SEMTEC Report Elmer FEM – Induction Machine Tutorial

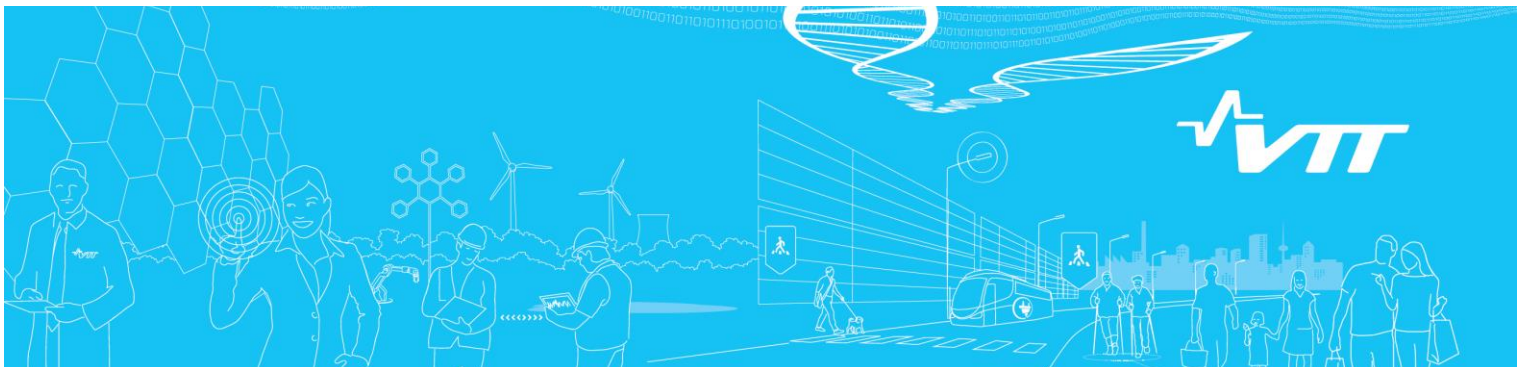
Technical Report · May 2017

DOI: 10.13140/RG.2.2.18599.75688

1 author:

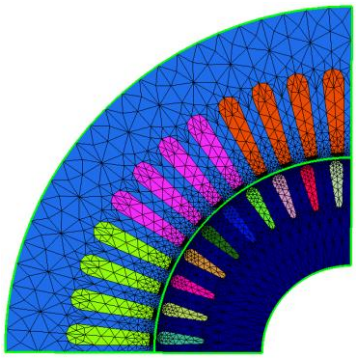
Pavel Ponomarev
ABB
41 PUBLICATIONS **522** CITATIONS

SEE PROFILE



RESEARCH REPORT

VTT-R-02819-17



Elmer FEM
Induction Machine Tutorial

Authors: Pavel Ponomarev

Confidentiality: Public

Report's title Elmer FEM. Induction Machine Tutorial	
Customer, contact person, address Tekes, Matti Säynätjoki	**Order reference** Tekes: 5488/31/2014
Project name SEMTEC: Mastering multiphysics in electrical machines - theory and computational solutions	**Project number/Short name** 104088/SEMTEC
Author(s) Pavel Ponomarev	**Pages** 95
Keywords Elmer FEM, Induction Machine	**Report identification code** VTT-R-02819-17

Summary

This report demonstrates a workflow to model an induction machine using multi-physics simulation software Elmer FEM within an open source toolchain. The test case is an induction motor for blowing applications. The models are also provided. The workflow demonstrates all the typical important simulation procedures usually undertaken to validate and verify an electrical machine design. This report is not in any way a part of Elmer documentation.

Confidentiality	Public

Espoo 19.5.2017

Written by	**Reviewed by**	**Accepted by**
Pavel Ponomarev Research Scientist	Janne Keränen Senior Scientist	Johannes Hyrynen Vice President

VTT's contact address
pavel.ponomarev@vtt.fi

Distribution (customer and VTT)

Tekes/Matti Säynätjoki,
VTT/archive,
other media

Preface

This report is written as a part of the SEMTEC project during years 2016–2017. The main aim of this report is to establish a typical workflow for modelling of electromagnetic devices using capabilities of open-source toolchain with Elmer FEM in its core. All participants of the SEMTEC project (http://www.semtec-project.fi/en/about/) have contributed to the creation of this tutorial. Particularly important contributions came from Peter Råback, Juhani Kataja, Juha Ruokolainen, Janne Keränen, and Eelis Takala.

Espoo 19.5.2017

Pavel Ponomarev

Contents

1. Introduction

Numerical modelling of electrical machines is a difficult task. Multidisciplinary knowledge is required from the modeler – electrical engineering, computer engineering, machine design, mathematics, numerical methods. Often, in order to perform simulation effectively, a very deep knowledge of all the tools is a must. For junior level engineers and those who just started using FEM simulation tools, the analysis could arise a lot of practical questions, answers to which could be obtained only with experience. E.g. what numerical technique, mesh density, solver tolerances, to use for a certain kind of problem; what boundary condition to use for certain situation; how to approach modelling of certain devices; what are the best practices to approach a problem. This document is compiled in an attempt to give some very basic answers to those kinds of questions and provide a documented example with established practices for modelling of electrical machines using open-source toolset with Elmer in its core [4]. This report is not in any way a part of Elmer documentation. The developed models have been tested in the `devel` branch of Elmer 8.2, and are expected to work in Elmer 8.3 release version. Their correct functionality is not guaranteed. The models are available at http://doi.org/10.5281/zenodo.581131 .

2. Model Description

The tutorial concentrates on the modelling and simulation process of one example case. The example case is a 5 kW induction motor for blower applications. Main dimensions are outlined in the Figure 1.

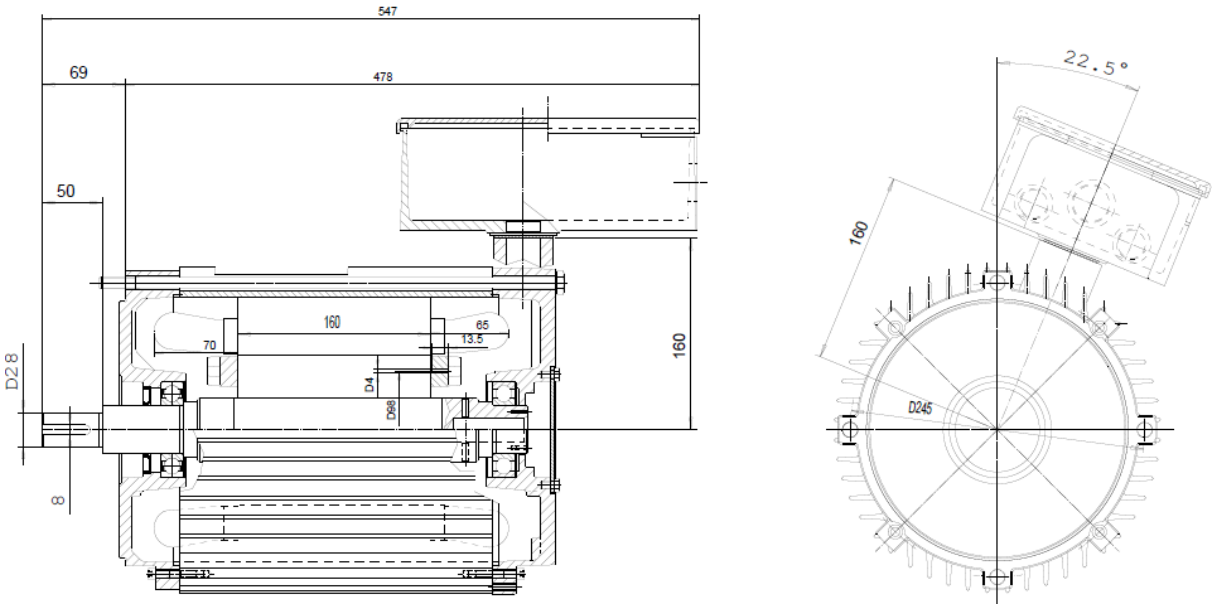


Figure 1. Main dimensions

The machine's main parameters are given in the Table 1.

