Assignment 2

# Subject: MCEN90044 Electromagnetic Technologies

# Name: Shiyu Zhang

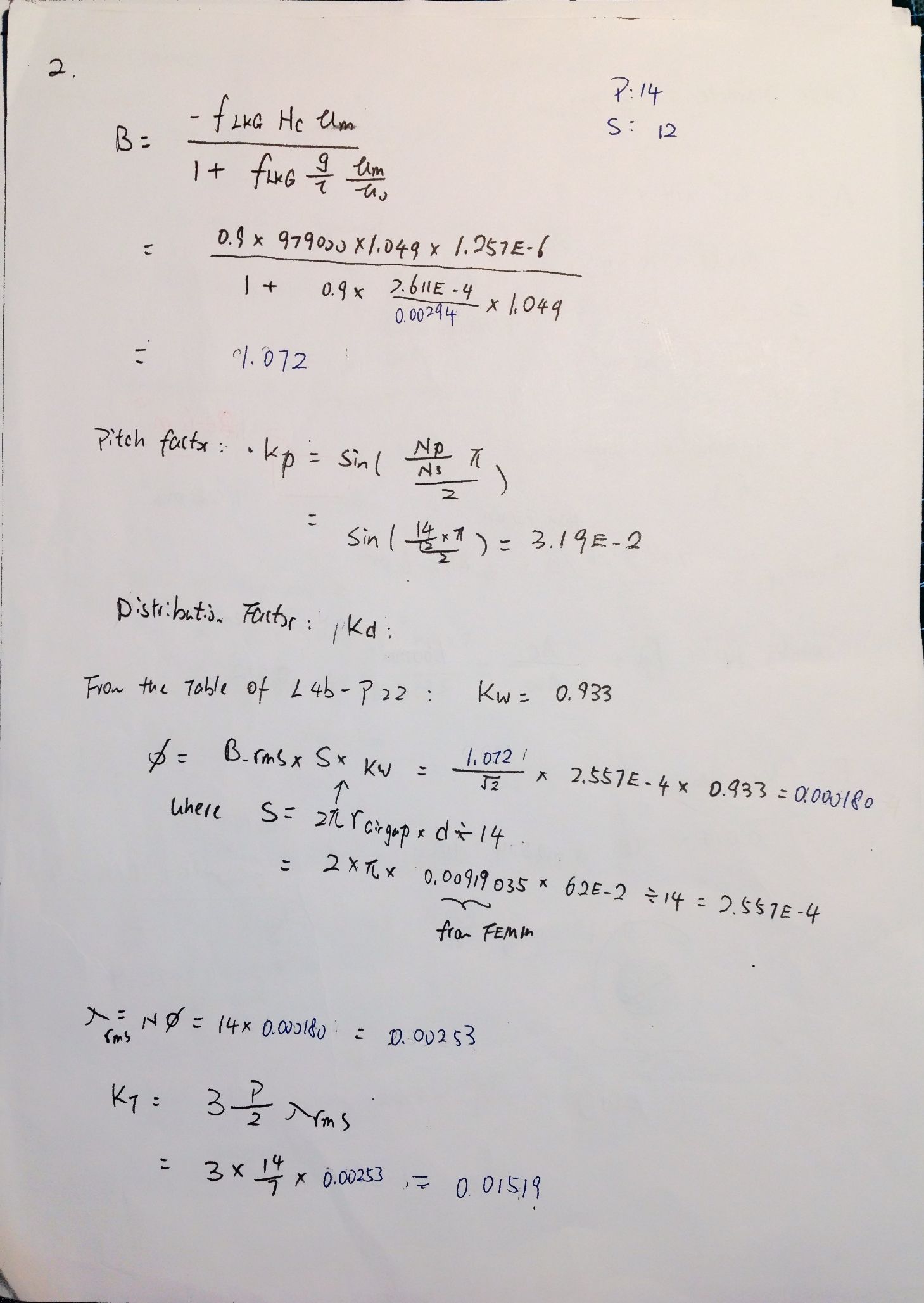
# Student Number: 683557

Part 1

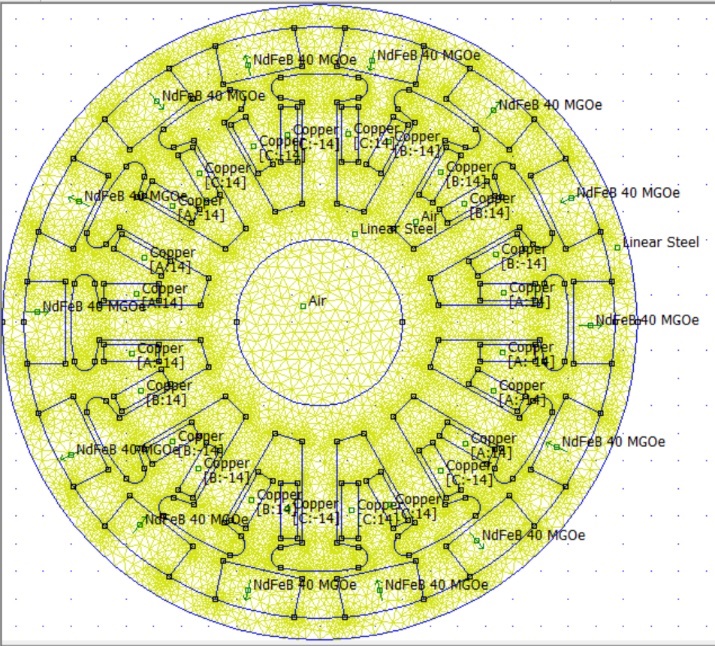
1. Group Photo



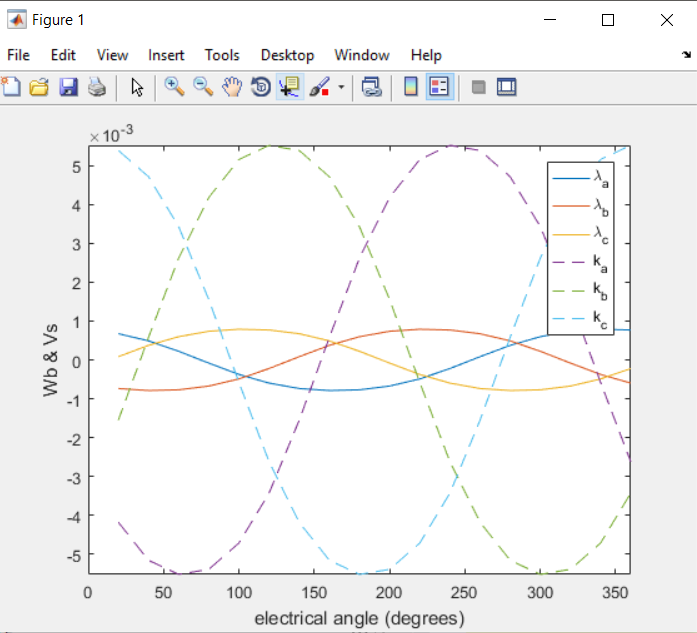
1. The hand calculation of the analytical expression for torque constant Kt is shown in the figure below.

