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

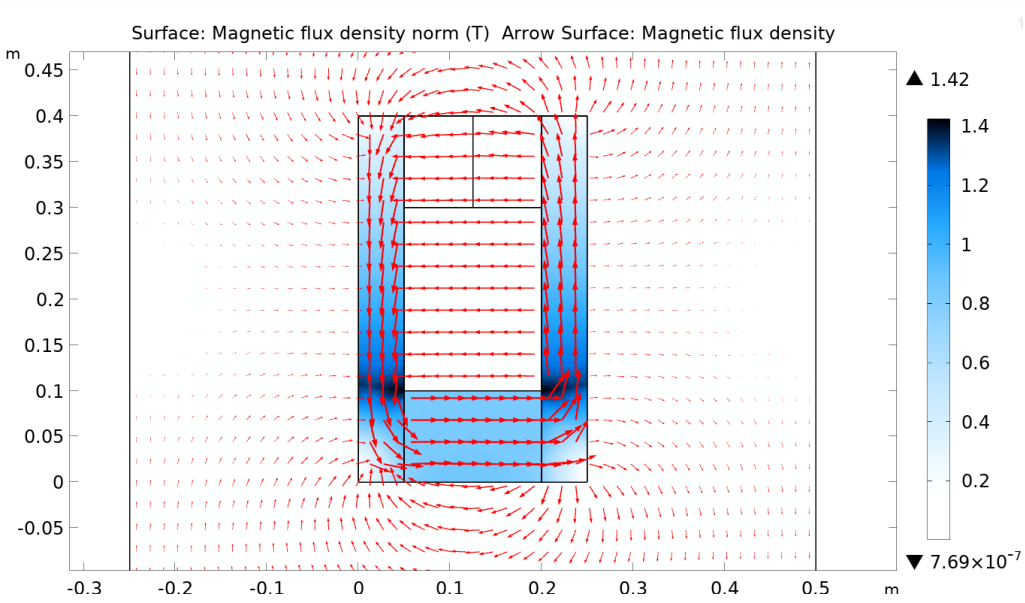

Figure 1: The magnetic distribution for changed geometry for hot active material (Gd).

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

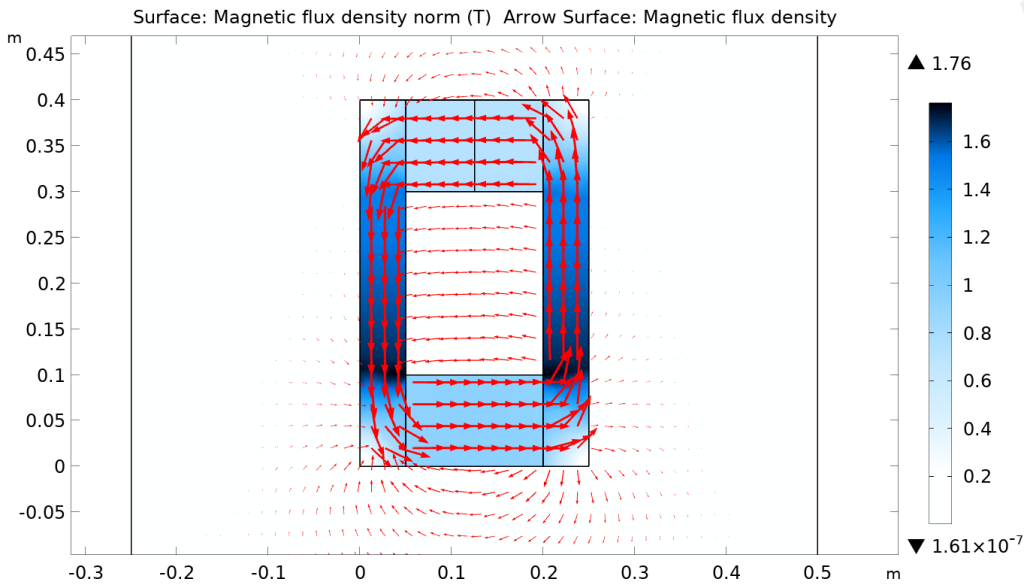

Figure 2: The magnetic distribution for changed geometry for cold active material (Gd).