Table 1. Main parameters.

Parameter	Value
Power	5 kW
Speed	1450 rpm @50Hz
Voltage	400 V
Current	10.4 A
No-load current	6.7 A
Connection	2Y
Number of phases	3
Number of pole pairs	2
Number of stator slots	48
Winding type	1-layer diamond
Number of slot conductors	32
Slots/pole/phase	4
Number of rotor slots	40
Skew	8.5 degrees
Torque	32,9 Nm

The main stator dimensions and stator slot dimensions are shown in the next two figures.

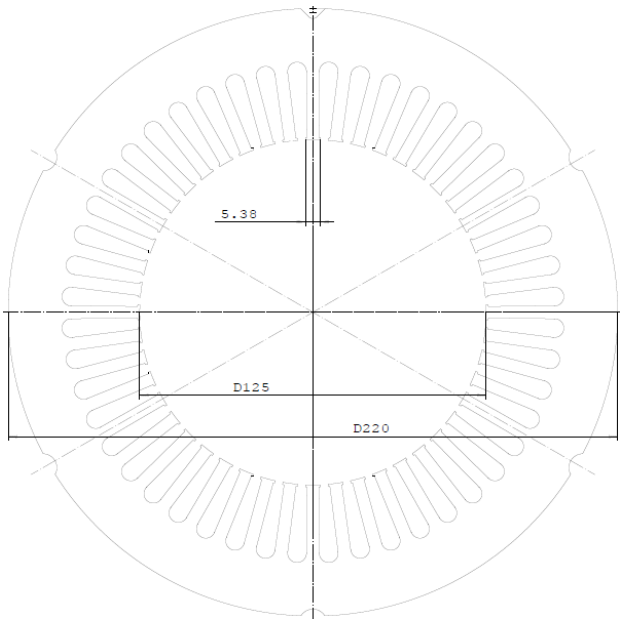


Figure 2. Stator lamination dimensions

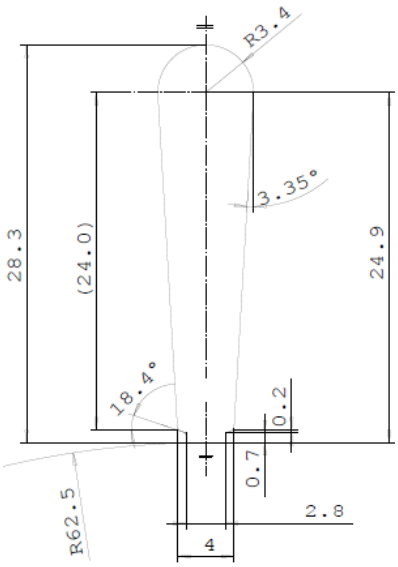


Figure 3. Stator slot dimensions

The rotor dimensions and the shape of the rotor slots for the squirrel cage winding are shown next.

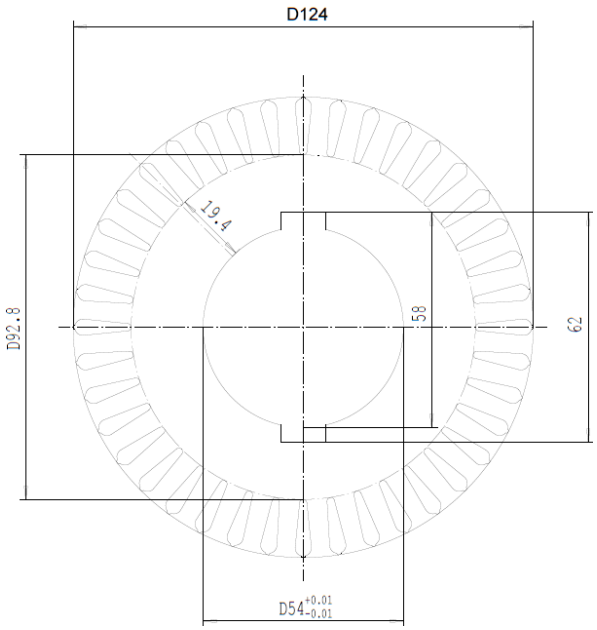


Figure 4. Rotor lamination main dimensions

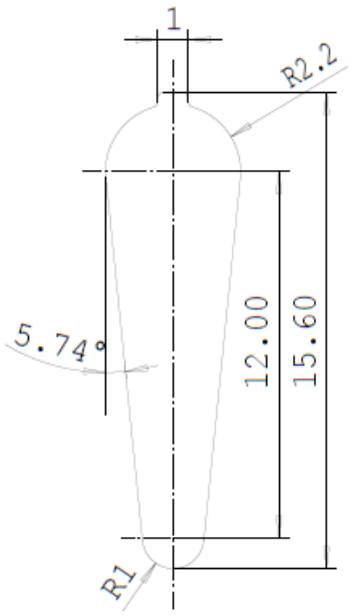


Figure 5. Rotor slot dimensions

The rotor winding is a squirrel cage winding made of aluminum. The winding is short-circuited by the end rings. Figure 6 shows the dimensions of the rotor end rings.

Short circuit rings

Length W	mm	15
Height h	mm	25
Average diameter Dk	mm	97
Inner diameter d	mm	72
Area A	mm^2	375

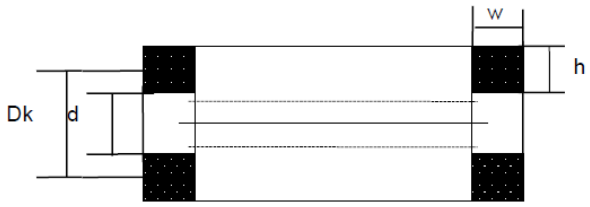


Figure 6. Dimensions of the short-circuiting end rings of the rotor.

The stator winding layout is periodic as the machine has 2 pole pairs. One periodic section of the winding layout is shown in the Figure 7.

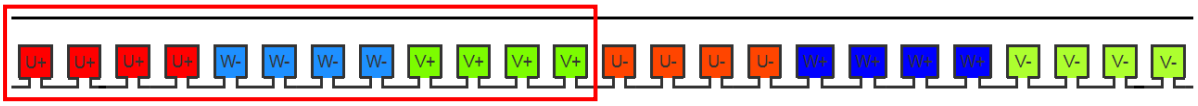


Figure 7. Stator winding layout of one pole pair. Red rectangle shows the modelled field FEM domain.

This periodicity will be used to reduce the model size and speed up the FEM computations. Moreover, just one pole needs to be modelled as fluxes are symmetrical between two poles. Therefore, the FEM model consists of just $1/4^{th}$ of the whole machine cross-section, representing one pole – 10 bars in the rotor and 12 slots in the stator.

The winding connection scheme is a double star as it is seen in Figure 8. The star point can be common or separate for two parallel stars. Each parallel phase coil has two sides - positive and negative. Only one of those sides for each phase needs to be modelled explicitly in FEM domain. Others are modelled implicitly using periodicities, symmetries and circuit relations.