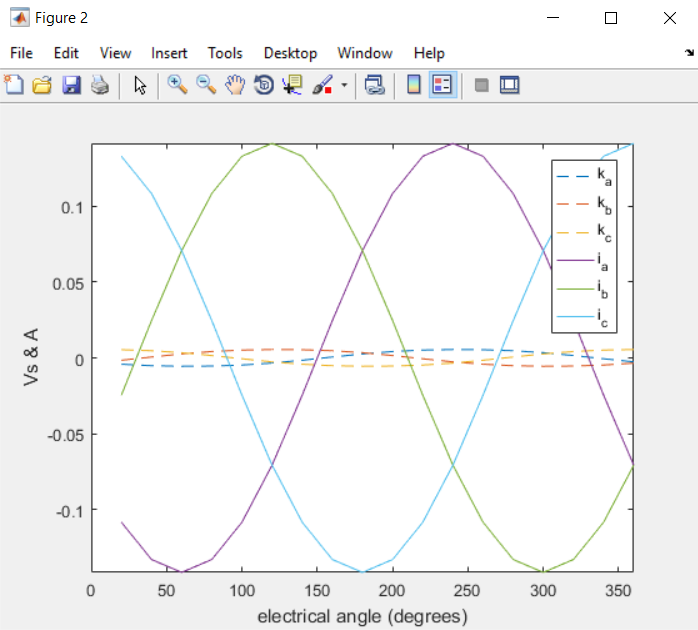


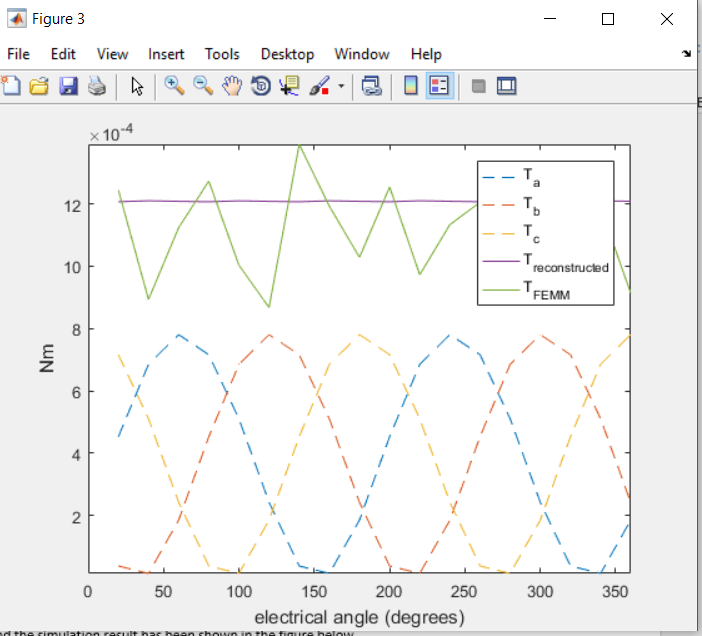
1. A FEMM model has been created as the figure below show.



The script Q3\_part1.m has run to analyse the CSV file generated by FEMM on the following parameters.







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

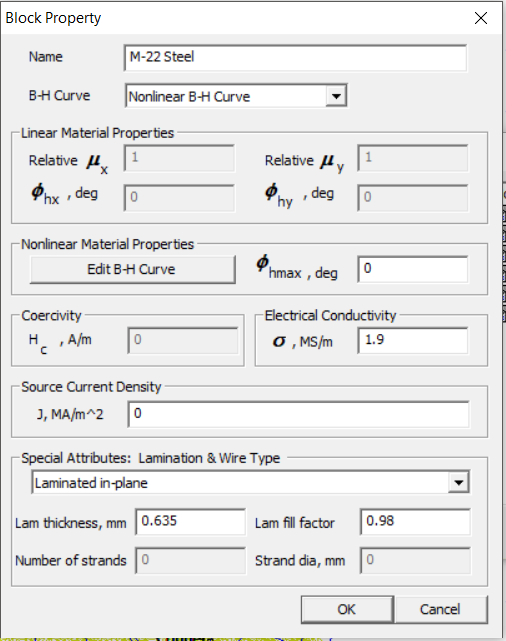
|  |  |
| --- | --- |
| Method | Torque Constant(Nm/A) |
| Hand Calculation | 0.01519 |
| FEMM output to Matlab | 0.01450 |

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.

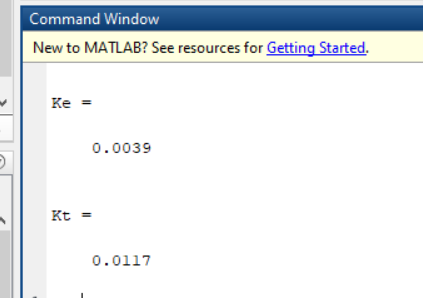


After looking up the 1806 motor catalogue, it can be found that the motor was built using laminated silicon steel to reduce hysteresis loss and core loss. Therefore, the B and H relationship would be nonlinear.

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

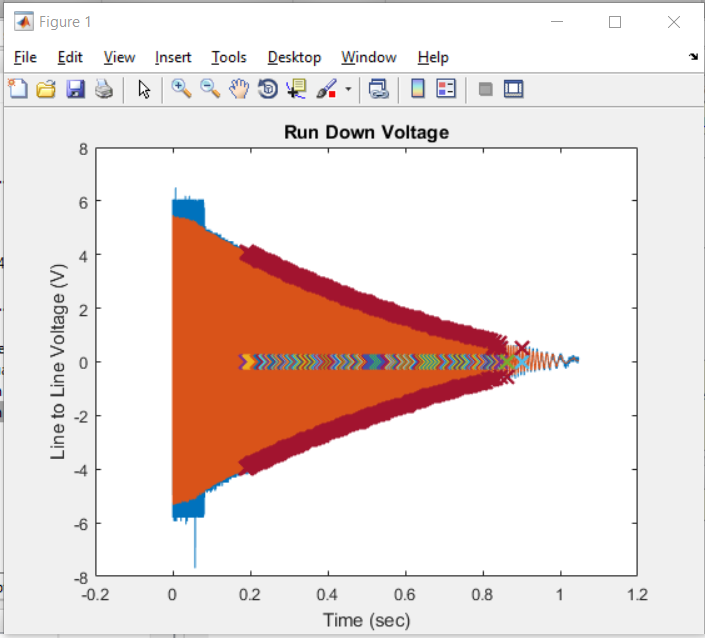


And the simulation result has been shown in the figure below.