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

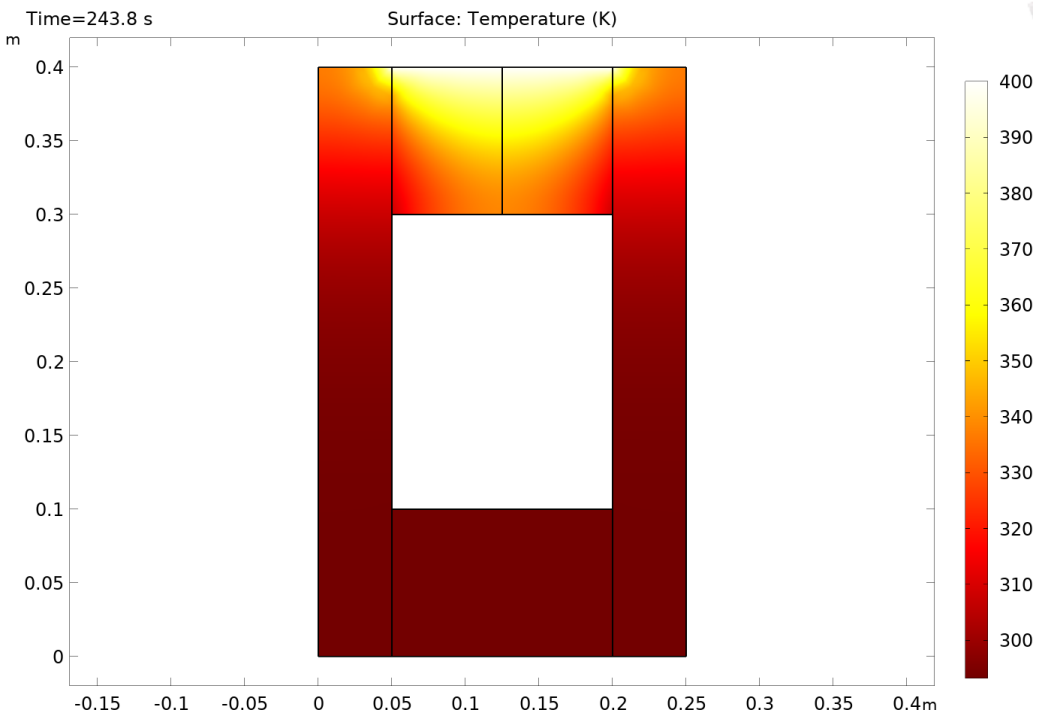

Figure 3: The temperature distribution for the changed shape, with an initial heat flux on the active material.

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.

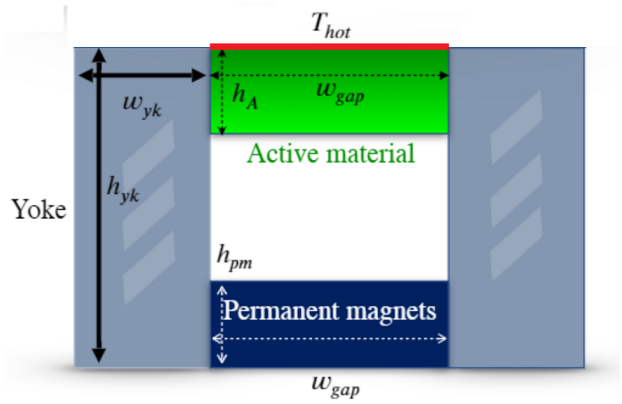

Figure 4: The schematic view for our setup of the thermal-magnetic generator (TMG) as illustrated in the last HW.