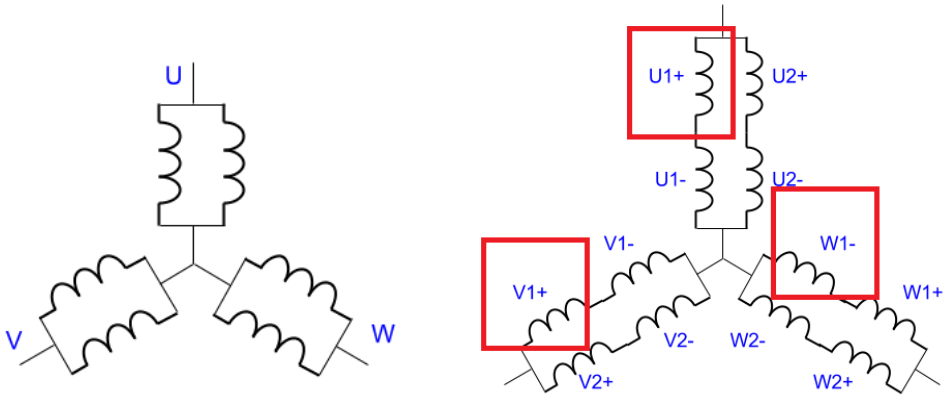


Figure 8. Stator winding connection diagram. Red rectangles show the field FEM domain.

3. Geometry and Meshing

This chapter describes the geometry definition and meshing.

Table 2 shows the geometrical parameters which are used in the construction of the model.

The geometry can be build using open-source CAD tools such as GMSH, FreeCAD, Salome – all those software packages can be run on Windows, Linux and MacOS. The GMSH has great meshing capabilities for 2D problems, and geometry can be effectively constructed using scripting. Also GUI input is supported in GMSH, but it is somewhat limited from user experience point of view. Simple 3D problems also can be defined using GMSH, but for complex 3D problems more tailored CAD tools could be better suited. The Salome and FreeCAD are more visual CAD tools where geometry and mesh can be effectively constructed using GUI.

If the geometry is already available in some widely used mesh format (e.g. unv), then it can be converted to Elmer mesh format using ElmerGrid. In certain cases, even the names of physical regions could be retained during the mesh conversion.

3.1 GMSH – Stator Geometry

The machine geometry will be built using GMSH scripting (it is also possible to use GMSH GUI for building the machine's geometry, see e.g. [7]). Stator and rotor geometry will be defined in separate files to simplify construction. The boundary mesh between stator and rotor does not need to be conformal as Elmer utilizes special mortar boundaries to enforce continuity over non conformal meshes. Outer stator boundary and inner rotor boundary will be modelled having zero normal flux component. This excludes effect of frame and shaft.

The model is based on GMSH geo files originally constructed by J. Gyselinck and R.V. Sabariego which are available here http://onelab.info/wiki/Electric_machines.

The main stator parameters are listed in the

Table 2. The slot dimensions notation is described by the Figure 9.

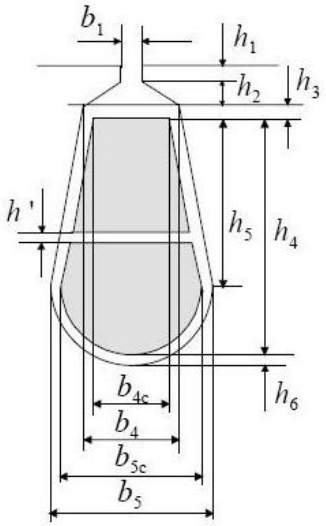


Figure 9. Slot dimensions.

Table 2. Geometrical parameters of the stator.

Parameter	Description	Value
Qs	Number of the stator teeth	48
N_ss	Number of the modelled slots	12
R_sin	Stator bore radius	62.5
R_sout	Outer stator radius	110
R_g	Radius of the middle of the air gap	62.25
b_5	Bottom slot width	6.8
b_1	Slot opening width	2.8
h_1	Height of the tooth tips	0.7
h_2	Height of the tips uplift	0.2
b_4	Width of the top of the slot	4
h_5	Height of the main slot section	24

The complete listing of the GMSH script is given in the Appendix.

The variables (parameters) are defined at the beginning of the file `im_stator.geo`.

```
//Stator
Qs = 48; // number of stator teeth
N_ss = 12;

//Main Stator parameters
R_sin = 62.5;      // inner stator radius
R_sout = 110;       // outer stator radius
R_g = 62.25;        // middle of the air gap radius

//Slot dimensions
b_5 = 6.8;
b_1 = 2.8;
h_1 = 0.7;
h_2 = 0.2;
b_4 = 4;
h_5 = 24;
```

In this tutorial the slot heights are defined referencing to the axis of the slot. Therefore, in order to account for stator inner surface curvature an offset *s* needs to be calculated as

```
//offset for the curvature of the slot opening in the gap
s = R_sin - Sqrt(R_sin^2-(b_1/2)^2);
```

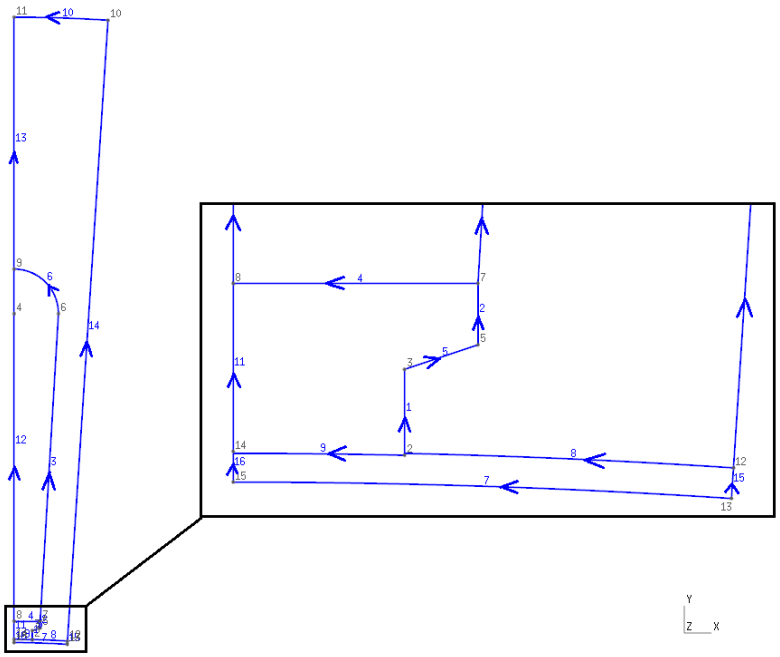


Figure 10. Stator slot point and line numbering, and line directions.

Figure 10 shows the point and line numbering for the stator slot construction. The point numbering starts from 2 at the tooth tip as the point number 1 is in the center of the rotor at the coordinate system origin. The line direction information is important for building arcs and defining line loops for physical regions. The line direction is defined by the order in which points are entered during line definition.

The points of the geometry of Figure 10, i.e. the points of the half of the slot geometry, are defined next.

```
//points of one half
  dP=newp;
  Point(dP+0)   = {0,0,0,m_coarse};

  //right toothtip points
  Point(dP+1)   = {b_1/2, R_sin-s, 0, 2*m_gap};
  Point(dP+2)   = {b_1/2, R_sin-s+h_1, 0, m_sl_bot};

  //center of the top circle
  Point(dP+3)   = {0, R_sin-s+h_1+h_2+h_5, 0, m_sl_top};

  //h2-b4 bottom teeth point
  Point(dP+4)   = {b_4/2, R_sin-s+h_1+h_2, 0, m_sl_bot};

  //h5-b5 top teeth point
  Point(dP+5)   = {b_5/2, R_sin-s+h_1+h_2+h_5, 0, m_sl_top};

  // point of lower winding
  Point(dP+6)   = {b_4/2, R_sin-s+h_1+h_2+0.5, 0, m_sl_bot/2};

  // central point of lower winding
  Point(dP+7)   = {0, R_sin-s+h_1+h_2+0.5, 0, m_sl_bot};

  // top slot point
  Point(dP+8)   = {0, R_sin-s+h_1+h_2+h_5+b_5/2, 0, m_sl_top};
```

```
// outer stator sector
Point(dP+9) = {R_sout*Sin(Pi/Qs), R_sout*Cos(Pi/Qs), 0, m_s_out};

// outer stator center
Point(dP+10) = {0, R_sout, 0, m_s_out};

// inner stator sector
Point(dP+11) = {R_sin*Sin(Pi/Qs), R_sin*Cos(Pi/Qs), 0, 2.5*m_gap};

// sliding sector
Point(dP+12) = {R_g*Sin(Pi/Qs), R_g*Cos(Pi/Qs), 0, 2.4*m_gap};

//inner stator center
Point(dP+13) = {0, R_sin, 0, 1.5*m_gap};

//sliding center
Point(dP+14) = {0, R_g, 0, 1.5*m_gap};
```

So, these 14 points describe the geometry of the sector of the rotor between center of the slot and center of the tooth.

