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 an implementation of Finite Element Analysis (FEA) that specialises in electro-magnetics. This tool, in combination with MATLAB or another scripting language such as Python, can be used to create a series of numerical situations, from which a electro-magnetic and then mechanical analysis can be extracted.

Here, FEMM is being used to characterise a magnetic actuator, made up of an armature and a core which share the same ferromagnetic properties, two sets of windings around the top and bottom of the core and three air gaps of note, as in figure 1. This characterisation will be done analytically, neglecting and accounting for fringing, and numerically, using both a linear and non-linear ferromagnetic material.

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, whilst not being too small so as to increase the computational time to an unreasonable length.

FEMM has a smart-meshing tool in which the software analyses the model and allocates a dense mesh where higher resolution analysis is required and a sparse mesh where it is not. This is evident in the difference between figures 2 & 3. Although useful, this feature increases the mesh elements (see table I) without knowing any information about the overarching aim of the problem. This is illustrated at the boundaries of the variable air gap between the movable armature and core, which is vital for the analysis of the force-displacement characteristic of the actuator. Smart-meshing fails to recognise these edges as paramount to the analysis and produces a grid seen in figure 4. By manually setting the spacing of meshing along these lines, FEMM can produce a much denser, more regular mesh seen in figure 5. This accuracy has its cost in terms of mesh elements and computational time, as seen in table I.

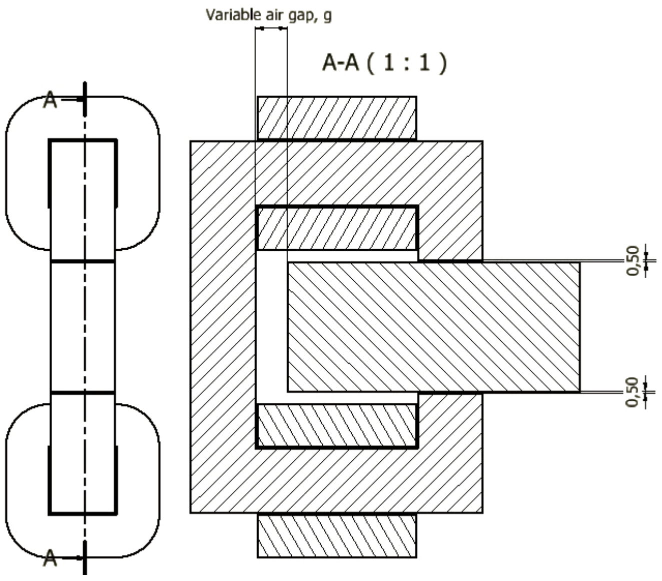

Fig. 1. Cross-sectional view of the actuator with the three air gaps higlighted (All measurements in mm).

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.

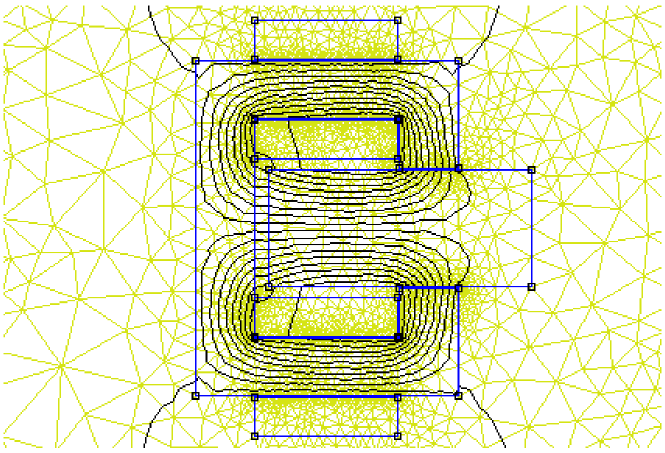

Fig. 2. Example mesh with smart-meshing disabled. Note the sparseness of the triangulation.

