Assignment 2

Subject: MCEN90044 Electromagnetic Technologies

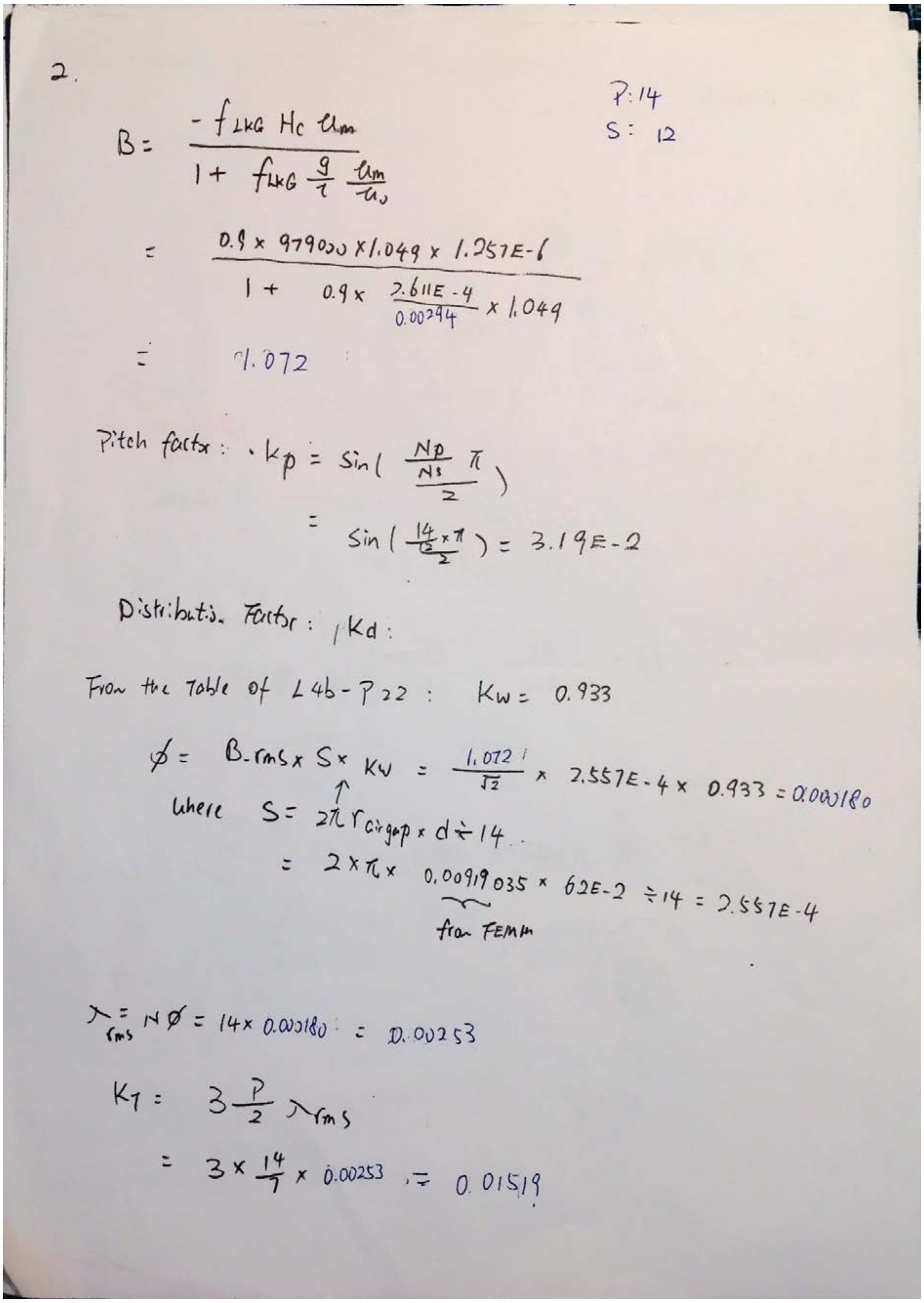
Name: Shiyu Zhang

Student Number: 683557

Part 1

1. Group Photo

2. The hand calculation of the analytical expression for torque constant Kt is shown in the figure below.



2.

P: 14
S: 12

$$B = \frac{-f_{LKG} H_c u_m}{1 + f_{LKG} \frac{g}{\tau} \frac{u_m}{u_0}}$$

$$= \frac{0.9 \times 979000 \times 1.049 \times 1.257E\text{-}6}{1 + 0.9 \times \frac{2.611E\text{-}4}{0.00294} \times 1.049}$$

$$= 1.072$$

Pitch factor: $\cdot k_p = \sin\left(\frac{\frac{N_p}{N_s} \pi}{2} \right)$

$$= \sin\left(\frac{\frac{14}{12} \times \pi}{2} \right) = 3.19E\text{-}2$$

Distribution Factor: K_d:

From the Table of L4b-P22: $K_w = 0.933$

$$\phi = \underset{\uparrow}{B_{rms} \times S} \times K_w = \frac{1.072}{\sqrt{2}} \times 2.557E\text{-}4 \times 0.933 = 0.000180$$

where $S = 2\pi r_{airgap} \times d \div 14$

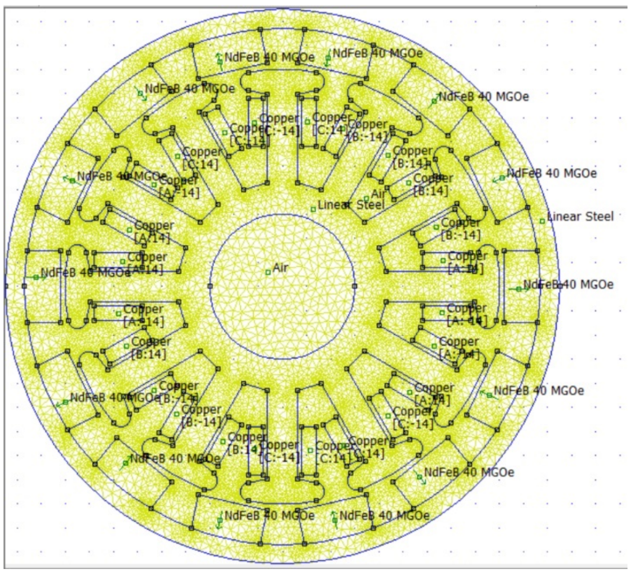
$$= 2 \times \pi \times \underbrace{0.00919035}_{\text{from FEMM}} \times 62E\text{-}2 \div 14 = 2.557E\text{-}4$$

$$\lambda_{rms} = N\phi = 14 \times 0.000180 = 0.00253$$

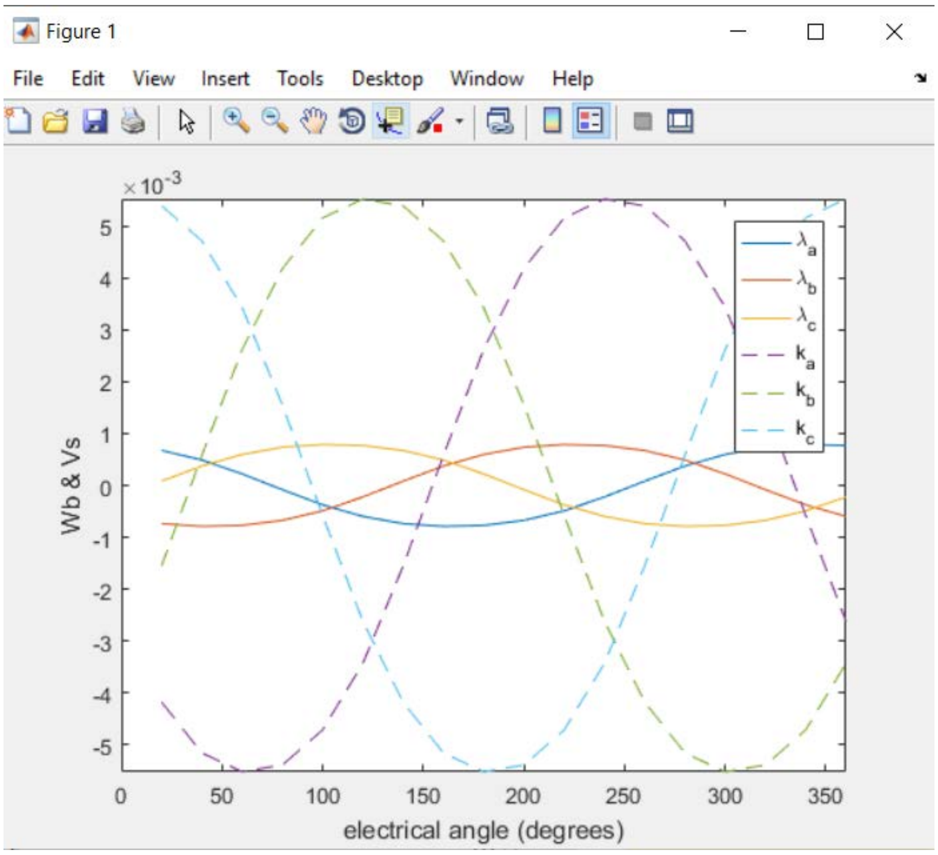
$$K_T = 3 \frac{P}{2} \lambda_{rms}$$

$$= 3 \times \frac{14}{7} \times 0.00253 \doteq 0.01519$$

3. A FEMM model has been created as the figure below show.



The script Q3_part1.m has run to analyse the CSV file generated by FEMM on the following parameters.



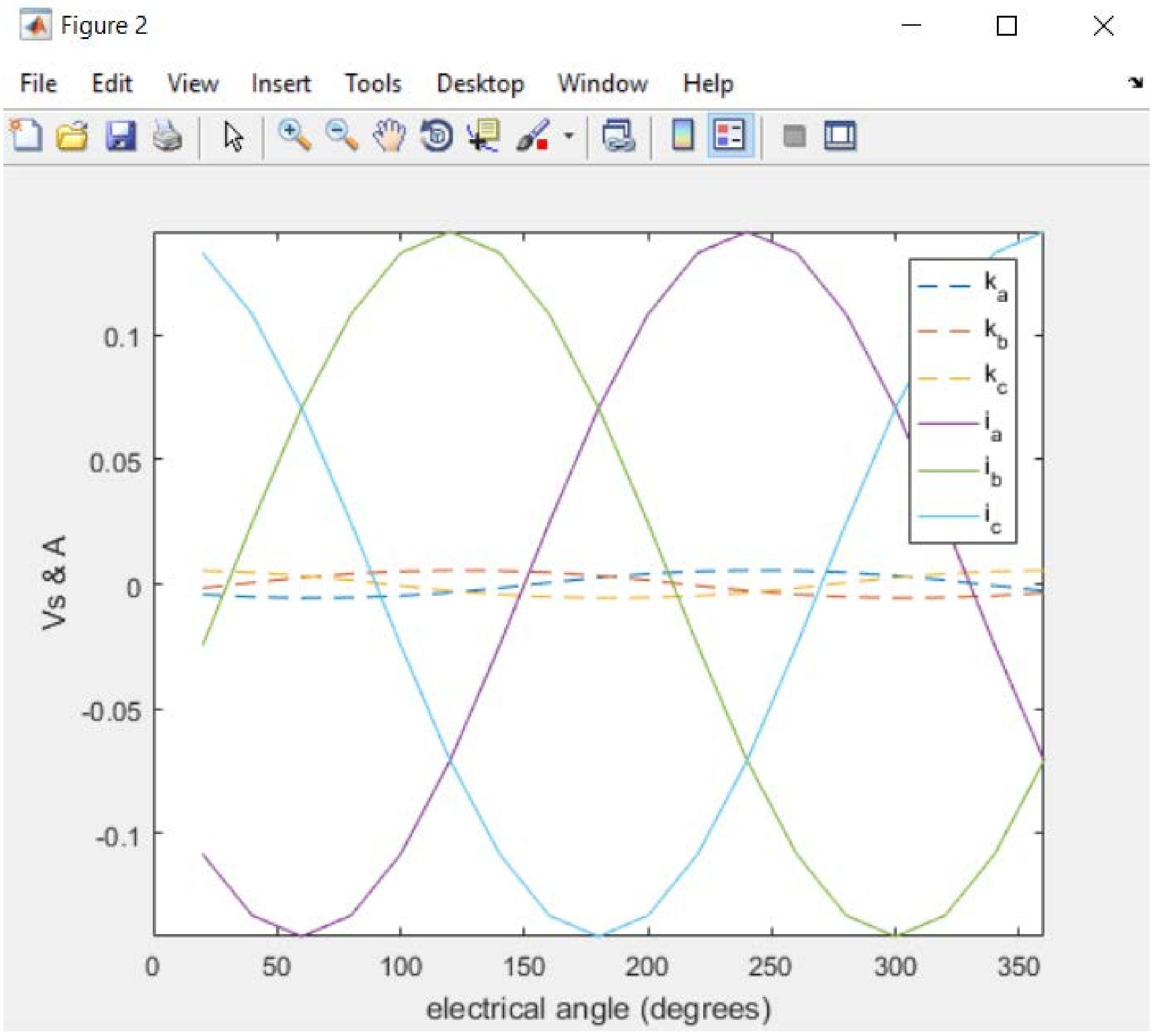
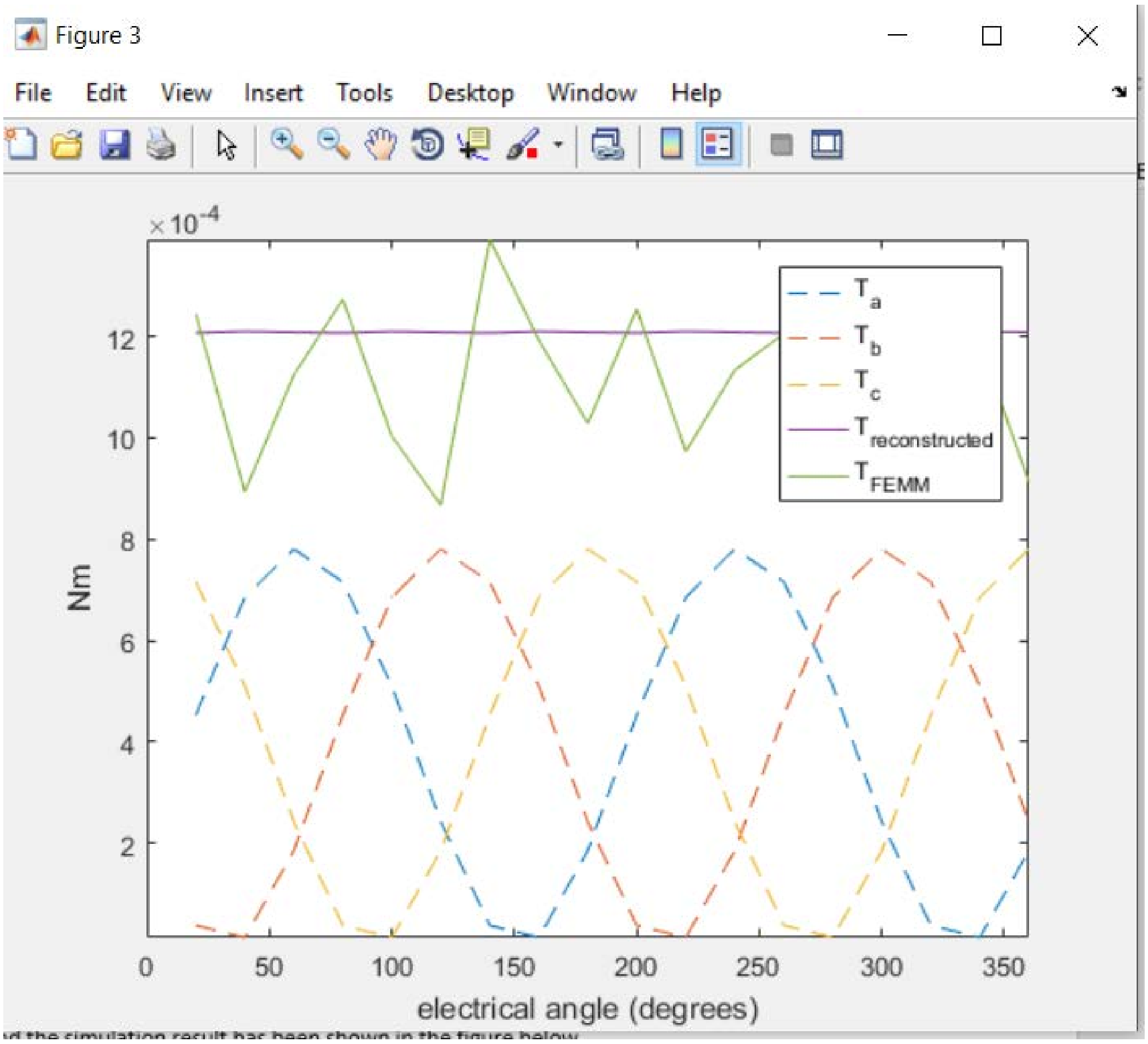