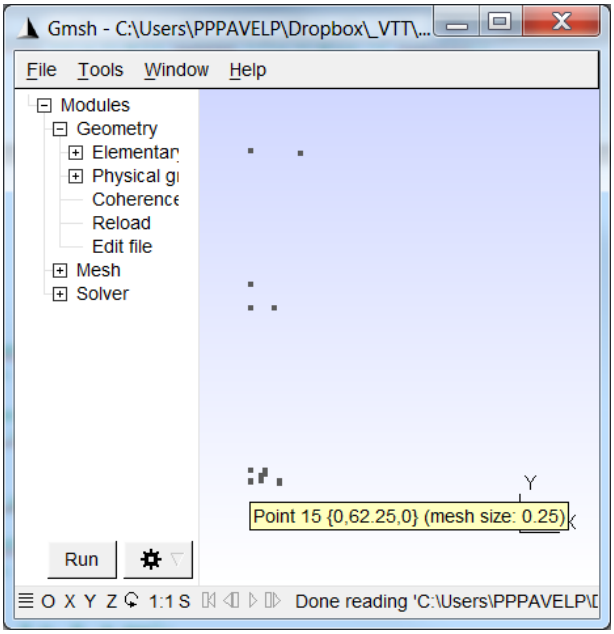


Figure 11. Points of half of the stator slot.

dP is a variable which allows to track building the geometry of all other modelled sectors in a for-loop and will be used later.

The third coordinate of each point definition is 0 as we are building 2D model. Fourth parameter is the ”Characteristic Length” (or mesh density) in each point. The following Characteristic Lengths were defined in the beginning of the geo file:

```
//Mesh density
m_coarse = 20;
m_normal = 12;
m_gap = 0.55;
m_sl_top = 7;
m_sl_bot = 1.5;
m_s_out = 12;
```

These values were selected to produce as coarse mesh as possible, but keeping all the details of the geometry resolved correctly. Also mesh should be dense enough at the airgap region.

Sometimes additional points need to be added to the geometry to control the mesh density more precisely. It is also possible to control mesh density by specifying distribution of the

mesh points over a line using “Transfinite Line”. GMSH has many other options to control the mesh density [1,2].

Next, the lines of the geometry are constructed based on the defined points.

```
dR=newl-1;
//vertical tooth tip line
Line(dR+1) = {dP+1,dP+2};

//line from tooth tip arc to start of the winding
Line(dR+2) = {dP+4,dP+6};

//vertical slot side
Line(dR+3) = {dP+6,dP+5};

//horizontal winding bottom
Line(dR+4) = {dP+6,dP+7};

//arc toothtip - side
Line(dR+5) = {dP+2,dP+4};

//arc top of the slot-side
Circle(dR+6) = {dP+5,dP+3,dP+8};

//sliding
Circle(dR+7) = {dP+12,dP+0,dP+14};

//slot opening arc
Circle(dR+8) = {dP+11,dP+0,dP+1};

//arc inner teeth surface
Circle(dR+9) = {dP+1,dP+0,dP+13};

//outer stator
Circle(dR+10) = {dP+9,dP+0,dP+10};

//vertical central slot opening
Line(dR+11) = {dP+13,dP+7};

//vertical in the center of the slot
Line(dR+12) = {dP+7,dP+8};

//vertical from top of the slot to the stator outer
Line(dR+13) = {dP+8,dP+10};

//sector border via steel
Line(dR+14) = {dP+11,dP+9};

//sector border via airgap
Line(dR+15) = {dP+12,dP+11};

//vertical sector center via gap
Line(dR+16) = {dP+14,dP+13};
```

The resultant geometry is shown in the Figure 12.

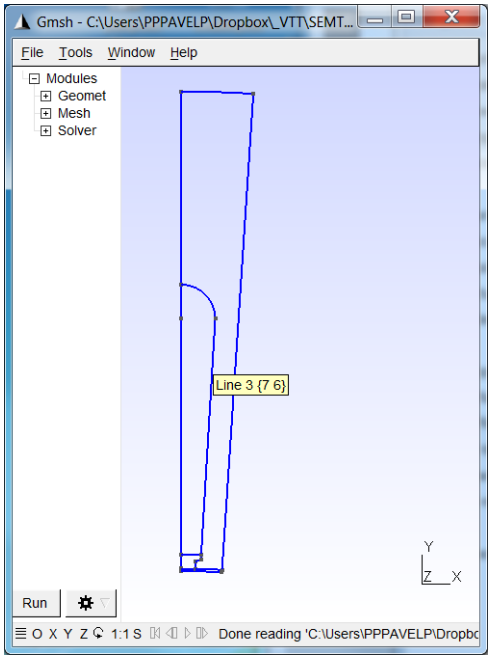


Figure 12. Lines of half of the stator slot.

GMSH removes all geometry entities with the same coordinates automatically. But as our geometry is built by propagation, this might ruin the process, and therefore this feature should be switched off by adding

```
Geometry.AutoCoherence = 0;
```

in the beginning of the script.

Now let us make the whole modelled geometry by nesting those defined lines into 2 for-loops – one for the symmetry parts of the slot, and the second for all the remaining slots. Each new slot points should be rotated to the position of the i^{th} slot.

```
//build stator slots
For i In {0:N_ss-1}
  //build two halfs
  For half In {0:1}

    //Points definitions
    …
    // rotate the built points to the i-th slot position
    For t In {dP+0:dP+14}
      Rotate {{0,0,1},{0,0,0}, 2*Pi*i/Qs+2*Pi/Qs/2} {Point{t};}
    EndFor

    If (half==1) //second half
      For t In {dP+0:dP+14}
        Symmetry {Cos(2*Pi*i/Qs+2*Pi/Qs/2),Sin(2*Pi*i/Qs+2*Pi/Qs/2),0,0} {Point{t};}
      EndFor
    EndIf

    //Lines definitions
    …
  EndFor
EndFor
```

The result is shown in the Figure 13.

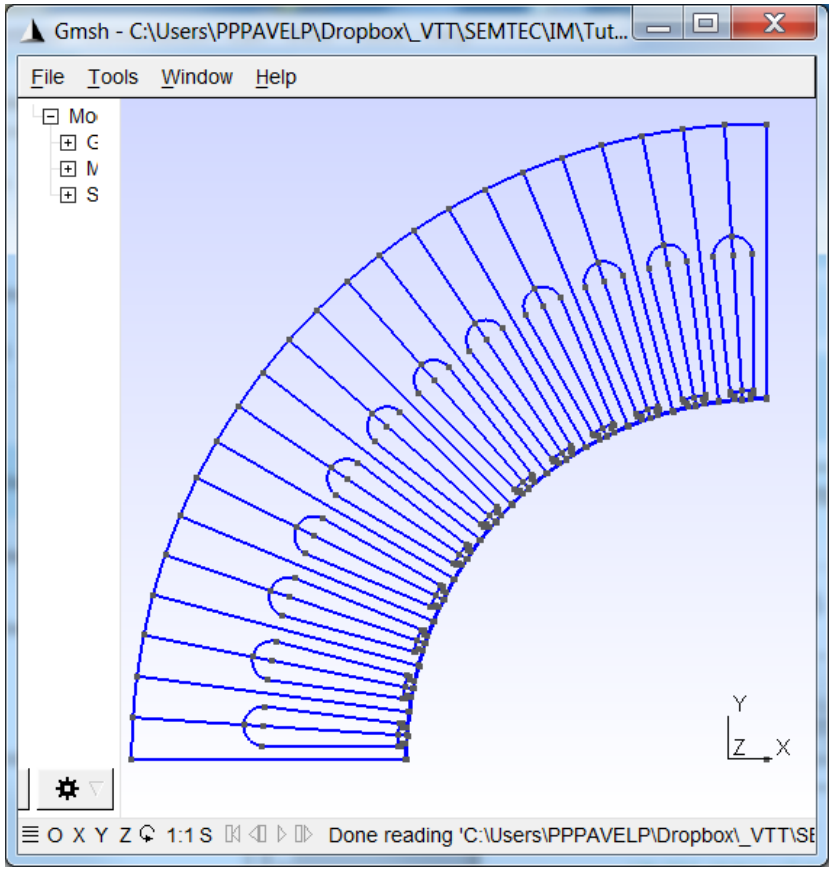


Figure 13. Stator pole.