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.

```
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
```

```
64
65          % find the column indices of all results where variable i is at the j level
66          indices = find(designVectors(i,:) == effectLevels(j));
67
68          jResultSum = 0;
69
70          for k = 1:length(indices)
71              % sum the results of cost when variable i is at j level
72              jResultSum = jResultSum + experimentResults(1,indices(k));
73          end
74
75          % compute average cost when variable i is at j level
76          jMean = jResultSum/length(indices);
77
78          % add effect to the effect matrix
79          costEffects(j+offset,3) = jMean - meanCost;
80
81      end
82      offset = offset + length(effectLevels);
83
84  end
85
86  %% compute effect of design variables on power
87
88  % initialize power effects matrix [variable, level, effect]
89  powerEffects = zeros(length(variables)*length(effectLevels),3);
90
91  % helper variable to ensure that each block of variables, levels, and effects are created in
        the right locations
92  offset = 0;
93
94  for i = 1:length(variables)
95      for j = 1:length(effectLevels)
96
97          % set the variable being measured (represented by a number)
98          powerEffects(j+offset,1) = i;
99
100         % set the variable level of the variable being measured
101         powerEffects(j+offset,2) = effectLevels(j);
102
103         % compute the effect of the variable at the given level
104
105         % find the column indices of all results where variable i is at the j level
106         indices = find(designVectors(i,:) == effectLevels(j));
107
108         jResultSum = 0;
```

```
109
110         for k = 1:length(indices)
111             % sum the results of power when variable i is at j level
112             jResultSum = jResultSum + experimentResults(2,indices(k));
113         end
114
115         % compute average power when variable i is at j level
116         jMean = jResultSum/length(indices);
117
118         % add effect to the effect matrix
119         powerEffects(j+offset,3) = jMean - meanPower;
120
121     end
122     offset = offset + length(effectLevels);
123
124 end
125
126 %% compute effect of design variables on efficiency
127
128 % initialize efficiency effects matrix [variable, level, effect]
129 efficiencyEffects = zeros(length(variables)*length(effectLevels),3);
130
131 % helper variable to ensure that each block of variables, levels, and effects are created in
      the right locations
132 offset = 0;
133