Figure 2
File
Edit
View
Insert
Tools
Desktop
Window
Help
k_a
k_b
k_c
i_a
i_b
i_c
0.1
0.05
0
-0.05
-0.1
Vs & A
0
50
100
150
200
250
300
350
electrical angle (degrees)



4.

The hand calculation result and the FEA model with high permeability linear steel of torque constant can be seen from the table below.

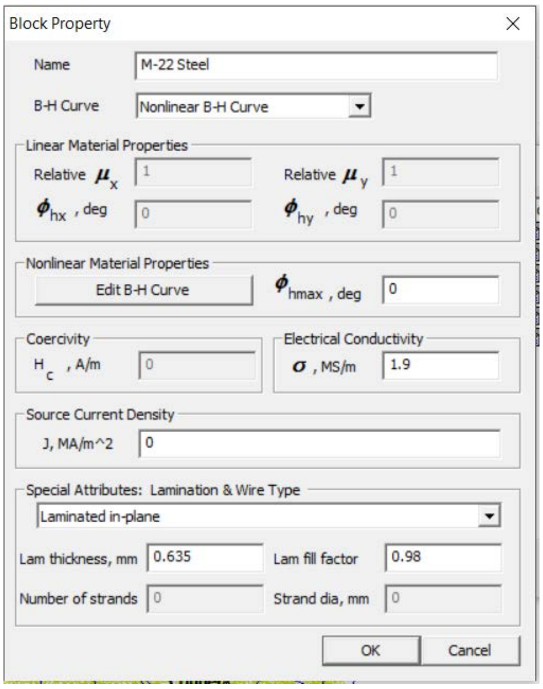
Method	Torque Constant(Nm/A)
Hand Calculation	0.01519
FEMM output to Matlab	0.01450

There are slight discrepancies between the two results. One of the reasons might be the flux leakage factor in hand calculation is a rough estimation, whereas FEMM can model this phenomenon quite well. The other reason might be the air gap between the coil area and the motor slot wasn't included into consideration in hand calculation whereas that has also been taken care of in the FEMM model.

5.

After looking up the 1806 motor catalogue, it can be found that the motor was built using laminated silicon steel to reduce hysteresis loss and core loss. Therefore, the B and H relationship would be nonlinear.

To improve this, the linear steel blocks in FEMM can be replaced with M22 Silicon Steel.



Block Property

Name	M-22 Steel
B-H Curve	Nonlinear B-H Curve

Linear Material Properties

Relative μ_x	1	Relative μ_y	1
ϕ_{hx}, deg	0	ϕ_{hy}, deg	0

Nonlinear Material Properties

Edit B-H Curve — ϕ_{hmax}, deg: 0

Coercivity — H_c, A/m: 0

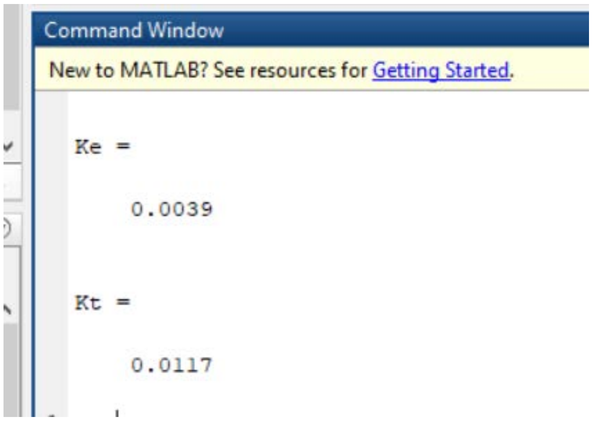
Electrical Conductivity — σ, MS/m: 1.9

Source Current Density — J, MA/m^2: 0

Special Attributes: Lamination & Wire Type — Laminated in-plane

Lam thickness, mm	0.635	Lam fill factor	0.98
Number of strands	0	Strand dia, mm	0

And the simulation result has been shown in the figure below.



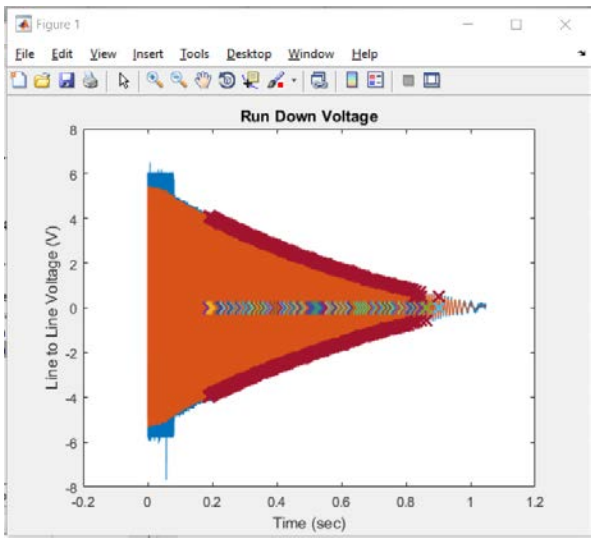
Command Window

New to MATLAB? See resources for Getting Started.

```
Ke =

    0.0039


Kt =

    0.0117
```

Kt is increased from 0.0014 to 0.017.

6.



The experimental torque can be found by modifying script *OutrunnerVideoExample_Part4.m* reading the back emf voltage csv file collected in the experiment (figure on the left), and the result has been found to be Kt = 0.0050.

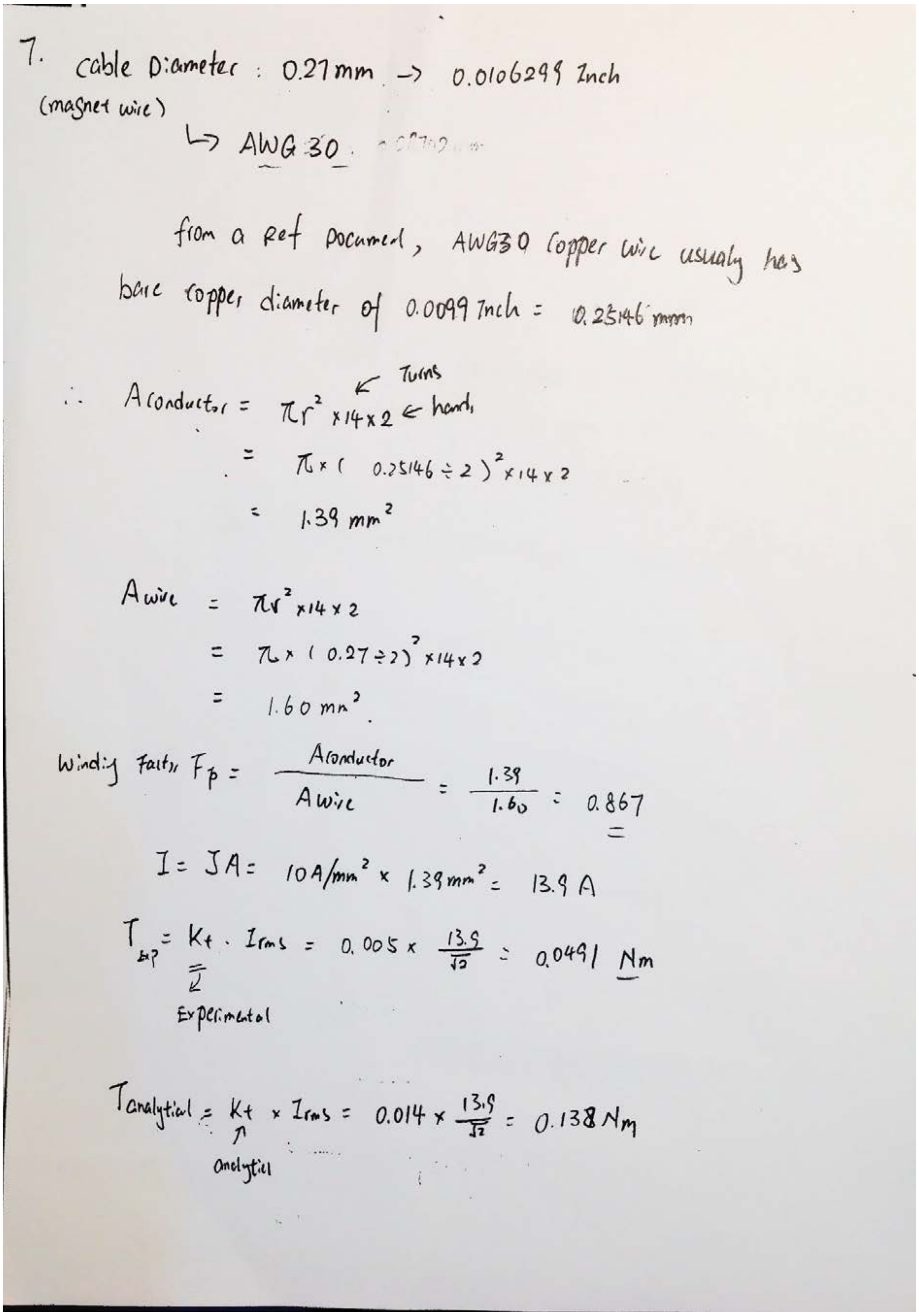
Matlab script Q6_part1.m has been used for this question.

To sum up with, the hand calculation, FEA, and experimental results has been found in the table below.

Method	Torque Constant(Nm/A)
Hand Calculation	0.01519
FEMM output to Matlab (Improved to M22 steel)	0.01450
Experimental	0.005

The cause of difference between Hand Calculation has been discussed in question 4. However, the Torque constant found by experiment is significantly smaller than that found by the rest of the method. This is due to different kind of loss that is not included in consideration when we model the system in FEMM and doing hand calculation. These losses include: 1. Conductor loss due to resistance in conductor; 2. Core loss due to hysteresis and eddy current; 3. Bearing loss due to the friction of the bearing in motor; 4. Solid loss due to eddy current in non-laminated materials.