While building the geometry we need also keep track of those lines which constitute physical entities – boundaries of the regions, periodic boundaries, and sliding boundaries. So, in every iteration after lines are defined, the arrays with indices of the physical boundaries are updated.

```
 //filling the lists for boundaries
OuterStator_[] += dR+10;
StatorBoundary_[] += {dR+10,dR+6,dR+3,dR+2,dR+5,dR+1,dR+8};
Sliding_[] += {dR+7};

//Periodic boundary
If (Qs != N_ss)
  //right boundary
  If (i==0 && half==0)
    StatorPeriod_Right_[] = {dR+14,dR+15};
  EndIf
  //left boundary
  If (i == N_ss-1  && half==1)
    StatorPeriod_Left_[] = {dR+14,dR+15};
  EndIf
EndIf
```

The `StatorBoundary_` array is not needed here, however it will be used later for prettier mesh visualization.

Then, the plane surfaces are defined to build physical regions:

```
//if mirrored, then the lines order is reversed
//direction is important defining the Line Loops
rev = (half ? -1 : 1);

//surface of the slot conductors
Line Loop(newll) = {dR+12,-dR-6,-dR-3,dR+4};
dH = news; Plane Surface(news) = -rev*{newll-1};
```

```
StatorConductor_[] += dH;

//surface of the stator iron
Line Loop(newll) = {dR+1,dR+5,dR+2,dR+3,dR+6,dR+13,-dR-10,-dR-14,dR+8};
dH = news; Plane Surface(news) = -rev*{newll-1};
StatorIron_[] += dH;

//wedges
Line Loop(newll) = {dR+11,-dR-4,-dR-2,-dR-5,-dR-1,dR+9};
dH = news; Plane Surface(news) = -rev*{newll-1};
StatorWedge_[] += dH;

//airgap stator
Line Loop(newll) = {-dR-16,-dR-7,dR+15,dR+8,dR+9};
dH = news; Plane Surface(news) = rev*{newll-1};
StatorAirgapLayer_[] += dH;
```

Next, the stator conductor plane surfaces are assigned to the physical surfaces (or regions). These physical entities names will be used later in the Elmer simulation file. For the 3-phase single-layer diamond integer-slot distributed winding next procedure can be used after the for-loops of the stator main geometry.

```
//Assigning StatorConductors to phase Regions
//number of slots in a belt
qq=4;

//for each phase side
For f In {0:5}
  Con[]={};
  For i In {0:N_ss/qq-1}
    If (Fmod(i,6) == f)
      For j In {0:qq-1}
        Con[] += StatorConductor_[{2*i*qq+2*j, 2*i*qq+2*j+1}];
      EndFor
    EndIf
  EndFor

  //color the phase and assign to a physical region
  If (#Con[] > 0)
    If (f == 0)
                          Color Red {Surface{Con[]};}
                          Physical Surface("U_plus") = {Con[]};
    EndIf
    If (f == 1)
                          Color SpringGreen {Surface{Con[]};}
                          Physical Surface("W_minus") = {Con[]};
    EndIf
    If (f == 2)
                          Color Gold {Surface{Con[]};}
                          Physical Surface("V_plus") = {Con[]};
    EndIf
    If (f == 3)
                          Color Pink {Surface{Con[]};}
                          Physical Surface("U_minus") = {Con[]};
    EndIf
    If (f == 4)
                          Color ForestGreen {Surface{Con[]};}
                          Physical Surface("W_plus") = {Con[]};
    EndIf
    If (f == 5)
                          Color PaleGoldenrod {Surface{Con[]};}
                          Physical Surface("V_minus") = {Con[]};
    EndIf
  EndIf
EndFor
```

Next, all the remaining physical lines and surfaces are defined for the stator:

```
Physical Surface("StatorIron") = {StatorIron_[]};
Physical Surface("StatorWedges") = {StatorWedge_[]};
Physical Surface("StatorAirgap") = {StatorAirgapLayer_[]};

Color SteelBlue {Surface{StatorIron_[]};}
Color Black {Surface{StatorWedge_[]};}
Color SkyBlue {Surface{StatorAirgapLayer_[]};}
```

```
Physical Line("OuterStator") = {OuterStator_[]};
Physical Line("StatorRight") = {StatorPeriod_Right_[]};
Physical Line("StatorLeft") = {StatorPeriod_Left_[]};

Physical Line("Sliding_Stator") = {Sliding_[]};
```

Next commands show just selected colored meshed geometry.

```
Coherence;

show_stator[] = CombinedBoundary{Surface{StatorIron_[]};};
show_stator[] +=
CombinedBoundary{Surface{StatorWedge_[],StatorAirgapLayer_[]};};

Hide { Point{ Point '*' }; }
Hide { Line{ Line '*' }; }
Show { Line{ show_stator[] }; }
Mesh 2;
```

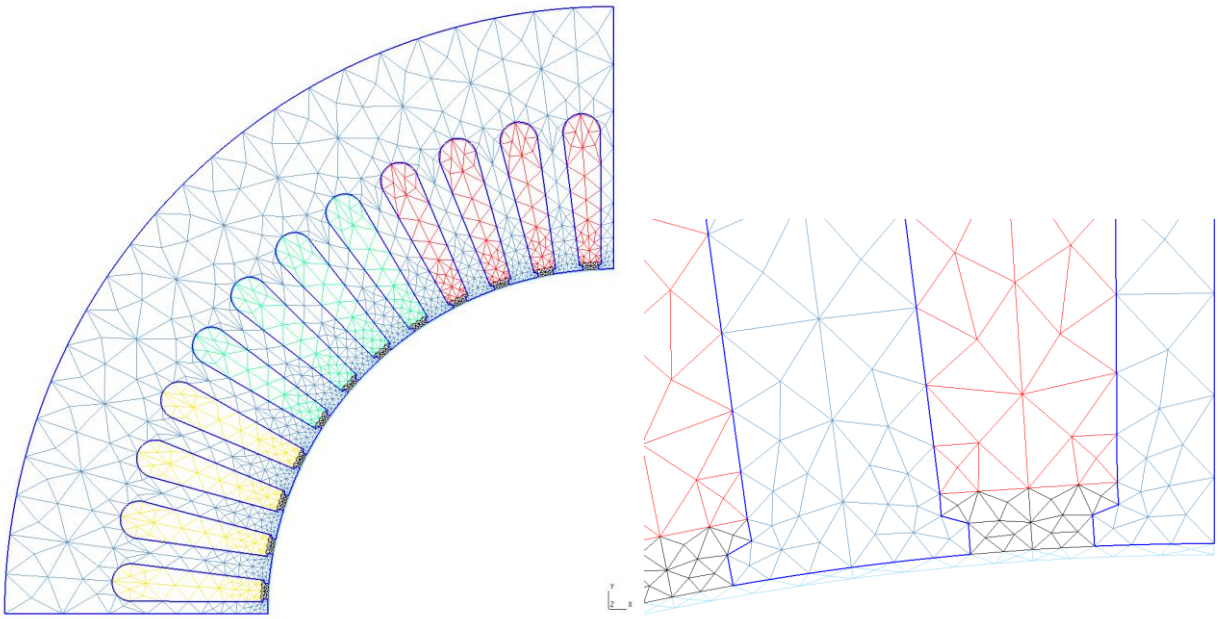


Figure 14. Stator pole meshed.

