Electromagnetic Characterisation of a Short-Stroke Ferromagnetic Actuator

R. M. Inston and H. Karimjee

***Abstract*—This experiment demonstrated the use of FEMM as an analysis tool for a short stroke ferromagnetic actuator.**

I. Nomenclature

R_w, Winding resistance, [Ω]
l_w, Length into the page of winding, [m]
A_w, Area of winding, [m^2]
V_w, Volume of winding, [m^3]
N, Number of turns in the winding, [-]
σ, Conductivity of winding, [Sm^{-1}]
k_{PF}, Packing factor of conductors in the winding, [-]
L, Inductance of the winding, [H]
$\mathcal{R}$, Reluctance, [H^{-1}]
A_{eff}, Effective area of air gap, [m^2]
W, Width of air gap, [m]
T, Thickness of air gap, [m]
g, Air gap length, [m]

II. Introduction

Finite Element Method Magnetics (FEMM) is a subset of Finite Element Analysis (FEA) that specialises in electromagnetics. This tool, in combination with MatLab, can be used to program a series of analytical situations, from which the mechanical aspects of the system can be characterised. The actuator and core of the system share the same ferromagnetic properties, windings cover the top and bottom of the core and three air gaps of note exist.

III. Model Meshing

To perform FEA, a model is split into elements. The elements must be small enough to output an accurate enough answer despite the linearisation of the physics occuring. Conversely, the elements cannot be too small as otherwise the computational time increases to an unfeasible (or uneconomical) amount.

FEMM has a smart-meshing tool in which the software analyses the model and allocates a dense mesh where the analysis must be of higher resolution and a sparse mesh where it does not. This is evident in figures 1 & 2. Although useful, this feature increases the mesh elements (see table I) and it does not know any information about the overarching complexity of the problem. One example of this is the boundaries involved with moving armature. Smart-meshing fails to recognise these edges as paramount to the analysis and produces a grid seen in figure 3. By manually increasing the density of meshing along these lines, FEMM can produce a more functional mesh seen in figure 4. This accuracy has its cost in terms of mesh elements and thus computational time, as seen in table I.

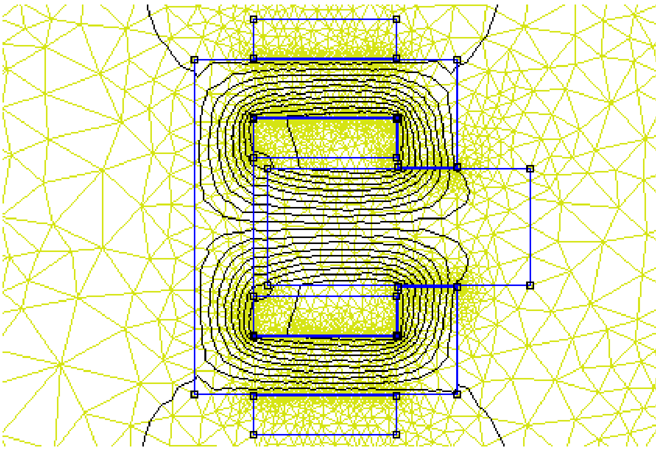

Fig. 1. Example mesh with smart-meshing disabled. Note the sparseness of the triangulation.

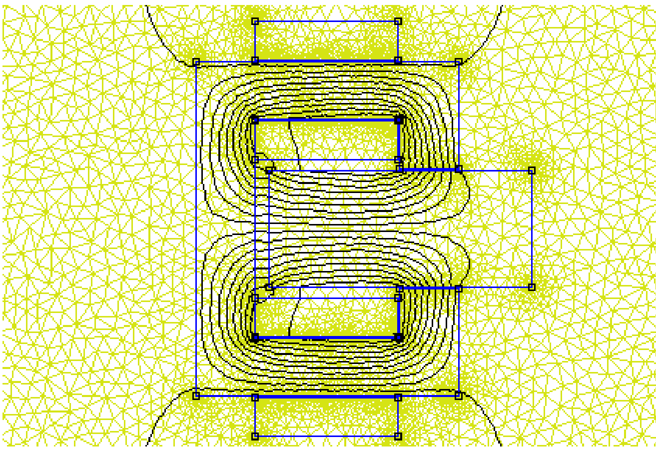

Fig. 2. Example mesh with smart meshing enabled. Note how the density increases around boundaries of interest.