7. The calculation of **wire fill factor** and **torque** with J = 10A/mm is shown below.



7. Cable Diameter : 0.27 mm $\rightarrow$ 0.0106299 Inch
(magnet wire)

$\hookrightarrow$ AWG 30 [illegible]

from a Ref Documed, AWG30 Copper wire usualy has bare copper diameter of 0.0099 Inch = 0.25146 mm

$\therefore$ $A_{conductor} = \pi r^2 \times 14 \times 2$ ← hand, ↙ Turns

$= \pi \times (0.25146 \div 2)^2 \times 14 \times 2$

$= 1.39\ mm^2$

$A_{wire} = \pi r^2 \times 14 \times 2$

$= \pi \times (0.27 \div 2)^2 \times 14 \times 2$

$= 1.60\ mm^2$.

Winding Factor $F_p = \dfrac{A_{conductor}}{A_{wire}} = \dfrac{1.39}{1.60} = \underline{\underline{0.867}}$

$I = JA = 10 A/mm^2 \times 1.39 mm^2 = 13.9\ A$

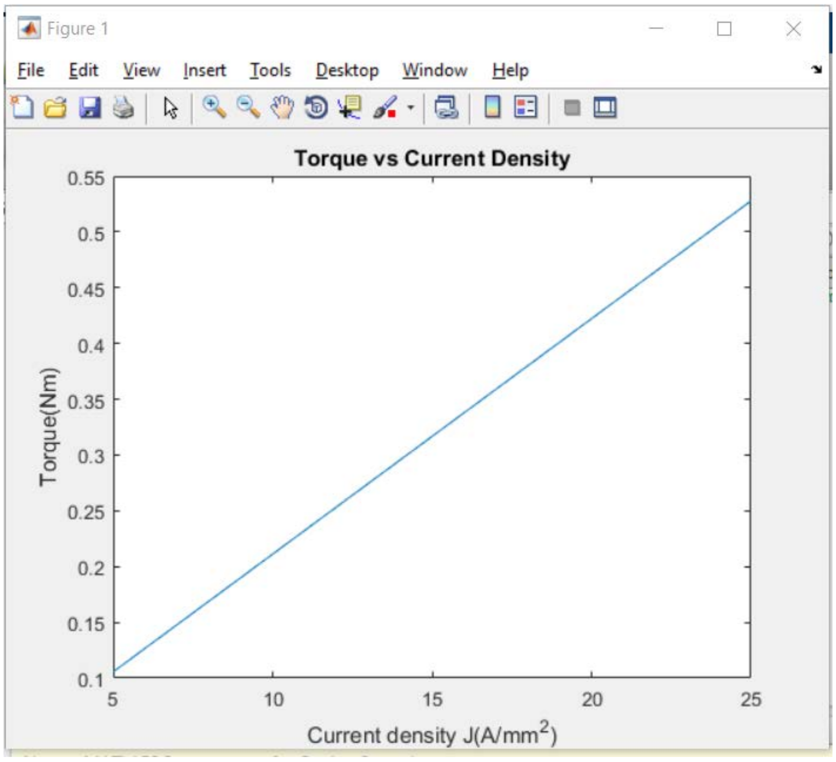
$T_{EXP} = \underset{\swarrow}{K_t} \cdot I_{rms} = 0.005 \times \dfrac{13.9}{\sqrt{2}} = 0.0491\ \underline{Nm}$

Experimental

$T_{analytical} = \underset{\uparrow}{K_t} \times I_{rms} = 0.014 \times \dfrac{13.9}{\sqrt{2}} = 0.138\ Nm$

analytical

8. The plot of torque vs current density has been shown in the figure below using Q8.m in the appendix.



9.

Know that the Motor velocity constant Kv is $K_v = \frac{\omega_{\text{No-Load}}}{V_{\text{Peak}}}$ = 2400 rpm/V = 251.33 rad/s*V.

Therefore, the back emf constant Ke = $\frac{1}{K_v} = \frac{1}{251.33} = 3.98E - 3.$

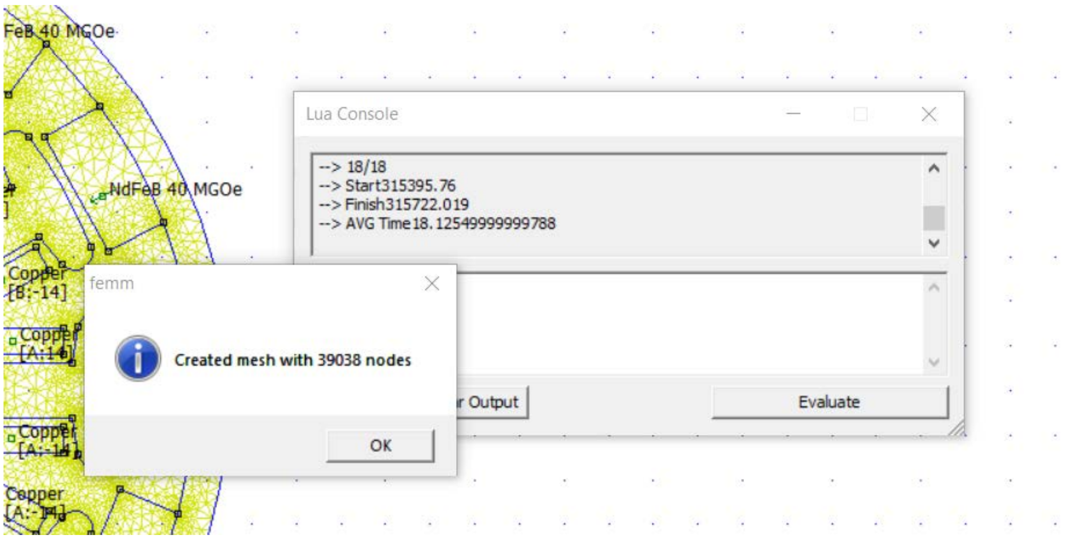
It is known that Ke = Kt when there is no loss.

However, Ke is very different from Kt_calculated_from_matlab = 0.014, and even smaller than Kt_experimental = 0.005.

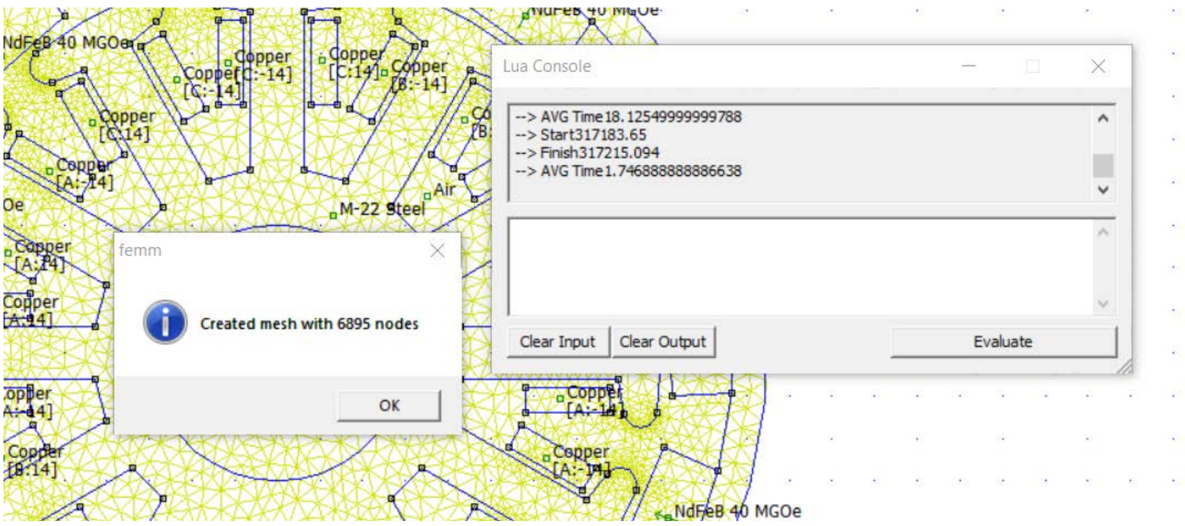
10.

By adjusting "local element size along the line" as well as "Maximum segment degrees" for arcs in the femm model, the following execution time has been estimated by printing the systems clock in the Lua script, and the Torque constant was estimated y running the provided Matlab script.

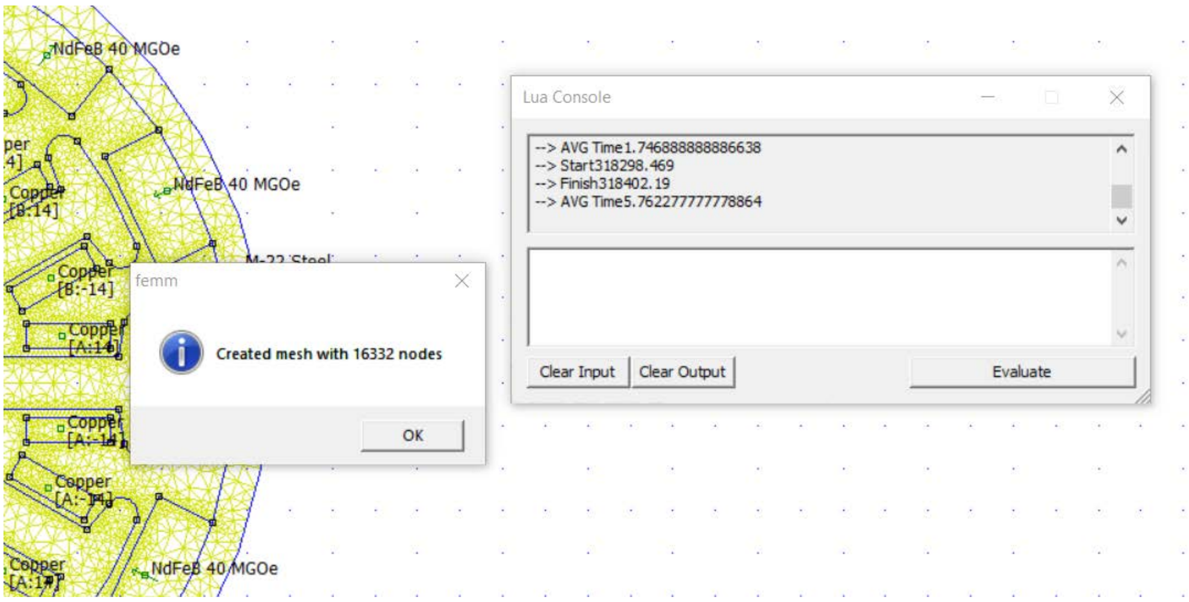
Lua script


```
--> 18/18
--> Start315395.76
--> Finish315722.019
--> AVG Time18.12549999999788
```

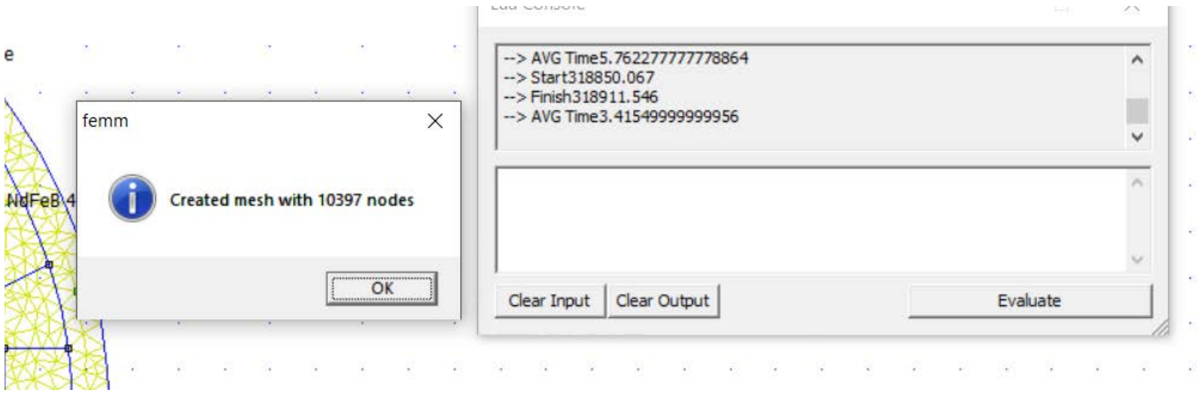
Created mesh with 39038 nodes


```
--> AVG Time18.12549999999788
--> Start317183.65
--> Finish317215.094
--> AVG Time1.746888888886638
```

Created mesh with 6895 nodes


```
--> AVG Time1.746888888886638
--> Start318298.469
--> Finish318402.19
--> AVG Time5.762277777778864
```