Kt is increased from 0.0014 to 0.017.





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.

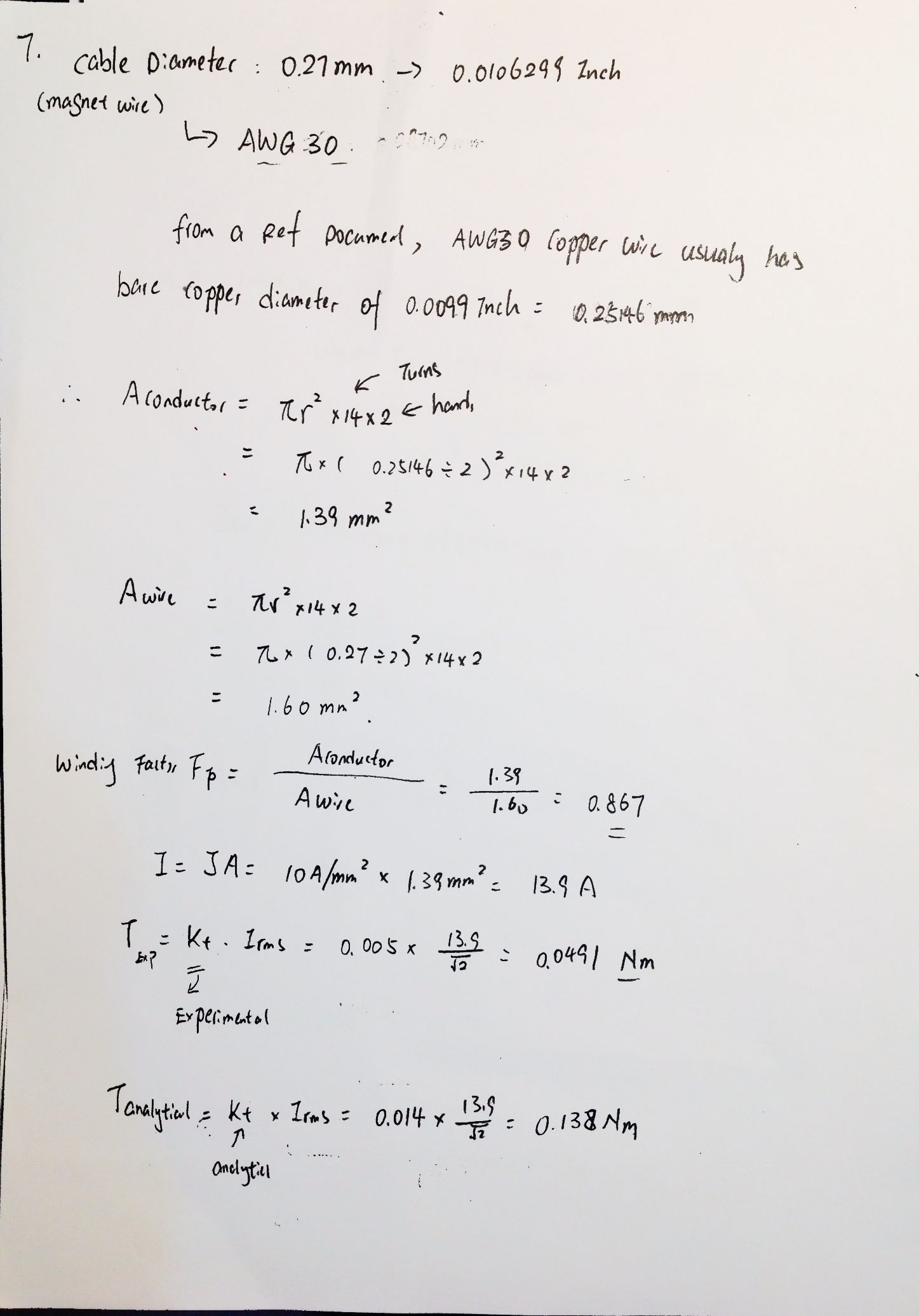
Matlab script Q6\_part1.m has been used for this question.

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

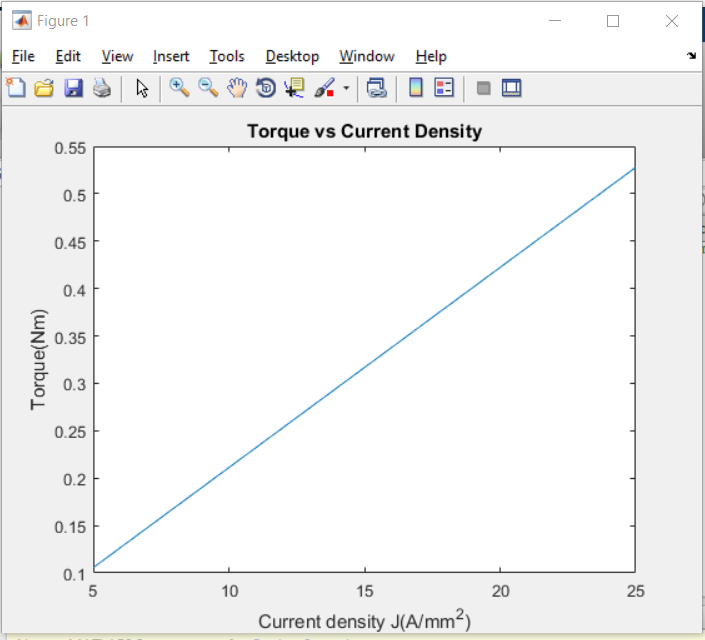
|  |  |
| --- | --- |
| Method | Torque Constant(Nm/A) |
| Hand Calculation | 0.01519 |
| FEMM output to Matlab (Improved to M22 steel) | 0.01450 |
| Experimental | 0.005 |

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.

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



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



1. 

Know that the Motor velocity constant Kv is = 2400 rpm/V = 251.33 rad/s\*V.

Therefore, the back emf constant Ke = .

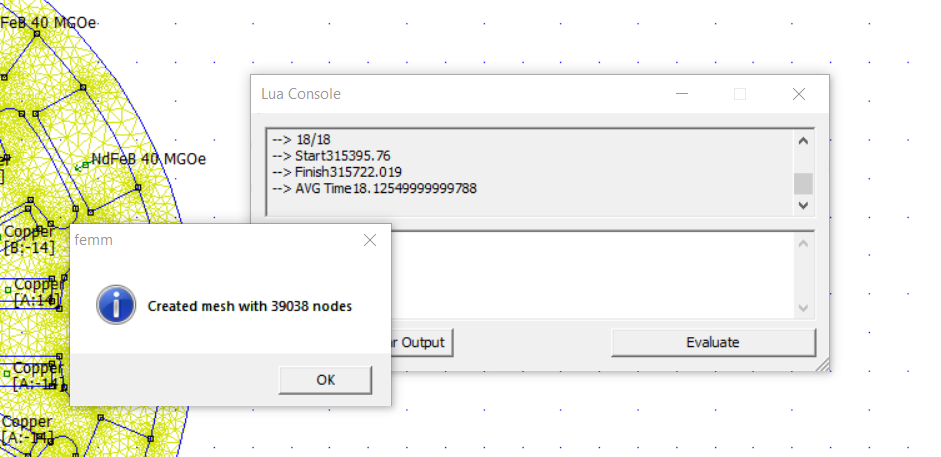
It is known that Ke = Kt when there is no loss.