IV. Winding Resistance

To calculate the winding resistance, R_w, two approaches were used: FEMM and analytical. FEMM can calculate the winding's resistance by getting the circuit properties of each winding and obtaining the voltage and current. The analytical

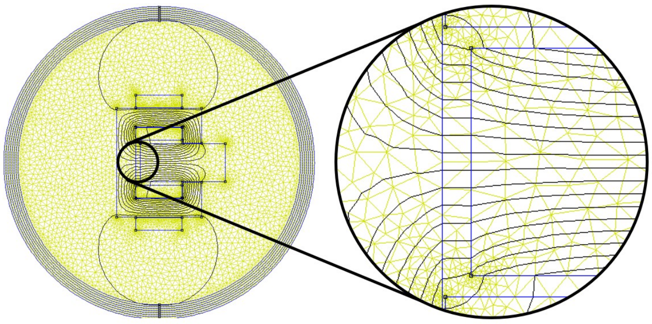

Fig. 3. FEMM smart-meshing output without specifying the area of interest. The density is low despite smart-meshing being on.

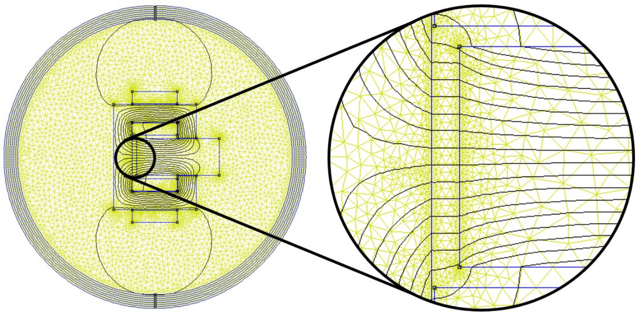

Fig. 4. FEMM smart-meshing output with specifying the area of interest and increasing the density of mesh elements within.

approach used block integrals in FEMM to obtain volume and area of the winding and hence the depth into the page can be found using equation 1:

$$l_w = \frac{V_w}{A_w} \tag{1}$$

The resistance is thus calculated using equation 2. The resistances of each calculation method are presented in table II.

$$R_w = \frac{N l_w}{\sigma \left(\frac{k_{PF} A_w}{N} \right)} \tag{2}$$

Notable in table II is the difference in estimated values. The analytical value uses the packing factor, k_{PF}, that compensates for the space used by the insulating material in a winding. FEMM sees the entire area as conducting material and hence the implied length of wire increases. Given the conductivity, σ, is the same in both predictions; the greater the implied length of winding, the greater the winding resistance, R_w. However, FEMM is working with a 2D model and the analysis was performed on the top section of the winding only. This means the analysis does not account for the mean path length around the winding, causing the FEMM estimate to be at least a factor of four out from the analytical prediction.

These two methods were plotted for many values of current and the power loss of the winding calculated. The resulting graph can be seen in figure 5.

Smart-meshing	**Dense Air gap**	**Mesh Elements**
OFF	OFF	14790
OFF	ON	16334
ON	OFF	22668
ON	ON	24368

TABLE I
A SUMMARY OF MESHING OPTIONS FOR THE MODEL.

Method	**Current [A]**	**Winding Resistance [Ω]**
FEMM	10	0.0107
Analytical	10	0.0557

TABLE II
A SUMMARY OF WINDING RESISTANCE CALCULATION FOR DIFFERENT METHODS.