Created mesh with 16332 nodes



Created mesh with 10397 nodes

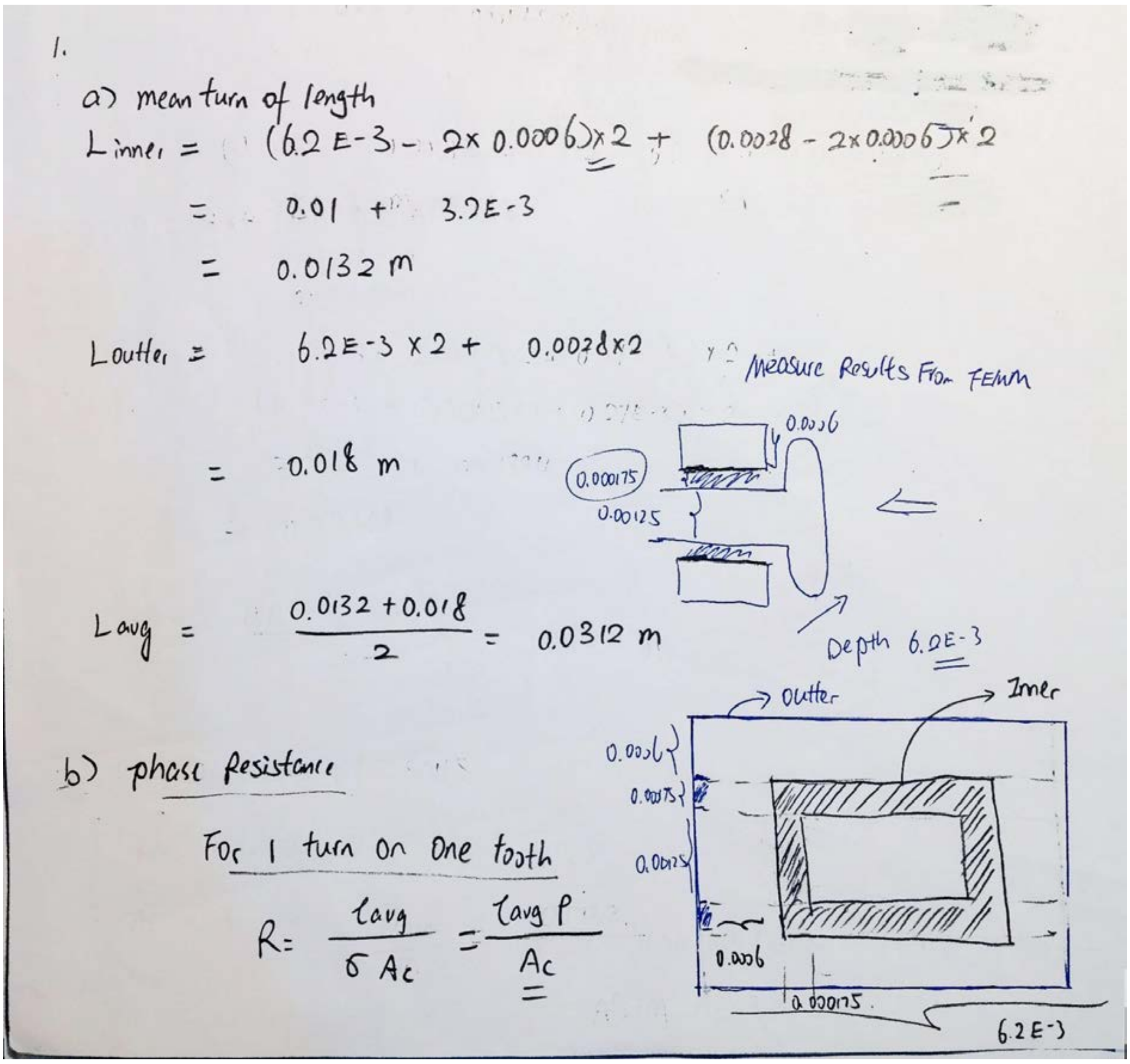
```
--> AVG Time5.762277777778864
--> Start318850.067
--> Finish318911.546
--> AVG Time3.41549999999956
```

Number of nodes	Execution Time(s)	Torque constant from Matlab (Nm/A)
6895	32	0.0118
10397	61.479	0.0118
16332	103	0.0117
39038	326.259	0.0117

As shown in the table above, number of nodes has significant impact on running time but only have minor effect on the Torque constant calculation.

Part 2

1.



1.

a) mean turn of length

$L_{inner} = (6.2E-3 - 2\times 0.0006)\times 2 + (0.0028 - 2\times 0.0006)\times 2$

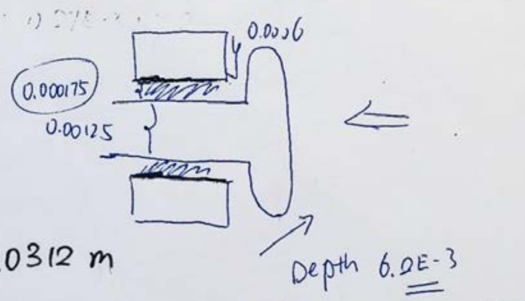
$= 0.01 + 3.2E-3$

$= 0.0132\ m$

$L_{outter} = 6.2E-3 \times 2 + 0.0028\times 2$

$= 0.018\ m$

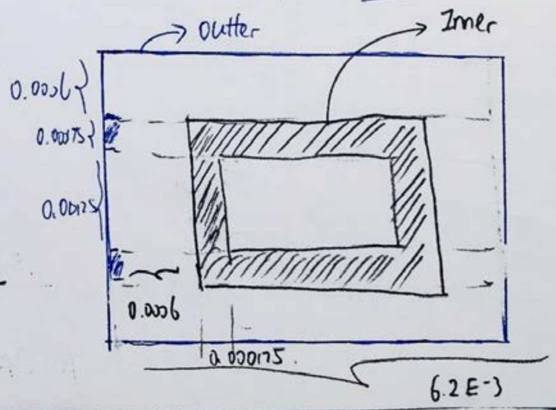
$$L_{avg} = \frac{0.0132 + 0.018}{2} = 0.0312\ m$$

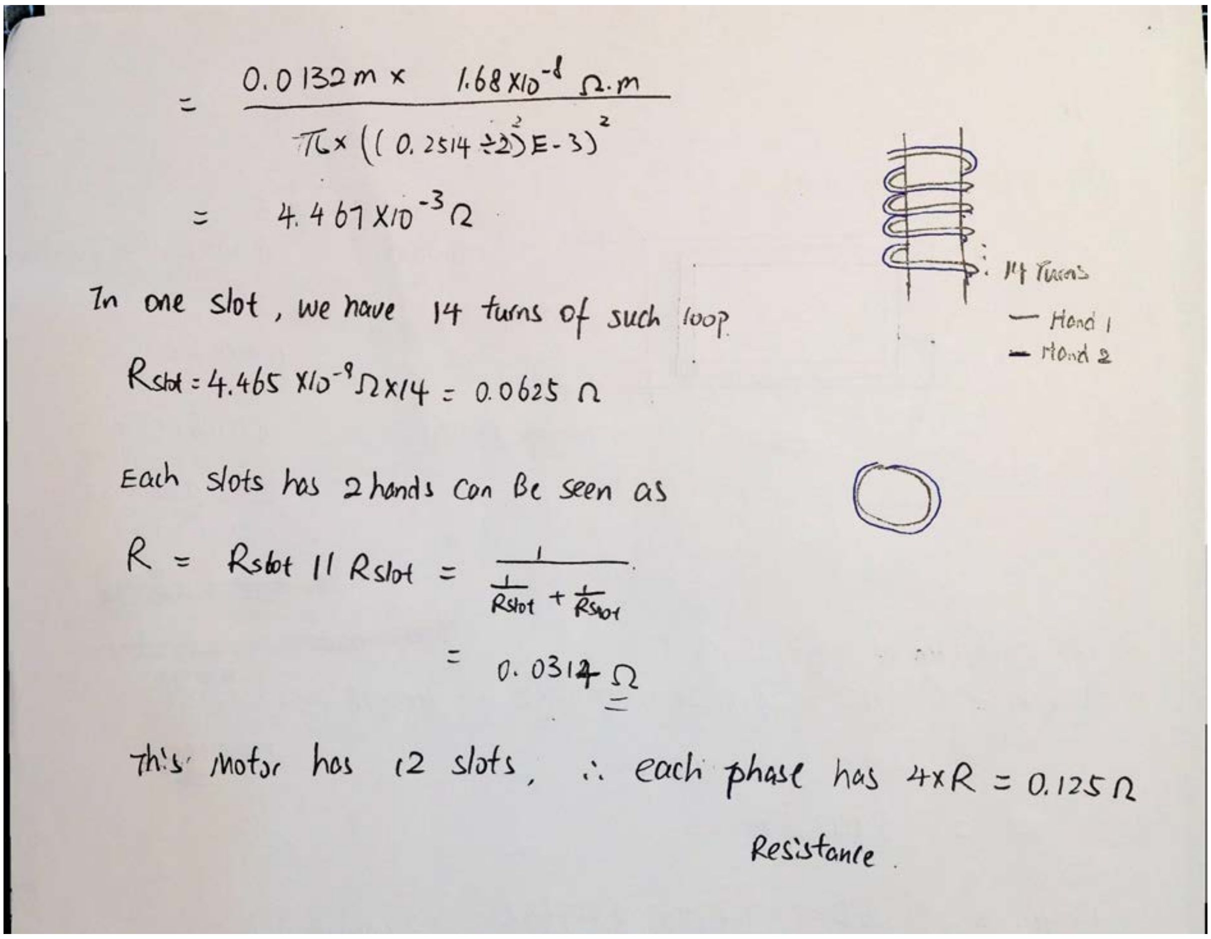


b) phase resistance

For 1 turn on one tooth

$$R = \frac{l_{avg}}{\sigma A_c} = \frac{l_{avg}\,\rho}{A_c}$$





$$= \frac{0.0132\,m \times 1.68\times10^{-8}\,\Omega.m}{\pi \times ((0.2514 \div 2)E{-}3)^2}$$

$$= 4.467\times10^{-3}\,\Omega$$

In one slot, we have 14 turns of such loop

$$R_{slot} = 4.465\times10^{-9}\,\Omega\times14 = 0.0625\,\Omega$$

Each slots has 2 hands can be seen as

$$R = R_{slot} \parallel R_{slot} = \frac{1}{\frac{1}{R_{slot}} + \frac{1}{R_{slot}}}$$

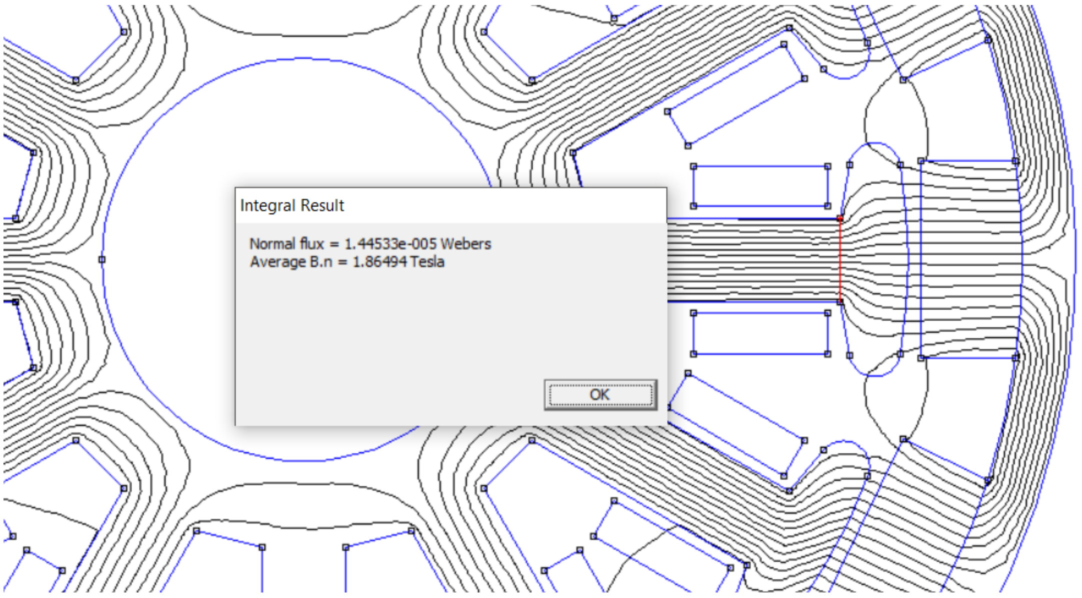
$$= \underline{\underline{0.0314\,\Omega}}$$

This motor has 12 slots, ∴ each phase has $4\times R = 0.125\,\Omega$ Resistance.

14 Turns
— Hand 1
— Hand 2

2. Find core loss

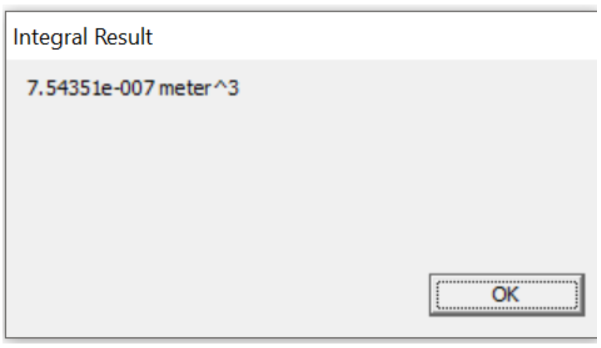
To find core loss at 50Hz, Table at Page 4 of document 1 in Appendix is used, because the thickness aim for this steel is 0.25mm, which is the closest to 1806 motors lamination thickness, 6.2mm/30 level = 0.21mm.



```
Integral Result

Normal flux = 1.44533e-005 Webers
Average B.n = 1.86494 Tesla
```

Recall from part 1, B = $\frac{1.86}{\sqrt{2}} = 1.272$, therefore from the table aforementioned, $\frac{P_{core}}{m} = 1.38\ W/Kg$.