134 for i = 1:length(variables)
135     for j = 1:length(effectLevels)
136
137         % set the variable being measured (represented by a number)
138         efficiencyEffects(j+offset,1) = i;
139
140         % set the variable level of the variable being measured
141         efficiencyEffects(j+offset,2) = effectLevels(j);
142
143         % compute the effect of the variable at the given level
144
145         % find the column indices of all results where variable i is at the j level
146         indices = find(designVectors(i,:) == effectLevels(j));
147
148         jResultSum = 0;
149
150         for k = 1:length(indices)
151             % sum the results of efficiency when variable i is at j level
152             jResultSum = jResultSum + experimentResults(3,indices(k));
153         end
```

```
154
155         % compute average efficiency when variable i is at j level
156         jMean = jResultSum/length(indices);
157
158         % add effect to the effect matrix
159         efficiencyEffects(j+offset,3) = jMean - meanEfficiency;
160
161     end
162     offset = offset + length(effectLevels);
163
164 end
165
166 %% Determine recommended start points X0 for numeric integration
167
168 % create copies of matrices with normalized effects
169 normalizedCostEffects = costEffects;
170 normalizedPowerEffects = powerEffects;
171 normalizedEfficiencyEffects = efficiencyEffects;
172
173 normalizedCostEffects(:,3) = normalize(costEffects(:,3));
174 normalizedPowerEffects(:,3) = normalize(powerEffects(:,3));
175 normalizedEfficiencyEffects(:,3) = normalize(efficiencyEffects(:,3));
176
177 % find variables and levels for greatest effect on each output
178
179 % best variable/level pairs to minimize cost:
180
181 % initialize array to hold the best variables/levels/effects for cost
182 bestCostVarsLevels = zeros(2,5); % row1 = level row2 = effect
183
184 % variable 1 (w_yk)
185 index = find(costEffects(1:5,3) == min(costEffects(1:5,3)));
186 bestCostVarsLevels(:,1) = [costEffects(index,2); costEffects(index,3)];
187 bestNormalCostVarsLevels(:,1) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];
188
189 % variable 2 (h_yk)
190 index = 5 + find(costEffects(6:10,3) == min(costEffects(6:10,3)));
191 bestCostVarsLevels(:,2) = [costEffects(index,2); costEffects(index,3)];
192 bestNormalCostVarsLevels(:,2) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];
193
194 % variable 3 (h_pm)
195 index = 10 + find(costEffects(11:15,3) == min(costEffects(11:15,3)));
196 bestCostVarsLevels(:,3) = [costEffects(index,2); costEffects(index,3)];
