968e+04	(-1.1529 -1.1527 1.6745)	-11.2403
10^{-5}	1.3990e+04	(-1.1527 1.6745 -1.1527)	-11.2403
10^{-6}	1.4013e+04	(-1.1527 -1.1527 1.6745)	-11.2403
10^{-7}	1.4029e+04	(-1.1527 -1.9868 3.4872)	-9.4272
10^{-8}	1.4097e+04	(-1.1527 1.6745 -1.1527)	-11.2403
10^{-9}	1.4111e+04	(-1.1527 -1.1527 -1.1527)	-11.6273

Table 7: The CPU time, solution and objectives corresponding to function tolerance.

options for tuning the parameters include (1) Function Tolerance; (2) Population Size; (3) Crossover Fraction; (4) Max Generations; and (5) Constraint Tolerance. According to multiple attempts, we observe that usually the optimization stops at about 100 iterations, which is evidently smaller than the original max ierations, 2000. And due to we don't have a highly nonlinear constraint in this problem, taking the constraint tolerance at a low value (10^{-6}) and fix it is reasonable.

Now, by tuning the above four parameters, we provide the solutions, the objective values and CPU time as shown in the following tables (Tables 7, 8, 9).

We here took similar strategies as in Q2: we first fix the rest parameters and only adjust one to check how it influence the solutions and check all the adjustable parameters manually. For function tolerance, we first fix population size = 50, crossover fraction = 0.7, max generations = 2000, and

Population Size	CPU time	Solution	Objective
50	1.4170e+04	(1.6745 1.6745 -1.1527)	-10.8534
60	1.4193e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
70	1.4207e+04	(-1.9868 -1.1527 -1.1527)	-11.0193
80	1.4225e+04	(-1.1527 -1.9868 1.6745)	-10.6323
90	1.4249e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
100	1.4260e+04	(-1.1528 -1.1527 -1.1527)	-11.6273

Table 8: The CPU time, solution and objectives corresponding to population size.

Crossover Fraction	CPU time	Solution	Objective
0.5	1.4921e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
0.6	1.4935e+04	(-2.7330 -1.1527 1.6745)	-10.1293
0.7	1.4946e+04	(-1.1527 1.6745 -1.1527)	-11.2403
0.8	1.4958e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
0.9	1.4968e+04	(-1.1527 -1.1527 -1.1527)	-11.6273
1	1.4978e+04	(-1.1527 -1.1527 1.6745)	-11.2403

Table 9: The CPU time, solution and objectives corresponding to crossover fraction.

constraint tolerance = 1e-6, and change the function tolerance to generate Table 7. From Table 7 we can observe that for each change of function tolerance of 10^{-1} there is a change of approximately 0.05×10^4 for CPU time, and there are some evident solution change for x_3 when function tolerance is about 10^{-8}.

We then fix the function tolerance to 10^{-9} and fix others to the same and change the population size, we then generate Table 8.

Similar as before, we fix the population size to 100 and fix others to the same and change the population size, we then generate Table 9.

(c) Based on your findings from (b), discuss which GA tuning parameters most affect the solution quality and computation time.

Solution: Based on our test data from Tables 7, 8, 9, we can conclude that the function tolerance influence both the CPU time and solution quality most evidently (Table 7: evident solution change from 10^{-8} to 10^{-10}.)

Appendix. Supplementary Code & Data

Main function of SA (SA.m):

```
Initial configuration:
function [xbest,Ebest,xhist]=SA(x0,file_eval,file_perturb);
% [xbest,Ebest,xhist]=SA(x0,file_eval,file_perturb,options);
%
% Single Objective Simulated Annealing (SA) Algorithm
%
% This function is a generic implementation of the single objective
% Simulated Annealing (SA) algorithm first proposed by Kirkpatrick,
% Gelatt and Vecchi. The algorithm tries to improve upon an initial
% configuration, x0, by evaluating perturbed configurations. When the
% system reaches the "frozen" state, the algorithm stops and the best
% configuration and search history are returned. The user can choose
% from one of two cooling schedules: linear or exponential.
%
% Input:
% x0           initial configuration of the system (a row vector)
% file_eval    file name (character string) of configuration evaluator;
%              assumes that E='file_eval'(x) is a legitimate function
%              call; set up function such that (scalar) output E will be
%              minimized.
% file_perturb file name (character string) of configuration perturbator;
%              assumes that xp='fname_perturb'(x) is a legitimate function
%              call. This function creates a "neighboring" configuration.
% options      algorithm option flags. Uses defaults, [ ], if left blank
%    (1)       To - initial system temperature - automatically determined if
%              left blank ([]). To should be set such that the expression
%              exp(-E(x0)/To)>0.99 is true, i.e. the initial system is "melted"
%    (2)       Cooling Schedule: linear=1, exponential=[2]
%    (3)       dT Temp. increment, e.g. [dT=0.9] for exp. cooling Tk=dT^k*To,
%              abs. temperature increment for linear cooling (Tk+1=Tk-dT);
%    (4)       neq = equilibrium condition, e.g. number of rearrangements
%              attempted to reach equilibrium at a given temperature, neq=[5]
%    (5)       frozen condition - sets up SA exit criterion
%              nfrozen = non-integer, e.g. 0.1 SA interprets this numbers as Tmin,
%              the minimum temperature below which the system is frozen.
%              nfrozen = integer ,e.g. 1,2..  SA interprets this as # of successive
%              temperatures for which the number of desired acceptances defined
%              under options(4) is not achieved, default: nfrozen=[3]
%    (6)       set to 1 to display diagnostic messages (=[1])
%    (7)       set to 1 to plot progress during annealing (=[0])
%
% Output:
% xbest        Best configuration(s) found during execution - row vector(s)
% Ebest        Energy of best configuration(s) (lowest energy state(s) found)
% xhist         structure containing the convergence history
%   .iter       Iteration number (number of times file_eval was called)
%   .x          current configuration at that iteration
%   .E          current system energy at that iteration
%   .T          current system temperature at that iteration
%   .k          temperature step index k
%   .C          specific heat at the k-th temperature
%   .S          entropy at the the k-th temperature
%   .Tnow       temperature at the k-th temperature step
%
% User Manual (article):   SA.pdf
%
% Demos:          SAdemo0 - four atom placement problem
%                 SAdemo1 - demo of SA on MATLAB peaks function
%                 SAdemo2 - demo of SA for Travelling Salesman Problem (TSP)
%                 SAdemo3 - demo of SA for structural topology optimization
%                 SAdemo4 - demo of SA for telescope array placement problem
%
% dWo,(c)  MIT 2004
%
% Ref: Kirkpatrick, S., Gelatt Jr., C.D. and Vecchi, M.P., "Optimization
% by Simulated Annealing", Science, Vol. 220, Number 4598, pp. 671-680, May
% 1983


%
dT=0.9;
neq=5;
nmax=neq*round(sqrt(max(size(x0)))); % nmax - maximum number of steps at one temperature, while
%                                       trying to establish thermal equilibrium
%
nfrozen=3;
diagnostics=1;
eval(['Eo=' file_eval '(x0);']);
To=abs(-Eo/log(0.99));

if nfrozen==round(nfrozen)
    % nfrozen is integer - look for nfrozen successive temperatures without
    % neq acceptances
    Tmin=0;
else
    Tmin=nfrozen; nfrozen=3;
end



% Step 1 - Show initial configuration
if diagnostics==1
```

```
disp('Initial configuration: ')
x0
end

% Step 2 - Evaluate initial configuration
eval(['Eo=' file_eval '(x0);']);
counter=1;
xnow=x0; Enow=Eo; nnow=1;
xhist(nnow).iter=counter;

xhist(nnow).x=x0;
xhist(nnow).E=Enow;
xhist(nnow).T=To;
%  still need to add .S         current entropy at that iteration
xbest=xnow;
Ebest=Enow;
Tnow=To;
if diagnostics==1
    disp(['Energy of initial configuration Eo: ' num2str(Eo)])
end

frozen=0;  % exit flag for SA
naccept=1; % number of accepted configurations since last temperature change
Tlast=1;   % counter index of last temperature change
k=1;       % first temperature step
ET=[];     % vector of energies at constant system temperature

% start annealing
while (frozen<nfrozen)&(Tnow>Tmin)

%Step 3 - Perturb xnow to obtain a neighboring solution

if diagnostics
    disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow) ' Perturbing configuration'])
end

eval(['xp=' file_perturb '(xnow);']);
xp
%Step 4 - Evaluate perturbed solution
eval(['Ep=' file_eval '(xp);'])
counter=counter+1;

%Step 5 - Metropolis Step

dE=Ep-Enow; % difference in system energy
PdE=exp(-dE/Tnow);
if diagnostics
    disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow) ' P(dE)= ' num2str(PdE)])
```

```
end

%Step 6 - Acceptance of new solution
if dE<=0 % energy of perturbed solution is lower , automatically accept
    nnow=nnow+1;
    xnow=xp; Enow=Ep;
    xhist(nnow).iter=counter;
    xhist(nnow).x=xp;
    xhist(nnow).E=Ep;
    xhist(nnow).T=Tnow;
    naccept=naccept+1;
   if diagnostics
   disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Automatically accept better
    configuration (downhill)'])
   end

else
   % energy of perturbed configuration is higher, but might still accept it
   randomnumber01=rand;
    if PdE>randomnumber01
        nnow=nnow+1;
        xnow=xp; Enow=Ep;
        xhist(nnow).iter=counter;
        xhist(nnow).x=xp;
        xhist(nnow).E=Ep;
        xhist(nnow).T=Tnow;
      if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Accepted inferior
    configuration (uphill)'])
      end

    else
        % keep current configuration
        xnow=xnow;
        Enow=Enow;
      if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Kept the current
    configuration'])
      end
    end
end
       ET=[ET; Enow];


if Enow<Ebest
    % found a new 'best' configuration
    Ebestlast=Ebest;
    Ebest=Enow;
    xbest=xnow;
    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' This is a new best
    configuration'])
    end

elseif Enow==Ebest
    same=0;
    for ib=1:size(xbest,1)
        if xbest(ib,:)==xnow
             if diagnostics
          disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Found same best
    configuration'])
             end
        same=1;
        end
    end

     if same ==0
       Ebestlast=Ebest;
       Ebest=Enow;
       xbest=[xbest ; xnow];
        if diagnostics
          disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Found another best
    configuration'])
        end
    end
end

%Step 7 - Adjust system temperature
Told=Tnow;
if (naccept<neq)&(counter-Tlast)<nmax
    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' Need to reach equilibrium at
    this temperature'])
    end
    % continue at the same system temperature
elseif (naccept<neq)&(counter-Tlast)>=nmax
    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' System nearly frozen'])
    end

    Eavg=mean(ET);
    Evar=mean(ET.^2);
    C=(Evar-Eavg^2)/Tnow^2;     % specific heat
    S=log(nmax*length(unique(ET))/length(ET));
    xhist(k).k=k;
    xhist(k).C=C;
    xhist(k).S=S;
    xhist(k).Tnow=Tnow;


    frozen=frozen+1;
    Tlast=counter;
    naccept=0;


%     switch schedule
%         case 1
%             % linear cooling
%             Tnow=Tnow-dT;
%               if Tnow<0
%                   frozen=nfrozen; %system temperature cannot go negative, exit
%               end
%         case 2
            % exponential cooling
              Tnow=dT*Tnow;
%         case 3
%               Tindex=Tindex+1;
%               if Tindex>size(Tuser,1)
%                   frozen=nfrozen; % have run through entire user supplied cooling schedule
%               else
%               Tnow=Tuser(Tindex,1);
%               neq=Tuser(Tindex,2);
%               end
%         otherwise
%               disp('Erroneous cooling schedule choice - option(2) - illegal')
%     end


    k=k+1;

    ET=[];

 elseif (naccept==neq)
    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' System reached equilibrium'])
    end

    Eavg=mean(ET);
    Evar=mean(ET.^2);
    C=(Evar-Eavg^2)/Tnow^2;     % specific heat
    S=log(nmax*length(unique(ET))/length(ET));
    xhist(k).k=k;
    xhist(k).C=C;
    xhist(k).S=S;
    xhist(k).Tnow=Tnow;


    Tlast=counter;
    naccept=0;

%     switch schedule
%         case 1
%             % linear cooling
%             Tnow=Tnow-dT;
%               if Tnow<0
%                   frozen=nfrozen; %system temperature cannot go negative, exit
%               end
%         case 2
              % exponential cooling
                Tnow=dT*Tnow;
%         case 3
%             % user supplied cooling
%               Tindex=Tindex+1;
%               if Tindex>size(Tuser,1)
%                   frozen=nfrozen; %have run through entire user supplied cooling schedule
%               else
%               Tnow=Tuser(Tindex,1);
%               neq=Tuser(Tindex,2);
%               end
%
%         otherwise
%               disp('Erroneous cooling schedule choice - option(2) - illegal')
%     end


     k=k+1;

  ET=[];
  % xbest];

 end

end %while (frozen<nfrozen)&(Tnow>tmin)


    if diagnostics
       disp(['Counter: ' num2str(counter) ' Temp: ' num2str(Tnow)  ' System frozen, SA ended'])
       disp(['Best configuration: '])
       xbest
       disp(['Lowest System Energy: ' num2str(Ebest) ])
    end
```

MAE 5350: HW #4

Multidisciplinary Design Optimization

Hanfeng Zhai[*]

Sibley School of Mechanical and Aerospace Engineering,

Cornell University, Ithaca, NY

November 16, 2021

Q1. Scaling

Consider the following optimization problem:

$$\begin{aligned} \text{minimize } & x_1^2 - 0.001x_2^2 + 1000x_3^2 \\ \text{subject to } & x_1^2 + x_2^2 + x_3^2 = 1 \end{aligned} \tag{1}$$

(a) Find the optimal solution to the problem.

Solution: As the semester is approaching to an end, and as I'm learning a lot from this course by coding with MATLAB®, I decided to try to solve problem with `python` this time for the homework to apply what we've learnt to a broader applications.

In this problem, we pick the `'SLSQP'` optimization method (Sequential Least Squares Programming), which can handle nonlinear constraints pretty well with set tolerance [Ref.]. The corresponding tolerance was given at a value of 10^{-6}. A randomly guess initial point $x_0 = (0.1, 0.3, 0.5)$ was given to initiate the optimization. We thence generate the following code for optimize the given problem:

Step 1. Define the objective function, constraints, and initial point:

[1]:
```
import scipy.optimize
fun = lambda x: x[0] ** 2 - 0.001 * x[1] ** 2 + 1000  * x[2] ** 2
cons = ({'type': 'eq', 'fun': lambda x:  x[0] ** 2 +  x[1] ** 2 +  x[2] ** 2 - 1})
x0 = [0.1, 0.3, 0.5]
```

[*]Email: hz253@cornell.edu

Step 2. Optimize the problem by recall the `minimize` function:

```
[2]: res = scipy.optimize.minimize(objective_function, x0, method='SLSQP', tol=1e-6,␣
     ↪constraints=cons)
```

Step 3. Print out the solution:

```
[3]: print(res)
```

```
     fun: -0.0010000000002877533
     jac: array([ 3.47680325e-05, -2.00000317e-03,  1.08021792e-04])
 message: 'Optimization terminated successfully'
    nfev: 44
     nit: 9
    njev: 9
  status: 0
 success: True
       x: array([-1.11711296e-07,  1.00000000e+00, -2.87906700e-09])
```

Therefore the optimized point is $(-8.45463244 \times 10^{-6}, 1, 3.42745663 \times 10^{-7})$ - which can be approximately considered as the point $(0, 1, 0)$.

(b) Find a non-singular transformation $x = Ly$ such that the condition number of the Hessian matrix of f is close to unity.

Solution: We first need to rescale the objective; assuming the new form of the objective agrees:

$$J(\mathbf{x}) = x_1^2 - \tilde{x}_2^2 + \tilde{x}_3^2 \qquad \text{where } x_1 = L_1\tilde{x}_1,\ x_2 = L_2\tilde{x}_2,\ x_3 = L_3\tilde{x}_3 \tag{2}$$

Therefore we can solve:

$$L_2 = \sqrt{1000} = 31.6228 \approx 10,\ L_3 = \sqrt{0.001} = 0.0316 \approx 0.01$$

In this way we can deduce $L_1 = 10$.

With the new $\tilde{x}_1$ and $\tilde{x}_2$, the optimization problem can be written in the new form:

$$\text{minimize } 1001x_1^2 - 0.1\tilde{x}_2^2 + 0.1\tilde{x}_3^2 \qquad \text{subject to} \quad 100x_1^2 + 100\tilde{x}_2^2 + 0.0001\tilde{x}_3^2 = 1 \tag{3}$$

Now we can compute the new Hessian of the objective, written as

$$\tilde{\mathbf{H}} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & -0.2 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}$$

which agrees with the condition $\mathcal{O}(\tilde{\mathbf{H}}) = -0.0800 \approx -0.1, \longrightarrow$ the problem is rescaled.

(c) Quantify the effect of rescaling the problem, either on the number of iterations or function evaluations required to find the solution (convergence), or the quality of the optimal solution itself, or a combination of the two.