Its know that core loss consist of hysteresis loss and eddy current loss, which is cause by the reversal of magnetisation, and the inducing current only in the part of the motor which has coil wrap around, which is the stator in this case. Therefore, the mass of the stator has been found by


Integral Result

```
7.54351e-007 meter^3
```

$$m = \rho v = 7.65\frac{\mathrm{g}}{\mathrm{cm}^3} * 7.54E - 01cm^3 = 5.7681g$$

So, the mass for the 1806 motor rotor is $5.7681g$ = 5.7681E-3 Kg, therefore

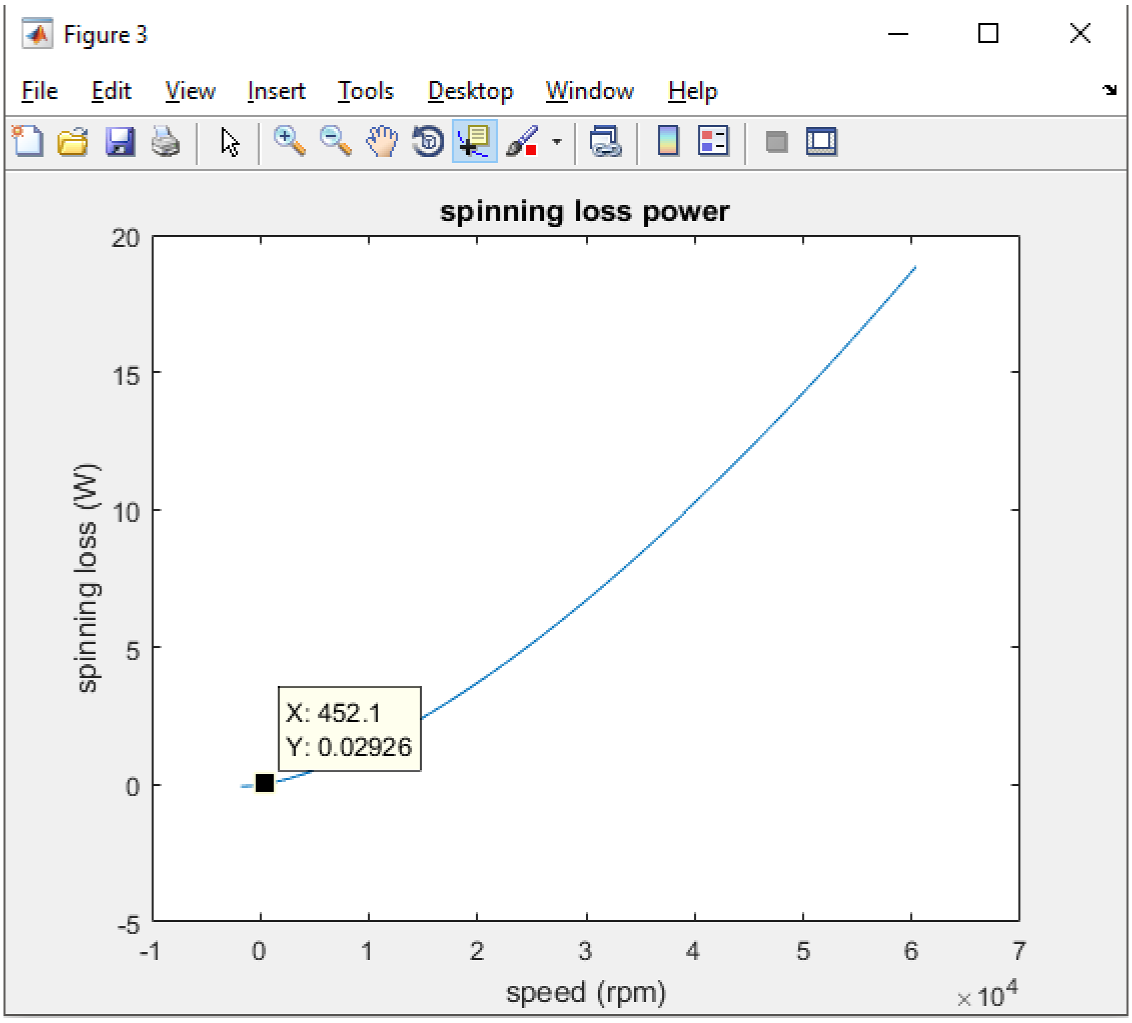
$\mathrm{P_{core}} = 1.38 * 5.7681\mathrm{E} - 3 * 2 \quad = 0.0159W$ where 2 is the fabrication factor due to the edge damage in lamination steel sheet during production process.

3. No load/ spinning loss

It can be seen from the appendix of the assignment sheet that the moment of inertia on z axis for 1806 motor is 796.66972e-9 Kg*m^2.

Modifying Example_L5c.m, the plot of Spinning power loss power Vs Speed is shown in the figure below.

Part2_Q3_Example_L5c.m was used in this question.



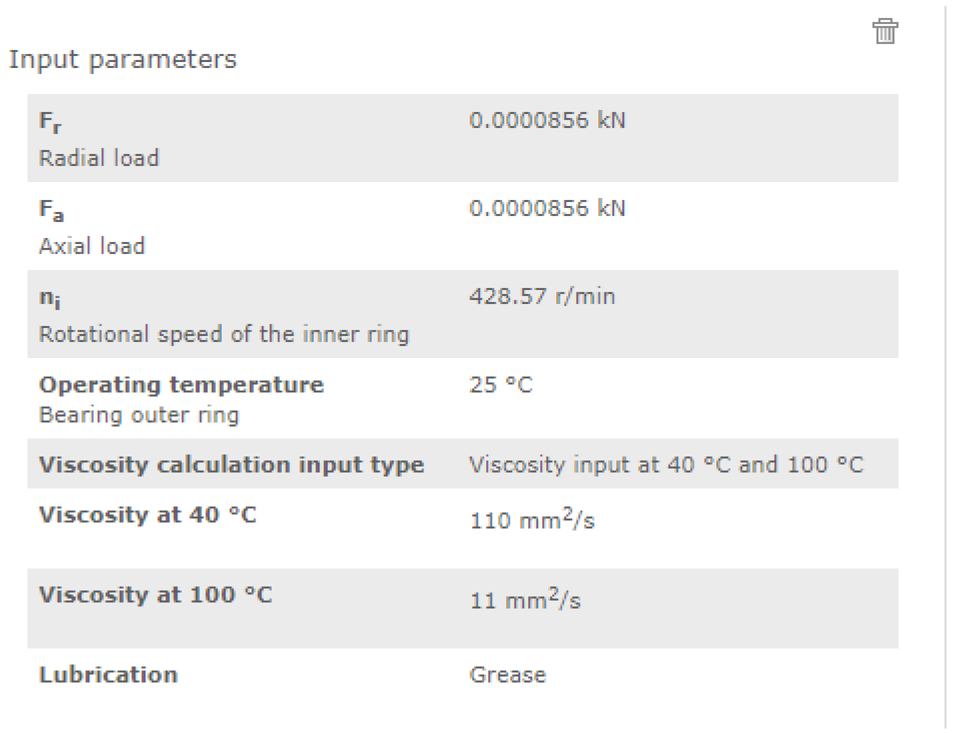
A reasonable running situation for this specific motor is under 11V will leads to 2400KV*11V = 26400rpm.

The RPM of the motor can be calculated by $RPM = \frac{(50*2\pi)}{P/2} * \frac{60}{2\pi} = 428.57$

Therefore, the spinning loss given by the curve above roughly 0.02926W.

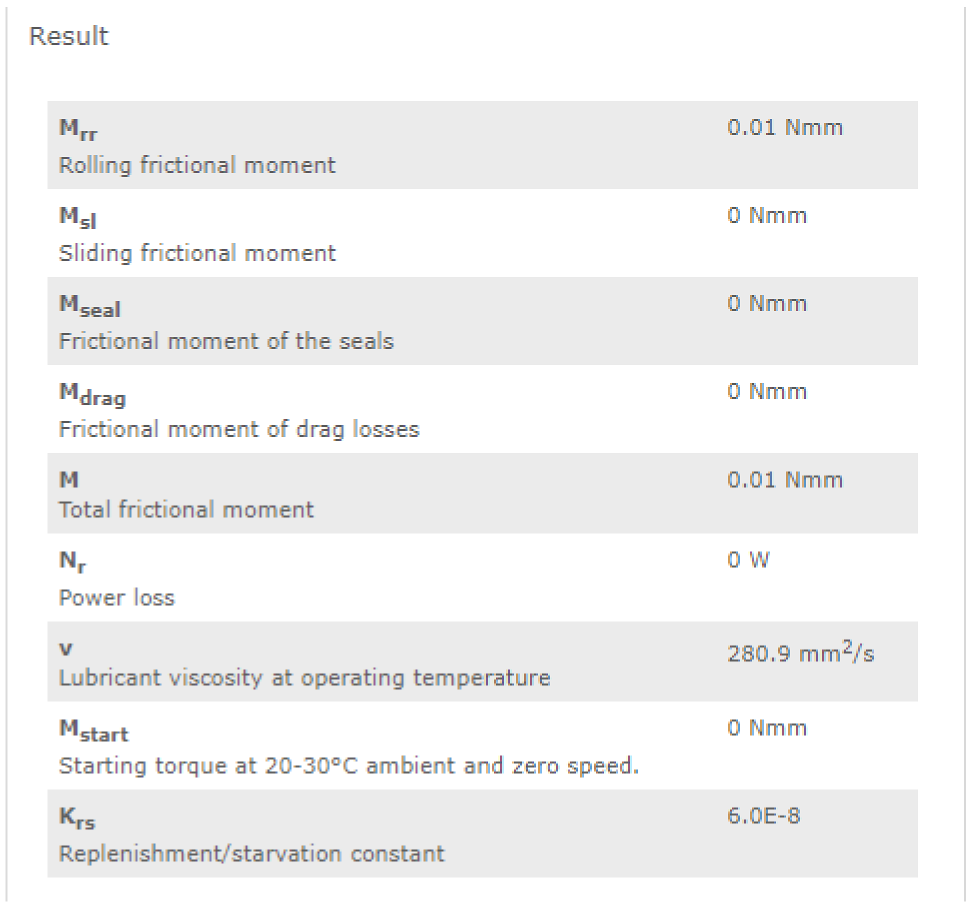
4.

The rotor mass is given by 8.72625g = 8.72625E-3Kg = 0.0856E-3 N = 0.000085 KN



Input parameters

F_r Radial load	0.0000856 kN
F_a Axial load	0.0000856 kN
n_i Rotational speed of the inner ring	428.57 r/min
Operating temperature Bearing outer ring	25 °C
Viscosity calculation input type	Viscosity input at 40 °C and 100 °C
Viscosity at 40 °C	110 mm^2/s
Viscosity at 100 °C	11 mm^2/s
Lubrication	Grease



Result

M_{rr} Rolling frictional moment	0.01 Nmm
M_{sl} Sliding frictional moment	0 Nmm
M_{seal} Frictional moment of the seals	0 Nmm
M_{drag} Frictional moment of drag losses	0 Nmm
M Total frictional moment	0.01 Nmm
N_r Power loss	0 W
v Lubricant viscosity at operating temperature	280.9 mm^2/s
M_{start} Starting torque at 20-30°C ambient and zero speed.	0 Nmm
K_{rs} Replenishment/starvation constant	6.0E-8

It is given by the skf website that Ploss = 1,05 x 10–4 *M*n, where M is the total friction moment which is 0.01Nmm, n is the rotational speed [r/min].

Therefore $P_{bearing-loss} = 1.05E - 4 * 0.01E - 3 * 428.57 = 4.49E - 7W.$

5.

Analytical estimation of spinning loss

To the extent of this assignment, the spinning loss is consisting of the core loss and the bearing loss.

And it can be seen from the previous part of this assignment that core loss at 50Hz is $0.0159W$

And the bearing loss is $4.49E - 7W$. Therefore P_spinning_analytical = 0.0159W.

The result of spinning loss can be sum up in the following table

Method	Spinning loss(W)
Analytical	0.0159
FEMM output to Matlab	
Experimental	0.02926

The different among three might cause by Solid loss due to eddy current in non-laminated materials which hasn't been account for in FEMM and analytical analysis, whereas that exist in the real world.

6.

Efficiency = $\frac{\mathrm{P_{out}}}{P_{in}}$, the detailed calculation has shown below.