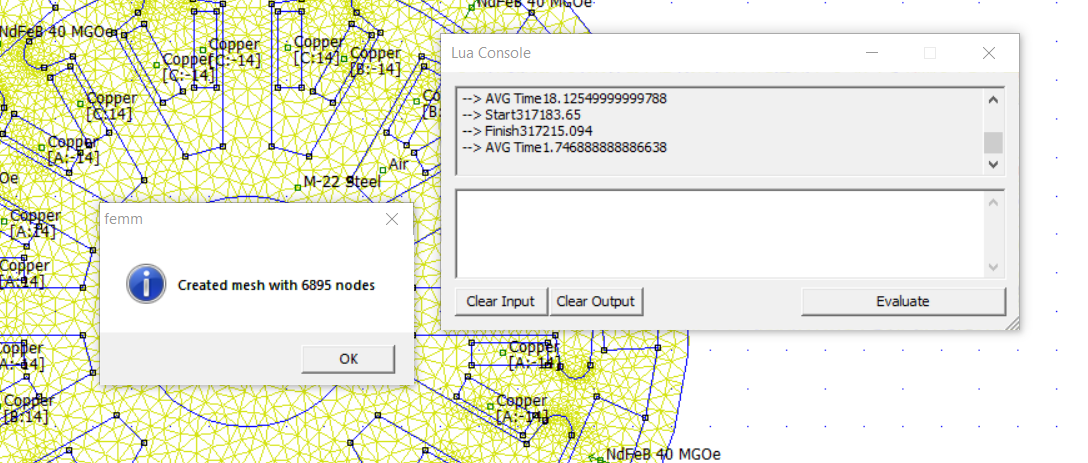
However, Ke is very different from Kt\_calculated\_from\_matlab = 0.014, and even smaller than Kt\_experimental = 0.005.

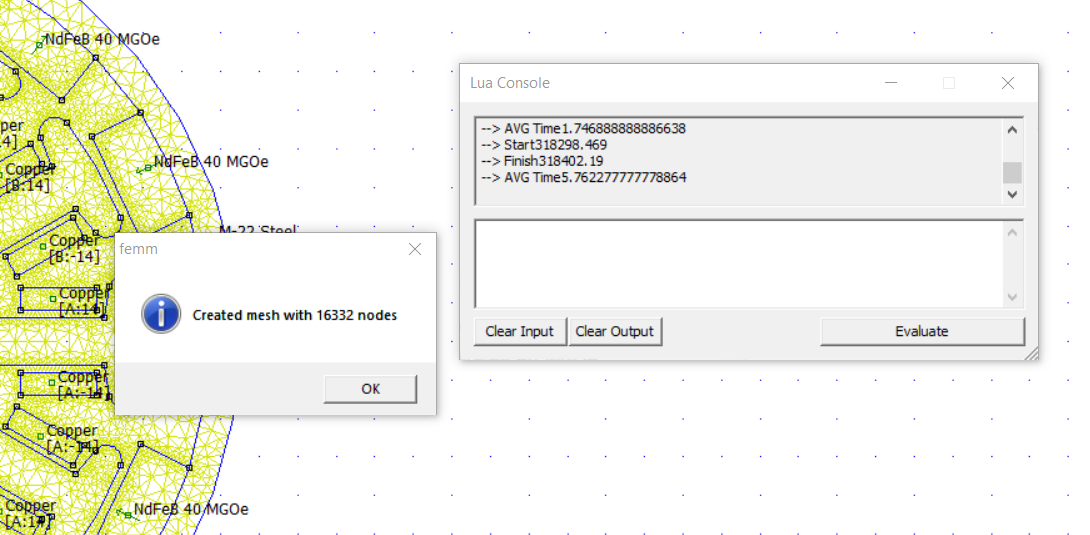


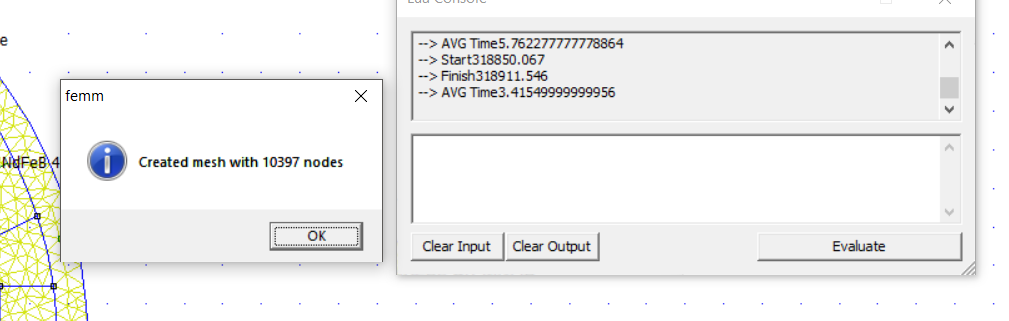
By adjusting “local element size along the line” as well as “Maximum segment degrees” for arcs in the femm model, the following execution time has been estimated by printing the systems clock in the Lua script, and the Torque constant was estimated y running the provided Matlab script.