Solution: Based on the rescaled problem in subquestion (b), we can re-optimize the problem with the following python code:

```
[4]: import scipy.optimize


fun = lambda x: 100 * x[0] ** 2 - 0.1 * x[1] ** 2 +  0.1 * x[2] ** 2
cons = ({'type': 'eq', 'fun': lambda x:  100 * x[0] ** 2 + 100 * x[1] ** 2 + 1e-4␣
 ↪* x[2] ** 2 - 1})


x0 = [0.1, 0.3, 0.5]
print(fun)
```

```
<function <lambda> at 0x7f8f08b444c0>
```

```
[5]: res = scipy.optimize.minimize(fun, x0, method='SLSQP', tol=1e-6, constraints=cons)
```

```
[6]: print(res)
```

```
     fun: -0.0009999951013620627
     jac: array([ 1.31615215e-03, -2.00000045e-02,  1.87065307e-05])
 message: 'Optimization terminated successfully'
    nfev: 31
     nit: 7
    njev: 7
  status: 0
 success: True
```

```
    x: array([6.57331016e-06, 1.00000015e-01, 9.35251772e-05])
```

From the output we know that the optimized point is $(x_1, \tilde{x}_2, \tilde{x}_3) = (6.57331016 \times 10^{-6}, 1.00000015 \times 10^{-1}, 9.35251772 \times 10^{-5})$. We already know that $x_2 = \tilde{x}_2 L_2$, $x_3 = \tilde{x}_3 L_3$; the scaled back optimal solution is $(6.57331016 \times 10^{-51}, 1.00000013, 9.35251772 \times 10^{-7})$ - the x_1 and x_3 values truns out to be verry small, which can be considered as 0: to verify this point, we use `fmincon` of MATLAB of `'sqp'` to recompute the optimization problem to verify, and generate the following code:

*The **main program** for running the optimization:*

```
% Set nondefault solver options
options2 = optimoptions('fmincon','PlotFcn',{'optimplotx','optimplotfval'},'Display','iter','
    Algorithm','sqp');

% Solve
[solution,objectiveValue] = fmincon(@obj,x0,[],[],[],[],[],[],@MDOcons,...
    options2);

% Clear variables
clearvars options2
```

*The **objective function**:*

```
function f = obj(x)
    f = 100*x(1)^2 - 0.1*x(2)^2 + 0.1*x(3)^2;
end
```

*The **constraints**:*

```
function [c,ceq] = MDOcons(x0)
    x = x0;
    c = [];
    ceq = 100*x(1)^2 + 100*x(2)^2 + 1e-4*x(3)^2 - 1;
end
```

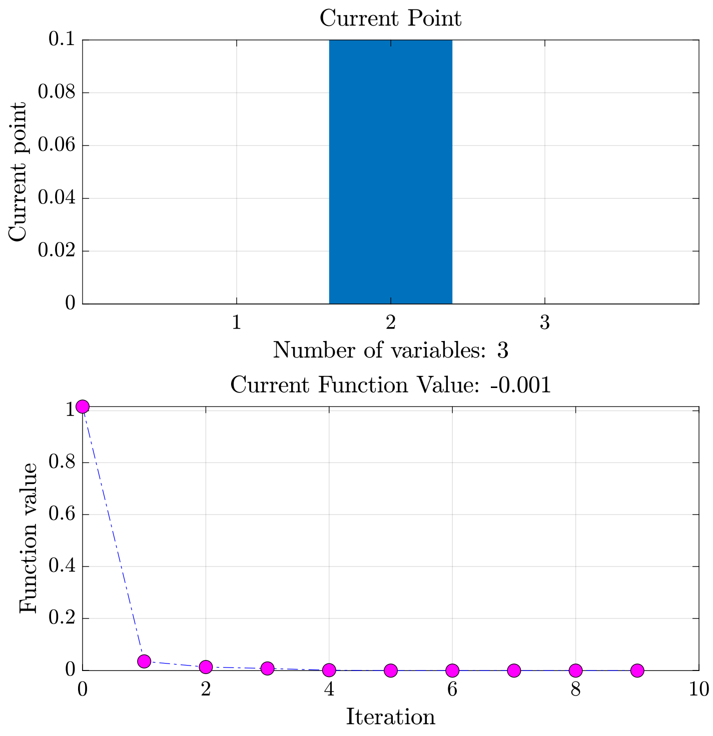
And generate the following output and figure:

```
Iter  Func-count            Fval   Feasibility   Step Length       Norm of    First-order
                                                                      step     optimality
    0           4    1.016000e+00     9.000e+00     1.000e+00     0.000e+00     2.000e+01
    1          23    3.449389e-02     9.770e+00     4.748e-03     9.017e-02     2.848e+00
    2          27    1.276882e-02     2.222e+00     1.000e+00     1.793e-01     7.996e-02
    3          31    7.254477e-03     3.834e-01     1.000e+00     1.274e-01     3.544e-01
    4          35    6.652805e-04     2.658e-02     1.000e+00     1.641e-01     2.220e-01
    5          39   -9.984504e-04     2.785e-04     1.000e+00     1.216e-01     1.423e-02
    6          43   -9.999895e-04     3.941e-07     1.000e+00     3.726e-03     2.007e-03
    7          47   -9.999993e-04     5.815e-09     1.000e+00     5.035e-05     4.821e-04
    8          51   -1.000000e-03     5.854e-10     1.000e+00     3.955e-05     3.599e-06
    9           55    -1.000000e-03      2.220e-16      1.000e+00      2.927e-11      3.599e-06

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```



We can deduce that the optimization point for the rescaled optimization is $(0, 0.1, 0)$; Therefore it is safe to say that the optimization point is $(0, 1, 0)$, agrees with our previous results before rescale the optimization problem - and verify our hypothesis from the `python` code.

Based on the output, we can deduce that the iterations of the optimization reduces from 9 to 7, which means that rescaling problem makes it easier to solve (converge).

Q2. Isoperformance

The Jacobian matrix at a point x_0 is given as:

$$\nabla J(x_0) = \begin{bmatrix} \frac{\partial J_1}{\partial x_1} & \frac{\partial J_2}{\partial x_1} & \frac{\partial J_3}{\partial x_1} & \frac{\partial J_4}{\partial x_1} \\ \frac{\partial J_1}{\partial x_2} & \frac{\partial J_2}{\partial x_2} & \dots & \frac{\partial J_4}{\partial x_2} \\ & \dots & & \\ & & & \\ & & \dots & \\ \frac{\partial J_1}{\partial x_6} & \dots & \dots & \frac{\partial J_4}{\partial x_6} \end{bmatrix}_{x_0} = \begin{bmatrix} 1 & 4 & 5 & 6 \\ 0.2 & 0.7 & 0.9 & 0.1 \\ 4 & 6 & 1 & 0 \\ 7 & 8 & 8 & 7 \\ 2.4 & 1.7 & 2.9 & -1.1 \\ 12 & 8 & 7 & 1 \end{bmatrix}$$

(a) What are the performance invariant directions that we can step to from x_0?

Solution: First, we apply a singular value decomposition (SVD) to the Jacobian matrix (based on the lecture slides):

$$\Delta J^{\mathsf{T}} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}}$$

We use MATLAB to conduct such a process:

```
>> A = [1 4 5 6; .2 .7 .9 .1; 4 6 1 0; 7 8 8 7; 2.4 1.7 2.9 -1.1; 12 8 7 1];
>> [U,S,V] = svd(A')

    U =

       -0.5922   -0.5492    0.2306    0.5428
       -0.5645   -0.0401   -0.7525   -0.3367
       -0.5010    0.2446    0.6132   -0.5596
       -0.2823    0.7981   -0.0671    0.5281


    S =

       23.5418         0         0         0         0         0
             0    8.1328         0         0         0         0
             0         0    3.3332         0         0         0
             0         0         0    1.7753         0         0


    V =

       -0.2994    0.6519   -0.0348   -0.2444   -0.6031   -0.2463
       -0.0422    0.0199    0.0194   -0.3256   -0.2033    0.9220
       -0.2658   -0.2696   -0.8939   -0.2304    0.0125   -0.0662
       -0.6221    0.4154    0.0090    0.1830    0.6167    0.1630
       -0.1497   -0.1912    0.3379   -0.8301    0.2828   -0.2406
       -0.6546   -0.5411    0.2918    0.2421   -0.3668   -0.0197
```

Therefore we know matrix $\mathbf{V}$ is written in the form:

$$\mathbf{V} = \begin{bmatrix} -0.2994 & 0.6519 & -0.0348 & -0.2444 & -0.6031 & -0.2463 \\ -0.0422 & 0.0199 & 0.0194 & -0.3256 & -0.2033 & 0.9220 \\ -0.2658 & -0.2696 & -0.8939 & -0.2304 & 0.0125 & -0.0662 \\ -0.6221 & 0.4154 & 0.0090 & 0.1830 & 0.6167 & 0.1630 \\ -0.1497 & -0.1912 & 0.3379 & -0.8301 & 0.2828 & -0.2406 \\ -0.6546 & -0.5411 & 0.2918 & 0.2421 & -0.3668 & -0.0197 \end{bmatrix}$$

where the red part indicate the null space.

Therefore we can compute the performance invariant direction:

$$\begin{aligned} \Delta x &= \alpha \cdot (\beta_1 v_{z+1} + \ldots + \beta_{n-z} v_n) \\ &= \alpha \cdot \left(\beta_1 \begin{bmatrix} -0.6031 \\ -0.2033 \\ 0.0125 \\ 0.6167 \\ 0.2828 \\ -0.3668 \end{bmatrix} + \beta_2 \begin{bmatrix} -0.2463 \\ 0.9220 \\ -0.0662 \\ 0.1630 \\ -0.2406 \\ -0.0197 \end{bmatrix} \right) \end{aligned} \tag{4}$$

(b) Show an example of a step direction (vector), Δx, such that $J(x_0+\Delta x)-J(x_0) \approx 10^{-4}$. **Solution:** Assuming the Hessian at the point x_0 for every objective function J_i is the identity matrix. We also assume the Taylor approximation of J_i writes:

$$J_i(x_0 + \Delta x) = J_i(x_0) + \Delta J_i^{\mathsf{T}}(x_0)\Delta x + \frac{1}{2}\Delta x^{\mathsf{T}} H(x_0)\Delta x + \ldots$$

which further reduces to

$$J_i(x_0 + \Delta x) = J_i(x_0) + \Delta J_i^{\mathsf{T}}(x_0)\Delta x + \frac{1}{2}\Delta x^{\mathsf{T}}\Delta x$$

Based on the instructions, we can establish:

$$\Delta J_i^{\mathsf{T}}(x_0)\Delta x + \frac{1}{2}\Delta x^{\mathsf{T}}\Delta x \approx 10^{-4} \tag{5}$$

By definition, we know that $\Delta J_i^{\mathsf{T}} x = 0$; therefore we need to solve

$$\frac{1}{2}\Delta x^{\mathsf{T}}\Delta x \approx 10^{-4} \tag{6}$$

To solve this equation, substitute Equation (4) back into Equation (6); we generate the following MATLAB® code:

```
A = [1 4 5 6; .2 .7 .9 .1; 4 6 1 0; 7 8 8 7; 2.4 1.7 2.9 -1.1; 12 8 7 1];
[U,S,V] = svd(A);
vec1 = U(:,5);vec2 = U(:,6);
syms Alpha B1 B2; deltaXdirections = vpa(Alpha*(B1*vec1 + B2*vec2),4);
deltaX = @(consts) consts(1)*(consts(2)*vec1 + consts(3)*vec2);
eqn = @(consts) A'*deltaX(consts) + 1/2*deltaX(consts)'*H*deltaX(consts) - 1e-4;
H = eye(6);
guess = [1;1;1];
[consts, fval, exitflag] = fsolve(eqn,guess);
```

We therefore generate the output variable `consts` of the function:

```
consts =


    0.0153
    0.6551
    0.6551
```

We therefore know the constants: $\alpha = 0.0481$, $\beta_1 = 0.6569$, $\beta_2 = 0.6569$, as the example of the step direction Δx:

$$\Delta x = \begin{bmatrix} -0.0085 \\ 0.0072 \\ -0.0005 \\ 0.0078 \\ 0.0004 \\ -0.0039 \end{bmatrix} \tag{7}$$

Verify it we can compute

$$\frac{1}{2}\Delta x^{\mathsf{T}}\Delta x = 1.0046 \times 10^{-4} \tag{8}$$

References

[1] Olivier L. de Weck & Marshall B. Jones. Isoperformance: Analysis and Design of Complex Systems with Known or Desired Outcomes. Fourteenth Annual International Symposium of the International Council on Systems Engineering (INCOSE) Toulouse, France, 21 June – 24 June 2004.

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

Hanfeng Zhai[*], Mads Berg[†], Jiayi Cao[‡], Will Hintlian[§], Kevin Pan[¶]

[a] *Sibley School of Mechanical and Aerospace Engineering, Cornell University*

[b] *School of Applied & Engineering Physics, Cornell University*

September 21, 2021

Q1. Project selection

Pick a multidisciplinary system to analyze. Form a team of students who are interested in the same system. For the multidisciplinary design problem that your team has chosen, write a short (≈ 2 pages) project proposal. You should address the following:

Formal problem statement

On the existing trend of global warming and the catastrophes caused by the increasing temperature, there is no doubt that shifting and revolutionizing our energy form is becoming one of the most important goals for scientists, engineers, and the whole human race [IPCC Report]. Thus, with the effort and collaborations between governments, corporations, institutions, we are making great progress in advancing wind power [U.S. Energy Information (a)], electrochemical energies [Region, and Segment Forecasts], nuclear powers [World Nuclear Energy Assoc.], and many related clean energies for substitutes of traditional fossil fuels. However, a new form of clean energy, thermo-magnetic power, was often neglected by the general public. In fact, adopting magnetic

[*a] Email: hz253@cornell.edu
[†b] Email: mpb99@cornell.edu
[‡a] Email: jc2732@cornell.edu
[§a] Email: wth42@cornell.edu
[¶a] Email: kp428@cornell.edu

power as a new form of green energy is not a novel thing, and emerging and growing drastically in recent years, since it can also be employed as part of many renewable energies and functional equipment [MMTA, 2016]. For example, magnetic materials play a pivotal role in the efficient performance of devices in a wide range of applications such as electric power generation, transportation, air-conditioning, and telecommunications [Matizamhuka, 2018]. In general, the drive towards improving electricity transmission efficiency and the replacement of oil-based fuels by electric motors in transportation technologies has motivated researchers to focus on magnetic material technologies [Gutfleisch *et al.*, 2011]. Physics tells us that a change in a magnetic field generates electricity that can support our daily energy needs as a form of clean energy [U.S. Energy Information (b)]. In addition, a change of temperature field can cause the magnetic variation for specific materials under set conditions in solid-state physics [Kittel, 1986]. Hence, it is straightforward that a temperature change can generate magnetic change thus generate electric energy that meets our needs, which we may term as thermo-magnetic energy. To employ such kind of energy, a particular machine, named thermo-magnetic generator (TMG) is designed. Currently, the research and real-world applications are not widely touched by both academia and industry, compared with other forms of clean energy. Notwithstanding, the potential of TMG is huge since its magnetic are ubiquitous and the world demands clean energy strong.

The genius idea of the thermo-magnetic generator can be traced back to Nicola Tesla [Tesla, 1889]: Originally named Pyromagneto-Electric Generator, whose idea employs two well-known laws: First, that electricity or electrical energy is developed in any conducting-body by subjecting such body to a varying magnetic influence. Second, the magnetic properties of iron or other magnetic substance may partially or entirely be destroyed or caused to disappear by raising it to a certain temperature, but it restored and caused to reappear by again lowering its temperature to a certain degree. The Tesla and Edison [Edison, 1892] patents originated more than 100 years ago formulated our basics to design such a machine.

In our design works, the TMG model was mainly adopted from Waske *et al.*'s work [2019], whose model was very similar to Tesla's original design yet more practical for to-

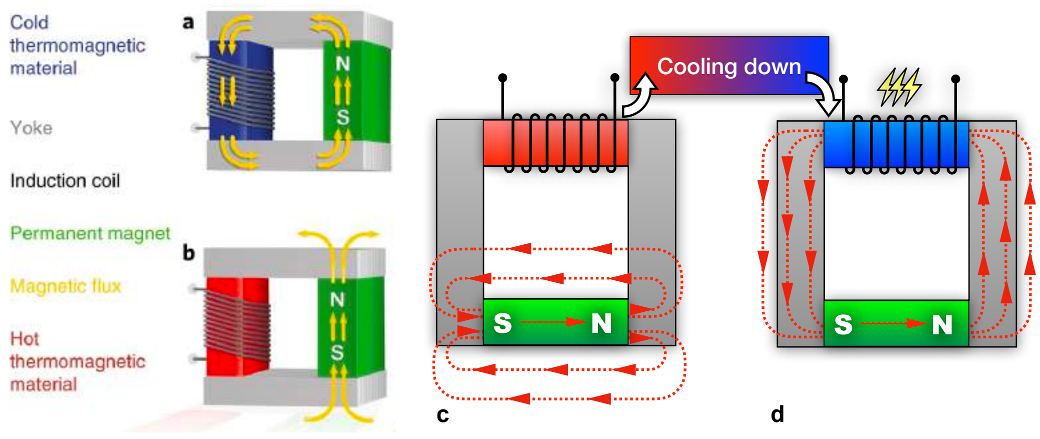


Figure 1: The schematic illustration of thermo-magnetic generator. Note that subfigures **a** and **b** are reproduced from Waske *et al.* [2019]. Subfigures **c** to **d** represents how the cooling down of the active magnetic materials generates electric power.

day's industrial applications, as illustrated in Figure 1 **a** and **b**. Our model is illustrated in Figure 1 **c** and **d**: a hollow squared-shape generator connects two pieces ferromagnetic materials - so-called *yoke* - without permanent magnetization on the two sides, rendered as different colors. The green material represents the permanent magnetic material, generating the magnetic field. The blue and red parts represent the active material when "hot" and "cold", respectively, where we study whose behavior through changing the temperature thus causing its magnetic properties changing, according to Tesla [Tesla, 1889]. Heating up thus stops connection of magnetic field, while the magnetic field is trapped in the permanent materials, as in Figure 1 **a**. Cooling down causes "activation" of active materials, thus making the magnetic field go through the whole generator, as in Figure 1 **b**. The variation process generates electricity.

In our project, we extend the system and also consider the mechanism by which the active material is heated up and cooled down. We propose a simple design scheme[1] where a flow channel is connected on top of the active material and is continuously supplying a flow of fluid through it. This fluid serves to enhance the heat-up and cool-down processes. These two processes, in conjunction, represent a thermal cycle, and we assume that the power output[2] of the device should simply scale with how long

[1]which is definitely not the optimal one
[2]energy produced per time

this takes.[3] Note that the work of the pump as it pushes the fluid through has to be considered in the overall power output and efficiency of the system too. The work of the pump can be computed from the specified pressure difference and the fluid flow. We will divide this by an efficiency factor of the pump and subtract that from the power output, hence obtaining an *effective* power output. This of course, goes back into the efficiency of the system too.

As the active material is cooled down and heated up, the so-called *magnetic permeability* changes. This parameter defines the ability of a material to be magnetized in response to an external magnetic field. Through a complicated mechanism, this can cause the active material to either be *"guiding"* the magnetic field of the permanent magnet through it or not. As the magnetic field in the active material thus changes, a current can be induced in a coil wrapped around it. This current will, according to Faraday's law of electromagnetic induction, be proportional to the total magnetic flux going through the material, and we can thus try to optimize that.

We already have a lot of physical simplifications in mind. Not all of them are implemented in Q2. and Q3. of this assignment, but probably will be later. At this point, we are in principle still open to keeping some of them

Here, a simplified heating/cooling process will be considered, where we separately simulate the cooling/heating and the magnetic field/magnetization field. From the heating/cooling process, one can extract how long that takes; from the magnetic fields, we extract the difference between the total flux at maximum and minimum temperature of the active material. This temperature will be determined by looking at the so-called *Curie temperature* of the active material. Thus we define the duration of a heating/cooling cycle as the time it takes to get sufficiently above/below the Curie temperature in order for the permeability of the material to change a given amount[4]. The relation between the two can be found experimentally.Additionally, the temperature distribution will not be homogeneous, that is, the time it takes to heat up and cool down every infinitesimal point of the active material will not be the same. In this project we decide that every single point should be above or below a certain tempera-

[3] We might, later on, decide that these two processes are roughly equal and only look at one of them.
[4] yet to be decided upon

ture in order for the material to be considered hot or cold [5]. The permeabilities of the active material at the hottest and coldest temperature are plugged into the magnetic module which will spit out the total magnetic flux at these extremes. The difference will be taken as proportional to total power output. We thus have to model outputs that will be proportional to the total power output, and we will make their product an objective of the optimization. In order to find the constant of proportionality, one would have to run an actual experiment, but for optimization in itself, it turns out not to be necessary. The temperatures around the Curie temperature to heat up and cool down around could definitely be made an object of optimization too, but we refrain from doing that since it would include a lot of physics that would take time away from the optimization itself.[6] A schematic diagram for our TMG systems are shown as in Figure 2. Note that essential definitions are made in this figure.

Here, we are curious how we can design the best TMG in this model considering as many modules involved with multidisciplinary optimization.

It is obvious that the TMG system involves complex disciplines, requires certain standards to optimize for a engineering solution, which can be tackled multidisciplinary optimization (MDO). In MDO, a problem involves multiple disciplines targeting specific objectives can be written in the following forms [Agte *et al.*, 2009]:

$$\begin{aligned} \min f(\mathbf{x}, \mathbf{p}) \\ \mathbf{x} = [x_1, ..., x_n]^T,\ \mathbf{p} = [p_1, ..., p_m]^T \\ x_{i,LB} \leq x_i \leq x_{i,UB}, \quad i = 1, 2, ..., n \\ \text{s.t.} \quad \mathbf{g}(\mathbf{x}, \mathbf{p}) < 0,\ \mathbf{h}(\mathbf{x}, \mathbf{p}) = 0 \end{aligned}$$

where f is the objective function that we aims to maximize or minimize. $\mathbf{x}$ is a n-dimensional vector of design variables with lower and upper bounds, $\mathbf{p}$ is a vector of fixed parameters that influence the behavior of the system but cannot be freely chosen (material properties, operating conditions, ...), and $\mathbf{g}$ and $\mathbf{h}$ are inequality and equality constraints, respectively. These variables for our TMG will be presented in the following

[5]This procedure is likely not optimal, but we can not optimize everything

[6]As of now, we are still listing choice of the active material as an input variable, but we are inclined to scratch that too and settle on Gadolinium, which is the conventional choice for a TMG.

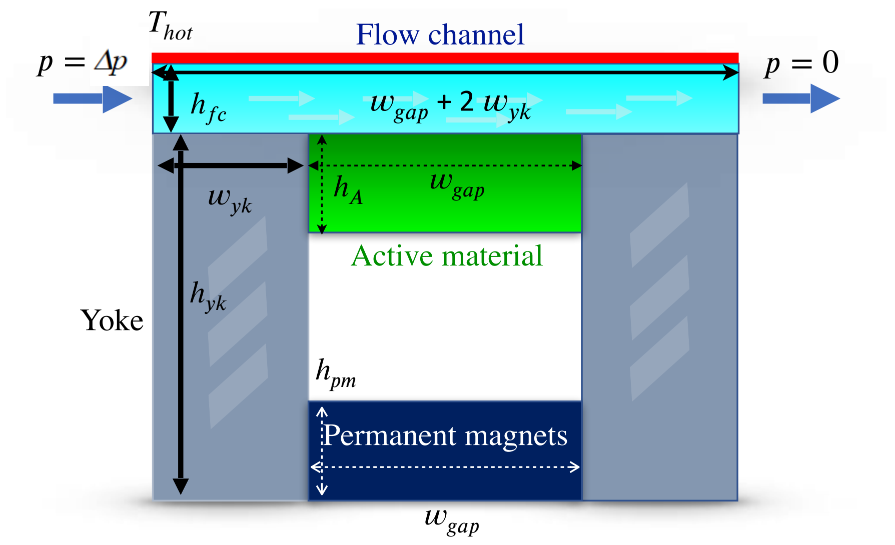


Figure 2: Our proposed design scheme. Notice in particular the flow channel in blue through which a fluid is pushed by a pump. This picture is to also serve as a reference to some of the variables/parameters mentioned later. In this picture, the green block is the active material; the dark blue is the permanent magnet; the grey ones are the yoke. The red bar on top signifies a *boundary condition on the temperature field.* During heating, it is "turned on", and is given by T_{hot} . During cooling it is "turned off" and is given by T_{cold}. The latter temperature also defines the boundary condition at all other outer boundaries of the configuration. Not that $p = 0$ is set as the reference pressure. We might as well have chosen $p = 1atm$, but in our simulation it makes no difference.

sections.

Technical estimation

For parameters, design variables and objective functions, we refer to table 1. Note that the geometric variables refer to all those shown in figure 2. We include also the dependent variables used as intermediate outputs of the model, which constitute inputs to other modules of it. If we were to explain everything, we would have to write twice as many pages, and have therefore described certain physical processes and variables/parameters a bit vaguely. We hope to make some of this more clear in the following assignments

Inequality constraints

(1) Total volume taken up by the device must fall below a certain threshold (it has to fit in a car. Maybe this can eventually be relaxed, so that the device only applies for large cars, or maybe for non-automobile applications, such as energy recouping at power plants) $(h_{yk} + h_{fc}) \cdot (2 \cdot w_{yk} + w_{\text{gap}}) \leq V_{\max}$

(2) We mention that the total weight of the device should perhaps not exceed a certain threshold either - one could imagine that the car would then get too heavy. This is closely coupled to the total volume however, since the densities of the raw material probably do not vary *too* much. Furthermore, in modern cars, the total volume will probably become a limiting factor long before total weight. In addition to the inequality constraint on volume, singular dimensions of the device should not exceed certain limits. For instance, even if the volume remains the same, could you imagine a device that is $10m$ long fit into a regular car? Probably not. Hence come the following inequality constraints,

(3) $(h_{yk} + h_{fc}) \leq L_{\max}$

(4) $(2 \cdot w_{yk} + w_{gap}) \leq L_{\max}$

We also say that we can not have material that *overlaps*. This creates the following constraint,

(5) $h_{yk} \geq h_{ac} + h_{pm}$ This says that the total height of the yoke has to be large enough to fit the combined heights of the active material and the permanent magnet. If this were not true, either the latter two would have to overlap, or we would need to change the fundamental design scheme.

Equality constraints

As of yet, we have not set any equality constraints. We do however "prophesize", that at some later stage we decide to turn the equality constaint, $(h_{yk} + h_{fc}) \cdot (2 \cdot w_{yk} + w_{\text{gap}}) \leq V_{\max}$ into an equality constraint, $(h_{yk} + h_{fc}) \cdot (2 \cdot w_{yk} + w_{\text{gap}}) = V_{\max}$, the reason being that more "room to work with" will *probably* be better. This assumption might turn out to be wrong though[7].

Bounds

We are formulating this problem in such a way that we do not have a lot of bounds. We have some constraints which could *sort of* be considered as bounds though, simply because they are very simple. These are the geometric constraints that the total "width" and "height" of the system can not exceed certain thresholds. If the width or height of all components except one becomes very small though, then we effectively have an upper bound on the remaining width or or heights.

We also have the bounds on every single continuous design variable - whether that be pump pressure or the width of the active material - that it can not be negative, which does not make physical sense.

For the discrete variables, *choice of active material* and *choice of intermediate fluid*, we have bounds in the sense that we only have a given selection to take from [8]. We have not yet settled on which fluids and solids to try out, but are very much aware that a long array of properties affiliated with the given substance will be relevant [9]

[7]Perhaps because more pump would be required, for instance

[8]One might even say, "there are only that many elements in the periodic system"

[9]For the intermediate fluid for example, both the magnetic permeability and the mechanical properties will be important. More on that later.

Symbol	Nomenclature	Unit	Type
Geo	Geometric parameters	$[m]$	Design var.
ΔP	Forced Pressure Difference	$[Pa]$	Design var.
M_F	Choice of Intermediate Fluid	$\times$	Design var.
M_A	Choice of Active Material	$\times$	Design var.
P_O	Power Output	W]	Objective
η	Efficiency	[1]	Objective
$\mathcal{C}$	Cost	[\$]	Objective
V	System Volume	$[m^3]$	Constraint
$\mathcal{C}$	Various geometric figures	[m]	Constraint
$\mathbb{C}$	Materials Cost	[\$]	Parameter
η_P	Pump Efficiency	[1]	Parameter
μ	Active Material Permeability	$[H/s]$	Parameter
T_{cold}	Temp. of the environment (300 K)	$[K]$	Parameter
T_{hot}	Temp. Maintained by Heat Source	$[K]$	Parameter
v_{fluid}	Velocity of Fluid	$[m/s]$	Dependent var.
B_{field}	The Magnetic Field	$[T]$	Dependent var.
$\mathbb{B}_{ind}$	Mag. Field Induced by Coil Current	$[T]$	Dependent var.
M_{field}	Magnetization field	$[A/m]$	Dependent var.
T_{outlet}	Temp. at Flow Channel Outlet	$[K]$	Dependent var.
P_{pump}	Pump Power Consumption	$[W]$	Dependent var.
P_{elec}	System Electrical Power Output	$[W]$	Dependent var.

Table 1: The design table for the thermal-magnetic generator applied to MDO. Note that different types of variables are marked with different colors. Variables are abbreviated as var.; Temperatures are abbreviated as temp. to save spaces.

Goal

Designing a thermo-magnetic generator *used in automobiles*. For a given volume[10], we want to minimize the total cost of the material used[11], maximize the total *effective*[12] power output[13], and maximize the efficiency[14]. It is emphasized that a competition between the latter two runs very deep and that they could never both reach their respective optima jointly.

[10]this volume being an estimate of what could fit in a car

[11]It turns out that the material used in this kind of system is very expensive and the main factor limiting the commercial potential

[12]as it was defined previously

[13]to be understood as the amount of power that is extracted from the system per *time*

[14]We will define to be the power output divided by the rate of energy loss as heat is ejected from a control volume around the whole system

Current status and outlook

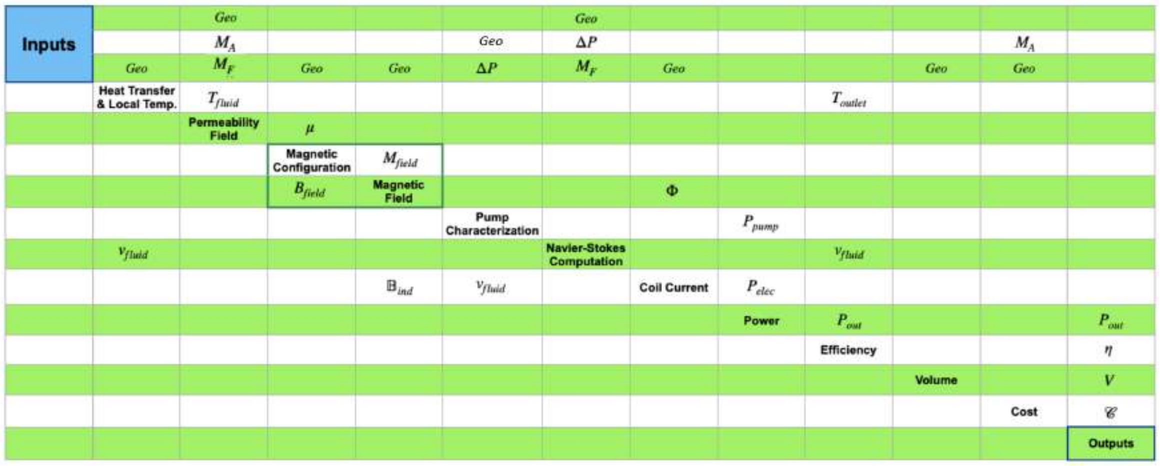
We have clearly defined the problem and the design and variables and parameters. We are still debating how complicated a physical model we want to create. We have defined the system boundary and already realized that we can only hope to optimize the efficiency and power output times some constant, and will not produce the actual numbers, which would have to be calculated experimentally.

We have also come very far in terms of simulation of the magnetic field and the magnetization of the different materials. A numerical model has been created, which can calculate the total flux through the coil at a given magnetic permeability of the active material. Much thought has also been given to the subject of fluid dynamics, although the convection of heat is still a matter to be researched. Another major difficulty will probably be the extraction of different simulation results from different COMSOL[15] models so that they can be analyzed in conjunction. Especially when it comes to evaluating that the temperature at no place in the model exceeds or falls below a certain threshold, there might be trouble ahead.

By the end of the semester, we hope to have spent most of our time on the actual optimization, and not the physics. We hope that we will have created a model that is closely enough related to reality so as to be a meaningful subject of optimization. Conducting a successful optimization of whatever design/function space we end up with, is however our main aim.

---

[15]or maybe some other simulation tool



Inputs		*Geo*				*Geo*						
		M_A			*Geo*	ΔP					M_A	
	Geo	M_F	*Geo*	*Geo*	ΔP	M_F	*Geo*			*Geo*	*Geo*	
	Heat Transfer & Local Temp.	T_{fluid}							T_{outlet}			
		Permeability Field	μ									
			Magnetic Configuration	M_{field}								
			B_{field}	Magnetic Field			Φ					
					Pump Characterization			P_{pump}				
	v_{fluid}					Navier-Stokes Computation			v_{fluid}			
				$\mathbb{B}_{ind}$	v_{fluid}		Coil Current	P_{elec}				
								Power	P_{out}			P_{out}
									Efficiency			η
										Volume		V
											Cost	$\mathscr{C}$
												Outputs

Figure 3: The N^2 diagram for the TMG system.

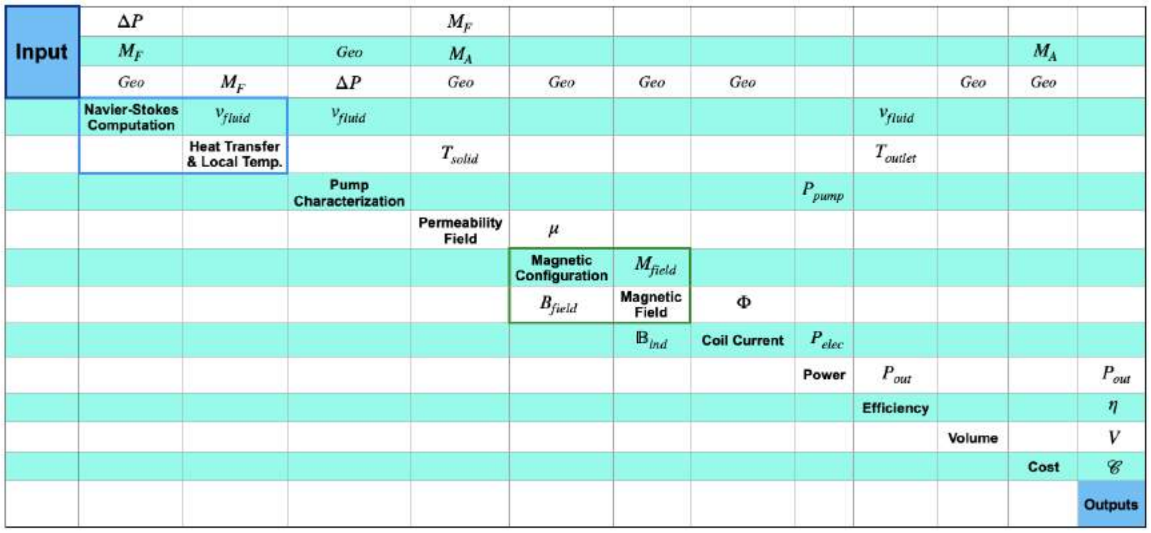
Q2. Coupling and N^2 Diagram

For the problem that you have chosen, identify the modules (see guidelines from Lecture 3), and identify the inputs and outputs for each module. For simplicity, limit the number of modules to about seven, plus or minus two (7+/-2) at this point.

For the original and rearranged N^2 diagram please refer to Figure 3 and Figure 4. As is clearly indicated by the two diagrams, huge progress was made as we went from the initial random order to the rearranged diagram.