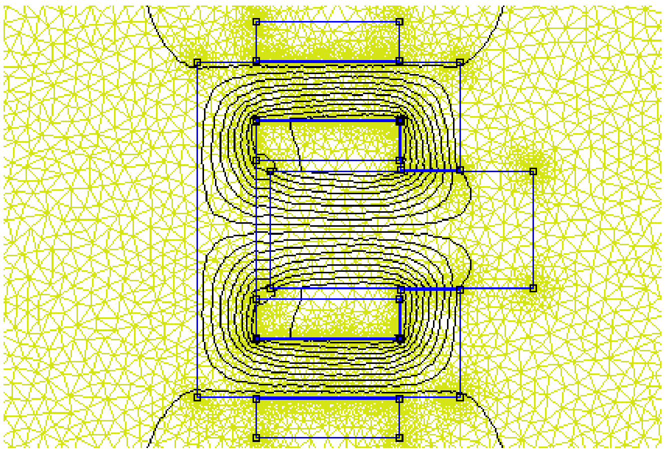

Fig. 3. Example mesh with smart meshing enabled. Note how the density increases around boundaries of interest.

IV. Winding Resistance

To calculate the winding resistance, R_w, two approaches were used: extracting resistance from FEMM and analytical. FEMM can calculate the winding resistance by obtaining the voltage and current across and through the winding respectively, and finding the resistance using Ohm's law. The analytical approach used block integrals in FEMM to obtain winding area and volume, and hence the winding depth (into the page) can be found using equation 1:

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

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

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

Table II shows a significant difference in estimated values. The analytical value uses a packing factor, $k_{PF} = 0.6$, that compensates for the space used by the insulating material in

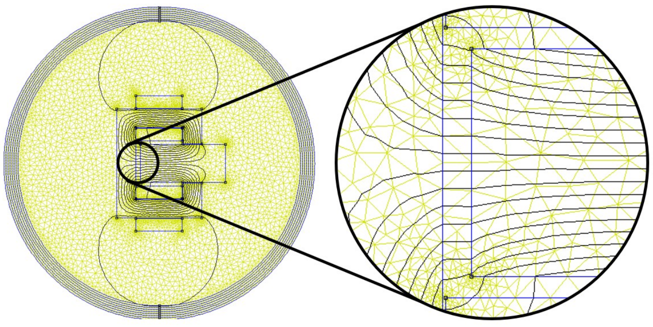

Fig. 4. FEMM smart-meshing output without specifying the area of interest. The density is low despite smart-meshing being on.

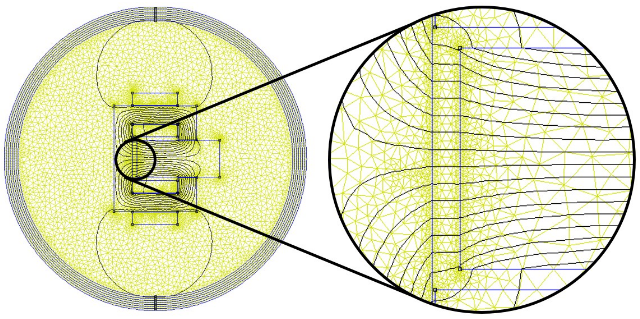

Fig. 5. FEMM smart-meshing output with specifying the area of interest and increasing the density of mesh elements within.

a winding. FEMM sees the entire area as conducting material and hence the implied cross sectional area of wire increases. Given the conductivity $\sigma = 58\mathrm{MS/m}$ 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 and bottom sections of the winding only. This means the analysis does not account for the whole length around the winding, notably the rounded corners and sides, causing the FEMM estimate to be at least a factor of four smaller than the analytical prediction.

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.

The resistances resulting from these two methods were used to calculate the power loss through the windings for a range of current values using equation 3. This was then plotted giving the graph in figure 6. Clearly, this shows the power loss based on the analytically calculated resistance quickly becomes far greater than that of the FEMM calculated resistance. As such, if the analysis performed on the actuator were to be taken further into real applications the analytical power loss should be considered when specifying power dissipation requirements.

$$P_{loss} = I^2 R \tag{3}$$

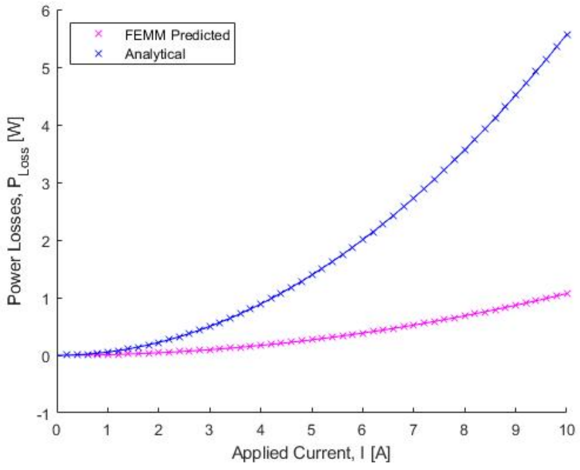

Fig. 6. Power loss against current based on equation 3 for analytical and FEMM calculated resistances. A quadratic curve produces the line of best fit.