The resultant stator mesh is shown in the Figure 14. This 2D mesh has 2760 triangular elements, total number of elements is 3891.

Mesh can be saved in msh format by

```
>gmsh im_stator.geo -2 -o im_stator.msh
```

3.2 GMSH – Rotor geometry

The main rotor parameters are listed in the Table 3. The slot dimensions notation is the same as for the stator.

Table 3. Geometrical parameters of the rotor.

Parameter	Description	Value
Qr	Number of the rotor teeth	40

N_rs	Number of the modelled rotor slots	10
R_rin	Rotor inner radius	28
R_rout	Outer rotor radius	62
R_g	Radius of the middle of the air gap	62.25
b_5	Bottom slot width	2
b_1	Slot opening width	1
h_1	Height of the tooth tips	0.4
h_5	Height of the main slot section	12
b_4	Width of the top of the slot	4.4

The complete listing of the GMSH script is given in the Appendix. Two additional parameters are defined:

```
//offset for the curvature of the slot opening in the gap
s = R_rout-Sqrt((R_rout)^2-(b_1/2)^2);
//offset of the top arc curvature at the tip
s2 = b_4/2-Sqrt((b_4/2)^2-(b_1/2)^2);
```

The following characteristic lengths are defined for the rotor points:

```
//Mesh density
m_coarse = 10;
m_normal = 8;
m_gap = 0.5;
m_sl_bot = 4;
m_sl_top = 1;
m_r_in = 10;
```

Next the points of one half of the slot are defined in the same way as for the stator:

```
//points of one half
dP=newp;
Point(dP+0)  = {0,0,0,m_coarse};

//right tooth tip points
Point(dP+1)  = {b_1/2, R_rout-s, 0, m_gap};
Point(dP+2)  = {b_1/2, R_rout-h_1-s2, 0, m_sl_top};

//center of the bottom circle
Point(dP+3)  = {0, R_rout-h_1-b_4/2-h_5, 0, m_sl_bot};

//b4 top circle side
Point(dP+4)  = {b_4/2, R_rout-h_1-b_4/2, 0, m_sl_top};

//b5 bottom circle side
Point(dP+5)  = {b_5/2, R_rout-h_1-b_4/2-h_5, 0, m_sl_bot};

// central point of top circle
Point(dP+6)  = {0, R_rout-h_1-b_4/2, 0, m_sl_top};

// bottom slot point
Point(dP+7)  = {0, R_rout-h_1-b_4/2-h_5-b_5/2, 0, m_sl_bot};

// inner rotor sector
Point(dP+8) = {R_rin*Sin(Pi/Qr), R_rin*Cos(Pi/Qr), 0, m_r_in};

// inner rotor center
Point(dP+9) = {0, R_rin, 0, m_r_in};

// outer rotor sector
Point(dP+10) = {R_rout*Sin(Pi/Qr), R_rout*Cos(Pi/Qr), 0, m_gap};

// sliding sector
Point(dP+11) = {R_g*Sin(Pi/Qr), R_g*Cos(Pi/Qr), 0, m_gap};

//outer rotor center
Point(dP+12) = {0, R_rout, 0, m_gap};
```

```
//sliding center
Point(dP+13) = {0, R_g, 0, m_gap};
```

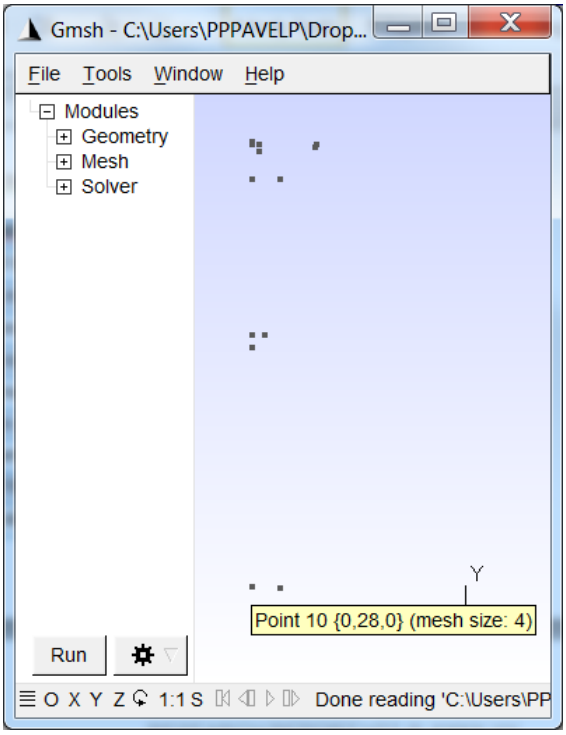


Figure 15. Rotor segment points.

Then the points should be connected using lines and arcs.

```
dR=newl-1;
//vertical tooth tip line
Line(dR+1) = {dP+1,dP+2};

//top arc
Circle(dR+2) = {dP+2,dP+6,dP+4};

//vertical slot side
Line(dR+3) = {dP+4,dP+5};

//arc slot bottom
Circle(dR+4) = {dP+5,dP+3,dP+7};

//sliding
Circle(dR+5) = {dP+11,dP+0,dP+13};

//tooth top arc
Circle(dR+6) = {dP+10,dP+0,dP+1};

//arc slot opening
Circle(dR+7) = {dP+1,dP+0,dP+12};

//inner rotor
Circle(dR+8) = {dP+8,dP+0,dP+9};

//vertical central slot opening
Line(dR+9) = {dP+12,dP+7};

//vertical from bot of the slot to the rotor inner
Line(dR+10) = {dP+7,dP+9};

//sector border via steel
Line(dR+11) = {dP+10,dP+8};

//sector border via airgap
Line(dR+12) = {dP+11,dP+10};

//vertical sector center via gap
Line(dR+13) = {dP+13,dP+12};
```

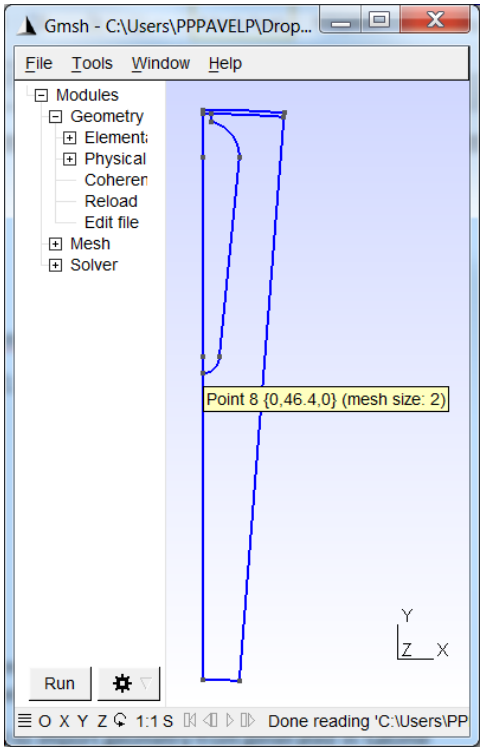


Figure 16. Half of the rotor slot.