Q3. Block diagram

Sketch a block diagram that shows how the modules from your previous answer to **(part b, Q2)** work together and how you would wrap a trade space exploration tool or optimizer around your simulation model. You don't actually have to implement this (yet). That will happen in assignment A2.



Input	ΔP			M_F								
	M_F		Geo	M_A							M_A	
	Geo	M_F	ΔP	Geo	Geo	Geo	Geo			Geo	Geo	
	Navier-Stokes Computation	v_{fluid}	v_{fluid}						v_{fluid}			
		Heat Transfer & Local Temp.		T_{solid}					T_{outlet}			
			Pump Characterization					P_{pump}				
				Permeability Field	μ							
					Magnetic Configuration	M_{field}						
					B_{field}	**Magnetic Field**	Φ					
						$\mathbb{B}_{ind}$	**Coil Current**	P_{elec}				
								Power	P_{out}			P_{out}
									Efficiency			η
										Volume		V
											Cost	$\mathscr{C}$
												Outputs

Figure 4: Rearranged N^2 diagram for the TMG system. You will notice the green box drawn around the "magnetic part", that will be done in one simulation, and the blue box around the "convective heat transfer part, that will be done in another.

For the block diagram please refer to Figure 5. Here, parameters are in yellow, design variables in red, objective functions in blue, and computational modules in green. Note that the *Number of turns* is very much in parenthesis. In this block diagram we have included *nearly* all the of the relevant physical processes. We show this and emphasize that cuts will be made and the whole analysis simplified. As this is done, *Number of turns* will overwhelmingly likely become irrelevant. In purple we have constraints, which are all geometric and included in *system volume*, although some of them are in fact one-dimensional and not two-dimensional.

As for the trade exploration, we have not settled on any method yet, but we think it might be useful try out certain fixed combinations of design variables in the "blue" and "green" modules[16], *only after having optimized design variables of these modules internally*.

[16]as indicated on the N^2 diagram

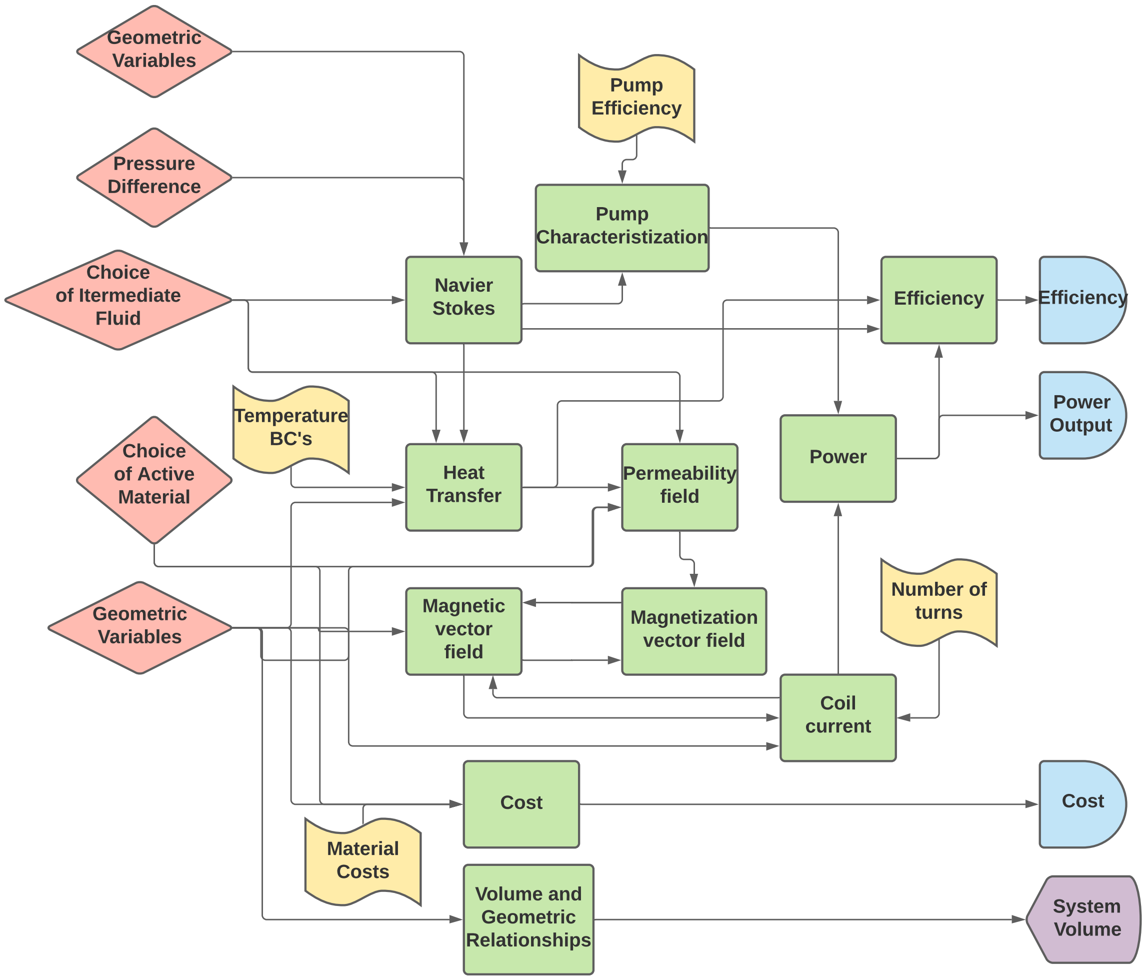


Figure 5: The block diagram for the TMG system.

References

[IPCC Report] Global Warming of 1.5 ºC. IPCC Report. URL: https://www.ipcc.ch/sr15/.

[U.S. Energy Information (a)] U.S. Energy Information Administration. Wind explained: Electricity generation from wind. (link)

[Region, and Segment Forecasts] Battery Market Size, Share & Trends Analysis Report By Product (Lead Acid, Li-ion, Nickle Metal Hydride, Ni-cd), By Application (Automotive, Industrial, Portable), By Region, And Segment Forecasts, 2020 - 2027.

[World Nuclear Energy Assoc.] World Energy Needs and Nuclear Power. World Nuclear Energy Association. (link).

[MMTA, 2016] Minor Metals Trade Association. Clean Energy Runs on Magnets. (link)

[Matizamhuka, 2018] Matizamhuka, M. (2018) The Impact of Magnetic Materials in Renewable Energy-Related Technologies in the 21st Century Industrial Revolution: The Case of South Africa. *Advances in Materials Science and Engineering*, vol. 2018, Article ID 3149412, 9 pages.

[Gutfleisch *et al.*, 2011] O. Gutfleisch, M. A. Willard, E. Brück, et al. Magnetic materials and devices for the 21st century: stronger, lighter, and more energy efficient. *Advanced Materials*, vol. 23, no. 7, pp. 821–842, 2011.

[U.S. Energy Information (b)] U.S. Energy Information Administration. Electricity explained: Magnets and electricity. (link)

[Kittel, 1986] Kittel, C. (1986) Introduction to Solid State Physics. John Wiley & Sons. ISBN 0-471-87474-4.

[Tesla, 1889] Tesla, N. Thermomagnetic motor. US Patent 396,121 (1889).

[Edison, 1892] Edison, T. A. Pyromagnetic generator. US patent 476,983 (1892).

[2019] Waske, A., Dzekan, D., Sellschopp, K. et al. (2019) Energy harvesting near room temperature using a thermomagnetic generator with a pretzel-like magnetic flux topology. *Nat Energy* **4**, 68–74.

[Agte *et al.*, 2009] Agte, J., de Weck, O., Sobieszczanski-Sobieski, J., Arendsen, P., Morris, A., & Spieck, M. (2009). MDO: assessment and direction for advancement—an opinion of one international group. Structural and Multidisciplinary Optimization, 40(1-6), 17–33.

[Dan'kov *et al.*, 1998] Dan'kov, S. Y., Tishin, A. M., Pecharsky, V. K., & Gschneidner, K. A. (1998). Magnetic phase transitions and the magnetothermal properties of gadolinium. Physical Review B, 57(6), 3478–3490.

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

Mads Berg[*], Will Hintlian[†], Hanfeng Zhai[‡], Jiayi Cao[§], Kevin Pan[¶]

[a] *Sibley School of Mechanical and Aerospace Engineering, Cornell University*

[b] *School of Applied & Engineering Physics, Cornell University*

October 7, 2021

1 Model Implementation

1.1 Module decomposition and analysis

The problem formulation is simplified as it was last described considerably. The fluid dynamics module is eliminated since it does not strongly variate the energy generation of TMG, and the heat transfer is facilitated purely by conduction, as the active material is in direct contact with an infinite heat source with a fixed temperature (the engine) (as shown in figure 4). Inspired by a bit of sporadic research, we decide that our particular engine is running at $400K$. It is running in an environment of $293.15K$.

We have created 2 numerical models which have to be run separately and then analyzed in conjunction in order to spit out the expression for predicted power output, and efficiency. We have also created a computer program which returns the total cost of the whole thermo-magnetic system, given a design vector.

The first numerical model we created was the magnetic one. It is implemented in COMSOL Multiphysics and will be able to spit out the total magnetic flux passing though our coil for

[*][b]Email: mpb99@cornell.edu
[†][a]Email: wth42@cornell.edu
[‡][a]Email: hz253@cornell.edu
[§][a]Email: jc2732@cornell.edu
[¶][a]Email: kp428@cornell.edu

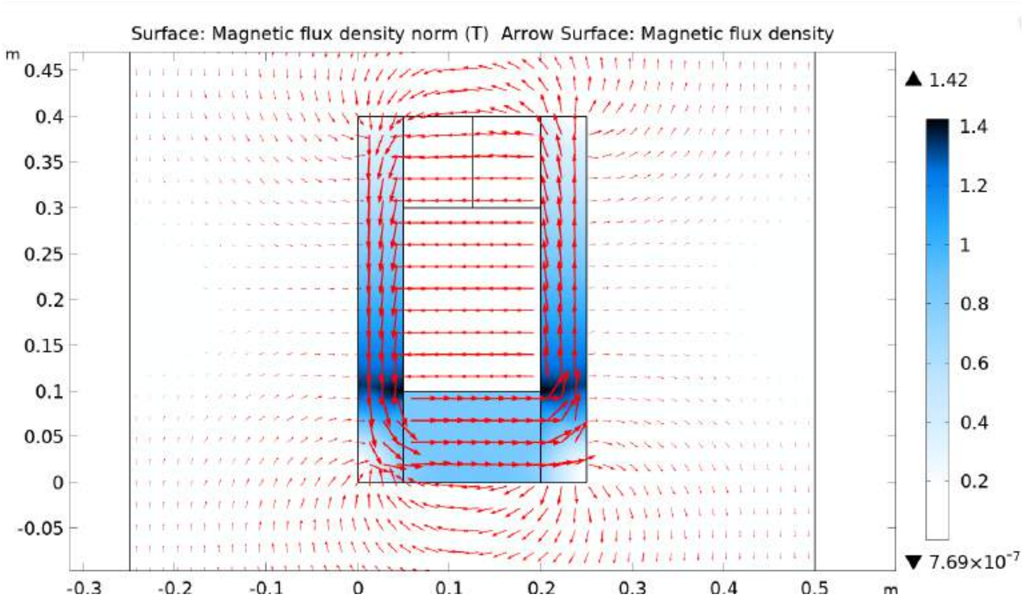


Figure 1: The magnetic distribution for changed geometry for hot active material (Gd).

any set of geometric parameters, or materials. We programmed an adaptive geometry, so that any input design vector can immediately translate into a fully functional model. This has been tested and has turned out to work. Let us demonstrate this for the same design vector as our initial guess in Q2. In figure 1 and figure 2, we show a graphical representation of the output. The red arrows represent the direction of the magnetic field, and the blue density plot shows the magnetic flux density. In figure 1 the active material is above the Curie temperature, and in figure 2 it is below the Curie temperature. The magnetic distinction between the two are realized by changing the magnetic permeability from 1 to 20, which represents a transition from non-magnetic to ferromagnetic which turns out to be as good as total, and certainly good enough for the purposes of this project. It very clear how the active material changes from being non-guiding to guiding, when the temperatures drops.

These results are exactly what we expected. Since the field is still emerging, there is not a whole lot of data that can be used for validation. I happen to have worked with the same physics before though, and have validated it by means of mesh convergence testing, at least. This, in conjunction with the fact that the same qualitative behaviour as we expected is produced, leads us to consider the model validated enough for further studying.

The second numerical model computes the heat distribution as a function of time. We here look at the time it takes to heat up the active material and take that as an expression

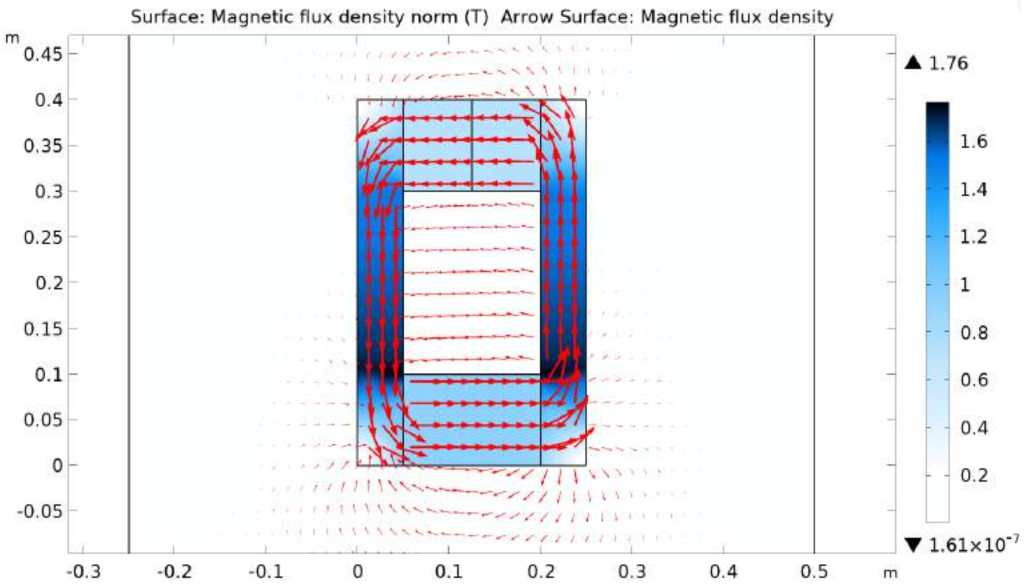


Figure 2: The magnetic distribution for changed geometry for cold active material (Gd).

for the total time of a thermal cycle[1], since it is basically the same process as the cooling, only in reverse. We are thus implicitly assuming that there is a delay between the switching of the heating and cooling processes, which is sufficiently long to ensure that a more or less uniform temperature distribution has been created at the onset of the cooling process. This may or may not be a good approximation, but we have had to restrict ourselves, and of course, the problem is just being redefined to one where cycling of cold and hot matter in contact with the device has not been optimized from the start itself. In conclusion, we are assuming that the total heating/cooling time scales linearly with the time it takes to heat up the active material from a uniform temperature distribution equal to that of the ambient surroundings. For an optimized cycling mechanism, this would not be the case, but would likely be close to it. In another project, the procedure for cooling/heating initialization might be very interesting to optimize too. We have programmed a stop condition which evaluates the geometric domain of the active material and finds the minimum temperature. When this minimum temperature is above $310K$, the whole active material is well above a complete transition to being non-ferromagnetic, and the stop criterion is activated and gives us the time it took to reach that point. As emphasized in the previous report, we have not set our hearts at optimizing the temperature to which one would want to heat up, simply because of the increased physical complexity which comes about from having to couple the exact

[1]both heating and cooling

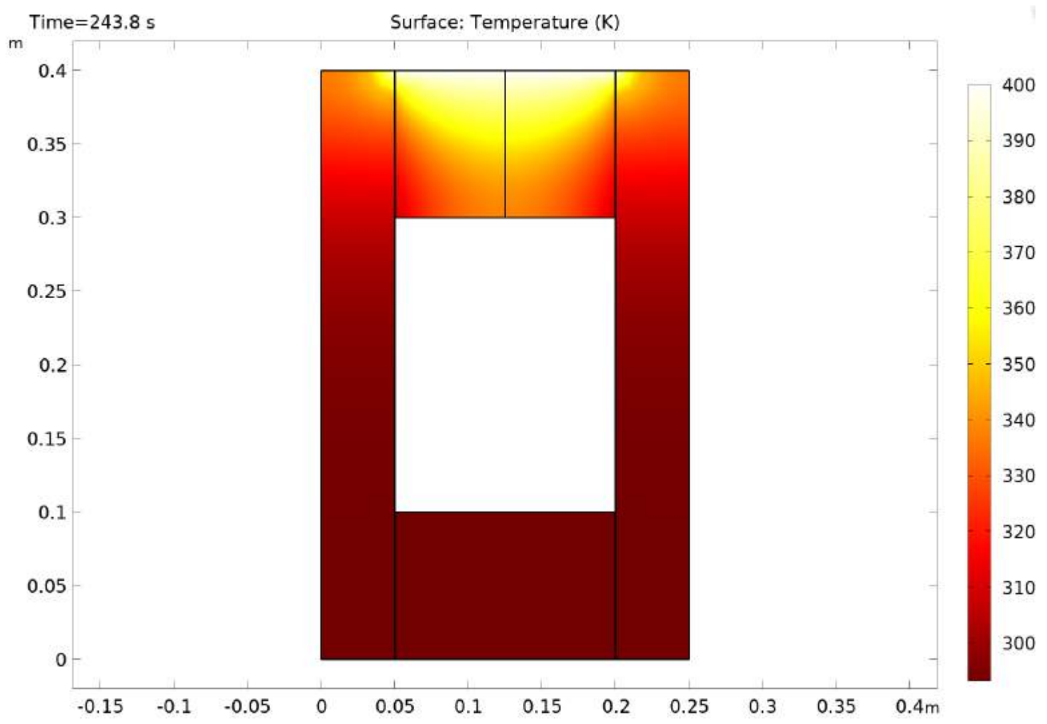


Figure 3: The temperature distribution for the changed shape, with an initial heat flux on the active material.

temperature distribution with a "magnetic permeability field", which we would then have to introduce in the geometric domain of the active material, and define based on experimental data on the temperature-dependence hereof[2].

Figure 3 shows the distribution of temperature at the time of activation of the stop condition. Validity of this model stands with the validity of the heat-diffusion equation. As was the case with the magnetic simulation, the physics itself[3] is well tested, and the results are backed up by intuition. This is just a heating process, and that it should take $243.8s$ to heat up a material with the indicated dimensions from $293.15K$ to a temperature distribution such as the one seen, is very reasonable considering that the material is in fact, metallic, and should thus have a very high conductivity. On that note, let it also be said that all the fundamental physical parameters[4] have been found in COMSOL's own library.

The power output of the system has been determined to be proportional to the total magnetic flux going though the active material raised to the power of two. Actually, let us just make the reasoning for this a little bit more clear. We can write the power output P:

[2] which is rare by the way
[3] the governing PDE's
[4] such as the coefficient of heat transfer

$$P = IV \tag{1}$$

Assuming the coil around the active material to be completely ohmic,

$$V = IR \quad \rightarrow I = \frac{V}{R} \tag{2}$$

Hence, substituting Eq. 2 in the Eq. 1 we have

$$P = \frac{V^2}{R} \tag{3}$$

V is induced by the electromotive force of the changing magnetic field, and hence, Faraday's law gives us,

$$V = \varepsilon = -N\frac{\Delta\Phi}{\Delta t} \tag{4}$$

The induced current will however in turn, produce a magnetic field in the opposite direction[5]. This is proportional to I, which in turn is proportional to the change of magnetic flux. We combine this effect with that of the electrical resistance, R, and the number of turns, N, and throw it all into a constant, K. We do not calculate K explicitly, but simply note that the power is proportional to $\Delta\Phi^2 \cdot \frac{1}{\Delta t}$. For optimization, we do not actually care what this value is. Interestingly, we can optimize our system without ever knowing exactly what the objective function is! We simply express it in "units" of K.

$$P = K \cdot \Delta\Phi^2 \cdot \Delta t^{-1} \tag{5}$$

The efficiency is taken as, $\frac{E_{out}}{Q_{in}}$. We just need to look at a single cycle. The total heat put into the system (and ejected again), will be taken as $Q_{in} = \int_{\delta V} C_V \cdot (T - 293.15K)\mathrm{d}V$. We simply take the added temperature and multiply with the energy associated with that. This is done for every infinitesimal point in the structure and it is all added together. C_V is kept within the integral, as the different components of the structure have different heat capacities.

[5]Lenz's law

$$\eta = \frac{E_{out}}{Q_{in}} = \frac{1}{Q_{in}} \frac{\Delta\Phi^2}{\Delta t} \cdot \Delta t \tag{6}$$

$$\eta = \frac{\Delta\Phi^2}{\int_{\delta V} C_V \cdot (T - 293.15K)\mathrm{d}V} \tag{7}$$

In this derivation, we have stuck with Δt and $\Delta\Phi$ for the total change in t and Φ over half a cycle. E_{out} was thus found simply by multiplying P by t. This approach should generalize to incremental changes in t and Φ. Either way, it is clear that the efficiency too can be found to be proportional to a (somewhat) simple expression. This time, we simply get,

$$\eta = G \cdot \frac{\Delta\Phi^2}{\int_{\delta V} C_V \cdot (T - 293.15K)\mathrm{d}V} \tag{8}$$

So for computing the efficiency, we do not need to worry about the time involved. We simply look at the total difference in magnetic flux of half a cycle and the temperature difference of half a cycle. We do (in principle) need to multiply the whole thing by 2 since the heating of half a cycle corresponds to the heat spent on a full cycle. During that same time the total magnetic flux will have changed two times. Anyway, this is of course just collected in the constant K anyway. The expressions for P and η given in equation 5 and 8 are the ones that we optimize in this project. Henceforth, the factors K and G will implicitly be multiplied on.

1.2 Cost module

The team gathered data on the cost of key materials used in our design and built a MATLAB script ”`costModule.m`” to evaluate the objective Cost for an individual or array of given design vectors. The cost data is summarized in Table 1 and ”`costModule.m`” is included in Appendix 1.2. Cost is computed by multiplying the area of each component by the cost of its material. Because the team is designing around a 2D model, we must note the true meaning of the units in our objective Cost variable. The pricing data gives cost in terms of cubic meters, but the combination of our geometric variables yields an area in square meters. Therefore, in computing $/m^2 from a rate given in $/m^3, the team recognizes that the objective Cost variable will have units of $/m ”into the page.” This is consistent with our methods in other sections of our analysis.

Material	**Cost**
Iron [Ref.]	1.4804e3 \$/m^3
Neodymium [Ref.]	1.628e5 \$/m^3
Gadolinium [Ref.]	1.71553e5 \$/m^3

Table 1: The price of the materials applied in the simulations.

Feasibility

In preparation for creating a Design of Experiments as well as more rigorous simulation and optimization later on, the team created a script ”`geometricConstraints.m`” in MATLAB to evaluate the feasibility of any design vector based on geometric constraints. Currently, all of the team's design variables are geometric. The team defined a set of equations which must all be true for a design vector to be physically valid. Equations 9-12 define the relationships required between geometric design variables shown in Figure 4 for a given input to be valid. The team established geometric constraints at this time. The maximum volume (area) of our design is 0.125 m^2. The maximum length of the design is 0.5 m. These constraints are mostly arbitrary at this time. While they do yield a ”reasonably” sized design, the team may adjust them later on as details of the design and optimization process begin to accumulate.

$$h_{yk}(2w_{yk} + w_{gap}) > V_{max} \tag{9}$$

$$h_{yk} > L_{max} \tag{10}$$

$$2w_{yk} + w_{gap} > L_{max} \tag{11}$$

$$h_A + h_{pm} > h_{yk} \tag{12}$$

The code to check the geometric feasibility of a design vector is included in Appendix 1.2. The code is written as a function so that it may be used by other scripts to assist with our Design of Experiments and optimization algorithm later on. Using the function, we verified an initial design vector shown in Table 2 for use with our model implementation. The material used here is iron for the yoke and neodymium for the permanent magnet. The active material is Gadolinium. For now, these material choices are more or less considered parameters, but we could potentially change them to variables later on.

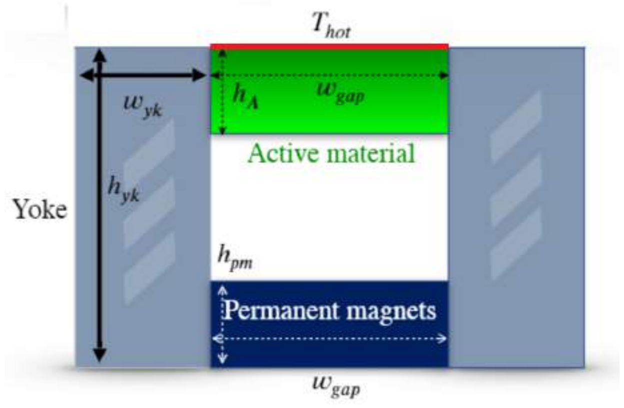


Figure 4: The schematic view for our setup of the thermal-magnetic generator (TMG) as illustrated in the last HW.

Variable	**Abbreviation**	**Level**
Yoke Width	w_{yk}	0.05 m
Yoke Height	h_{yk}	0.4 m
Permanent Magnet Height	h_{pm}	0.1 m
Active Material Height	h_A	0.1 m
Gap Width	w_{gap}	0.15 m

Table 2: Initial design vector for the TMG experiments.

Design of Experiments

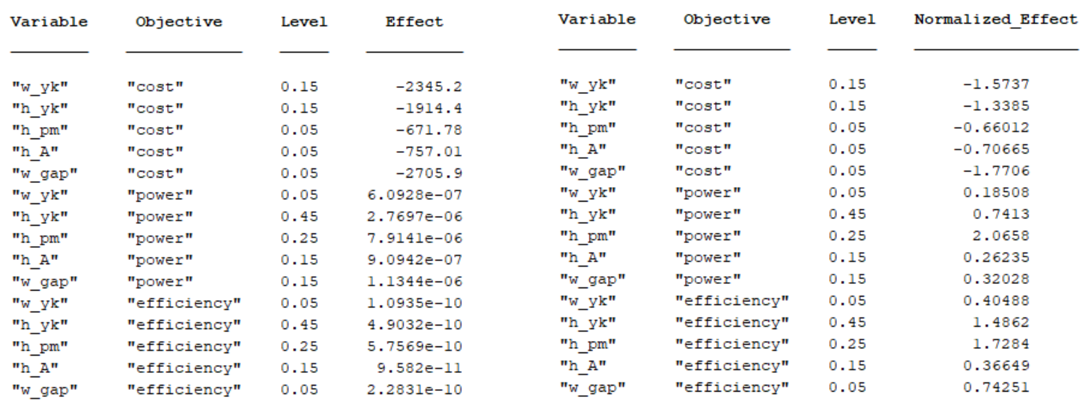
The team carried out a Design of Experiments exercise to explore the design space and evaluate how different objective variables are affected by changes in specific design variables. Although the team has not yet fully integrated their COMSOL models with MATLAB to run autonomously, the team identified a method of running many simulations in series within the COMSOL interface. This capability combined with the observation that each simulation only took 30s-60s to run led the team to perform a "full factorial" design of experiments. The team wrote a function "`designVectorBuilder.m`" taking inputs of lower bound, step size, and upper bound to generate vectors based on every combination of design variables between the range of the lower and upper bounds. The function is included in Appendix 1.2. Next, the matrix of design vectors is fed into the geometric constraint function, which evaluates the feasibility of each input. "`designVectorBuilder.m`" then returns a full factorial matrix of valid design vectors with help from "`geometricConstraints.m`". Since our design space is continuous, the step size does limit the number of design vectors used in our "full factorial" experiment. For our DoE, we used a lower bound of .05 m, step size of .1 m, and upper bound of .05 m for each design variable. The design vector building function initially created 3125 combinations of input variables, but reduced them down to only 68 vectors which met the design constraints. The team exported the matrix of valid design vectors into a reformatted .txt file which could be directly imported into COMSOL.

After running the simulations, the team exported data from COMSOL and manipulated it further in Excel to achieve the desired numeric outputs. Later, the team plans to fully automate this manipulation in MATLAB as it was fairly time consuming and not intuitive to look at. The team wrote the function "`initialDoE.m`" to carry out the combined analysis of the full factorial experiment. The script uses "`designVectorBuilder.m`", "`costModule.m`", and two helper functions (for importing data) to build the DoE. The full "`initialDoE.m`" script is included in Appendix 1.2. Through a series of loops the script computes the effect of every design variable at every level on each objective outcome. The full lists of effects are stored as matrices in the MATLAB workspace, but the script has two outputs. First, the script returns a table showing the variable, level, and effect associated with the greatest effect for that variable on each objective outcome shown in Figure 5. Due to the methods used to compute objective variables, the scale of effects may not be intuitive for some objectives. This is a result of scaling constants which are associated with our Power and Efficiency calculations but not computed in our model at this time. In order to better show the relative effects of each design variable, the script returns a second table identical to the first but with normalized effects for each objective shown in Figure 5. The variable and level pairs in Figure 5 represent initial X* design vectors which we can recommend for optimization in the direction of each of our objectives based on our analysis. Notably, there is not a single instance of any variable and level pair causing the greatest effect for all of our objective outcomes at once. This suggests that our objectives may be difficult to achieve simultaneously and makes our problem an especially strong candidate for multidisciplinary optimization later on in the project.

Variable	Objective
w_{yk}	Cost
h_{yk}	Cost
h_{pm}	Cost
h_A	Cost
w_{gap}	Cost
w_{yk}	Power
h_{yk}	Power
h_{pm}	Power
w_{gap}	Power
h_A	Power
w_{yk}	Efficiency
h_{yk}	Efficiency
h_{pm}	Efficiency
h_A	Efficiency
w_{gap}	Efficiency

Table 3: The variable table for DoE considering different variables with regards to different objectives. The actual computation results are shown in Figure 5.

.