V. Winding Inductance

The inductance of the windings can be found analytically and numerically. To estimate the value analytically, equation 3 is followed.

$$L = \frac{N^2}{\sum \mathcal{R}} \tag{3}$$

To find an analytical expression for the inductance where flux is assumed to be conserved:

$$\mathcal{R} = \frac{l}{\mu_0 \mu_r A} \tag{4}$$

$$\sum \mathcal{R} = \mathcal{R}_{core} + \mathcal{R}_{air} + \mathcal{R}_{armature} + \mathcal{R}_{gap} \tag{5}$$

Or fringing effects can be taken into account by substituting equation 6 into equation 4 for $\mathcal{R}_{gap}$ and $\mathcal{R}_{air}$.

$$A_{eff} = (W + 2g)(T + 2g) \tag{6}$$

This gives a total of two analytical values. Numerically, two values of inductance can be estimated using FEMM by using a linear or non-linear approximation of core and armature properties.

VI. Force on the Armature

VII. Conclusion

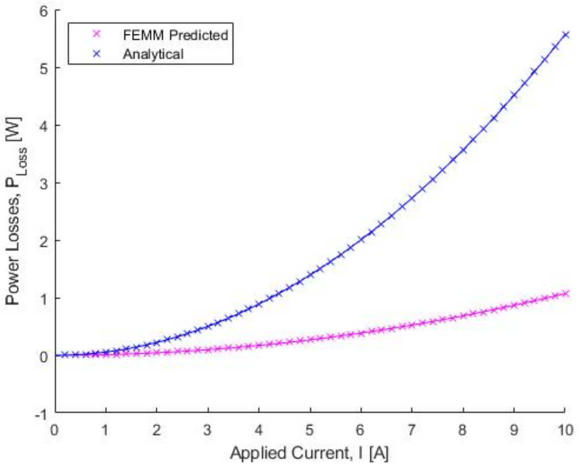

Fig. 5. Power loss against current shown for the FEMM predicted values and Analytical evaluation. A quadratic curve produces the line of best fit.