6.

$$P_{out} = P_{mech\_out} = T_m W_m$$

where $T_m = 0.0491\ Nm$ is an experimental result from Part 1.

$$W_m = 428.57\ rpm \times 2\pi \div 60$$

$$= 44.879745\ rad/s$$

$$\therefore P_{out} = 0.0491\ Nm \times 44.879745\ rad/s = 2.20W$$

$$P_{in} = I^2 R = \left(\frac{13.9}{\sqrt{2}}\right)^2 \times (12 \times R)$$

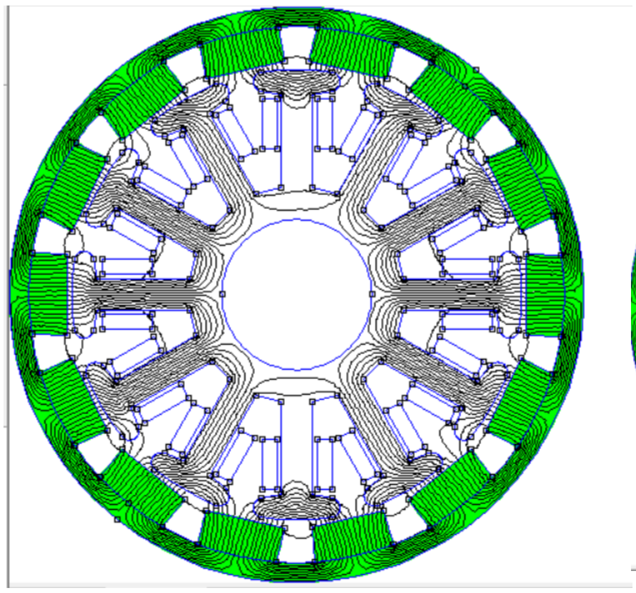
$$= 36.4\ W$$

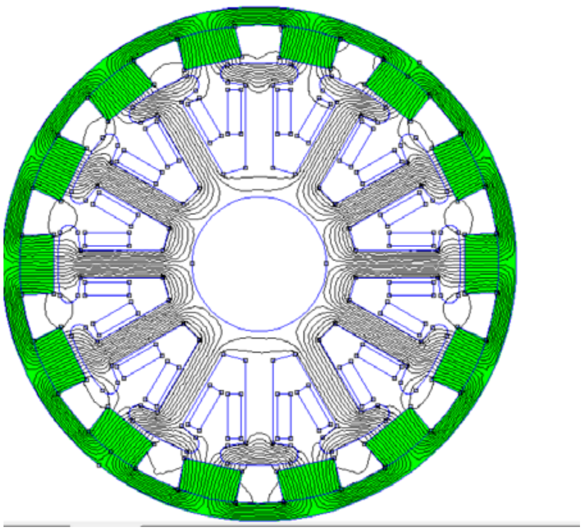
$$\frac{P_{out}}{P_{in}} \times 100\% = \frac{2.20}{36.4} \times 100\% = 6.04\%\ \text{Efficiency.}$$

7. Cogging torque

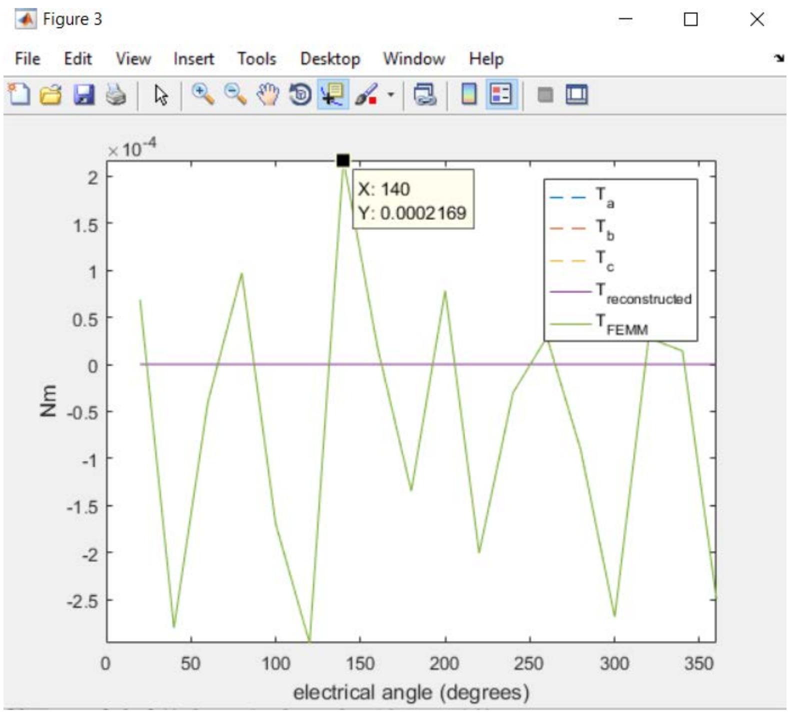
The cogging torque can be found by chaging the rms current value to 0 and find the maximum T_FEMM point from Matlab script Q3_part1.m.

The fllowing diagrams shows the change of geometry in the motor, 10% wider and 10% narrower.



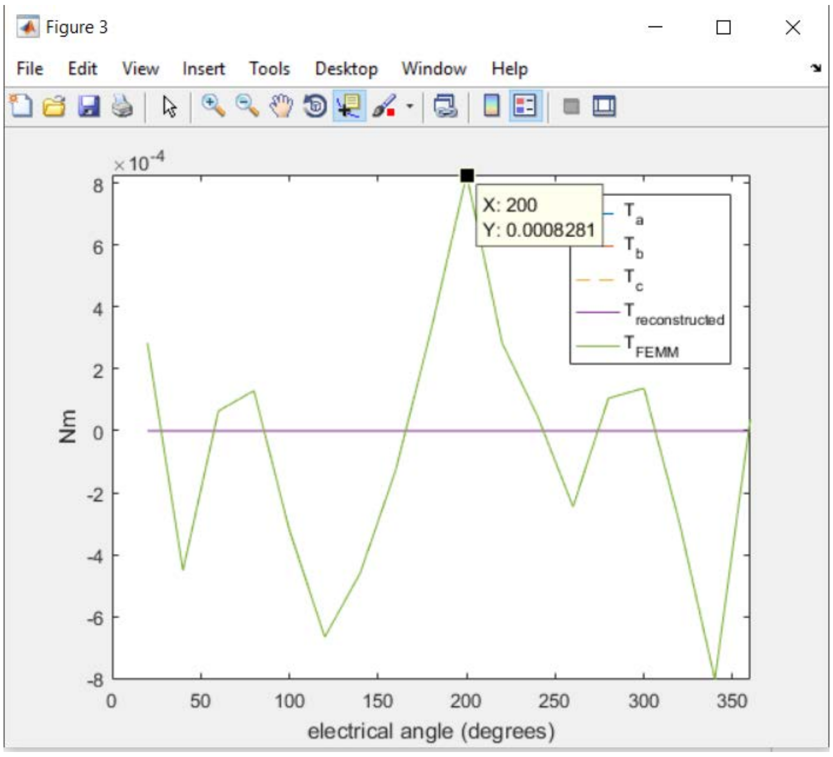


a. Standard magnet width



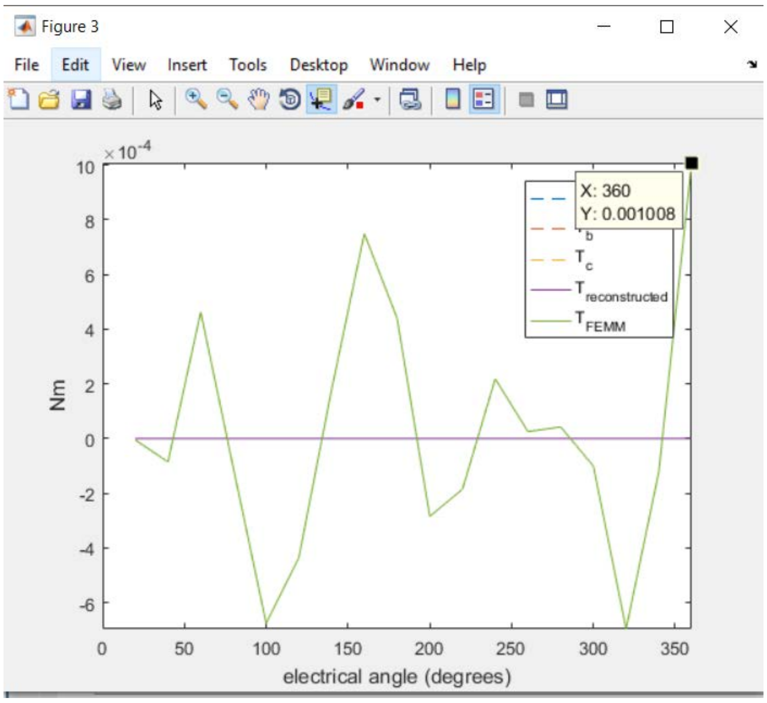
The cogging torque in this case has been found to be 0.0002Nm.

b. A magnet width 10% wider



The cogging torque in this case has been found to be 0.0008Nm.

c. Magnet 10% narrower



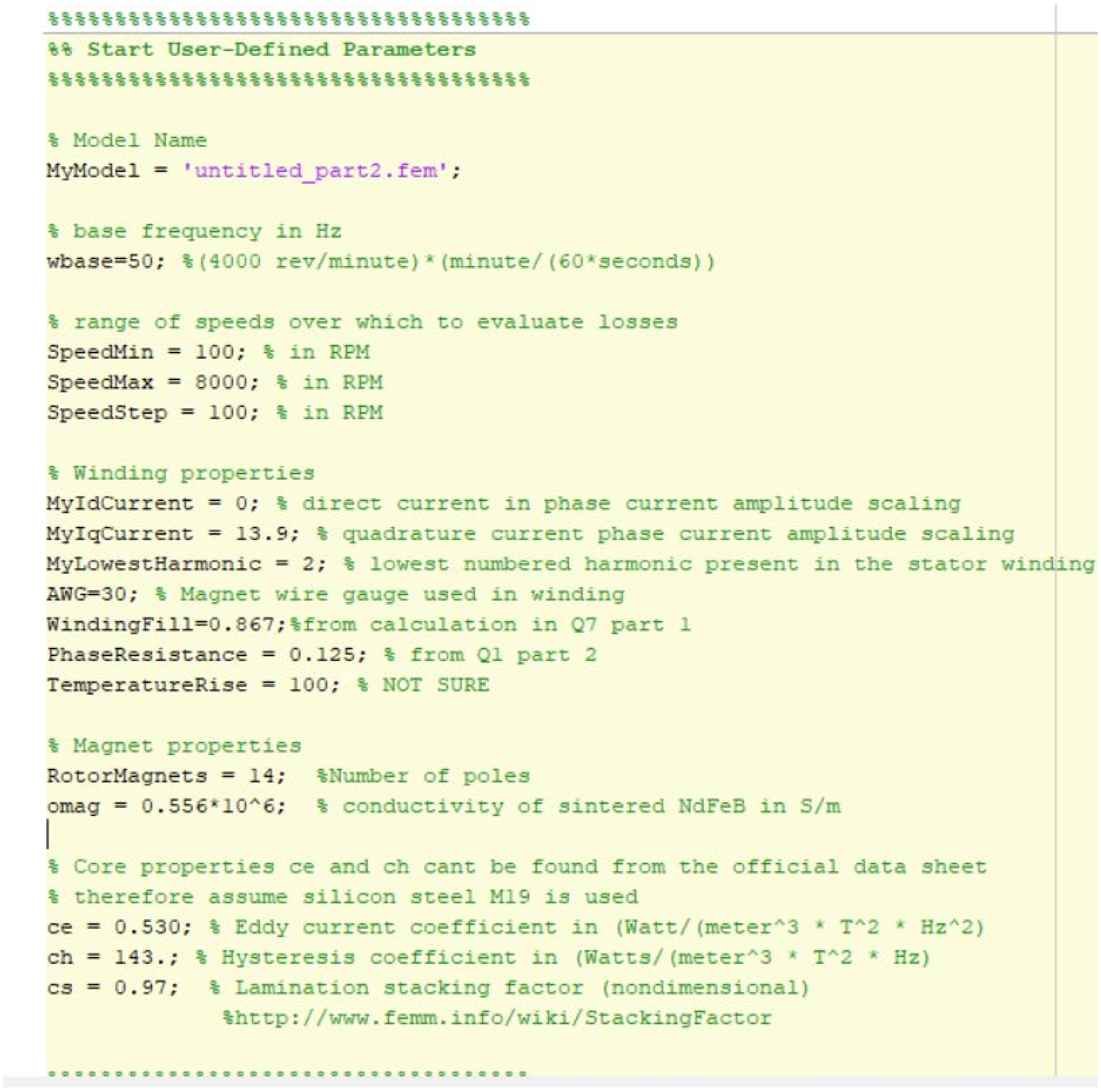
The cogging torque in this case has been found to be 0.001Nm.

Comparing among three, narrowing the magnet by 10% gives the highest cogging torque.