```
Variable    Objective      Level      Effect
________    ___________    _____    __________

"w_yk"      "cost"         0.15        -2345.2
"h_yk"      "cost"         0.15        -1914.4
"h_pm"      "cost"         0.05        -671.78
"h_A"       "cost"         0.05        -757.01
"w_gap"     "cost"         0.05        -2705.9
"w_yk"      "power"        0.05     6.0928e-07
"h_yk"      "power"        0.45     2.7697e-06
"h_pm"      "power"        0.25     7.9141e-06
"h_A"       "power"        0.15     9.0942e-07
"w_gap"     "power"        0.15     1.1344e-06
"w_yk"      "efficiency"   0.05     1.0935e-10
"h_yk"      "efficiency"   0.45     4.9032e-10
"h_pm"      "efficiency"   0.25     5.7569e-10
"h_A"       "efficiency"   0.15      9.582e-11
"w_gap"     "efficiency"   0.05     2.2831e-10
```

```
Variable    Objective      Level    Normalized_Effect
________    ___________    _____    _________________

"w_yk"      "cost"         0.15          -1.5737
"h_yk"      "cost"         0.15          -1.3385
"h_pm"      "cost"         0.05         -0.66012
"h_A"       "cost"         0.05         -0.70665
"w_gap"     "cost"         0.05          -1.7706
"w_yk"      "power"        0.05          0.18508
"h_yk"      "power"        0.45           0.7413
"h_pm"      "power"        0.25           2.0658
"h_A"       "power"        0.15          0.26235
"w_gap"     "power"        0.15          0.32028
"w_yk"      "efficiency"   0.05          0.40488
"h_yk"      "efficiency"   0.45           1.4862
"h_pm"      "efficiency"   0.25           1.7284
"h_A"       "efficiency"   0.15          0.36649
"w_gap"     "efficiency"   0.05          0.74251
```

Figure 5: Variables and levels of greatest effect on objectives. Raw (left), normalized effects (right).

Appendix

costModule.m: The function to estimate the cost for materials during the optimization process. Note that the price of the materials corresponds to Table 1.

```matlab
function cost = costModule(inputs)

    % function determines the cost of an input vector of geometric
    % parameters. Returns a cost values corresponding to vector
    % inputs.

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

    % extract input geometry
    %h_fc = inputs(1);
    w_yk = inputs(1,:);
    h_yk = inputs(2,:);
    h_pm = inputs(3,:);
    h_A = inputs(4,:);
    w_gap = inputs(5,:);

    % initialize cost output vector
    cost = zeros(1,length(inputs));

    for i = 1:length(inputs)

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

        % compute cost
        cost(i) = Gadolinium*activeMaterialArea + Iron*yokeArea + Neodymium*magnetArea; % $/
    m into the page

    end

end
```

geometricConstraints.m: The function to enforce and encode the geometric constraint through the intermediate varaible `valid`.

```
function valid = geometricConstraints(inputs,V_max,L_max)

    % function determines the spatial validity of an input vector of
    % geometric parameters. Returns 1 if the input is valid. Returns 0 if
    % the input is invalid.

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

    valid = 1;

    if h_yk*(2*w_yk + w_gap) > V_max
        valid = 0;
    elseif h_yk > L_max
        valid = 0;
    elseif (2*w_yk + w_gap) > L_max
        valid = 0;
    elseif (h_A + h_pm) > h_yk
        valid = 0;
    end

end
```

---

designVectorBuilder.m: The function to build up the design vector for further analysis.

```
function designVectors = designVectorBuilder(lb,step,ub)

    % Function returns a set of design vectors which satisfy geometric
    % design constraints

    % declare constraint constants
    V_max = .125;
    L_max = .5;

    % initialize wide ranges for design variables
    w_yk0  = lb:step:ub;
    h_yk0  = lb:step:ub;
    h_pm0  = lb:step:ub;
    h_A0   = lb:step:ub;
    w_gap0 = lb:step:ub;

    % note that this creates a stupidly large vector
    sheet = combvec(w_yk0, h_yk0, h_pm0, h_A0, w_gap0);

    % initialize vector of valid geometric inputs
    validSheet = zeros(5,length(sheet));

    for i = 1:length(sheet)

        input = sheet(:,i);

        % evaluate the geometric configuration based spatial validity
        if geometricConstraints(input,V_max,L_max) == 1
            % valid inputs will be copied to the new sheet
            validSheet(:,i) = input(:);
        end

    end

    % remove zero columns
    designVectors = validSheet(:,any(validSheet,1));

end
```

---

initialDoE.m: The function to initialize the whole design space for design of experiments.

```
%% Perform initial setup: import data, initialize variables

clear
clc
close all

% Generate design vectors
lb = .05;
step = .1;
ub = .45;
designVectors = designVectorBuilder(lb,step,ub);

% 'variables' corresponds to variables w_yk, h_yk, h_pm, h_A, w_gap
variables = 1:5;

% import power data
DoEpowerResults = importPowerfile("DoEpowerResults.xlsx", "Ark1", [7, 74]);

```

```
% import efficiency data
DoEefficiencyResults = importEfficiencyfile("DoEefficiencyResults.xlsx", "Ark1", [85, 152]);

% to help later with checking variable effects
effectLevels = lb:step:ub;

% initialize array to hold cost, power, and efficiency outcomes for each design vector
experimentResults = zeros(3,length(designVectors));

%% Fill in experimental results and compute means

% compute cost outcomes with cost module and add to results
experimentResults(1,:) = costModule(designVectors);

% enter power outcomes computed from COMSOL (later will call for these
% computations through matlab)
experimentResults(2,:) = DoEpowerResults;

% enter efficiency outcomes computed from COMSOL (later will call for these
% computations through matlab)
experimentResults(3,:) = DoEefficiencyResults;

% compute overall mean cost, power, and efficiency outcomes
meanCost = mean(experimentResults(1,:));
meanPower = mean(experimentResults(2,:));
meanEfficiency = mean(experimentResults(3,:));

%% compute effect of design variables on cost

% initialize cost effects matrix [variable, level, effect]
costEffects = zeros(length(variables)*length(effectLevels),3);

% helper variable to ensure that each block of variables, levels, and effects are created in
     the right locations
offset = 0;

for i = 1:length(variables)
    for j = 1:length(effectLevels)

        % set the variable being measured (represented by a number)
        costEffects(j+offset,1) = i;

        % set the variable level of the variable being measured
        costEffects(j+offset,2) = effectLevels(j);

        % compute the effect of the variable at the given level

        % find the column indices of all results where variable i is at the j level
        indices = find(designVectors(i,:) == effectLevels(j));

        jResultSum = 0;

        for k = 1:length(indices)
            % sum the results of cost when variable i is at j level
            jResultSum = jResultSum + experimentResults(1,indices(k));
        end

        % compute average cost when variable i is at j level
        jMean = jResultSum/length(indices);

        % add effect to the effect matrix
        costEffects(j+offset,3) = jMean - meanCost;

    end
    offset = offset + length(effectLevels);

end

%% compute effect of design variables on power

% initialize power effects matrix [variable, level, effect]
powerEffects = zeros(length(variables)*length(effectLevels),3);

% helper variable to ensure that each block of variables, levels, and effects are created in
     the right locations
offset = 0;

for i = 1:length(variables)
    for j = 1:length(effectLevels)

        % set the variable being measured (represented by a number)
        powerEffects(j+offset,1) = i;

        % set the variable level of the variable being measured
        powerEffects(j+offset,2) = effectLevels(j);

        % compute the effect of the variable at the given level

        % find the column indices of all results where variable i is at the j level
        indices = find(designVectors(i,:) == effectLevels(j));

        jResultSum = 0;

        for k = 1:length(indices)
            % sum the results of power when variable i is at j level
            jResultSum = jResultSum + experimentResults(2,indices(k));
        end

        % compute average power when variable i is at j level
        jMean = jResultSum/length(indices);

        % add effect to the effect matrix
        powerEffects(j+offset,3) = jMean - meanPower;

    end
    offset = offset + length(effectLevels);

end

%% compute effect of design variables on efficiency

% initialize efficiency effects matrix [variable, level, effect]
efficiencyEffects = zeros(length(variables)*length(effectLevels),3);

% helper variable to ensure that each block of variables, levels, and effects are created in
     the right locations
offset = 0;

for i = 1:length(variables)
    for j = 1:length(effectLevels)

        % set the variable being measured (represented by a number)
        efficiencyEffects(j+offset,1) = i;

        % set the variable level of the variable being measured
        efficiencyEffects(j+offset,2) = effectLevels(j);

        % compute the effect of the variable at the given level

        % find the column indices of all results where variable i is at the j level
        indices = find(designVectors(i,:) == effectLevels(j));

        jResultSum = 0;

        for k = 1:length(indices)
            % sum the results of efficiency when variable i is at j level
            jResultSum = jResultSum + experimentResults(3,indices(k));
        end

        % compute average efficiency when variable i is at j level
        jMean = jResultSum/length(indices);

        % add effect to the effect matrix
        efficiencyEffects(j+offset,3) = jMean - meanEfficiency;

    end
    offset = offset + length(effectLevels);

end

%% Determine recommended start points X0 for numeric integration

% create copies of matrices with normalized effects
normalizedCostEffects = costEffects;
normalizedPowerEffects = powerEffects;
normalizedEfficiencyEffects = efficiencyEffects;

normalizedCostEffects(:,3) = normalize(costEffects(:,3));
normalizedPowerEffects(:,3) = normalize(powerEffects(:,3));
normalizedEfficiencyEffects(:,3) = normalize(efficiencyEffects(:,3));

% find variables and levels for greatest effect on each output

% best variable/level pairs to minimize cost:

% initialize array to hold the best variables/levels/effects for cost
bestCostVarsLevels = zeros(2,5); % row1 = level row2 = effect

% variable 1 (w_yk)
index = find(costEffects(1:5,3) == min(costEffects(1:5,3)));
bestCostVarsLevels(:,1) = [costEffects(index,2); costEffects(index,3)];
bestNormalCostVarsLevels(:,1) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];

% variable 2 (h_yk)
index = 5 + find(costEffects(6:10,3) == min(costEffects(6:10,3)));
bestCostVarsLevels(:,2) = [costEffects(index,2); costEffects(index,3)];
bestNormalCostVarsLevels(:,2) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];

% variable 3 (h_pm)
index = 10 + find(costEffects(11:15,3) == min(costEffects(11:15,3)));
bestCostVarsLevels(:,3) = [costEffects(index,2); costEffects(index,3)];
bestNormalCostVarsLevels(:,3) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];

% variable 4 (h_A)
index = 15 + find(costEffects(16:20,3) == min(costEffects(16:20,3)));
bestCostVarsLevels(:,4) = [costEffects(index,2); costEffects(index,3)];
bestNormalCostVarsLevels(:,4) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];

% variable 5 (w_gap)
index = 20 + find(costEffects(21:25,3) == min(costEffects(21:25,3)));
bestCostVarsLevels(:,5) = [costEffects(index,2); costEffects(index,3)];
bestNormalCostVarsLevels(:,5) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];

% best variable/level pairs to maximize power:

% initialize array to hold the best variables/levels/effects for power
bestPowerVarsLevels = zeros(2,5); % row1 = level row2 = effect

% variable 1 (w_yk)
index = find(powerEffects(1:5,3) == max(powerEffects(1:5,3)));
bestPowerVarsLevels(:,1) = [powerEffects(index,2); powerEffects(index,3)];
bestNormalPowerVarsLevels(:,1) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];

% variable 2 (h_yk)
index = 5 + find(powerEffects(6:10,3) == max(powerEffects(6:10,3)));
bestPowerVarsLevels(:,2) = [powerEffects(index,2); powerEffects(index,3)];
bestNormalPowerVarsLevels(:,2) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];

% variable 3 (h_pm)
index = 10 + find(powerEffects(11:15,3) == max(powerEffects(11:15,3)));
bestPowerVarsLevels(:,3) = [powerEffects(index,2); powerEffects(index,3)];
bestNormalPowerVarsLevels(:,3) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];

% variable 4 (h_A)
index = 15 + find(powerEffects(16:20,3) == max(powerEffects(16:20,3)));
bestPowerVarsLevels(:,4) = [powerEffects(index,2); powerEffects(index,3)];
bestNormalPowerVarsLevels(:,4) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];

% variable 5 (w_gap)
index = 20 + find(powerEffects(21:25,3) == max(powerEffects(21:25,3)));
bestPowerVarsLevels(:,5) = [powerEffects(index,2); powerEffects(index,3)];
```

```matlab
bestNormalPowerVarsLevels(:,5) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];

% best variable/level pairs to maximize efficiency:

% initialize array to hold the best variables/levels/effects for power
bestEfficiencyVarsLevels = zeros(2,5); % row1 = level row2 = effect

% variable 1 (w_yk)
index = find(efficiencyEffects(1:5,3) == max(efficiencyEffects(1:5,3)));
bestEfficiencyVarsLevels(:,1) = [efficiencyEffects(index,2); efficiencyEffects(index,3)];
bestNormalEfficiencyVarsLevels(:,1) = [normalizedEfficiencyEffects(index,2);
    normalizedEfficiencyEffects(index,3)];

% variable 2 (h_yk)
index = 5 + find(efficiencyEffects(6:10,3) == max(efficiencyEffects(6:10,3)));
bestEfficiencyVarsLevels(:,2) = [efficiencyEffects(index,2); efficiencyEffects(index,3)];
bestNormalEfficiencyVarsLevels(:,2) = [normalizedEfficiencyEffects(index,2);
    normalizedEfficiencyEffects(index,3)];

% variable 3 (h_pm)
index = 10 + find(efficiencyEffects(11:15,3) == max(efficiencyEffects(11:15,3)));
bestEfficiencyVarsLevels(:,3) = [efficiencyEffects(index,2); efficiencyEffects(index,3)];
bestNormalEfficiencyVarsLevels(:,3) = [normalizedEfficiencyEffects(index,2);
    normalizedEfficiencyEffects(index,3)];

% variable 4 (h_A)
index = 15 + find(efficiencyEffects(16:20,3) == max(efficiencyEffects(16:20,3)));
bestEfficiencyVarsLevels(:,4) = [efficiencyEffects(index,2); efficiencyEffects(index,3)];
bestNormalEfficiencyVarsLevels(:,4) = [normalizedEfficiencyEffects(index,2);
    normalizedEfficiencyEffects(index,3)];