Lua script





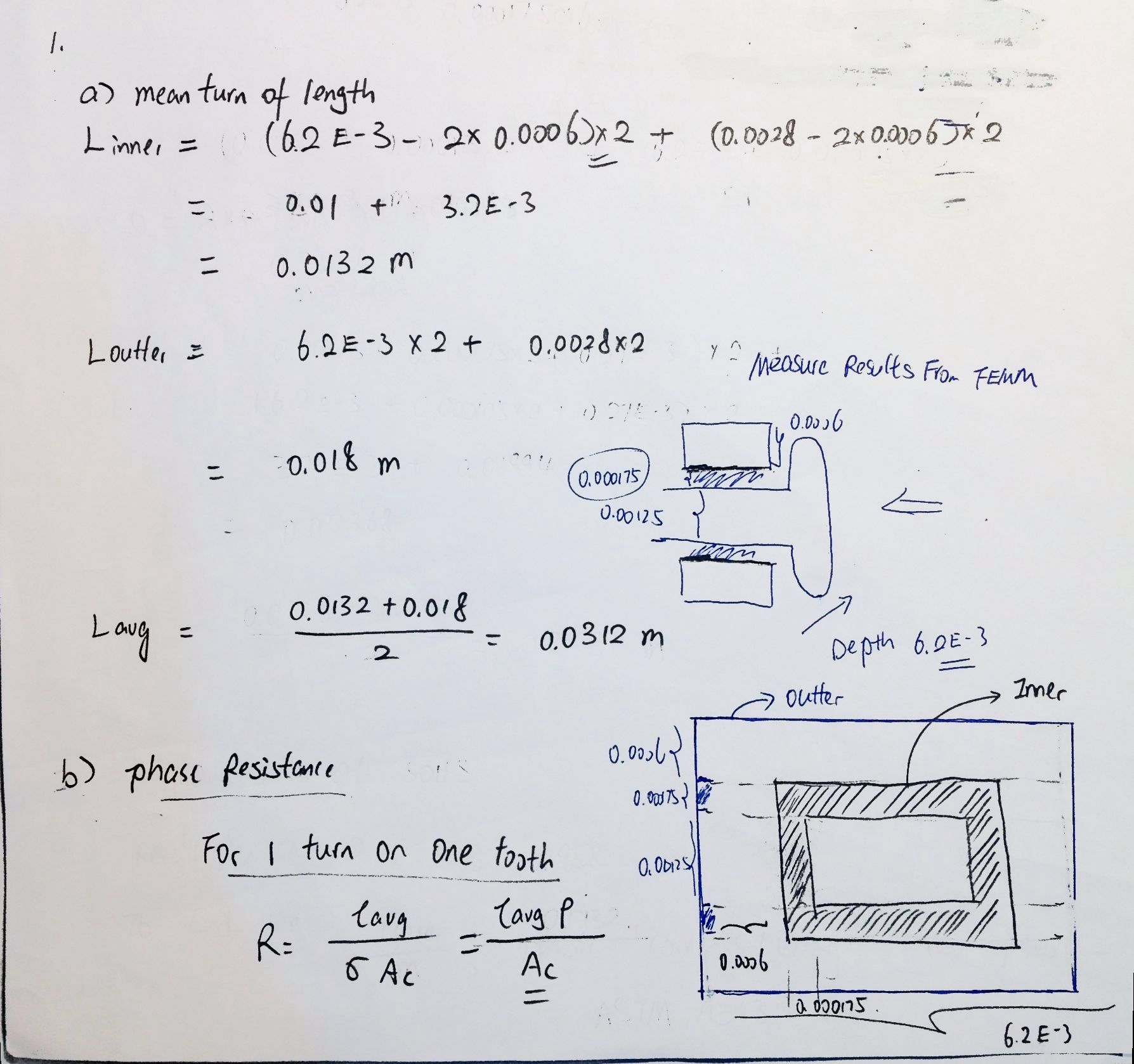


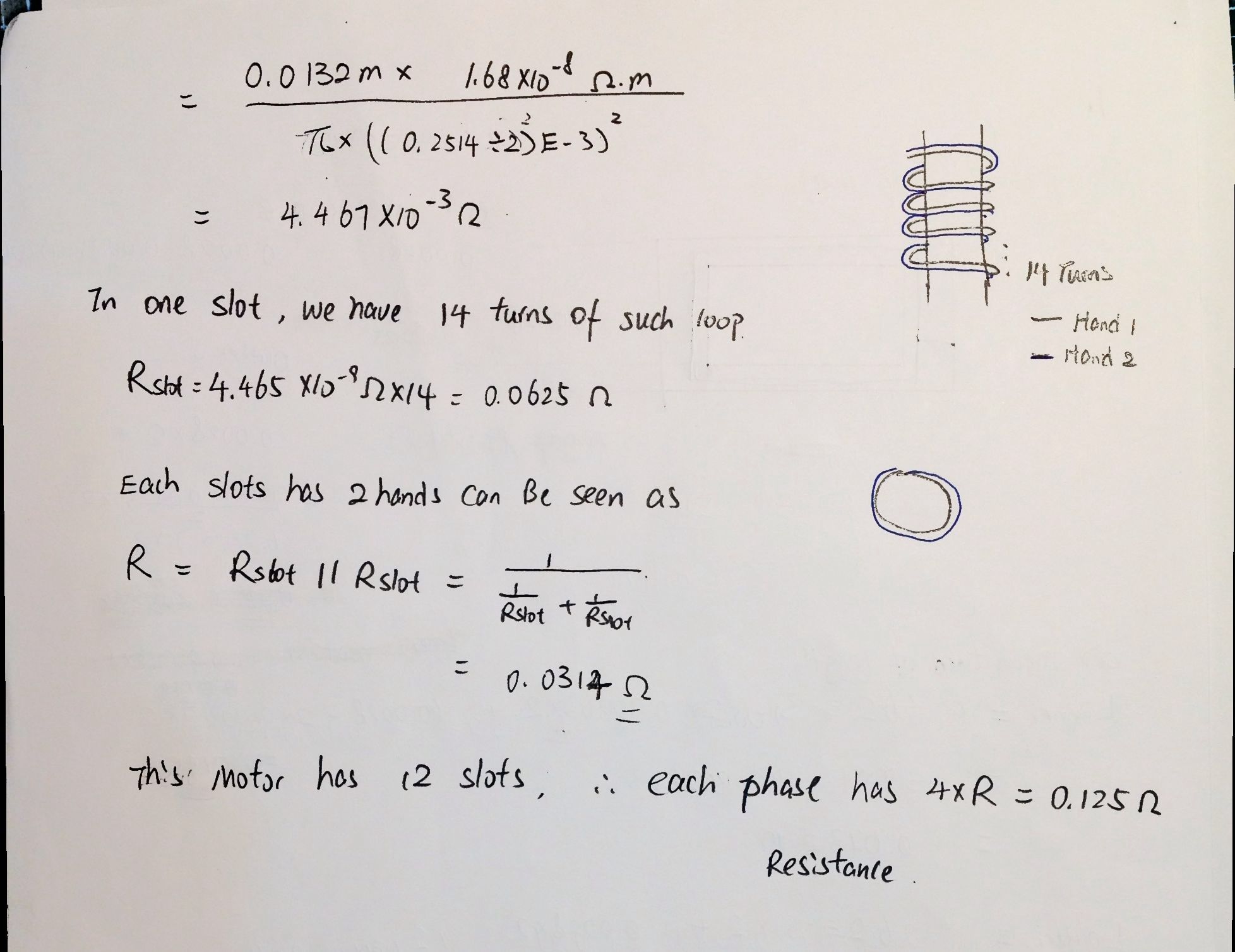


|  |  |  |
| --- | --- | --- |
| Number of nodes | Execution Time(s) | Torque constant from Matlab (Nm/A) |
| 6895 | 32 | 0.0118 |
| 10397 | 61.479 | 0.0118 |
| 16332 | 103 | 0.0117 |
| 39038 | 326.259 | 0.0117 |