VIII. Appendix

The following is a listing of the Matlab script written to achieve this analysis.

```
% CI ACTUATOR FEMM ANALYSIS PROGRAM

% show matlab where femm files are and setup
addpath(genpath('C:\femm42'));
openfemm()
opendocument('femm_template.fem');
mi_saveas('actuator.fem');

% load in position and group no. for all components.
load('coil1p.mat');
load('coil2p.mat');
load('coil3p.mat');
load('coil4p.mat');
load('corep.mat');
load('moverp.mat');

% set create component array
components = {corep moverp coil1p coil2p coil3p coil4p};

mi_probdef(0, 'millimeters', 'planar', 1e-008, 20, 30, 0)

% add all nodes from data
mi_addnode(components{1}(:,1), components{1}(:,2))
mi_addnode(components{2}(:,1), components{2}(:,2))
mi_addnode(components{3}(:,1), components{3}(:,2))
mi_addnode(components{4}(:,1), components{4}(:,2))
mi_addnode(components{5}(:,1), components{5}(:,2))
mi_addnode(components{6}(:,1), components{6}(:,2))

% set node groups and draw lines for all components
for i = 1:length(components)
    modifyNodes(components{i});
    addLines(components{i});
end

% create array with coordinates of centre's of component blocks
blockCoords = [(min(components{1}(:,1)) + 3) mean(components{1}(:,2));
               mean(components{2}(:,1)) mean(components{2}(:,2));
               mean(components{3}(:,1)) mean(components{3}(:,2));
               mean(components{4}(:,1)) mean(components{4}(:,2));
               mean(components{5}(:,1)) mean(components{5}(:,2));
               mean(components{6}(:,1)) mean(components{6}(:,2));
               -25 0];

% create array of block properties
blockProps = {'core_linear' 1 0 '<None>' 0 1 0;
              'core_linear' 1 0 '<None>' 0 2 0;
              'copper' 1 0 'winding_1' 0 3 100;
              'copper' 1 0 'winding_1' 0 4 -100;
              'copper' 1 0 'winding_2' 0 5 100;
              'copper' 1 0 'winding_2' 0 6 -100;
              'air' 1 0 '<None>' 0 7 0};
```

```
% for all the different blocks, add labels and set its properties
for i = 1:max(size(blockCoords))
    mi_addblocklabel(blockCoords(i,1), blockCoords(i,2));
    mi_selectlabel(blockCoords(i,1), blockCoords(i,2));
    mi_setblockprop(blockProps{i,1}, blockProps{i,2}, ...
                    blockProps{i,3}, blockProps{i,4}, ...
                    blockProps{i,5}, blockProps{i,6}, ...
                    blockProps{i,7});
    mi_clearselected
end

% create system boundary, switch on smart-meshing & set winding currents
mi_makeABC();
smartmesh(1);
mi_setcurrent('winding_1', 0);
mi_setcurrent('winding_2', 0);




% set number of steps for the armature to move and initialise 3D psi array
numOfSteps = 10;
psi = zeros(numOfSteps,4,10);

% for all the different currents, get inductances and hence psi values
for i = 10:1:30
    mi_setcurrent('winding_1', i);
    mi_setcurrent('winding_2', i);
    inductances = getInductances(blockProps, blockCoords, numOfSteps);
    psi(:,:,i) = inductances(:,2:5) * i;
end

% figure('Name', 'Psi - I')
% hold on
%
% plot(2:2:10, psi(1,4,:), 'x')
% plot(2:2:10, psi(2,4,:), 'x')
% plot(2:2:10, psi(3,4,:), 'x')
% plot(2:2:10, psi(4,4,:), 'x')
% plot(2:2:10, psi(5,4,:), 'x')
% hold off

figure('Name', 'Inductance - displacement')
hold on
plot(inductances(:,1), inductances(:,2), 'x');
plot(inductances(:,1), inductances(:,3), 'x');
plot(inductances(:,1), inductances(:,4), 'x');
plot(inductances(:,1), inductances(:,5), 'x');
hold off

mi_saveas('actuator.fem');

function inductances = getInductances(blockProps, blockCoords, numOfSteps)
    % reset armature to zero, define max & min positions & step size
    armaturePos = 0.0;
```

```matlab
maxPos = -4.9;
minPos = 0.0;
stepSize = (maxPos-minPos)/(numOfSteps-1);

% initialise arrays for each type of analysis (linear/non-linear)
linearInductances = zeros(numOfSteps, 4);
nonlinearInductance = zeros(numOfSteps,1);

% set the core and mover properties to linear
for i = 1:2
    mi_selectlabel(blockCoords(i,1), blockCoords(i,2));
    mi_setblockprop('core_linear', blockProps{i,2}, ...
                    blockProps{i,3}, blockProps{i,4}, ...
                    blockProps{i,5}, blockProps{i,6}, ...
                    blockProps{i,7});
    mi_clearselected
end

% for all the linear states, analyse and get inductances
for i = 1:numOfSteps