% variable 5 (w_gap)
index = 20 + find(efficiencyEffects(21:25,3) == max(efficiencyEffects(21:25,3)));
bestEfficiencyVarsLevels(:,5) = [efficiencyEffects(index,2); efficiencyEffects(index,3)];
bestNormalEfficiencyVarsLevels(:,5) = [normalizedEfficiencyEffects(index,2);
    normalizedEfficiencyEffects(index,3)];

% package the results in a pretty table

Level = [bestCostVarsLevels(1,:)';bestPowerVarsLevels(1,:)';bestEfficiencyVarsLevels(1,:)'];
Effect = [bestCostVarsLevels(2,:)';bestPowerVarsLevels(2,:)';bestEfficiencyVarsLevels(2,:)
    '];
Objective = ["cost";"cost";"cost";"cost";"cost";"power";"power";"power";"power";"power";"
    efficiency";"efficiency";"efficiency";"efficiency";"efficiency";];
Variable = ["w_yk";"h_yk";"h_pm";"h_A";"w_gap";"w_yk";"h_yk";"h_pm";"h_A";"w_gap";"w_yk";"
```

```
    h_yk";"h_pm";"h_A";"w_gap";];
effectAnalysis = table(Variable,Objective,Level,Effect)

Level = [bestNormalCostVarsLevels(1,:)';bestNormalPowerVarsLevels(1,:)';
    bestNormalEfficiencyVarsLevels(1,:)'];
Normalized_Effect = [bestNormalCostVarsLevels(2,:)';bestNormalPowerVarsLevels(2,:)';
    bestNormalEfficiencyVarsLevels(2,:)'];
Objective = ["cost";"cost";"cost";"cost";"cost";"power";"power";"power";"power";"power";"
    efficiency";"efficiency";"efficiency";"efficiency";"efficiency";];
Variable = ["w_yk";"h_yk";"h_pm";"h_A";"w_gap";"w_yk";"h_yk";"h_pm";"h_A";"w_gap";"w_yk";"
    h_yk";"h_pm";"h_A";"w_gap";];
normalizedEffectAnalysis = table(Variable,Objective,Level,Normalized_Effect)
```

Coupled model: A simplified case study

The core of the Thermal-Magnetic Generator (TMG) is always about the energy transformation between the magnetic to electric energy. But thermal convection, or heat transfer, is of importance in the magnetic generation since the permeability of the active materials is heavily influenced by the temperature. Therefore, the study of heat transfer here is still of focus on the magnetic field yet investigating the variables related to the thermal field (or block). Focusing on such a point and hope to provide a benchmark for our DoE we here carried out simplified study with the coupled study of the two modules. As mentioned in the previous homework, we designed a complex system involves interaction between fluid dynamics, heat transfer, magnetic in a complex system. But at a current stage we only start with a very generalized and simplified model as shown in Figure 6, specific outlining the characteristics of the heat transfer module.

Within the model, there is initial heat flux set on the active material, corresponding to the real-world applications that temperature change on active materials causes magnetic changes, which leads to generation of electricity. Here, for soft iron, relative permeability $P_r = 1$; The thermal conductivity $k_\epsilon = 240$ $[W/m \cdot K]$ [Ref.]; The density $\rho = 7000$ $[kg/m^3]$ [Ref.]; Heat capacity at constant pressure $C_{pMag} = 450$ $[J/kg \cdot J]$ [Ref.]. Listed above are properties of soft iron, which are considered as **parameters** based on our previous Assignment.

For the permanent magnets, considering choice of such a material will strongly influence the performance of the system, we re-modify such as a variable. Here, relative permeability $P_r = 1$; The thermal conductivity $k_\epsilon = 500$ $[W/m \cdot K]$ [Ref.]; The density $\rho = 7000$ $[kg/m^3]$

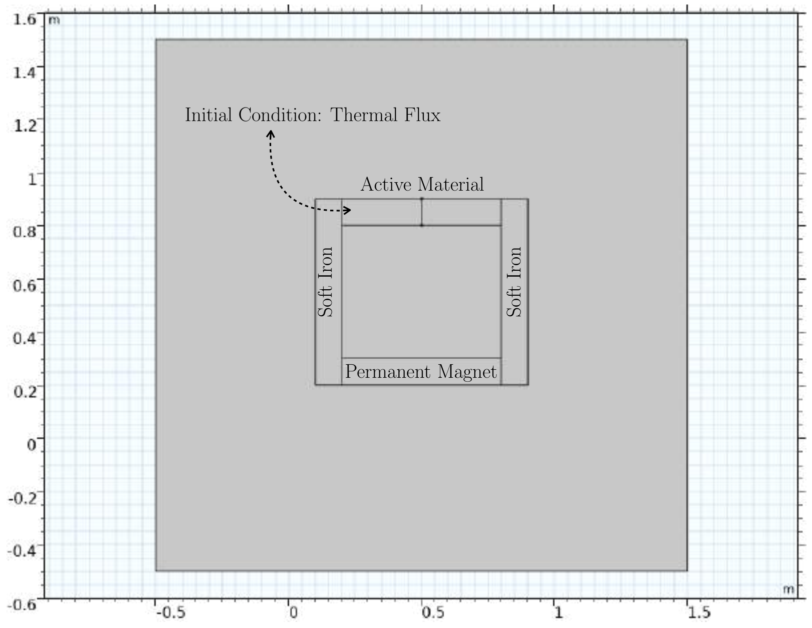


Figure 6: A schematic illustration of the simplied model for the thermal convection module.

[Ref.]; Heat capacity at constant pressure $C_{pAM} = 450\ [J/kg \cdot J]$ [Ref.]. Since the choice of permanent magnets are already a variable, listed above are all considered as **variables**.

For active materials, we choose Gd as the materials, since this is the most commonly used and applied active magnetic material[6]. The Gd is applied with the COMSOL inner material library, where the material properties is inner connected[7]. The initial conditions are set corresponding to Figure 6, where an initial thermal flux are set on the active material, with a linear heat source of $Q_0 = q_s \cdot T$, of $q_s = 500[W/m^3 \cdot K]$ We run the coupled thermal and magnetic modules in a **Time Dependent** general coupling study. Since in the previous section (Sec. 1) we found out that it will take $\approx 240s$ for the system to heat up, thus here we run the simulation for 250s to check how the numerical results look like. For the general thermal-magnetic coupled model, the thermal contour are shown in Figure 7, where the magnetic field

[6]Pyykkö, P. *Nature Chem.* **7**, 680 (2015).
[7]Will be detailed further in future works

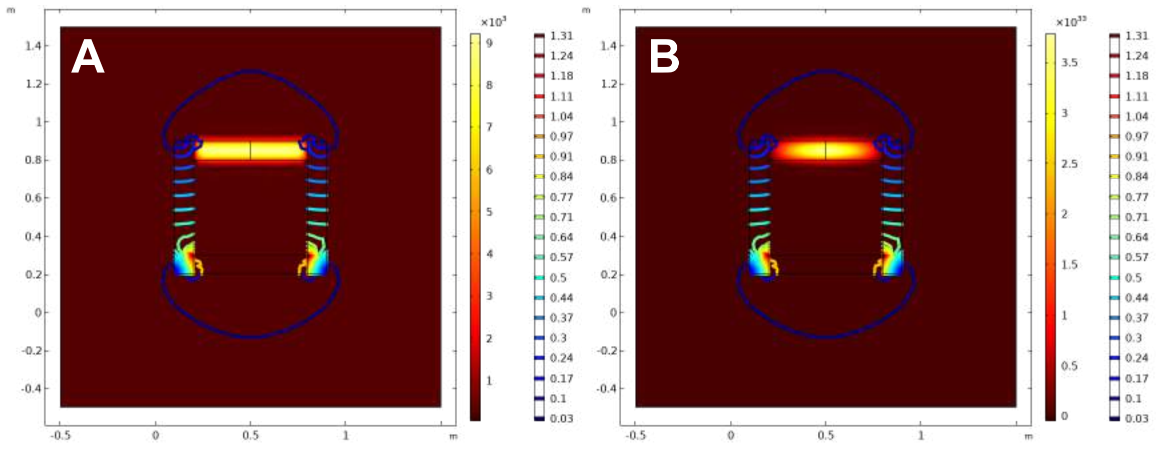


Figure 7: The thermal contour for the coupled model

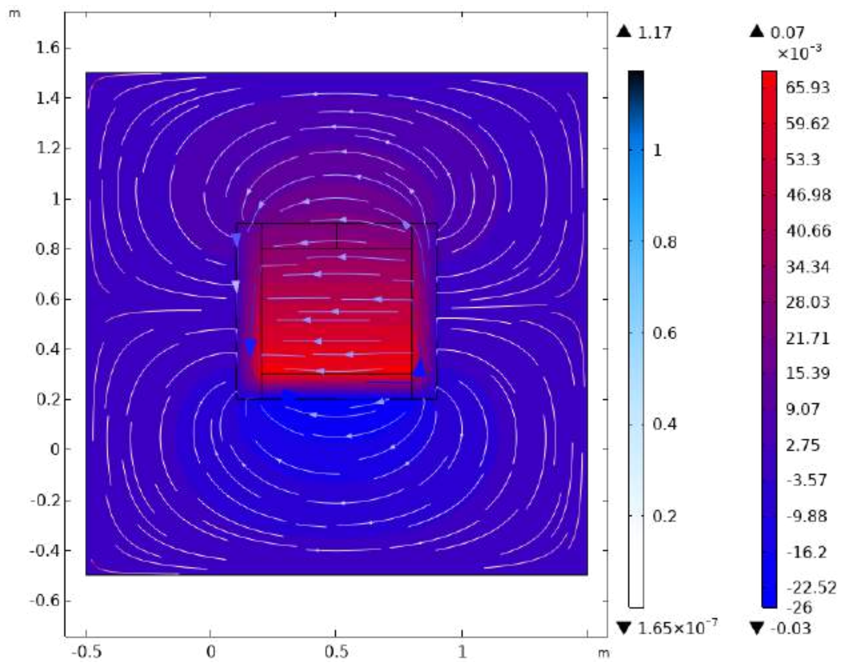


Figure 8: The magnetic flux contour for the coupled model.

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.

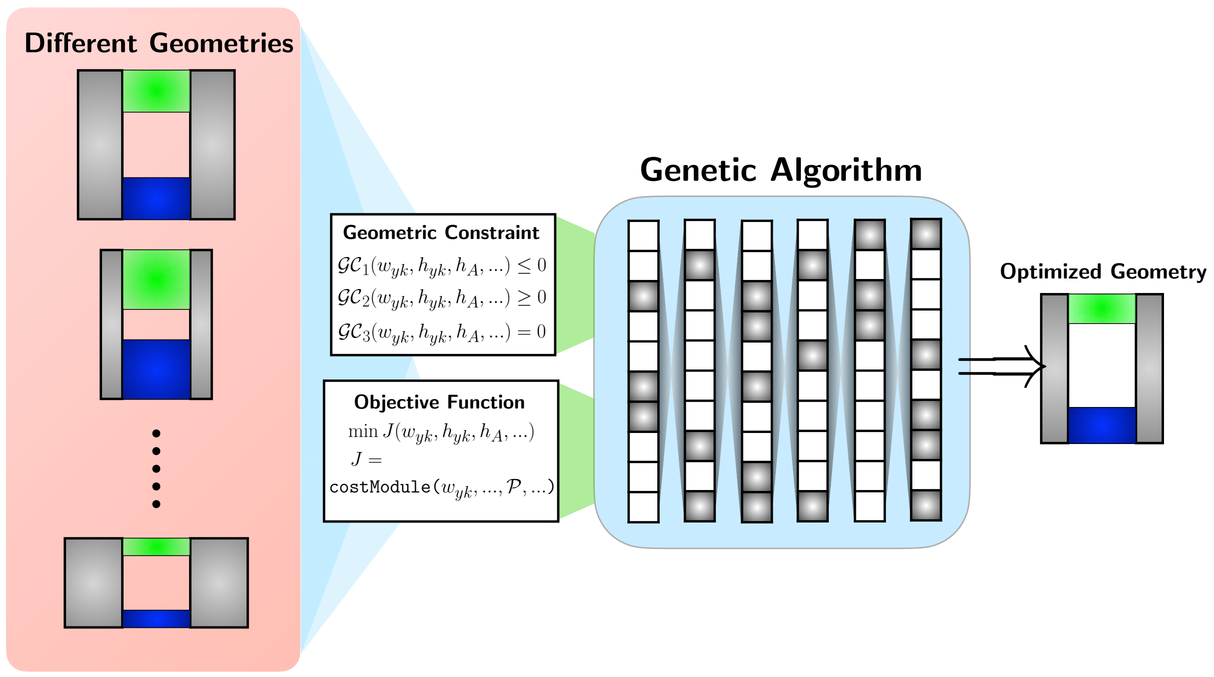

Different Geometries
Genetic Algorithm
Geometric Constraint
$\mathcal{GC}_1(w_{yk}, h_{yk}, h_A, ...) \leq 0$
$\mathcal{GC}_2(w_{yk}, h_{yk}, h_A, ...) \geq 0$
$\mathcal{GC}_3(w_{yk}, h_{yk}, h_A, ...) = 0$
Objective Function
$\min J(w_{yk}, h_{yk}, h_A, ...)$
$J =$
costModule$(w_{yk}, ..., \mathcal{P}, ...)$
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 $[0.05, 0.45]$.

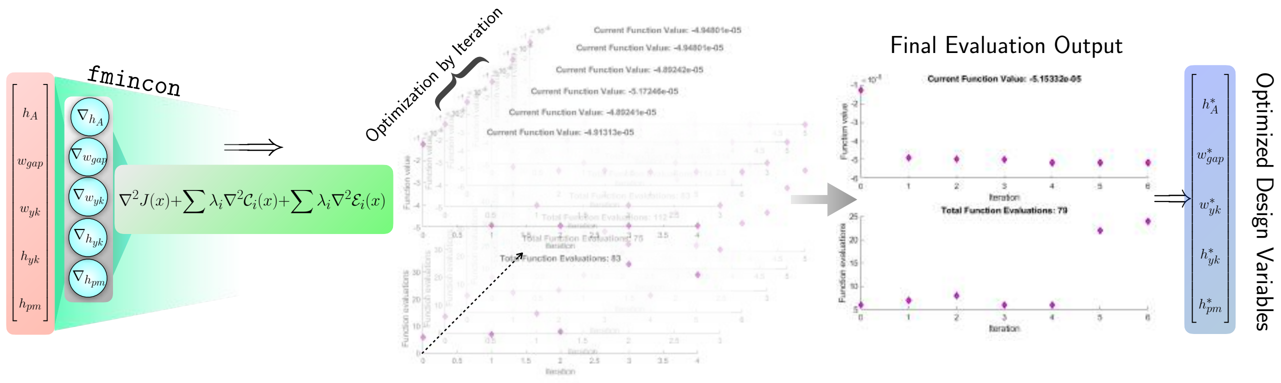

fmincon
h_A
w_{gap}
w_{yk}
h_{yk}
h_{pm}
∇h_A
∇w_{gap}
∇w_{yk}
∇h_{yk}
∇h_{pm}
$\nabla^2 J(x)+\sum \lambda_i \nabla^2 \mathcal{C}_i(x)+\sum \lambda_i \nabla^2 \mathcal{E}_i(x)$
Optimization by Iteration
Final Evaluation Output
Current Function Value: -5.15332e-05
Total Function Evaluations: 79
h_A^*
w_{gap}^*
w_{yk}^*
h_{yk}^*
h_{pm}^*
Optimized Design Variables

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.

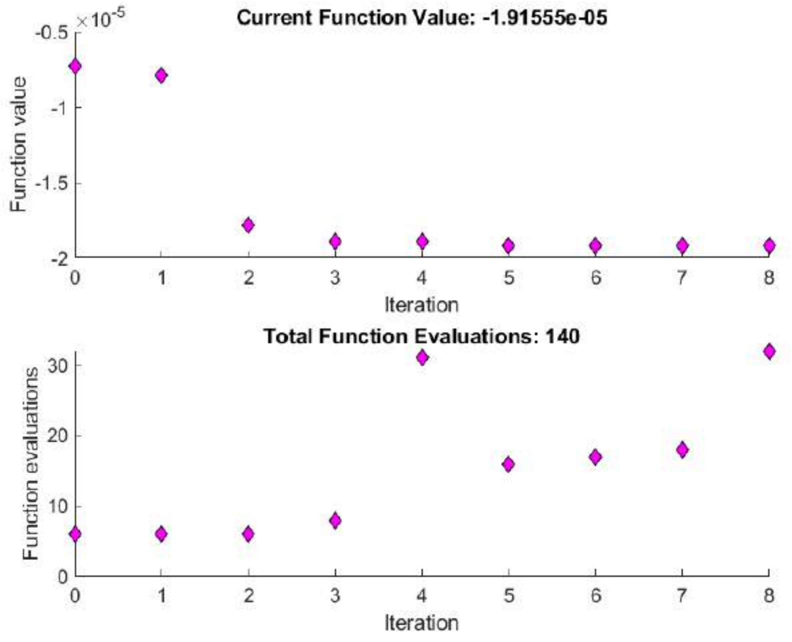


Figure 3: Plot showing the function evaluation at each iteration (top) and the number of function evaluations for the given iteration (bottom) in the case where we do gradient based optimization of the efficiency based on the aforementioned arbitrary design vector as the initial guess. The design space 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.
```

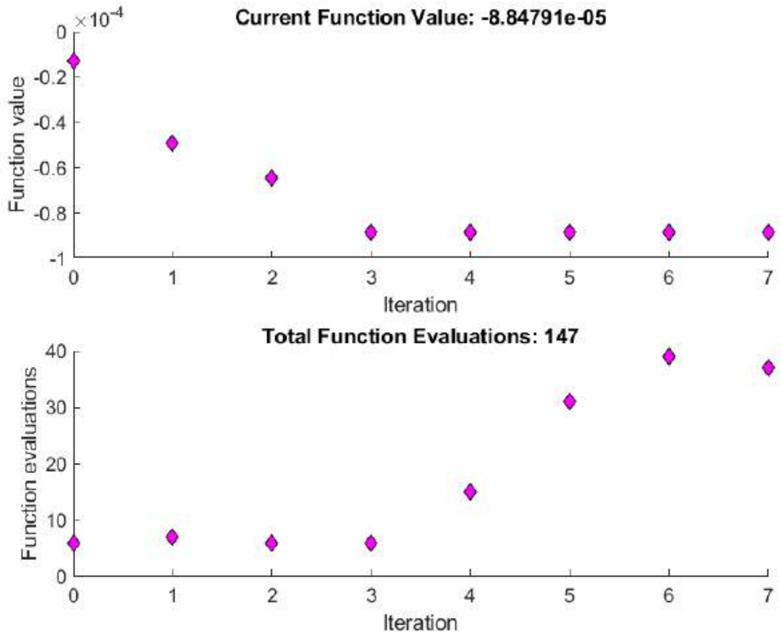


Figure 4: Plot showing the function evaluation at each iteration (top) and the number of function evaluations for the given iteration (bottom) in the case where we do gradient based optimization of the efficiency based on the aforementioned "good" design vector as the initial guess. The descent towards 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`:

```matlab
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

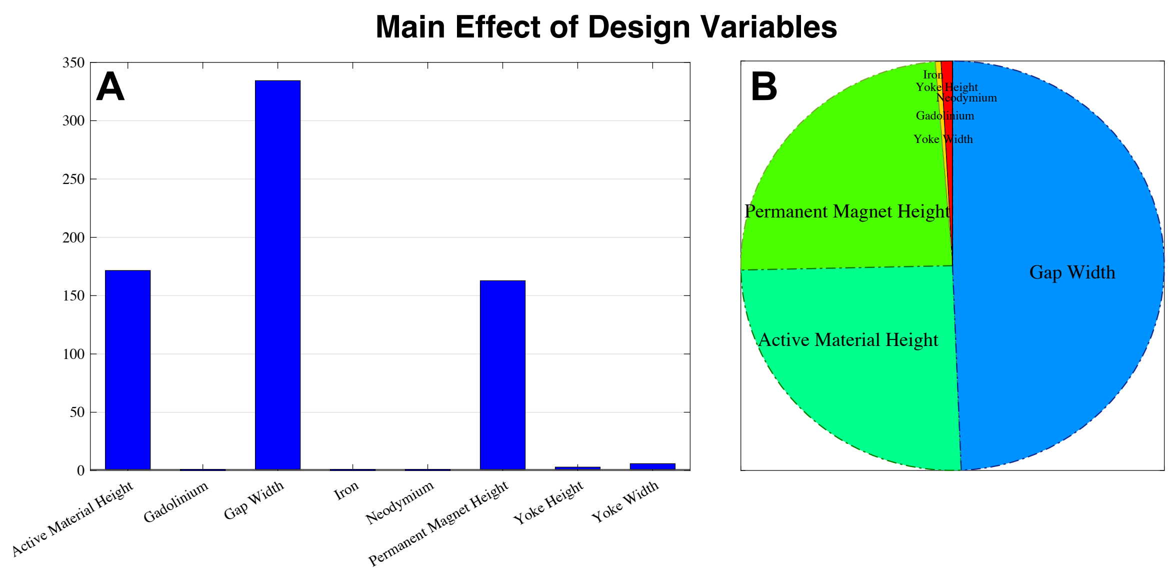


Figure 5: The graph representing each effects for design variables and parameters. Subfigure **A** is the bar plot for effects and **B** is the π plot.

4 Appendix

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

```matlab
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);
```

```matlab
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

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 20, 2021

1 Scaling

1.1 Cost

As referred back to the original optimization problem, we still focus on the `cost` module as for the scaling problem with gradient based method `fmincon`.