**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.**

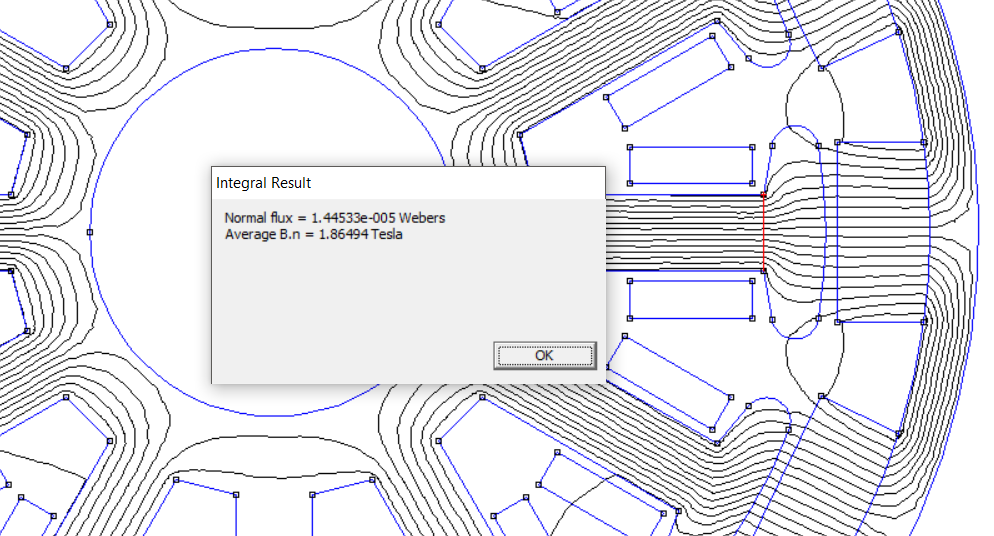
Part 2





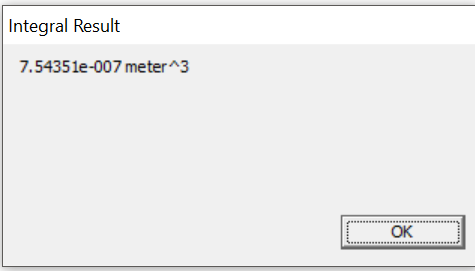
1. Find core loss

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.



Recall from part 1, B = , therefore from the table aforementioned, .

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



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

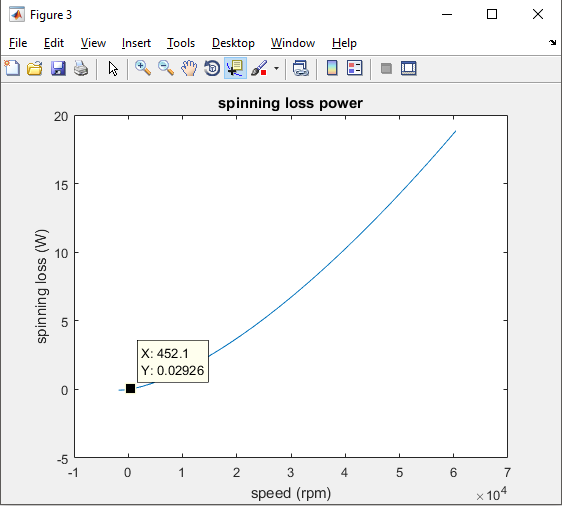
where 2 is the fabrication factor due to the edge damage in lamination steel sheet during production process.

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.



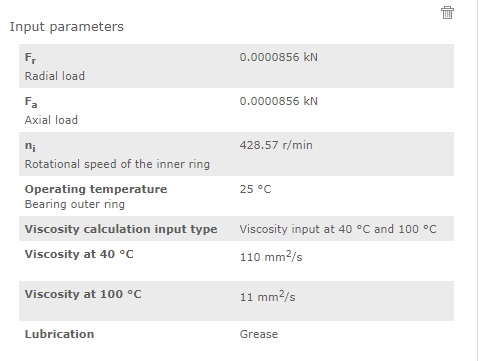
A reasonable running situation for this specific motor is under 11V will leads to 2400KV\*11V = 26400rpm.

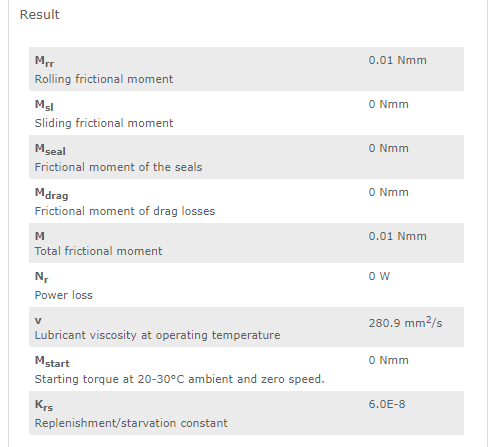
The RPM of the motor can be calculated by

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





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 .

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

And the bearing loss is . Therefore P\_spinning\_analytical = 0.0159W.