V. Winding Inductance

The inductance of the windings can be found analytically and numerically. To estimate the value analytically, equation 4 is followed.

$$L = \frac{N^2}{\sum \mathcal{R}} \tag{4}$$

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

$$\mathcal{R} = \frac{l}{\mu_0 \mu_r A} \tag{5}$$

$$\sum \mathcal{R} = \mathcal{R}_{core} + \mathcal{R}_{air} + \mathcal{R}_{armature} + \mathcal{R}_{gap} \tag{6}$$

Or fringing effects can be taken into account by substituting equation 7 into equation 5 for $\mathcal{R}_{gap}$ and $\mathcal{R}_{air}$.

$$A_{eff} = (W + 2g)(T + 2g) \tag{7}$$

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. The inductance varies with the displacement of the armature, evident in figure 7.

VI. Force on the Armature

The force on the armature of the actuator is exponentially related to its position relative to the core. As the armature approaches the core the air-gap narrows, and so the reluctance of the magnetic circuit decreases and hence the inductance of the coil increases. This allows a stronger magnetic field for a given winding current, and therefore a greater force applied to the armature.

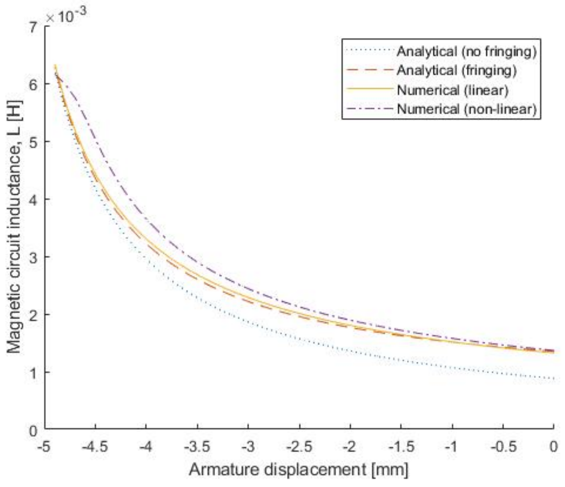

Fig. 7. Graph showing inductance versus armature displacement for the varying methods of calculation.

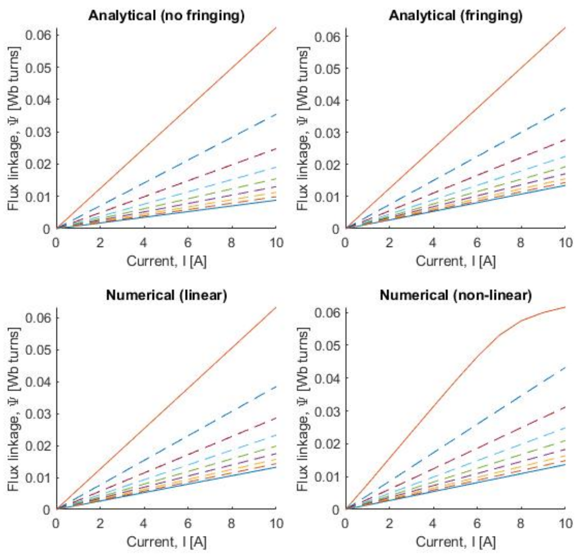

Fig. 8. Ψ-I diagrams for the four methods of calculating inductance. Note saturation reached in the non-linear graph. Shown for zero displacement, maximum displacement (-49mm) and seven linearly spaced intervals between (dashed lines).