We adopt the `sqp` algorithm on the **cost** module with the five input design variables $(0.05; 0.1; 0.05; 0.05; 0.05)$ for running the optimization; the objective of cost module, the geometric constraint and the main optimization program are same as our previous work (*as attached in the Appendix*). Note that the upper and lower bounds are $(.45, .45, .45, .45, .45)$ and $(.05, .05, .05, .05, .05)$. By running the optimization we generate the following output:

```

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

```

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

```
<stopping criteria details>
No scaling: The best solution is efficiency = 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'
            message: ' Local minimum found that satisfies the constraints.  Optimization
    completed because the objective function is non-decreasing in  feasible directions, to
    within the value of the optimality tolerance, and constraints are satisfied to within
    the value of the constraint tolerance.  <stopping criteria details>  Optimization
    completed: The relative first-order optimality measure, 2.228358e-11, is less than
    options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint
    violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.  '
    constrviolation: 0
           stepsize: 1.4867e-17
       lssteplength: 1
      firstorderopt: 7.4506e-09
       bestfeasible: [1x1 struct]

The unscaled hessian is:

   1.0e+05 *

    0.0005    0.0003    0.0142    0.0149    0.0291
    0.0003    0.0001    0.0071    0.0075    0.0145
    0.0142    0.0071    0.3895    0.4104    0.7998
    0.0149    0.0075    0.4104    0.4325    0.8428
    0.0291    0.0145    0.7998    0.8428    1.6427

The unscaled condition number is 7.144017e+05
Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

```

```
<stopping criteria details>
Cost scaling: The best solution is cost = 0.00192677, 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'
            message: ' Local minimum found that satisfies the constraints.  Optimization
    completed because the objective function is non-decreasing in  feasible directions, to
    within the value of the optimality tolerance, and constraints are satisfied to within
    the value of the constraint tolerance.  <stopping criteria details>  Optimization
    completed: The relative first-order optimality measure, 1.332268e-15, is less than
    options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint
    violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.  '
    constrviolation: 0
           stepsize: 9.8027e-17
       lssteplength: 1
      firstorderopt: 1.3323e-15
       bestfeasible: [1x1 struct]

The scaled hessian is:

   25.9973   12.3112    2.0591    2.1765    4.3607
   12.3112    6.7806    0.8421    0.9008    1.9928
    2.0591    0.8421    1.0649    0.0751    0.2650
    2.1765    0.9008    0.0751    1.0859    0.2859
    4.3607    1.9928    0.2650    0.2859    1.6759

The scaled condition number is 6.270862e+01
The scaled optimization found a 0 percent smaller optimal value in the same number of
    iterations and -0 percent fewer function evaluations

The scaled optimization found a solution 87.90 percent faster than the unscaled optimization
    >>
```

Therefore we can write out the unscaled Hessian:

$$\mathbf{H} = \begin{bmatrix} 52.4001 & 25.5125 & 1416.4316 & 1492.5933 & 2909.1499 \\ 25.5125 & 13.3813 & 708.0283 & 746.1091 & 1454.3874 \\ 1416.4316 & 708.0283 & 38945.6567 & 41038.5367 & 79983.3184 \\ 1492.5933 & 746.1091 & 41038.5367 & 43245.9949 & 84283.6567 \\ 2909.1499 & 1454.3874 & 79983.3184 & 84283.6567 & 164268.1001 \end{bmatrix} \tag{1}$$

With simple observation we can deduce that the problem is obviously ill conditioned; A scaling of the problem is hence required.

The five input design variables are $[w_{yk}, h_{yk}, h_{pm}, h_A, w_{gap}]$, to approximate the final form of $\mathcal{O}(\mathbf{H}) \approx 1$; we hope to make each terms of $\mathbf{H} \sim \mathbf{1}$. Therefore, the scaling factor should be $[10^{-1.5}; 10^{-.5}; 10^{-2}; 10^{-2}; 10^{-2.5}]$. Same as our previous settings, the lower and upper bounds of the optimization system are $(0.05, 0.05, 0.05, 0.05, 0.05)$ and $(0.45, 0.45, 0.45, 0.45, 0.45)$, respectively. Applying the factors and scale the new design variables as $[\tilde{w}_{yk}, \tilde{h}_{yk}, \tilde{h}_{pm}, \tilde{h}_A, \tilde{w}_{gap}] = [10^{-1.5}w_{yk}, 10^{-.5}h_{yk}, 110^{-2}h_{pm}, 10^{-2}h_A, 10^{-2.5}w_{gap}]$ and reoptimize the problem (*the new code for the reoptimization are also attached in the Appendix*).

From the output we can deduce that the new Hessian $\tilde{\mathbf{H}}$ is

$$\tilde{\mathbf{H}} = \begin{bmatrix} 25.9973 & 12.3112 & 2.0591 & 2.1765 & 4.3607 \\ 12.3112 & 6.7806 & 0.8421 & 0.9007 & 1.9928 \\ 2.0591 & 0.8421 & 1.0649 & 0.07509 & 0.2650 \\ 2.1765 & 0.9007 & 0.0751 & 1.0859 & 0.2859 \\ 4.3607 & 1.9928 & 0.2649 & 0.2859 & 1.6759 \end{bmatrix} \tag{2}$$

It can be deduced that the problem is no longer ill-conditioned. Also, surprisingly, the scaled and original optimization both land on the same solution for the design variables: $(0.05, 0.05, 0.05, 0.1, 0.05)$.

To estimate the how scaling works for the cost module, we mainly investigate *compute time, condition number, number of function evaluations, and quality of the optimized solution* before and after the scaling.

Considering computation time, we observe that using scaling on the cost function allows the optimization to run 87% faster while performing the same number of function evaluations.

Considering the condition number, we can apply the MATLAB® built-in function `cond()` to compute the condition number of our Hessian, given by

$$\kappa(\mathbf{H}) = ||\mathbf{H}||||\mathbf{H}^{-1}|| \tag{3}$$

Before the scaling the condition number is $\kappa(\mathbf{H}) = 7.144 \times 10^5$; and after the scaling the condition is $\kappa(\tilde{\mathbf{H}}) = 6.271 \times 10$. It is obvious that the condition number dropped significantly after scaling, indicating an effective scaling.

Eventually, when looking at the quality of the optimization, the `First-order optimality` could help us deduce. First-order optimality is a measure of how close a point x is to optimal [4]. Here, in our approach, the objective takes the form

$$\min \texttt{cost}(w_{yk}, h_{yk}, ...) \tag{4}$$

the first-order optimality measure is the infinity norm (meaning maximum absolute value) of

$$\max_i |(\Delta\texttt{cost}(\text{geomtric vars.}))_i| = ||\Delta\texttt{cost}(\text{geomtric vars.})||_\infty \tag{5}$$

The first order optimality drops from 7.45×10^{-9} to 1.33×10^{-15} after scaling, indicating an extremely effective scaling.

1.2 Efficiency and General Optimization Considerations

When it comes to efficiency and power output, things become complicated. If the optimum solution is taken as the one that optimizes efficiency, then the situation is very different. The optimization algorithm is run anew without scaling, and the resultant Hessian[1] at the optimum, is given by,

$$\mathbf{H}_{\mathbf{x_{opt}}} = \begin{bmatrix} 4.891 & -4.1653 & 1.150 & 0.0 & -2.3737 \\ -4.1653 & 9.9191 & -1.1196 & -0.0 & 6.2427 \\ 1.1501 & -1.1196 & 0.8515 & 0 & -0.5468 \\ 0.0 & 0.0 & 0.0 & 1 & 0.0 \\ -2.373 & 6.2427 & -0.5468 & 0.0 & 3.972 \end{bmatrix} \tag{6}$$

[1]that is, the numerical approximation for it using MATLAB's built-in function (FD)

- computed using finite differencing. This looks fairly innocent, but it turns out that the condition number is extremely large. That is, $\kappa = 5041$. Also, since large and small numbers are so sporadically distributed in the matrix, there is no easy way of finding a good scaling. Rather, a systematic approach is undertaken. First, let us write up the new matrix in terms of the scaling parameters, $\mathbf{L}$. This is computed by taking first taking the outer product of $\mathbf{L}$ with itself; then taking the Hadamard product between the resulting matrix and $\mathbf{H}_{\mathbf{x_{opt}}}$ before the scaling. This yields

$$\hat{\mathbf{H}}_{\mathbf{x_{opt}}} = \begin{bmatrix} 4.891L_1^2 & -4.1653L_1L_2 & 1.150L_1L_3 & 0. & -2.3737L_5L_1 \\ -4.1653L_1L_2 & 9.9191L_2^2 & -1.1196L_3L_2 & -0. & 6.2427L_5L_2 \\ 1.1501L_1L_3 & -1.1196L_3L_2 & 0.8515L_3^2 & 0 & -0.5468L_5L_3 \\ 0 & 0 & 0 & L_4^2 & 0 \\ -2.373L_5L_1 & 6.2427L_5L_2 & -0.5468L_5L_3 & 0 & 3.972L_5^2 \end{bmatrix} \tag{7}$$

It should be noted that this approach is just a suggestion of our own, whose sole purpose is to bring down the computed condition number, at the given optimum. It builds on the Taylor approximation to second order, where the function is assumed to behave quadratically along the diagonal, and linear in both variables in the cross terms. Of course, there is no way of guaranteeing that the scaling actually translates into something that is well approximated by a Taylor series like that very far from the optimum[2]. Maybe, the actual objective function[3] could even have a sort of delta-function like behavior up until right before the optimum.

Anyway, for optimizing the condition number, a numerical approach is used where we loop through each component of $\mathbf{L}$ and optimize individually. The optimum is then fixed and used for computing the optimum of the next component, and so forth. There will no doubt be more intelligent ways to undertake a joint optimization of all the components and once[4], but this approach turns out to be extremely effective too. As every other component is held fixed, the condition number is plotted as the one in question varies. The optimum is then determined graphically. As nice as the graphs look, let us skip ahead to the results though.

[2]should of course always be possible right at it

[3]this is recognized as a sort of abstract idea, since no objective function is actually defined analytically

[4]somewhat ironically, we could have used an advanced optimizing algorithm to optimize this

The final scaling vector looks like this, $\mathbf{L} = [0.25; 0.6; 1; 0.15; 0.9]$. The condition number has thus been reduced from 5041 to 310. This is more impressive than it looks. Randomly varying components of the scaling vector was attempted too. This was never able to yield a condition number below 4000, so the systematic approach definitely proved itself useful. Looking at the diagonal elements, $\mathbf{\hat{H}_{i,i}}(\mathbf{x_{opt}})$, there was not a lot to be gained. $\mathbf{\hat{H}_{2,2}}(\mathbf{x_{opt}})$ and $\mathbf{\hat{H}_{3,3}}(\mathbf{x_{opt}})$ did vary by an order of magnitude, but all are already $\sim O(1)$. Furthermore, it turned out that trying to re-scale without considering the condition number almost always increased it. Computation time only increased. All this may have seemed like overkill, as we are not being asked for it anyway. However, it turned out we were in dire need of good scaling as our problem is of a complexity which makes the computation time take hours otherwise. When rerunning with the scaling, the condition number of the new optimum had come all the way down to just 11.5. It converged with a time reduction of only 14% though, and the quality of the optimum deteriorated considerably, going from an efficiency of $3.15 \cdot 10{-}5$ to $8.8^{\cdot}10^{-6}$, in units of the scaling factor mentioned in report 3. This emphasizes an important point, to which we will return in *Multiobjective Optimization*; the robustness of the gradient-based method is extremely poor for our particular problem, unless settings are chosen which make it run much too slow.

Due to the extreme degree of non-linearity in our problem, local minima will likely be plentiful and *deep*. This owes partly to the fact that the modelling of the heat transfer is from the two-dimensional heat diffusion equation, where the heating time in worst case goes up exponentially to the power of 2, as the solution of the time and position dependent temperature is on a form *similar* to, $K(t,x,y) = \frac{1}{(4\pi t)^{d/2}} e^{-|x-y|^2/4t}$ [5].

First of all though, the magnetic aspect of the problem makes it extremely non-linear. The model has been programmed with adaptive evaluation boundaries, and when the width of the active material decreases, the magnetic flux[6] can become extremely small if the bounds on the variables allows for a more complete exploration. In our project, we were ambitious and allowed for the dimensions of the active material to change by two orders of magnitude

[5]The two-dimensional behavior will come into play when the width of the active material becomes small compared to the width of the yoke. Combined with a large thickness of active material, the power output becomes as good as 0

[6]to the **square** of which power is proportional

for most of the optimization we have done. As a first order approximation, that causes a change in the power output of a 10^4 on a linear scale. Furthermore, this is just the obvious consequence of only wrapping our coil around a smaller rod; much more impactful, and unpredictable, is how the magnetic vector potential reacts to these changes. The curl will stay the same but how much flux ends up being guided around in the "magnetic circuit" is highly susceptible to change in any geometric parameter and extremely complicated from the perspective of the governing equations. The magnetic field is by nature divergence-less and rotational. It is also highly adaptive to changing geometry under the right permeability conditions, as it may condense many orders of magnitude.

Both power and efficiency depend on the magnetic simulation, and power furthermore depends on the heating time.

The actual scaling to be used in multi objective optimization took this as a starting point at was afterwards experimented with several times to get the best results for the *multiobjective optimization*. In the end, the fastest computed Pareto front and the best results came from running a multi-objective optimization based on a genetic algorithm, not a gradient-based one. This was implemented with no scaling at all. With such a high degree of non-linearity and unpredictable behavior, the genetic algorithm proved superior. As noted in report 3, it was slower, but the results were also less prone to producing spurious conclusions.

2 Multiobjective Optimization

Select two objective functions for your project

Multiobjective optimization turned out to be extremely troublesome. At first, the AWS and NBI methods were employed with SQP used for optimizing the weighted sum. This was done in the two-dimensional spaces of efficiency and power output, efficiency and cost, and power output and cost. We believe that both the AWS and NBI algorithms were successfully implemented. However, despite our best efforts to tune these algorithms, we were unable to get them to produce reasonable results within a reasonable time-frame and the algorithms took extraordinarily long times to run. We largely attribute this difficulty to the complicated nature of our COMSOL-coupled objectives. The team concluded that MATLAB's fmmincon function and all of its algorithmic settings are poorly suited for our problem.

Multiobjective optimization of power output and cost turned out to be especially problematic. This could perhaps have been predicted from the start, although we must admit that we were very optimistic at first. There are two main reasons for this, as we see it:

- Both objectives are extremely non-linear and the combination of them, even more so.
- Both objectives are largely co-directional up until a level of *hard-to-resolve finesse*, and co-dependent by a myriad of mechanisms

The latter requires a bit of explanation. There are a few common features that will both enhance the power output and the efficiency. As has been explained in *report 2*, both of them scale with the total magnetic flux squared and will thus benefit from increased height of the active material. They will also both benefit from having a larger permanent magnet. As long as we can change variables and improve both objectives however, we are simply not at Pareto optimum. Getting to the point where one of them will have to hurt the other turns out to be a matter of finesse which was too much for simple optimization tools to be able to resolve to a level of[7].

Consider on the other hand yoke widths that are large. Then the heating time and the heat input will be increased, as the heat diffusion becomes increasingly two-dimensional in nature and more "waste material" has to be heated up alongside the active material. At the same time, this will mean that there is less fringing field, so that more magnetic flux can be guided through the coil. Less fringing leads to more power output, which leads to more efficiency. At the same time, increased heating time leads to *less* power output and increased heat input leads to *less* efficiency. This is just taking the variation of one variable into consideration and considering only the obvious effects, and *some* of the feedback mechanisms. The optimum where the sign of the effect on one of the objectives differs from that one the other will come down to the exact nature of the non-linearities and will be highly sensitive

[7]this is mainly a critique of *fmincon*

to the state of all the other variables.

Neither NBI or AWS could produce reliable results for multiobjective optimization of power and efficiency. We know that both of these methods do have their limitations and can produce erroneous results. Combined with the tendency of *fmincon* to produce spurious results for these particular objective functions, it is not a huge surprise. AWS was implemented with some success for optimizing efficiency and cost jointly. It was painfully slow though, and a full and evenly space Pareto front will only be available the morning after the deadline of this report.

The team turned back to revisit the use of a genetic algorithm, which had previously been discarded due to its unreasonably large compute time. The number of function evaluations needed to be reduced for a genetic algorithm to suit our needs. So, the team turned to MATLAB's documentation to understand why the GA performed more function evaluations than there were population members in each generation. We found that while the GA handles linear constraints without trouble, nonlinear constraints rapidly increase the computational cost of the algorithm. Where one function evaluation takes place per population member in each generation with linear constraints, nonlinear constraints ”confuse” the algorithm and require many more function evaluations per individual. The constraints related to length and width of our thermomagnetic generator are in fact already linear, but had been represented as nonlinear in MATLAB previously. The volume constraint is nonlinear. For the sake of experimentation, we relaxed the volume constraint entirely to explore the GA's performance. We may linearize the volume constraint later on, or remove it as we feel it may be unimportant in the scheme of our problem. Although we were able to make fmincon work for single objective optimizations, the success largely came from our ability to run the sqp algorithm many times and cherry pick the best results. To further reduce computing cost, we reduced the resolution of the mesh for both the thermal and the magnetic COMSOL model. During this process we were careful to not make the mesh so coarse that our objective functions returned inaccurate results.