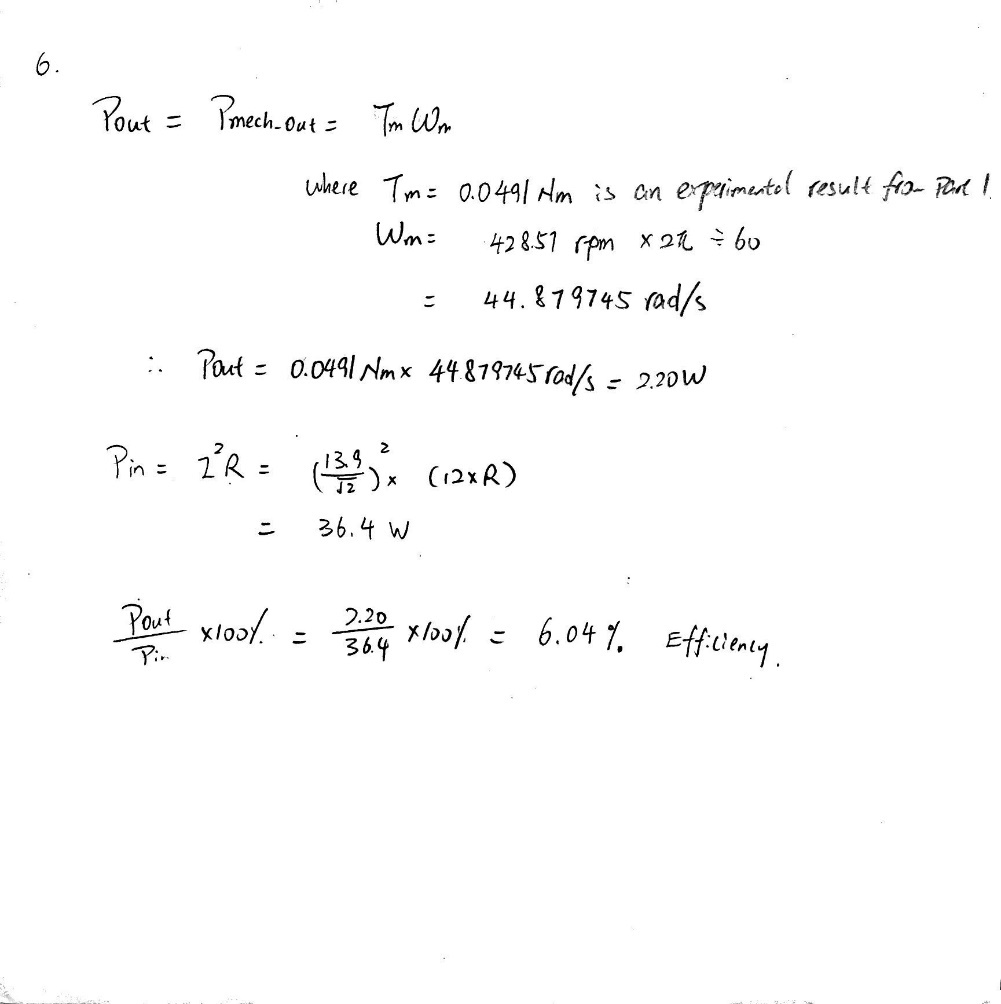
The result of spinning loss can be sum up in the following table

|  |  |
| --- | --- |
| 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.

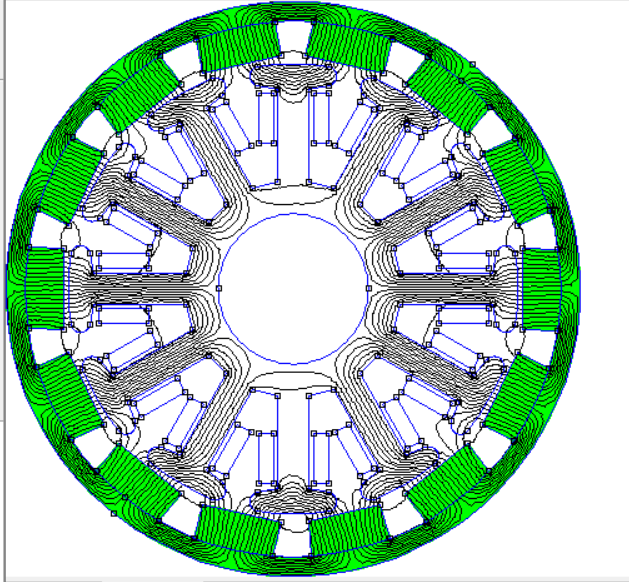
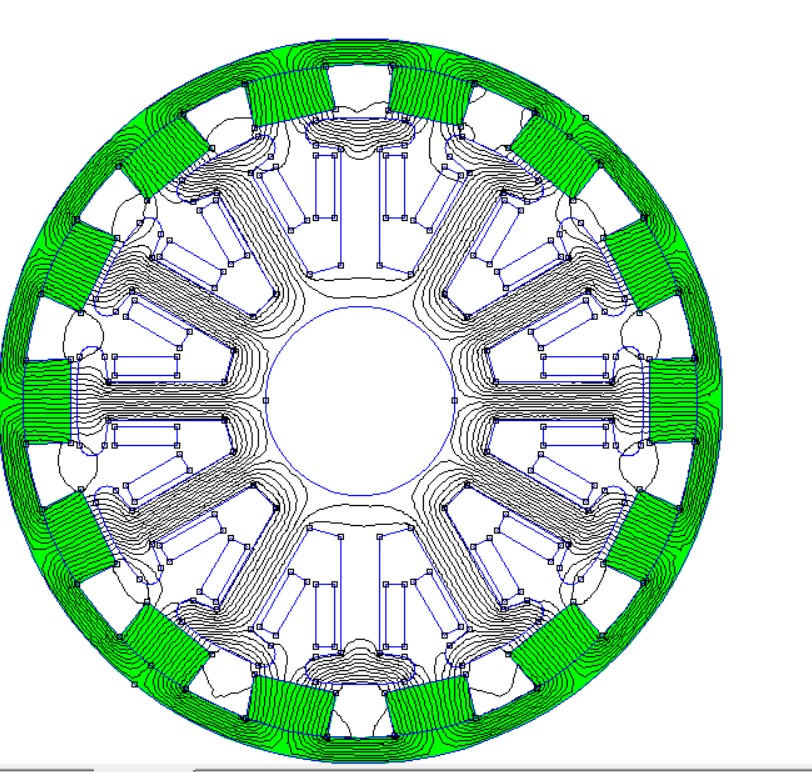
Efficiency = , the detailed calculation has shown below.



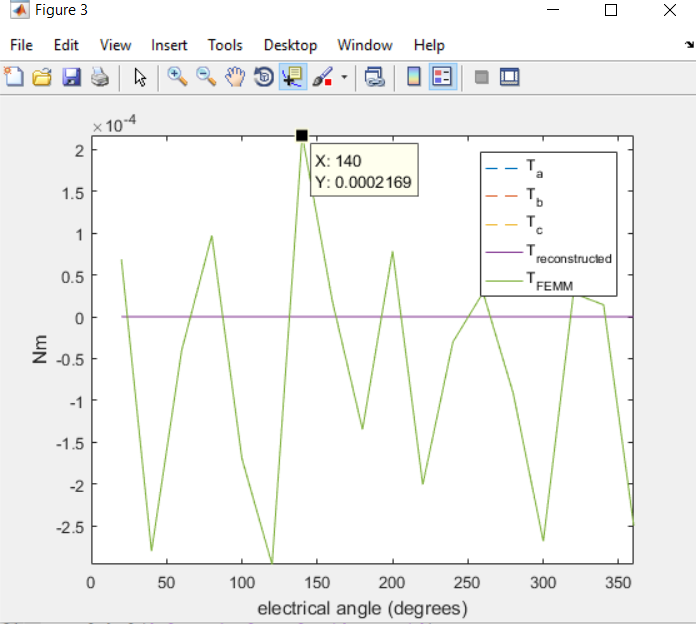
7. Cogging torque

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.