Bonus work attempt

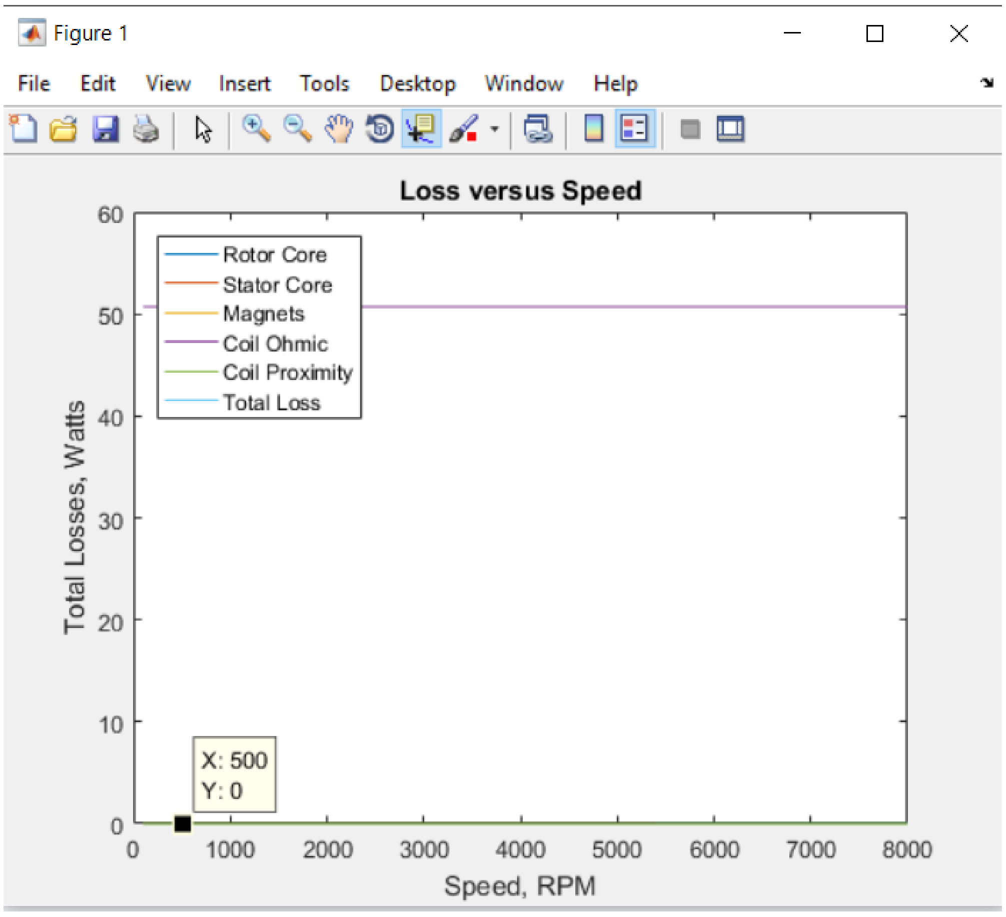
The Matlab script that used to control and collect data from FEMM has been downloaded and modified.

The `Start User-Defined Parameters` are set as following with reasoning in the comment. The full code is attached in the appendix section.



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Start User-Defined Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Model Name
MyModel = 'untitled_part2.fem';

% base frequency in Hz
wbase=50; %(4000 rev/minute)*(minute/(60*seconds))

% range of speeds over which to evaluate losses
SpeedMin = 100; % in RPM
SpeedMax = 8000; % in RPM
SpeedStep = 100; % in RPM

% Winding properties
MyIdCurrent = 0; % direct current in phase current amplitude scaling
MyIqCurrent = 13.9; % quadrature current phase current amplitude scaling
MyLowestHarmonic = 2; % lowest numbered harmonic present in the stator winding
AWG=30; % Magnet wire gauge used in winding
WindingFill=0.867;%from calculation in Q7 part 1
PhaseResistance = 0.125; % from Q1 part 2
TemperatureRise = 100; % NOT SURE

% Magnet properties
RotorMagnets = 14;  %Number of poles
omag = 0.556*10^6;  % conductivity of sintered NdFeB in S/m

% Core properties ce and ch cant be found from the official data sheet
% therefore assume silicon steel M19 is used
ce = 0.530; % Eddy current coefficient in (Watt/(meter^3 * T^2 * Hz^2)
ch = 143.; % Hysteresis coefficient in (Watts/(meter^3 * T^2 * Hz)
cs = 0.97;  % Lamination stacking factor (nondimensional)
             %http://www.femm.info/wiki/StackingFactor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The plot for different losses VS Speed is attached below.



The result is not really making sense at this stage, so further improvement is needed.

Appendix

Q3_part1.m

```
close all; clear all; clc

%% input data
NumberPoles = 14;

%% extract FEMM data
Data = csvread('OutrunnerVideoExample_Part3.csv',1,0);
Angle = Data(:,1);
Torque = Data(:,2);
FluxLinkage = Data(:,3:5);
 Current = Data(:,6:8);

%% calculations
K = circshift(FluxLinkage, [-
round(length(Angle)*(90)/360),0])*NumberPoles/2;  % differentiate in
frequency domain
T = K.*Current;

%% plotting
figure
plot(Angle, FluxLinkage)
hold on
plot(Angle,  K,'--')
hold off
legend('\lambda_a', '\lambda_b', '\lambda_c', 'k_a', 'k_b', 'k_c')
xlabel('electrical angle (degrees)')
ylabel('Wb & Vs')
axis([0 360 -inf inf])

figure
plot(Angle,  K,'--')
hold on
plot(Angle, Current)
hold off
legend('k_a', 'k_b', 'k_c', 'i_a', 'i_b', 'i_c')
xlabel('electrical angle (degrees)')
ylabel('Vs & A')
axis([0 360 -inf inf])



figure
plot(Angle,  T,'--')
hold on
plot(Angle, sum(T,2),Angle , Torque)
hold off
legend('T_a', 'T_b', 'T_c', 'T_{reconstructed}', 'T_{FEMM}')
xlabel('electrical angle (degrees)')
ylabel('Nm')
axis([0 360 -inf inf])

FluxLinkageRMS = max(max(FluxLinkage))/sqrt(2);
AverageTorque = mean(Torque);


Ke = FluxLinkageRMS*NumberPoles/2
```

```
Kt = Ke*3
```

Q6_part1.m

```
close all; clear all; clc

%% motor parameters
VoltagePeakMin = 0.5;
VoltagePeakMax = 4;
N_RunningAverage =100;

%% Outrunner
N_poles = 14;
Data = csvread('RunDown_Outrunner_v1.csv',1,0);

%% extract run down data
Data = Data(10:end, :);
time = Data(:,1);
voltage = Data(:,2);

%% filter data
B = 1/N_RunningAverage*ones(N_RunningAverage,1);
voltage_filtered = filter(B,1,voltage);

%% determine times for zero crossing and peaks
index_zc = find(voltage_filtered.*circshift(voltage_filtered,[1 1])<=0);
time_zc = time(index_zc);
index_peak = index_zc(1:end-1) + round(gradient(index_zc(1:end-1))/2);
time_peak = time(index_peak);
voltage_peak = voltage_filtered(index_peak);

%% trim points from before run down (start at 10th point after drops below
threshold)
index_start = find(abs(voltage_peak)<VoltagePeakMax);
time_zc = time_zc(index_start(10):end);
time_peak = time_peak(index_start(10):end);
voltage_peak = voltage_peak(index_start(10):end);

%% trim points at end of curve to avoid spurious zero crossings
time_zc = time_zc(abs(voltage_peak)>VoltagePeakMin);
time_peak = time_peak(abs(voltage_peak)>VoltagePeakMin);
voltage_peak = voltage_peak(abs(voltage_peak)>VoltagePeakMin);

%% determine velocity
period = gradient(time_zc)*2; % two zerocrossings per full wave period
Hz_electrical = 1./period;
Hz_mechanical = Hz_electrical/(N_poles/2);
AngularVelocity = Hz_mechanical*2*pi;
VoltageRMS = abs(voltage_peak)/sqrt(2)/(sqrt(3)); % also convert from line-
line to phase voltage (for star)
% VoltageRMS = abs(voltage_peak)/sqrt(2); % also convert from line-line to
phase voltage (same for delta)

%% fit curve to determine torque constant
p_Ke = polyfit(AngularVelocity,VoltageRMS,1)
VoltageRMS_linear = polyval(p_Ke, AngularVelocity);
Kt = p_Ke(1)*3

%% plotting
```

```
figure
plot(time,voltage,time,voltage_filtered)
hold on
plot(time_zc,0,'x', time_peak, voltage_peak, 'x', 'markersize',10,
'linewidth', 2)
hold off
title('Run Down Voltage')
ylabel('Line to Line Voltage (V)')
xlabel('Time (sec)')

figure
plot(AngularVelocity(2:end-1),VoltageRMS(2:end-
1),'.',AngularVelocity(2:end-1), VoltageRMS_linear(2:end-1), 'linewidth',
2)
title('Voltage constant (Ke)')
ylabel('Phase Voltage (vrms)')
xlabel('Angular Velocity (rad/sec)')
axis([0 inf 0 inf])
```

```
--------------------------------------------
-- outrunner motor example
-- greg@heins.com.au, 20187
-- based on script from edie_currants@yahoo.com
-- Note: Requires FEMM 4.2
--------------------------------------------

-- set up model
steps_per_electrical_cycle = 18; --should be multiple of 3
number_of_electrical_cycles = 1;

number_of_poles = 14;
phase_current_rms = 0.1;

steps = steps_per_electrical_cycle*number_of_electrical_cycles;
current_mag = sqrt(2)*phase_current_rms;
current_offset_angle = -120; --aline current with back emf

-- load geometry
mydir="./"
open(mydir .. "Q10.FEM")

-- and save as a temporary file name so we don't
-- ovewrite the original geometry
mi_saveas(mydir .. "temp_Q10.fem")

-- show the Lua console window so that we can see the
-- program report its progress
showconsole()

-- move the rotor through one pole pitch in small
-- increments, recording the flux linkage of each
-- phase at each rotor position
d = {};-- to store things


start_clock = clock();
finish_clock = 0;
print("Start" .. start_clock);

for k = 1,(steps+1) do
    angle_elec_degrees_increment = 360/steps_per_electrical_cycle;
    angle_mech_degrees_increment =
angle_elec_degrees_increment/(number_of_poles/2);
    angle_elec_degrees = (k-1)*angle_elec_degrees_increment;
    angle_mech_degrees = angle_elec_degrees/(number_of_poles/2);


    mi_modifycircprop("A",1,current_mag*cos(rad(angle_elec_degrees+current_off
set_angle-120)));
    mi_modifycircprop("B",1,current_mag*cos(rad(angle_elec_degrees+current_off
set_angle)));
    mi_modifycircprop("C",1,current_mag*cos(rad(angle_elec_degrees+current_off
set_angle+120)));

    --print((k-1) .. "/" .. steps);


    mi_analyze(1);
    mi_loadsolution();
    d[k]={};
    d[k][1] = angle_elec_degrees;

    -- collect the flux linkage and current for each phase
    d[k][6],v2,d[k][3] = mo_getcircuitproperties("A"); -- current, voltage,
flux linkage
    d[k][7],v2,d[k][4] = mo_getcircuitproperties("B");
    d[k][8],v2,d[k][5] = mo_getcircuitproperties("C");

    -- compute the torque
    mo_groupselectblock(2);
    d[k][2] = mo_blockintegral(22);

    -- increment the rotor's position
    mi_selectgroup(2);
    mi_moverotate(0, 0, angle_mech_degrees_increment , 4);
    mi_clearselected();
end

finish_clock = clock();
calc_time_per_cyc = (finish_clock - start_clock)/18.0;
```

```
print("Finish" .. finish_clock);
print("AVG Time" .. calc_time_per_cyc);


-- write results to disk for further analysis.
fp=openfile(mydir .. "mesh_size_calc.csv","w")
for k = 1,(steps+1) do

write(fp,d[k][1],",",d[k][2],",",d[k][3],",",d[k][4],",",d[k][5],",",d[k][6],"
,",d[k][7],",",d[k][8],"\n")
end
closefile(fp);
```