Then, again, two nested for-loops are used to mirror and propagate this rotor sector to obtain complete geometry of the modelled rotor pole. Rotation of the defined points to their appropriate position should be implemented for each iteration.

```
//build rotor slots
For i In {0:N_rs-1}
  //build two halfs
  For half In {0:1}

//Points definition
….

    // rotate the built points to the i-th slot position
    For t In {dP+0:dP+13}
      Rotate {{0,0,1},{0,0,0}, 2*Pi*i/Qr+2*Pi/Qr/2} {Point{t};}
    EndFor

    If (half==1) //second half
      For t In {dP+0:dP+13}
        Symmetry {Cos(2*Pi*i/Qr+2*Pi/Qr/2),Sin(2*Pi*i/Qr+2*Pi/Qr/2),0,0} {
Point{t}; }
      EndFor
    EndIf

//Lines definition
….

  EndFor
EndFor
```

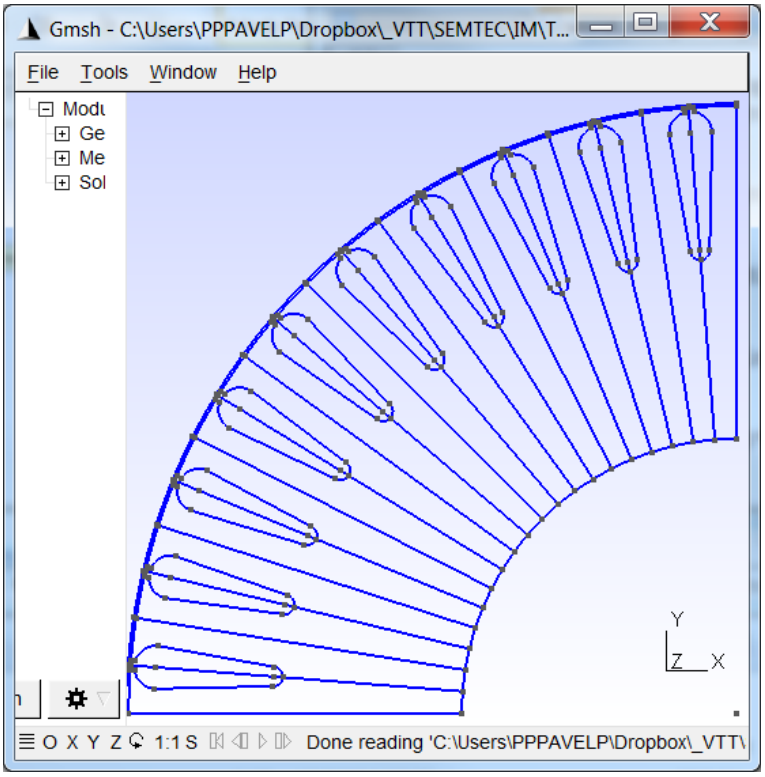


Figure 17. Rotor pole geometry.

Then the boundaries are saved in each iteration, and geometrical surfaces are created for all regions.

```
//filling the lists for boundaries
InnerRotor_[] += dR+8;

RotorBoundary_[] += {dR+8,dR+4,dR+3,dR+2,dR+1,dR+6};

SlidingR_[] += {dR+5};

//Periodic boundary
If (Qr != N_rs)
  //right boundary
  If (i==0 && half==0)
    RotorPeriod_Right_[] = {dR+11,dR+12};
  EndIf
  //left boundary
  If (i == N_rs-1  && half==1)
    RotorPeriod_Left_[] = {dR+11,dR+12};
  EndIf
EndIf

//if mirrored, then the lines order is reversed
//direction is important defining the Line Loops
rev = (half ? -1 : 1);

//surface of the slot conductors
Line Loop(newll) = {dR+9,-dR-4,-dR-3,-dR-2,-dR-1,dR+7};
dH = news; Plane Surface(news) = -rev*{newll-1};
RotorConductor_[] += dH;

//surface of the stator iron
Line Loop(newll) = {dR+1,dR+2,dR+3,dR+4,dR+10,-dR-8,-dR-11,dR+6};
dH = news; Plane Surface(news) = -rev*{newll-1};
RotorIron_[] += dH;

//airgap stator
Line Loop(newll) = {-dR-13,-dR-5,dR+12,dR+6,dR+7};
dH = news; Plane Surface(news) = rev*{newll-1};
RotorAirgapLayer_[] += dH;
```

Then, plane surfaces and boundaries are assigned to Physical Lines and Physical Surfaces.

```
//+----------------------------------------------------------------
// Physical regions
//+----------------------------------------------------------------

Color Yellow {Surface{RotorConductor_[]};}

Physical Surface("Bar1") = {RotorConductor_[{0,1}]};
Physical Surface("Bar2") = {RotorConductor_[{2,3}]};
Physical Surface("Bar3") = {RotorConductor_[{4,5}]};
Physical Surface("Bar4") = {RotorConductor_[{6,7}]};
Physical Surface("Bar5") = {RotorConductor_[{8,9}]};
Physical Surface("Bar6") = {RotorConductor_[{10,11}]};
Physical Surface("Bar7") = {RotorConductor_[{12,13}]};
Physical Surface("Bar8") = {RotorConductor_[{14,15}]};
Physical Surface("Bar9") = {RotorConductor_[{16,17}]};
Physical Surface("Bar10") = {RotorConductor_[{18,19}]};

Physical Surface("RotorIron") = {RotorIron_[]};
Physical Surface("RotorAirgap") = {RotorAirgapLayer_[]};

Color SteelBlue {Surface{RotorIron_[]};}
Color SkyBlue {Surface{RotorAirgapLayer_[]};}

Physical Line("InnerRotor") = {InnerRotor_[]};
Physical Line("RotorRight") = {RotorPeriod_Right_[]};
Physical Line("RotorLeft") = {RotorPeriod_Left_[]};
Physical Line("Sliding_Rotor") = {SlidingR_[]};
```

Then, the rotor is meshed by

```
Coherence;

show_rotor[] = CombinedBoundary{Surface{RotorIron_[]};};

Hide { Point{ Point '*' }; }
Hide { Line{ Line '*' }; }
//Show { Line{ show_rotor[] }; }

Mesh 2;
```

The result is shown here:

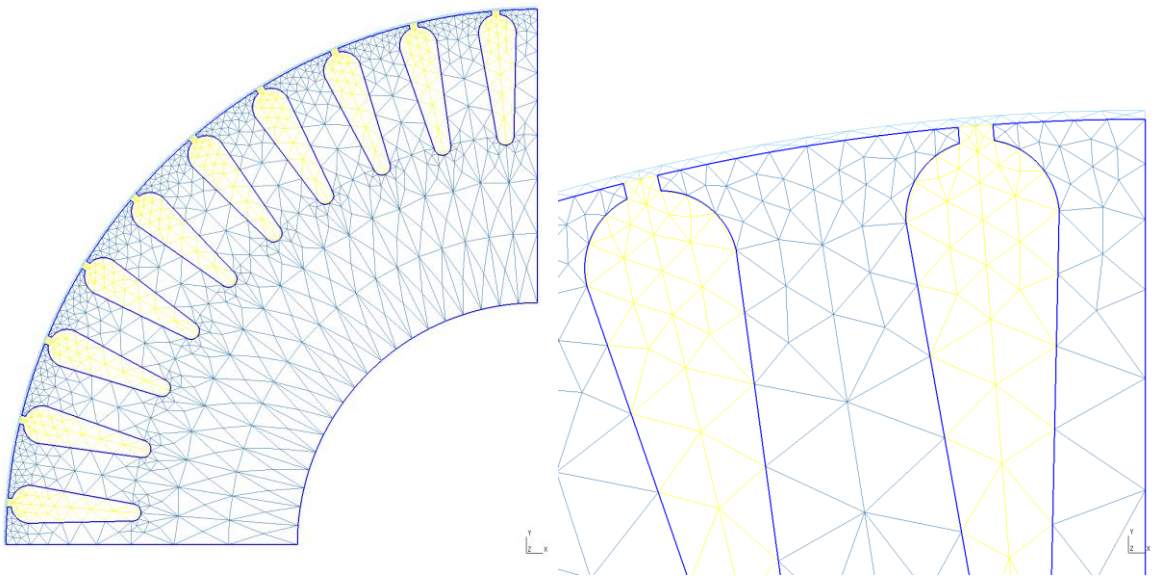


Figure 18. Rotor pole meshed.