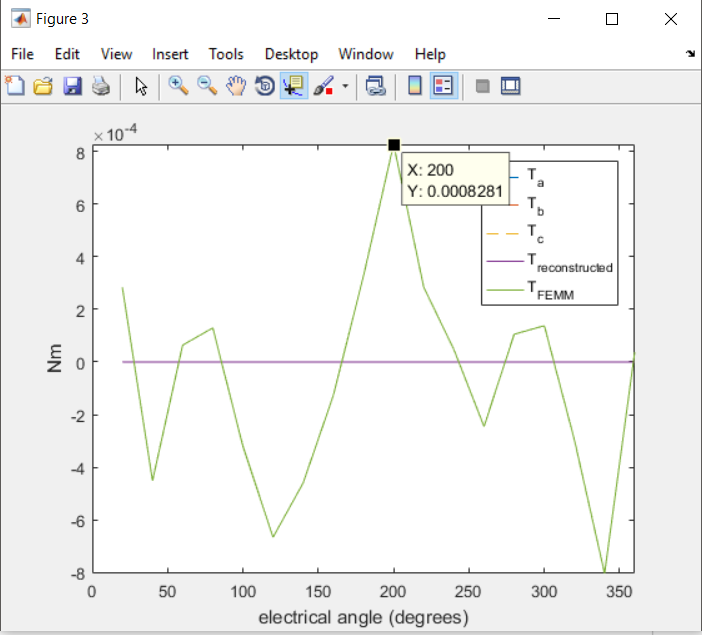


1. Standard magnet width



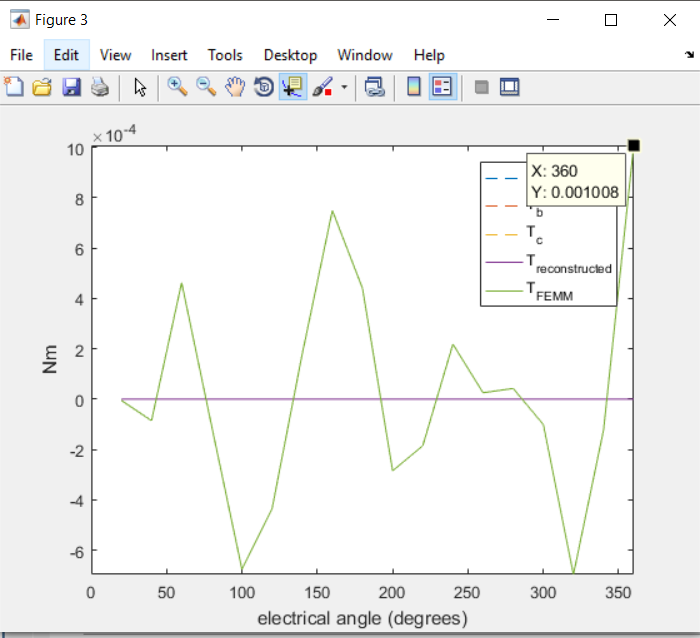
The cogging torque in this case has been found to be 0.0002Nm.

1. A magnet width 10% wider



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

1. Magnet 10% narrower



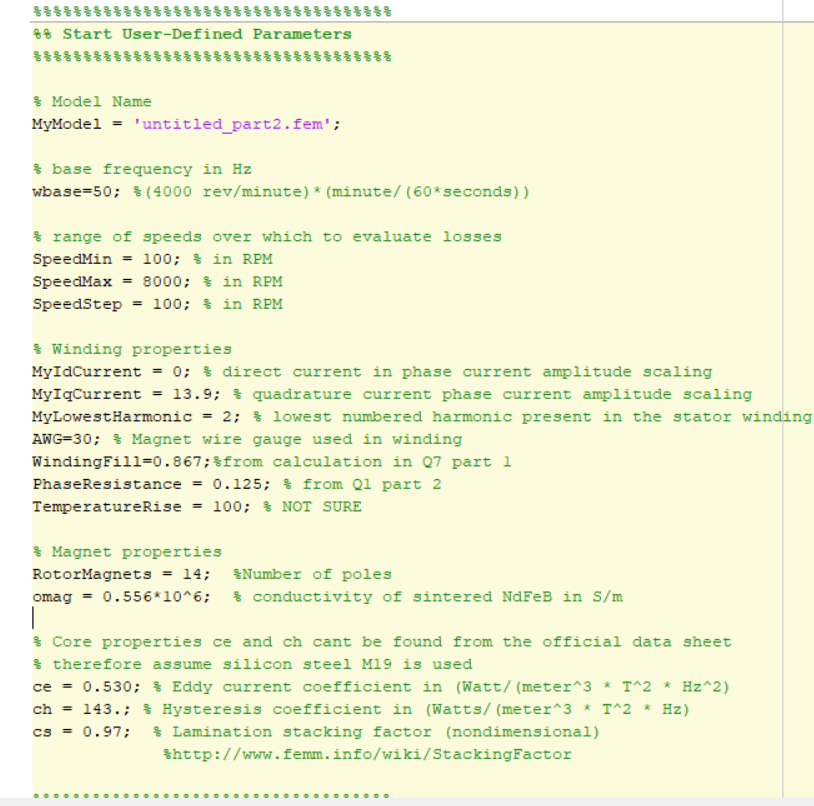
The cogging torque in this case has been found to be 0.001Nm.

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

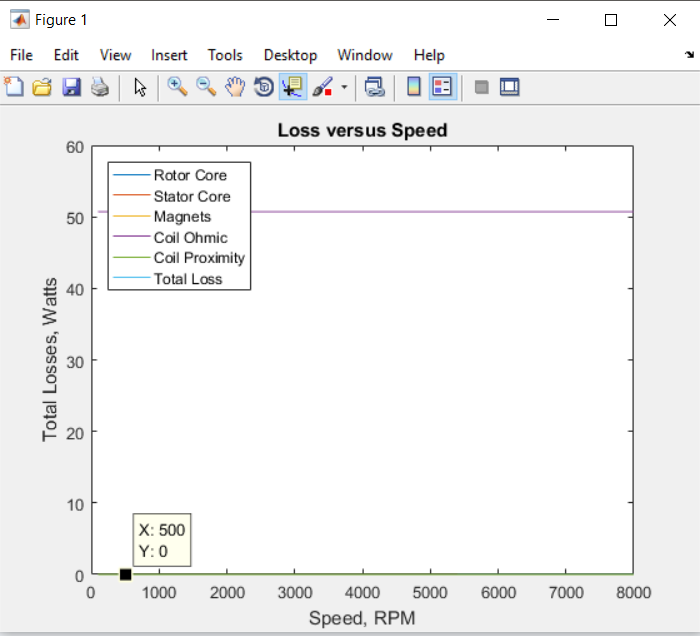
Bonus work attempt

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.



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\_offset\_angle-120)));

  mi\_modifycircprop("B",1,current\_mag\*cos(rad(angle\_elec\_degrees+current\_offset\_angle)));

  mi\_modifycircprop("C",1,current\_mag\*cos(rad(angle\_elec\_degrees+current\_offset\_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');