Part2_Q3_Example_L5c.m

```
close all; clear all; clc

%% motor parameters
VoltagePeakMin = 0.5;
VoltagePeakMax = 4;
N_RunningAverage =100;

%% Outrunner
J = 796.66972e-9; %from CAD: ABS - assume plastic is 42.9g
N_poles = 14;
Data = csvread('RunDown_Outrunner_v1.csv',1,0);

%% extract run down data
Data = Data(10:end, :);
time = Data(:,1)*10;
voltage = Data(:,2);

%% filter data
B = 1/N_RunningAverage*ones(N_RunningAverage,1);
voltage_filtered = filter(B,1,voltage);

%% determine times for zero crossing and peaks
index_zc = find(voltage_filtered.*circshift(voltage_filtered,[1 1])<=0);
time_zc = time(index_zc);
index_peak = index_zc(1:end-1) + round(gradient(index_zc(1:end-1))/2);
time_peak = time(index_peak);
voltage_peak = voltage_filtered(index_peak);

%% trim points from before run down (start at 10th point after drops below
threshold)
index_start = find(abs(voltage_peak)<VoltagePeakMax);
time_zc = time_zc(index_start(10):end);
time_peak = time_peak(index_start(10):end);
voltage_peak = voltage_peak(index_start(10):end);

%% trim points at end of curve to avoid spurious zero crossings
time_zc = time_zc(abs(voltage_peak)>VoltagePeakMin);
time_peak = time_peak(abs(voltage_peak)>VoltagePeakMin);
```

```
voltage_peak = voltage_peak(abs(voltage_peak)>VoltagePeakMin);

%% determine velocity
period = gradient(time_zc)*2; % two zerocrossings per full wave period
Hz_electrical = 1./period;
Hz_mechanical = Hz_electrical/(N_poles/2);
AngularVelocity = Hz_mechanical*2*pi;
VoltageRMS = abs(voltage_peak)/sqrt(2)/(sqrt(3)); % also convert from line-
line to phase voltage
% VoltageRMS = abs(voltage_peak)/sqrt(2); % also convert from line-line to
phase voltage (same for delta)

%% fit curve to determine torque constant
p_Ke = polyfit(AngularVelocity,VoltageRMS,1)
VoltageRMS_linear = polyval(p_Ke, AngularVelocity);
Kt = p_Ke(1)*3

%% polynominal fit for velocity and acceleration
TimePoly = linspace(min(time_zc)-2, max(time_zc));  % take 2 seconds off
the start time to simulate if the motor started faster
p_vel = polyfit(time_zc, AngularVelocity, 2);  % fit polynomial to
AngularVelocity Data
p_acc = polyder(p_vel); % determine the acceleration as the gradient of the
AngularVelocity Data

%% evaluate polynomial
VelocityPoly = polyval(p_vel, TimePoly);  % evaluate the polynomal for
AngularVelocity
AccelerationPoly = polyval(p_acc, TimePoly);  % evaluate the polynomal for
Acceleration

%% T = J*omega
T_spinning_loss = -J*AccelerationPoly;  % as per eq 53 in reference
P_spinning_loss = T_spinning_loss.*VelocityPoly;
Rpm = VelocityPoly * 60/(2*pi);  % define the rpm for plotting

Polynomial_SpinningLoss_Rpm = polyfit(Rpm, P_spinning_loss, 2);
Rpm_Plot = 0:600;
P_SpinningLoss_Plot = polyval(Polynomial_SpinningLoss_Rpm, Rpm_Plot);

%% plotting
figure
plot(time,voltage,time,voltage_filtered)
hold on
plot(time_zc,0,'x', time_peak, voltage_peak, 'x', 'markersize',10,
'linewidth', 2)
hold off
title('Run Down Voltage')
ylabel('Line to Line Voltage (V)')
xlabel('Time (sec)')
%
figure
plot(AngularVelocity(2:end-1),VoltageRMS(2:end-
1),'.',AngularVelocity(2:end-1), VoltageRMS_linear(2:end-1), 'linewidth',
2)
title('Voltage constant (Ke)')
ylabel('Phase Voltage (vrms)')
xlabel('Angular Velocity (rad/sec)')
axis([0 inf 0 inf])
```

```
figure
plot(Rpm, P_spinning_loss, Rpm_Plot, P_SpinningLoss_Plot)
title('spinning loss power')
xlabel('speed (rpm)')
ylabel('spinning loss (W)')
```

Q8.m

```
%It has been calculated that the conductive area is Ac = 1.39mm2

%J in A/mm2
close all;
clear all;
clc;
J = [5:5:25];

I = J.*1.39;
Kt = 0.01519;%hand calculation and FEMM result
T = Kt.*I;

plot(J, T);
title('Torque vs Current Density');
xlabel('Current density J(A/mm^2)');
ylabel('Torque(Nm)');
```

Q8_part2_mySPMLossScript.m

```
%% Core Loss Calculation Script

% David Meeker
% dmeeker@ieee.org
% 17Oct2017
clear all
clc
close all
addpath('C:\femm42\mfiles');
savepath;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Start User-Defined Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Model Name
MyModel = 'untitled_part2.fem';

% base frequency in Hz
wbase=50; %(4000 rev/minute)*(minute/(60*seconds))

% range of speeds over which to evaluate losses
SpeedMin = 100; % in RPM
SpeedMax = 8000; % in RPM
SpeedStep = 100; % in RPM

% Winding properties
MyIdCurrent = 0; % direct current in phase current amplitude scaling
MyIqCurrent = 13.9; % quadrature current phase current amplitude scaling
MyLowestHarmonic = 2; % lowest numbered harmonic present in the stator
winding
```

```
AWG=30; % Magnet wire gauge used in winding
WindingFill=0.867;%from calculation in Q7 part 1
PhaseResistance = 0.125; % from Q1 part 2
TemperatureRise = 100; % NOT SURE

% Magnet properties
RotorMagnets = 14;  %Number of poles
omag = 0.556*10^6;  % conductivity of sintered NdFeB in S/m

% Core properties ce and ch cant be found from the official data sheet
% therefore assume silicon steel M19 is used
ce = 0.530; % Eddy current coefficient in (Watt/(meter^3 * T^2 * Hz^2)
ch = 143.; % Hysteresis coefficient in (Watts/(meter^3 * T^2 * Hz)
cs = 0.97;  % Lamination stacking factor (nondimensional)
             %http://www.femm.info/wiki/StackingFactor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% End User-Defined Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% helpful unit definitions
PI=pi; Pi=pi;
deg=Pi/180.;

% angle through which to spin the rotor in degrees
n = 360/MyLowestHarmonic;

% angle increment in degrees
dk = 1;

% A similar loss/volume expression can be derived to compute proximity
% effect losses in the windings.  Use the low frequency approximiation
% from the paper, since in this case, wire size is a lot smaller than skin
% depth at the frequencies of interest.

% Get parameters for proximity effect loss computation for phase windings
dwire=0.324861*0.0254*exp(-0.115942*AWG); % wire diameter in meters as a
function of AWG
owire = (58.*10^6)/(1+TemperatureRise*0.004); % conductivity of the wire in
S/m at prescribed deltaT
cePhase = (Pi^2/8)*dwire^2*WindingFill*owire;

%% Perform a series of finite element analyses

openfemm;
opendocument(MyModel);
smartmesh(1);
mi_saveas('temp.fem');

% Run an analysis through an entire spin of the rotor and record
% element centroid flux density and vector potential (using the mesh from
the first iteration)
% at every step.  This information will then be used to estimate
% core losses
for k = 0:dk:(n-1)
    starttime=clock;
    kk = k/dk + 1;
    % make sure that the current is set to the appropriate value for this
iteration.
    tta = (RotorMagnets/2)*k*deg;
    Id = [cos(tta), cos(tta-2*pi/3), cos(tta+2*pi/3)];
    Iq =-[sin(tta), sin(tta-2*pi/3), sin(tta+2*pi/3)];
    Itot =  MyIdCurrent*Id + MyIqCurrent*Iq;
    mi_setcurrent('A', Itot(1));
    mi_setcurrent('B', Itot(2));
    mi_setcurrent('C', Itot(3));

    mi_analyze(1);
    mi_loadsolution;
    if (k == 0)
        % Record the initial mesh elements if the first time through the
loop
        nn = mo_numelements;
        b = zeros(floor(n/dk),nn); % matrix that will hold the flux density
info
        A = zeros(floor(n/dk),nn); % matrix that will hold the vector
potential info
        z = zeros(nn,1);
        a = zeros(nn,1);
        g = zeros(nn,1);
        tq = zeros(floor(n/dk),1);
        for m = 1:nn
            elm = mo_getelement(m);
            % z is a vector of complex numbers that represents the location
of
            % the centroid of each element.  The real part is x, the
            % imaginary part is y.  The purpose of representing the
            % location in this way is that it is easy to rotate the
            % location of the centroid (by multiplying by a complex number)
            % to find the point that corresponds to the centroid when the
            % rotor is rotated.
            z(m) = elm(4) + 1j*elm(5);
            % element area in the length units used to draw the geometry
            a(m) = elm(6);
            % group number associated with the element
            g(m) = elm(7);
        end
    end

    % Store element flux densities *)
    u=exp(1j*k*pi/180.);
    for m = 1:nn
        % Element is on the rotor.  Elements on the rotor have been
        % assigned to groups 10 and higher (by assigning the block label
that marks them
        % to group 10, 11, 12, etc.) so that the program can tell if an
element is part of the rotor.
        if(g(m)>=10)
            % the location in the original mesh is rotated so that the
            % flux density at the same point with the rotor rotated through
            % angle k can be evaluated. Multiplying by the complex number u
            % does the rotation.
            p = z(m)*u;
            % Flux densities bx and by are evaluated and rolled into a
complex number.
            % Dividing by the complex number u rotates the flux density
            pv = mo_getpointvalues(real(p),imag(p));
            % back into a rotor-fixed reference frame.
            if (g(m)==10)
                % store flux density for elements in rotor core
                b(kk,m) = (pv(2)+1j*pv(3))/u;
            else
                % store vector potential for elements that are in PMs
                A(kk,m) = pv(1);
            end
        elseif (g(m) > 0) % element is on the stator
            % since the stator doesn't move, no games have to be played
            % with rotations.
            p = z(m);
            b(kk,m) = (mo_getb(real(p),imag(p))*[1;1j]);
        end
    end

    % mo_getprobleminfo returns, among other things, the depth of the
    % machine in the into-the-page direction and the length units used to
    % draw the geometry. Both of these pieces of information will be needed
    % to integrate the losses over the volume of the machine.
    probinfo=mo_getprobleminfo;

    % select all blocks on the rotor and compute the torque
    for m=10:(10+RotorMagnets)
        mo_groupselectblock(m);
    end
    tq(kk)=mo_blockintegral(22);
    mo_close;

    % rotate the rotor to the position for the next iteration.
    for m=10:(10+RotorMagnets)
        mi_selectgroup(m);
    end
    mi_moverotate(0, 0, dk);
    mi_clearselected();

    fprintf('% i of % i :: %f seconds ::  %f N*m
\n',k,n,etime(clock,starttime),tq(kk));
end

% clean up after finite element runs are finished
closefemm;
delete('temp.fem');
delete('temp.ans');

%% Add Up Core Losses

% Compute the square of the amplitude of each harmonic at the centroid of
% each element in the mesh. Matlab's built-in FFT function makes this easy.
ns=n/dk;
bxfft=abs(fft(real(b)))*(2/ns);
byfft=abs(fft(imag(b)))*(2/ns);
bsq=(bxfft.*bxfft) + (byfft.*byfft);

% Compute the volume of each element in units of meter^3
h = probinfo(3);            % Length of the machine in the into-the-page
direction
lengthunits = probinfo(4);  % Length of drawing unit in meters
v = a*h*lengthunits^2;
```

```
% compute fft of A at the center of each element
Jm=fft(A)*(2/ns);
for k=1:RotorMagnets
    g3=(g==(10+k));
    % total volume of the magnet under consideration;
    vmag=v'*g3;
    % average current in the magnet for each harmonic
    Jo=(Jm*(v.*g3))/vmag;
    % subtract averages off of each each element in the magnet
    Jm = Jm - Jo*g3';
end

Iphase=sqrt(MyIdCurrent^2+MyIqCurrent^2)/sqrt(2);
PhaseOhmic = 3*(PhaseResistance*(1+TemperatureRise*0.004))*Iphase^2;

results=[];

for thisSpeed=SpeedMin:SpeedStep:SpeedMax

    thisFrequency = thisSpeed/60; % mechanical speed in Hz

    % Make a vector representing the frequency associated with each
harmonic
    % The last half of the entries are zeroed out so that we don't count
each
    % harmonic twice--the upper half of the FFT a mirror of the lower half
    w=0:(ns-1);
    w=MyLowestHarmonic*thisFrequency*w.*(w<(ns/2));

    % Now, total core loss can be computed in one fell swoop...
    % Dividing the result by cs corrects for the lamination stacking factor
    g1=(g==10);
    rotor_loss = ((ch*w+ce*w.*w)*bsq*(v.*g1))/cs;

    g2=(g==1);
    stator_loss = ((ch*w+ce*w.*w)*bsq*(v.*g2))/cs;

    % and prox losses can be totalled up in a similar way
    g4=(g==2);
    prox_loss = ((cePhase*w.*w)*bsq*(v.*g4));

    % Add up eddy current losses in the magnets
    magnet_loss = (1/2)*((omag*(2*pi*w).^2)*(abs(Jm).^2)*v);

    total_loss = rotor_loss + stator_loss + prox_loss + PhaseOhmic +
magnet_loss;

    results = [results; thisSpeed, rotor_loss, stator_loss, magnet_loss,
PhaseOhmic, prox_loss, total_loss];
end

% save('c:\\temp\\myLossData.m','results','-ASCII');

%% Loss plot
plot(results(:,1),results(:,2:7));
xlabel('Speed, RPM');
ylabel('Total Losses, Watts');
title('Loss versus Speed');
```

```
legend('Rotor Core','Stator Core','Magnets','Coil Ohmic','Coil
Proximity','Total Loss','Location','northwest');

% %% Torque plot
% plot(0:dk:(n-1),tq);
% xlabel('Rotor Angle, Degrees');
% ylabel('Torque, N*m');
% title('Torque on Rotor vs. Angle');

% %% Plot of heating
%
% wbase=4000/60; %(4000 rev/minute)*(minute/(60*seconds))
% w=0:(ns-1);
% w=MyLowestHarmonic*wbase*w.*(w<(ns/2));
%
% % Rotor loss
% g1=(g==10);
% ptloss = (transpose ((ch*w+ce*w.*w)*bsq).*g1)/cs;
%
% % Stator loss
% g2=(g==1);
% ptloss = ptloss + (transpose ((ch*w+ce*w.*w)*bsq).*g2)/cs;
%
% % Prox loss
% g4=(g==2);
% ptloss = ptloss + (transpose ((cePhase*w.*w)*bsq).*g4);
%
% % PM contribution
% ptloss = ptloss + ((1/2)*(omag*(2*pi*w).^2)*(abs(Jm).^2))';
%
% %% point location in z
% heating = [real(z),imag(z),ptloss];
% save('c:\\temp\\myLossData','heating','-ASCII');
```