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. Introduction

II. Winding Resistance

III. Winding Inductance

IV. Force on the Armature

V. Conclusion

VI. 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};
```

```matlab
% 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;
```

```
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
```

```matlab
    % 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
```