VII. Conclusion

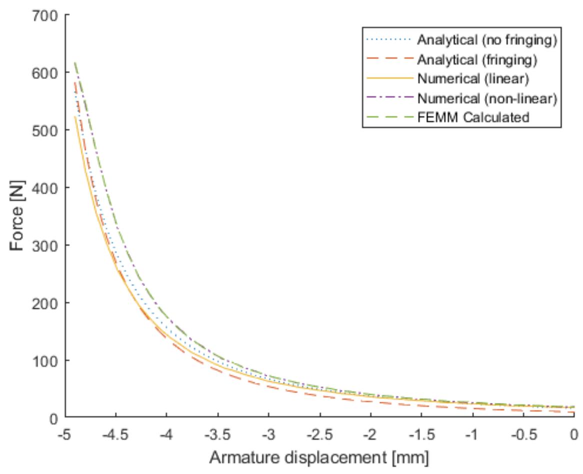

Fig. 9. Force versus displacement for the four methods of calculating inductance. Tabulated values evident in table III.

Armature Displacement [mm]	**Change in Co-Energy [J]**			
	Analytical (no fringing)	**Analytical (fringing)**	**Numerical (linear)**	**Numerical (non-linear)**
0.200	7.867	6.620	8.442	9.110
0.300	8.158	6.888	8.264	8.826
0.400	8.467	7.172	8.930	9.635
0.500	8.793	7.473	8.977	9.694
0.600	9.138	7.791	9.511	10.27
0.700	9.505	8.128	9.589	10.33
0.800	9.893	8.486	10.09	10.93
0.900	10.31	8.867	10.74	11.68
1.000	10.75	9.272	10.91	11.83
1.100	11.21	9.703	11.67	12.68
1.200	11.71	10.16	11.64	12.64
1.300	12.25	10.66	12.39	13.55
1.400	12.82	11.18	12.58	13.72
1.500	13.43	11.75	13.20	14.43
1.600	14.09	12.35	14.20	15.62
1.700	14.79	13.00	14.82	16.31
1.800	15.55	13.71	15.34	16.87
1.900	16.37	14.46	16.03	17.72
2.000	17.26	15.28	17.30	19.13
2.100	18.22	16.17	17.62	19.60
2.200	19.26	17.14	18.63	20.66
2.300	20.40	18.19	19.74	22.06
2.400	21.64	19.33	21.20	23.68
2.500	23.00	20.59	22.11	24.81
2.600	24.48	21.97	23.88	26.92
2.700	26.12	23.48	24.86	28.09
2.800	27.93	25.16	26.49	29.95
2.900	29.93	27.01	28.66	32.69
3.000	32.15	29.07	30.61	34.99
3.100	34.64	31.38	32.77	37.65
3.200	37.42	33.96	35.27	40.75
3.300	40.55	36.87	38.03	44.09
3.400	44.09	40.16	41.48	48.39
3.500	48.11	43.90	44.66	52.40
3.600	52.71	48.19	49.25	58.24
3.700	58.01	53.12	53.90	64.19
3.800	64.14	58.85	59.57	71.51
3.900	71.30	65.54	65.68	79.57
4.000	79.74	73.42	73.48	89.84
4.100	89.76	82.81	82.65	102.2
4.200	101.8	94.10	93.52	116.9
4.300	116.4	107.9	107.3	136.0
4.400	134.5	124.8	122.6	157.4
4.500	157.0	146.1	145.1	188.2
4.600	185.8	173.4	170.7	222.4
4.700	223.3	209.0	206.0	261.7
4.800	273.5	256.8	253.1	298.5
4.900	342.6	323.0	319.8	328.5

TABLE III
TABULATED DATA FOR THE CHANGE IN CO-ENERGY FOR ALL CHANGES IN POSITION, GIVEN TO 4 SIGNIFICANT FIGURES.