The genetic algorithm applied with only linear constraints quickly showed promising results in a timely manner. Where previously we had observed poor results with runs lasting

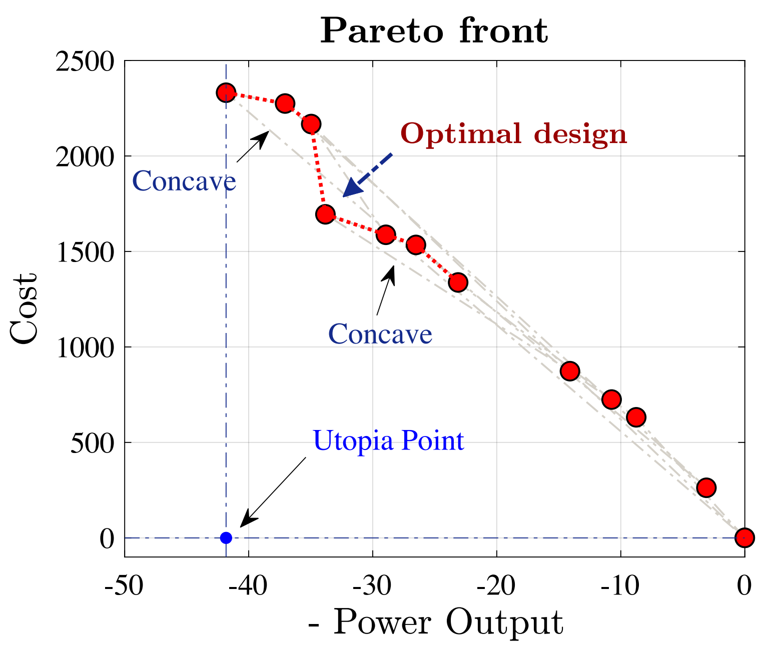


Figure 1: The Pareto front for optimizing both the cost and power output module. This is exactly what we would expect. As cost is being minimized with an eye for power too, there will be a common interest in removing active material from the structure, which is both the most expensive and results in exponentially larger heating times (exponentially smaller power outputs), but as power output becomes dominantly weighed, it will be more advantageous to morph active material into a horizontally elongated structure, rather than removing it, which hurts the magnetic flux and thus the power.

24 hours or more, we now could achieve reasonable results in less than 30 minutes. We first confirmed with single objective optimizations that the genetic algorithm offered comparable performance to fmincon with greater consistency. Then, we applied MATLAB's gamultiobj() function to our cost and power objectives and plot the resulting Pareto front in Figure 1.

We see that while cost is mostly linear with power, the optimal design does lie on a convex point and corresponds to a final selected design vector of:

Yoke Width	0.0709
Yoke Height	0.3625
Permanent Magnet Height	0.1464
Active Material Height	0.1499
Gap Width	0.1313

And gives the results:

$$\text{Cost} = \$1{,}694.4$$

$$\text{Power} = 33.8145$$

The optimal point marked in Figure 1 This convex point could potentially be a result of a high performing outlier from the optimization. However, we believe that the Pareto front is likely accurate because the algorithm performed more than 5000 function evaluations on a constant population of 35 individuals to gather the data. Had we realized that the algorithm would run for so long, we would have reduced the function tolerance and increased the population size to create a higher resolution curve. We plan to do this for the final presentation and report. Note that while the cost axis represents accurate dollar amounts, the power axis is on an arbitrary scale because per the setup of our problem, power is multiplied by an unknown constant K. Note also that power is negative. This is because the algorithm tries to minimize the objective, but we want to maximize power. Thus, we can multiply power by -1 and represent maximum power by optimizing for the most negative objective value. This approach is also taken with the efficiency objective. The following settings were used to create Figure 1:

```
funcTol = 1e-4;
conTol = 1e-5;
popSize = 35;
crossoverRatio = 1.2;
crossoverFraction = .8;
maxStallGenerations = 50;
```

For `gamultiobj()`, the algorithm is set to stop when the geometric average change in Pareto spread across all generations is less than the function tolerance divided by maximum stall generations. The above settings resulted in an impractically long computation time, and the algorithm was halted after approximately eight hours and the data was used to create the Pareto front in Figure 1.

It is an easy matter to take a point and show that it is non-dominated, and luckily, this can be shown for every single point on our Pareto front[8]. The chosen point is marked in figure 2. This figure contains the explanation as to why the point is no-dominated in its caption.

Using what we learned from the dual-objective optimization of cost and power, the team revised the MATLAB scripts to create a Pareto front for all three objectives: power, cost, and efficiency. The script and all helper functions used to create the three-objective Pareto front are included in the Appendix. The same script was used with a modified objective

[8]- meaning that is actually a Pareto front

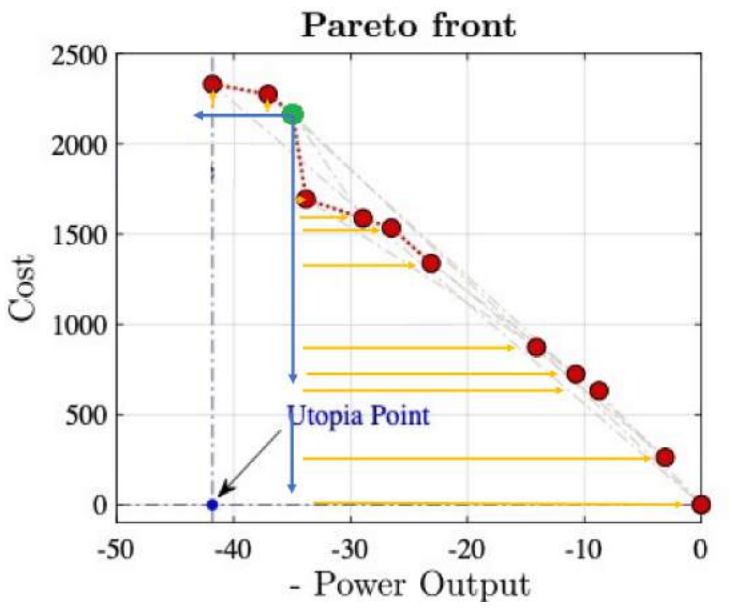


Figure 2: Consider the green point on the Pareto front. This is clearly non-dominated, as can be shown by drawing the blue arrows in the direction of decreasing cost and increasing power output. If we go along that line, then we will only be able to go to points that have also increased in cost, or decreased in power, respectively. The green point will thus always be optimal, as long as the particular weighing used to produce it, is considered. For a particular niche in terms of weighing, no other point can beat it.

function to produce the two-objective optimization. The 3D Pareto front produced is shown in Figure 3 and appears to be consistent with the 2D power-cost Pareto front when viewed from its side. We attempted to rush the completion of this optimization and thus the result does not show enough points along the Pareto front to draw a clear optimal line. The team is excited to run the algorithm again with a higher population and lower tolerances to produce a more complete graph for the final report and presentation.

Contributions

HW3:

- **Q1**: Mads: Writing & COMSOL/MATLAB integration, Problem formulation; Will: COMSOL/MATLAB integration
- **Q2**: Hanfeng: Writing & graph, Heuristic algorithm, GA programming; Will: programming that worked (Single objective GA), results generation, analysis and writing.
- **Q3**: Will: Main programming (cost, power output, and efficiency), figures generation (running from main program) results generation, & revision. Mads: Results analysis,

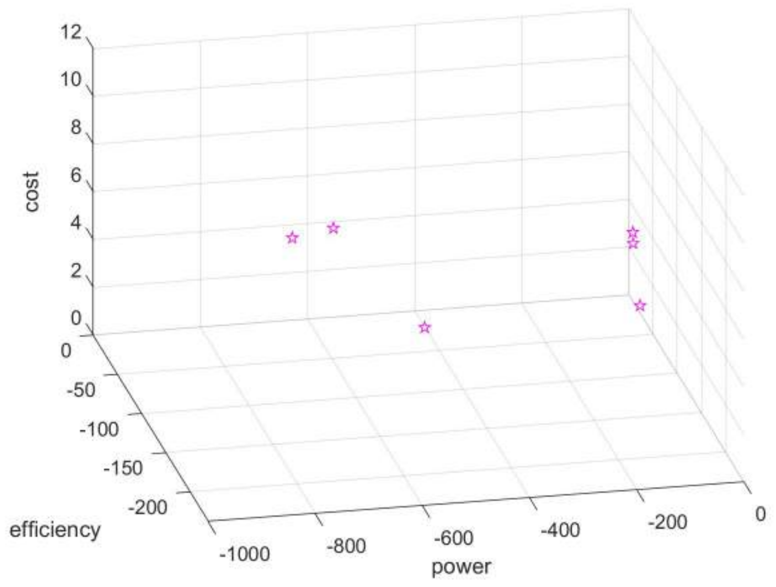


Figure 3: 3D Pareto front of power, efficiency, and cost. This was created with a rushed algorithm and did not produce enough data points to be meaningful enough to draw conclusions from.

and writing. Hanfeng: Writing, programming & analysis for sensitivity analysis & single objective optimization for cost, & graphic representation (schematics).

- **General**: Hanfeng: Beautiful figures (added by Will) & latex formatting. Putting up website with results and project description

HW4:

- **Q1**: Hanfeng: Scaling for cost module & writing. Will: Programming for both modules, editing & revising. Mads: scaling for the efficiency module & writing.

- **Q2**: Will: Programming NBI, programming MOO GA, optimization running, results analysis, writing. Mads: Programming, results analysis, writing. Hanfeng: Schematic & Graphic representation, revision.

- **General**: Mads: Discussion on optimization considerations, SQP AWS programming. Will: Extremely labour intensive and productive programming (added by Mads). Hanfeng: Latex formatting & schematics and the figures

Appendix

TherMaG_gradoptim_L_single_scaling.m: Script for exploring design variable scaling of different objective functions.

```matlab
clear
clc
close all

% starting point for optimization
x0 = [.05;.1;.05;.05;.05]; % default
x0cost = [.15;.15;.05;.05;.05]; % cost
x0power = [.05;.45;.25;.15;.15]; % power
x0efficiency = [.05;.45;.25;.15;.05]; % efficiency

TolF = 1e-6;
TolX = 1e-6;
TolCon = 1e-6;

% Set nondefault solver options
options = optimoptions('fmincon','Algorithm','sqp','FunctionTolerance',TolF,'StepTolerance',
    TolX,'ConstraintTolerance',TolCon,'PlotFcn',{@optimplotfval, @optimplotfunccount});

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

% make L scaling vectors
L = [1; 1; 1; 1; 1];

% make anonymous functions
powerFcn = @(input) powerFcnSingleL(input, L);
efficiencyFcn = @(input) efficiencyFcnSingleL(input, L);
costFcn = @(cost) objee_scaling(cost, L);

% constants
V_max = .125;
L_max = .5;

% set linear constraints
A = [0, 1, 0, 0, 0; 2, 0, 0, 0, 1; 0, -1, 1, 1, 0];
b = [L_max; L_max; 0];

tic;

```

```
% run the optimization
% CHANGE THIS TO THE CORRECT VARIABLE
[solution,objectiveValue,exitflag,output,lambda,grad,hessian] = fmincon(costFcn,x0,A,b
    ,[],[],lb,ub,[],options);

unscaledTime = toc;

%% Run the above section once, then only run this section to check new scaling vectors

L_efficiency = [1; 10^(-3.5); 10^(-4); 1; 10^(-3.5)];
L_power = [1; 1; 10^(-.5); 1; 1];
L_cost = [10^(-1.5); 10^(-1); 1; 1; 1];

fprintf('No scaling: The best solution is efficiency = %.8f, corresponding to a design
    vector of: \n',objectiveValue);
fprintf('\n');
disp(solution);
fprintf('\n');
fprintf('This solution had the following simulation output info: \n');
fprintf('\n');
disp(output);

eigs = eig(hessian);
maxeig = max(eigs);
mineig = min(eigs);
conditionNumber = abs(maxeig/mineig);

fprintf('The unscaled hessian is:\n');
fprintf('\n');
disp(hessian);
fprintf('The unscaled condition number is %d',conditionNumber);

powerFcn = @(input) powerFcnSingleL(input, L_power);
efficiencyFcn = @(input) efficiencyFcnSingleL(input, L_efficiency);
costFcn = @(cost) objee_scaling(cost, L_cost);

tic;

% run the optimization again with the new scaling
% CHANGE THE FUNCTION TO THE CORRECT VARIABLE
[scaledSolution,scaledObjectiveValue,scaledExitflag,scaledOutput,scaledLambda,scaledGrad,
    scaledHessian] = fmincon(costFcn,x0,A,b,[],[],lb,ub,[],options);

scaledTime = toc;

[realScaledPower, realScaledEfficiency] = efficiency_power_modules(scaledSolution);
```

```
realScaledCost = objee(scaledSolution);

% CHANGE THIS TO THE RIGHT VARIABLE
realObjectiveValue = realScaledCost;

% CHANGE THIS TEXT TO THE RIGHT VARIABLE
fprintf('Efficiency scaling: The best solution is efficiency = %.8f, corresponding to a
    design vector of: \n',realObjectiveValue);
fprintf('\n');
disp(scaledSolution);
fprintf('\n');
fprintf('This solution had the following simulation output info: \n');
fprintf('\n');
disp(scaledOutput);

eigs = eig(scaledHessian);
maxeig = max(eigs);
mineig = min(eigs);
scaledConditionNumber = abs(maxeig/mineig);

funcCountDiff = -(scaledOutput.funcCount - output.funcCount)/output.funcCount*100;
fvalDiff = -(realObjectiveValue - objectiveValue)/objectiveValue*100;
timeDiff = -(scaledTime - unscaledTime)/unscaledTime*100;

fprintf('The scaled hessian is:\n');
fprintf('\n');
disp(scaledHessian);
fprintf('The scaled condition number is %d',scaledConditionNumber);
fprintf('\n');
fprintf('The scaled optimization found a %.2f percent smaller optimal value in the same
    number of iterations and %.0f percent fewer function evaluations\n',fvalDiff,
    funcCountDiff);
fprintf('\n');
fprintf('The scaled optimization found a solution %.2f percent faster than the unscaled
    optimization',timeDiff);
```

powerFcnSingleL.m: Helper function to return power.

```
function [power] = powerFcnSingleL(x,L)
    [power,~] = efficiency_power_modules_L_single_scaling(x,L);
end
```

efficiencyFcnScaling.m: Helper function to return efficiency.

```
function [efficiency] = efficiencyFcnSingleL(x,L)
    [~,efficiency] = efficiency_power_modules_L_single_scaling(x,L);
end
```

objee.m: The scaled objective function for running minimizing the cost of the TMG.

```
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 = 10^(-7)*x(1);%10^(-7)*
  h_yk = 10^(1)*x(2);%10^(1)*
  h_pm = x(3);%1*
  h_A = 10^(1)*x(4);%10^(1)*
  w_gap = 10^(-10)*x(5);%10^(-10)*
  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: The geometric constraint for running minimizing the cost of the TMG.

```
function [c,ceq] = geocon(x0)
    x = x0;
    inputmatrx = x;
    h_A = inputmatrx(1);w_gap = inputmatrx(2);
    w_yk = inputmatrx(3);h_yk=inputmatrx(4);h_pm=inputmatrx(5);

    c(1) = h_yk*(2*w_yk + w_gap) - .125;
    c(2) = h_yk - .5;
    c(3) = (2*w_yk + w_gap) - .5;
    c(4) = (h_A + h_pm) - h_yk;
    ceq = [];
end
```

objee.m:

The unscaled objective function for running minimizing the cost of the TMG.

```
function f = objee(x)

    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
```

TherMaG_GA_Multi.m: The script used to run multiobjective genetic algorithm optimizations.

```
clear
clc
close all

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

% set optimization options
funcTol = 10^(-2.5);
conTol = 1e-5;
popSize = 30;
crossoverRatio = 1.2;
crossoverFraction = .8;
maxStallGenerations = 5;

options = optimoptions('gamultiobj', ...
        'CrossoverFcn',{@crossoverheuristic,crossoverRatio},...
        'FunctionTolerance', funcTol, ...
        'PopulationSize', popSize, ...
        'CrossoverFraction', crossoverFraction,...
        'ConstraintTolerance', conTol,...
        'MutationFcn',{@mutationadaptfeasible},...
        'MaxStallGenerations', maxStallGenerations,...
        "PlotFcn", {@gaplotPareto, @gaplotgenealogy, @gaplotscorediversity, @gaplotrankhist,
     @gaplotstopping},...
        'Display','diagnose');

% constants
V_max = .125;
L_max = .5;

% set linear constraints
A = [0, 1, 0, 0, 0; 2, 0, 0, 0, 1; 0, -1, 1, 1, 0];
b = [L_max; L_max; 0];

% run the GA
```

```
[solution, fval, exitflag, output, population, scores] = gamultiobj(
    @powerCostEfficiencyMulti,5,A,b,[],[],lb,ub,options);
```

powerCostEfficiencyMulti.m: A helper function returning all three objective values for optimization.

```
function [evaluation] = powerCostEfficiencyMulti(input)

    [power, efficiency] = efficiency_power_modules(input);

    evaluation = zeros(3,1);
    evaluation(1) = power;
    evaluation(2) = efficiency;
    evaluation(2) = objee(input);

end
```

efficiency_power_modules.m: The objective function coupled with COMSOL to return power and efficiency.

```

function [power, efficiency] = efficiency_power_modules(vector)

    % define constants
    Cp_A = 450;
    Cp_everythingElse = 3.5433e6;

    % extract design parameters
    w_yk = vector(1);
    h_yk = vector(2);
    h_pm = vector(3);
    h_A = vector(4);
    w_gap = vector(5);

    % run models to get outputs for result calculations

    % open the thermal comsol model and run the study
    thermalModel = mphopen('therm_assign_3.mph');

    % set basic thermal model parameters
    thermalModel.param.set('yoke_width',w_yk);
    thermalModel.param.set('permmag_height',h_pm);
    thermalModel.param.set('activemat_height',h_A);
    thermalModel.param.set('yoke_height',h_yk);
    thermalModel.param.set('actperm_width',w_gap);

    % run the thermal model
    thermalModel.study('std2').run;

    % extract the solution (end of the time array)
    time = thermalModel.result.numerical('pev1').getReal();
    heatTime = time(end);

    % run the thermal model to find integrated temperatures
    integratedTemperaturesActive = thermalModel.result.numerical('int1').getReal();
    T_int_active = integratedTemperaturesActive(end);

    integratedTemperaturesEverythingElse = thermalModel.result.numerical('int2').getReal();
    T_int_everythingElse = integratedTemperaturesEverythingElse(end);

    % open the magnetic comsol model
    magneticModel = mphopen('simple_for_assign_3.mph');

    % set basic magnetic model parameters
    magneticModel.param.set('yoke_width',w_yk);
    magneticModel.param.set('permmag_height',h_pm);
    magneticModel.param.set('activemat_height',h_A);
    magneticModel.param.set('yoke_height',h_yk);
    magneticModel.param.set('actperm_width',w_gap);

    % compute flux before (mu = 1)
    magneticModel.param.set('mu_r',1);
    magneticModel.study('std1').run;
    fluxBefore = magneticModel.result.numerical('int1').getReal();

    % compute flux after (mu = 20)
    magneticModel.param.set('mu_r',20);
    magneticModel.study('std1').run;
    fluxAfter = magneticModel.result.numerical('int1').getReal();

    % calculate results

    % compute power result
    deltaFlux = fluxBefore - fluxAfter;
    power = -deltaFlux^2/heatTime*1e8;

    % compute efficiency
    efficiency = -deltaFlux^2/(Cp_A*T_int_active + Cp_everythingElse*T_int_everythingElse)*1
    e11;

end
```

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`

[4] First-Order Optimality Measure. Mathworks, Inc. URL: `https://www.mathworks.com/help/optim/ug/first-order-optimality-measure.html`

[5] Professor Walter Murray, Systems Optimization Laboratory. Advanced Methods in Numerical Optimization. URL: `https://web.stanford.edu/class/msande312/restricted/OPTconditions.pdf`