    mi_purgemesh();
    mi_createmesh();
    linearInductances(i,:) = getLinearInductances(armaturePos);
    % catch to stop the mover being displaced into the core on last
    % iteration
    if i < numOfSteps
        moveArmature(stepSize);
        armaturePos = armaturePos + stepSize;
    end
end

% reset armature to zero displacement
moveArmature(-armaturePos);
armaturePos = 0.0;

% set core and mover properties to non-linear
for i = 1:2
    mi_selectlabel(blockCoords(i,1), blockCoords(i,2));
    mi_setblockprop('core_nonlinear', blockProps{i,2}, ...
                    blockProps{i,3}, blockProps{i,4}, ...
                    blockProps{i,5}, blockProps{i,6}, ...
                    blockProps{i,7});
    mi_clearselected
end

% for all the non-linear states, analyse and get inductances
for i = 1:numOfSteps
    mi_purgemesh();
    mi_createmesh();
    nonlinearInductance(i) = nonlinearInductances();
    if i < numOfSteps
        moveArmature(stepSize);
        armaturePos = armaturePos + stepSize;
    end
end
```

```
    % reset armature to zero displacement
    moveArmature(-armaturePos);
    armaturePos = 0.0;

    % collate all data outcomes
    inductances = [linearInductances(:,1) linearInductances(:,2) ...
                   linearInductances(:,3) linearInductances(:,4) ...
                   nonlinearInductance(:,1)];
end

function linearInductances = getLinearInductances(armaturePos)
    g = 5 + armaturePos;
    Rcore = (134.5e-3)/(4e-7 * pi * 1000 * 400e-6);
    Rair = (0.5e-3)/(4e-7 * pi * 400e-6);
    Rarmature = ((70 - g)*10^-3)/(4e-7 * pi * 1000 * 400e-6);
    Rairvariable = (g*10^-3)/(4e-7 * pi * 400e-6);
    Rtot = Rcore + Rair + Rarmature + Rairvariable;
    Rairfringe = (0.5e-3)/(4e-7 * pi * (20e-3 + g*10^-3)^2);
    Rairvariablefringe = (g*10^-3)/(4e-7 * pi * (20e-3 + g*10^-3)^2);
    Rtotfringe = Rcore + Rairfringe + Rarmature + Rairvariablefringe;
    Lanalytical = (100^2)/Rtot;
    Lanalyticalfringe = (100^2)/Rtotfringe;
    mi_saveas('linear.fem');
    mi_analyze();
    mi_loadsolution();
    CP = mo_getcircuitproperties('winding_1');
    Lnumericallinear = CP(3)/CP(1);
    linearInductances = [armaturePos Lanalytical ...
                         Lanalyticalfringe Lnumericallinear];
end



function Lnumericalnonlinear = nonlinearInductances()

    mi_saveas('nonlinear.fem');
    mi_analyze();
    mi_loadsolution();
    CP = mo_getcircuitproperties('winding_1');
    Lnumericalnonlinear = CP(3)/CP(1);

end

function moveArmature(dx)
    mi_selectgroup(2)
    mi_movetranslate(dx,0)
    mi_clearselected()
end

function modifyNodes(component)
    for j = 1:size(component,1)
        mi_selectnode(component(j,1), component(j,2));
        mi_setnodeprop(component, component(j,3));
        mi_clearselected
    end
end
```

```
function addLines(component)
    for j = 1:size(component,1)
        if j<size(component,1)
            mi_addsegment(component(j,1), component(j,2), ...
                          component(j+1,1), component(j+1,2));
            if j == 10 && component(j,3) == 1
                mi_selectsegment(midpoint([component(j,1) component(j,2)...
                                 component(j+1,1) component(j+1,2)]));
                mi_setsegmentprop('<None>', 0.5, 0, 0, component(j,3));
            end
            mi_selectsegment(midpoint([component(j,1) component(j,2) ...
                                 component(j+1,1) component(j+1,2)]));
            mi_setsegmentprop('<None>', 0, 1, 0, component(j,3));
            mi_clearselected
        else
            mi_addsegment(component(j,1), component(j,2), ...
                          component(1,1), component(1,2));
            if component(j,3) == 2
                mi_selectsegment(midpoint([component(j,1) component(j,2)...
                                 component(1,1) component(1,2)]));
                mi_setsegmentprop('<None>', 0.5, 0, 0, component(j,3));
            end
            mi_selectsegment(midpoint([component(j,1) component(j,2) ...
                                 component(1,1) component(1,2)]));
            mi_setsegmentprop('<None>', 0, 1, 0, component(j,3));
            mi_clearselected
        end
    end
end

function mid = midpoint(coords)
    mid = [(coords(1) + coords(3))/2 (coords(2) + coords(4))/2];
end
```