VIII. Appendix

The following is a listing of the Matlab script written to construct the actuator:

```
% 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);

mi_saveas('actuator.fem')


% function to add nodes to groups
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 to create lines between nodes in each group
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
```

THE FOLLOWING IS A LISTING OF THE MATLAB SCRIPT WRITTEN TO ANALYSE THE ACTUATOR:

```
addpath(genpath('C:\femm42'));

construct_actuator

% set number of steps for the armature to move and initialise 3D psi array
numOfSteps = 2;
psi = zeros(numOfSteps,4,11);

% for all the different currents, get inductances and hence psi values
for i = 0:1:10
    mi_setcurrent('winding_1', i);
    mi_setcurrent('winding_2', i);
    inductances = getInductances(blockProps, blockCoords, numOfSteps);
    psi(:,:,i+1) = inductances(:,2:5) * i;
end

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

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

function inductances = getInductances(blockProps, blockCoords, numOfSteps)
    % reset armature to zero, define max & min positions & step size
    armaturePos = 0.0;
    maxPos = -5;
    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
```

THE FOLLOWING IS A LISTING OF THE MATLAB SCRIPT WRITTEN TO PRODUCE THE GRAPHS AND TABLES:

```
load('L.mat');
linetypes = [":", "--", "-", "-."];

figure()
hold on
for i = 1:1:4
    x = L(1:49,1,11);
    y = L(1:49,i+1,11);
    plot(x, y, linetypes(i));
end
xlabel('Armature displacement [mm]');
ylabel('Magnetic circuit inductance, L [H]');
legend('Analytical (no fringing)', 'Analytical (fringing)', ...
       'Numerical (linear)', 'Numerical (non-linear)')
hold off

figure()
subplot(2,2,1)
hold on
for i = 1:6:49
    x = 0:1:10;
    y = [];
    for j = 1:1:11
        if j ~= 1
            y(j) = L(i,2,j)*(j-1);
        else
            y(j) = 0;
        end
    end
    if i == 1 || i == 49
        plot(x, y, '-');
    else
        plot(x, y, '--');
    end
end
xlabel('Current, I [A]');
ylabel('Flux linkage, \Psi [Wb turns]');
title('Analytical (no fringing)');
hold off

subplot(2,2,2)
hold on
for i = 1:6:49
    x = 0:1:10;
    y = [];
    for j = 1:1:11
        if j ~= 1
            y(j) = L(i,3,j)*(j-1);
        else
            y(j) = 0;
        end
    end
    if i == 1 || i == 49
        plot(x, y, '-');
    else
```

```
        plot(x, y, '--');
    end
end
xlabel('Current, I [A]');
ylabel('Flux linkage, \Psi [Wb turns]');
title('Analytical (fringing)');
hold off

subplot(2,2,3)
hold on
for i = 1:6:49
    x = 0:1:10;
    y = [];
    for j = 1:1:11
        if j ~= 1
            y(j) = L(i,4,j)*(j-1);
        else
            y(j) = 0;
        end
    end
    if i == 1 || i == 49
        plot(x, y, '-');
    else
        plot(x, y, '--');
    end
end
xlabel('Current, I [A]');
ylabel('Flux linkage, \Psi [Wb turns]');
title('Numerical (linear)');
hold off

subplot(2,2,4)
hold on
for i = 1:6:49
    x = 0:1:10;
    y = [];
    for j = 1:1:11
        if j ~= 1
            y(j) = L(i,5,j)*(j-1);
        else
            y(j) = 0;
        end
    end
    if i == 1 || i == 49
        plot(x, y, '-');
    else
        plot(x, y, '--');
    end
end
xlabel('Current, I [A]');
ylabel('Flux linkage, \Psi [Wb turns]');
title('Numerical (non-linear)');
hold off

figure()
hold on
```

```
linetypes = [":", "--", "-", "-."];
for count = 1:1:4
    for i = 1:1:49
        x = 0:1:10;
        y = [];
        for j = 1:1:11
            if j ~= 1
                y(j) = L(i,count+1,j)*(j-1);
            else
                y(j) = 0;
            end
        end
        CoEnergy(count,i) = trapz(x, y);
    end
end
for n = 1:1:4
   for i = 1:1:48
      F(i,n) = (CoEnergy(n,i+1) - CoEnergy(n,i))/((4.9e-3)/48);
   end
   plot(linspace(-4.9,0,48),flip(F(:,n)),linetypes(n));
end

xlabel('Armature displacement [mm]');
ylabel('Force [N]');
legend('Analytical (no fringing)', 'Analytical (fringing)', ...
        'Numerical (linear)', 'Numerical (non-linear)')
hold off

Ftable = round(F, 4, 'significant');
```