197 bestNormalCostVarsLevels(:,3) = [normalizedCostEffects(index,2); normalizedCostEffects(index
```

```
    ,3)];
198
199 % variable 4 (h_A)
200 index = 15 + find(costEffects(16:20,3) == min(costEffects(16:20,3)));
201 bestCostVarsLevels(:,4) = [costEffects(index,2); costEffects(index,3)];
202 bestNormalCostVarsLevels(:,4) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];
203
204 % variable 5 (w_gap)
205 index = 20 + find(costEffects(21:25,3) == min(costEffects(21:25,3)));
206 bestCostVarsLevels(:,5) = [costEffects(index,2); costEffects(index,3)];
207 bestNormalCostVarsLevels(:,5) = [normalizedCostEffects(index,2); normalizedCostEffects(index
    ,3)];
208
209 % best variable/level pairs to maximize power:
210
211 % initialize array to hold the best variables/levels/effects for power
212 bestPowerVarsLevels = zeros(2,5); % row1 = level row2 = effect
213
214 % variable 1 (w_yk)
215 index = find(powerEffects(1:5,3) == max(powerEffects(1:5,3)));
216 bestPowerVarsLevels(:,1) = [powerEffects(index,2); powerEffects(index,3)];
217 bestNormalPowerVarsLevels(:,1) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];
218
219 % variable 2 (h_yk)
220 index = 5 + find(powerEffects(6:10,3) == max(powerEffects(6:10,3)));
221 bestPowerVarsLevels(:,2) = [powerEffects(index,2); powerEffects(index,3)];
222 bestNormalPowerVarsLevels(:,2) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];
223
224 % variable 3 (h_pm)
225 index = 10 + find(powerEffects(11:15,3) == max(powerEffects(11:15,3)));
226 bestPowerVarsLevels(:,3) = [powerEffects(index,2); powerEffects(index,3)];
227 bestNormalPowerVarsLevels(:,3) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];
228
229 % variable 4 (h_A)
230 index = 15 + find(powerEffects(16:20,3) == max(powerEffects(16:20,3)));
231 bestPowerVarsLevels(:,4) = [powerEffects(index,2); powerEffects(index,3)];
232 bestNormalPowerVarsLevels(:,4) = [normalizedPowerEffects(index,2); normalizedPowerEffects(
    index,3)];
233
234 % variable 5 (w_gap)
235 index = 20 + find(powerEffects(21:25,3) == max(powerEffects(21:25,3)));
236 bestPowerVarsLevels(:,5) = [powerEffects(index,2); powerEffects(index,3)];
```

```
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 $\rho = 7000$ $[kg/m^3]$

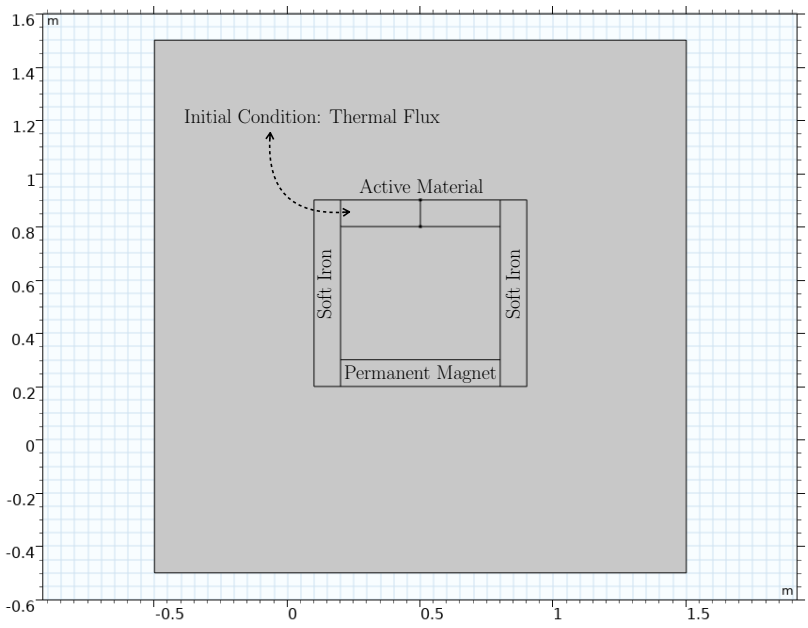

Figure 6: A schematic illustration of the simplied model for the thermal convection module.

[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

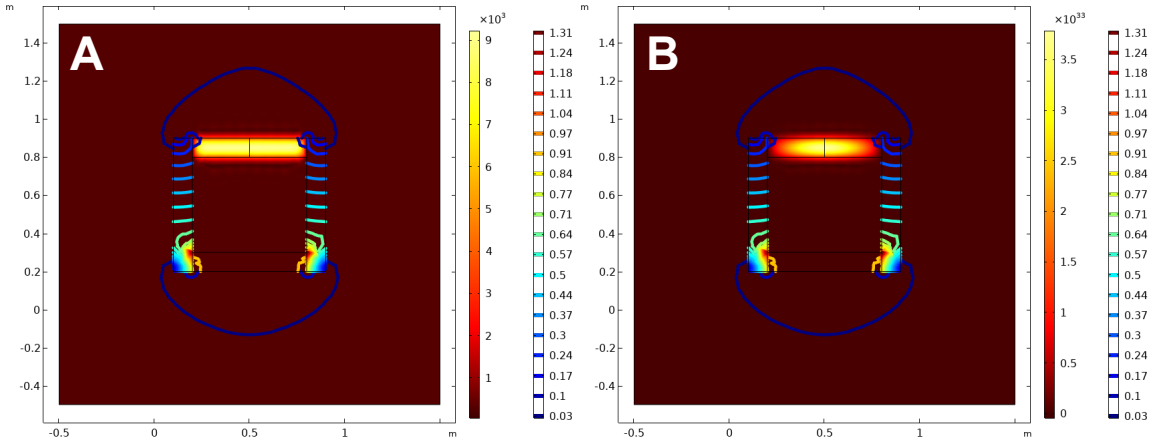

Figure 7: The thermal contour for the coupled model

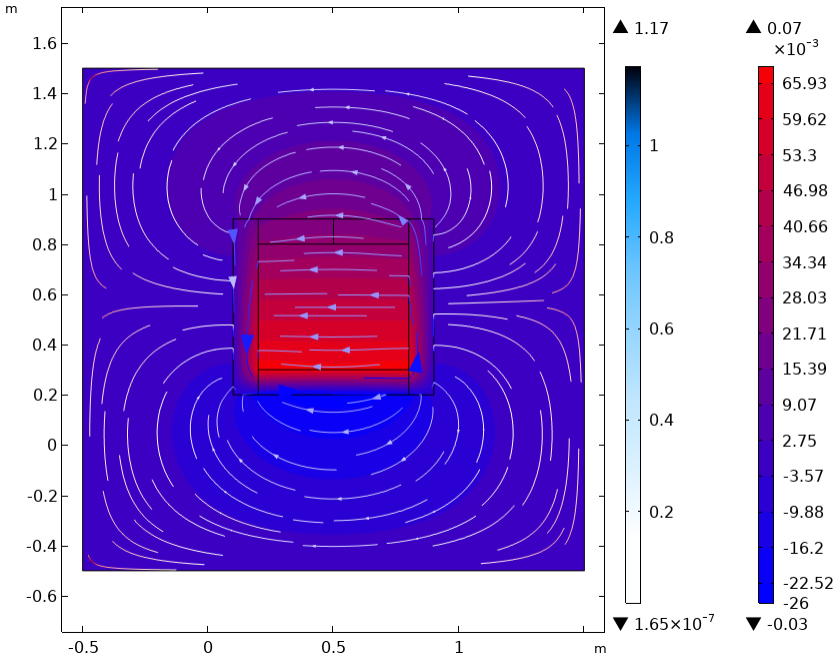

Figure 8: The magnetic flux contour for the coupled model.