This 2D mesh has 2220 triangular elements, total number of elements is 3094.

Mesh can be saved in msh format by

```
>gmsh im_rotor.geo -2 -o im_rotor.msh
```

3.3 Importing

Next, the generated mesh in GMSH .msh-format needs to be exported to the Elmer format. This is done using command line utility ElmerGrid [3].

ElmerGrid supports various formats of mesh files. In some cases the named physical regions can be imported as well (e.g. unv, msh, tra, pf3). If a format is not yet supported, it takes just a minor coding effort to add support of any ASCII format to ElmerGrid.

The following command is used to import stator geometry from generated in GMSH msh.-format to Elmer format. It also preserves the physical regions naming.

```
>ElmerGrid 14 2 im_stator.msh -2d -autoclean -names
```

Here:

- `14` is the input mesh format (for the full list of acceptable formats see ElmerGrid documentation).
- `2` is the output format – Elmer mesh format in this case.
- `-2d` ignores the third coordinate for 2D cases.
- `-autoclean` cleans the unused nodes and conveniently renumbers physical entities.
- `-names` preserves names of physical regions if possible.

Importing is done also for the rotor mesh

```
>ElmerGrid 14 2 im_rotor.msh -2d -autoclean -names
```

Then, those two meshes are united within one single mesh by

```
>ElmerGrid 2 2 im_stator -in im_rotor -unite -autoclean -names -out im
```

The output is folder `im` with the mesh of the whole machine. The resultant mesh has 5060 elements.

Appendix lists file `run.sh` which contains the sequence of commands required for the simulation in Linux environment. The same sequence (or equivalent) can be executed also in MacOSX and Windows environments, provided that all the dependent software (gmsh, Elmer, python) is installed and the paths are known by the console. Some of the commands can be commented out depending on the simulation.

4. SIF – Simulation Input File

In this section a minimal solver input file (SIF) is constructed in order to perform a very basic simulation to verify that the model and boundary conditions are set correctly.

The solving process starts from the preparation of the solver input file (SIF), which contains basic simulation settings, description of the physical regions and boundary conditions, definition of the materials, solver specific settings, and settings for the numerical solvers (direct or iterative). Before reading this section it is advisable to read [5,6].

It is possible, and perhaps even easier for beginners, to prepare the SIF file using ElmerGUI in interactive mode. However, in this section the SIF is created manually. An example SIF with periodicities in a rotating electrical machine can be found also in Elmer source code `Elmer/fem/tests/mgdyn2D_em`.

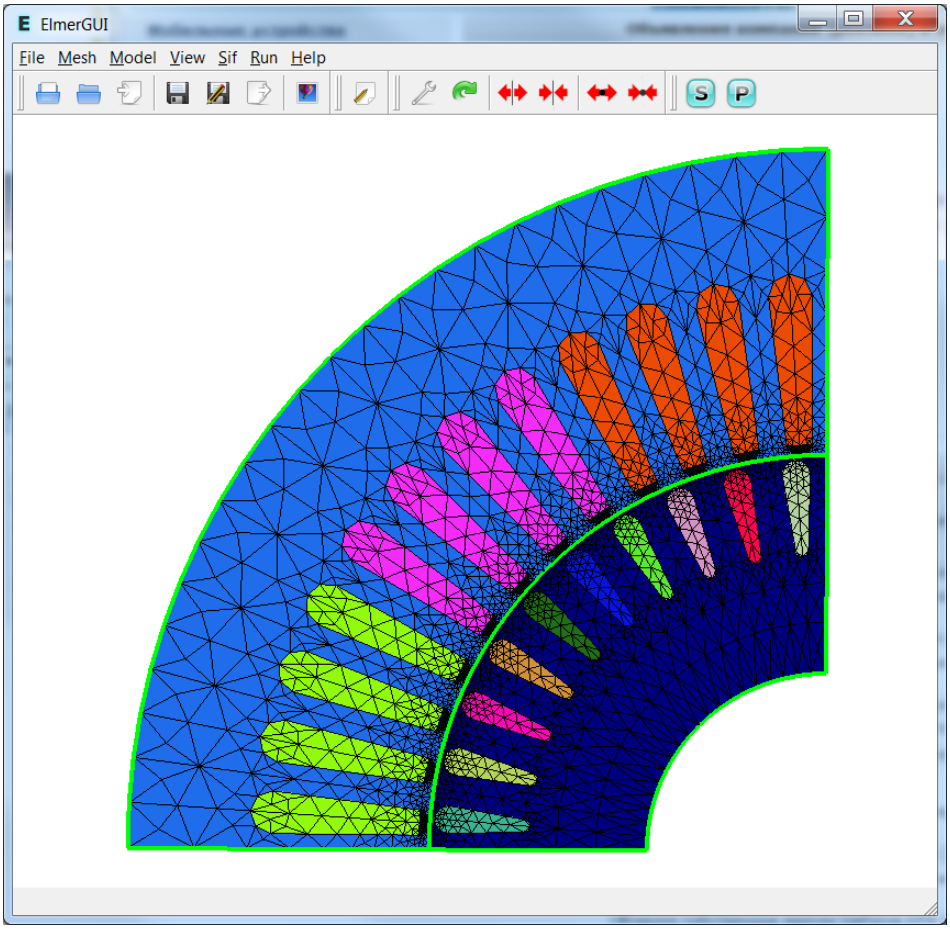


Figure 19. Model mesh loaded to ElmerGUI.

In this example case only the flux density distribution under DC stator excitation will be simulated in order to verify that boundary conditions are set properly in the model. Such a magneto-static simulation does not require modelling of the rotor circuit – the rotor bars can be modelled as air due to constant non-changing stator excitation.

4.1 Structure of the SIF

Folder 'im' now should contain 6 files; this is a result of uniting rotor and stator meshes in the ElmerGrid using option `-names`. This option adds files `mesh.names` and `entities.sif` to the folder. Other files in the folder contain mesh information in the Elmer format. File `mesh.names`

```
! ----- names for bodies -----
$ U_plus = 1
$ W_minus = 2
$ V_plus = 3
$ StatorIron = 4
$ StatorWedges = 5
$ StatorAirgap = 6
$ Bar7 = 7
$ Bar8 = 8
$ Bar9 = 9
$ Bar10 = 10
$ RotorIron = 11
$ RotorAirgap = 12
$ Bar1 = 13
$ Bar2 = 14
$ Bar3 = 15
$ Bar4 = 16
$ Bar5 = 17
$ Bar6 = 18
! ----- names for boundaries -----
$ OuterStator = 1
$ StatorRight = 2
$ StatorLeft = 3
$ Sliding_Stator = 4
$ Sliding_Rotor = 5
$ InnerRotor = 6
$ RotorRight = 7
$ RotorLeft = 8
```

keeps track of numbering of physical mesh entities (volumes, bodies, boundaries) and their names. These names are the same names which were used in GMSH to define physical surfaces and boundaries. The same names are also used in the skeleton file `entities.sif`, where Body and Boundary Condition sections are already prefilled with the corresponding physical entities.

```
!------ Skeleton for body section -----
Body 1
  Name = U_plus
End
Body 2
  Name = W_minus
End
Body 3
  Name = V_plus
End
Body 4
  Name = StatorIron
End
Body 5
  Name = StatorWedges
End
Body 6
  Name = StatorAirgap
End
Body 7
  Name = Bar7
End
Body 8
  Name = Bar8
End
Body 9
  Name = Bar9
End
Body 10
  Name = Bar10
```

```
End
Body 11
  Name = RotorIron
End
Body 12
  Name = RotorAirgap
End
Body 13
  Name = Bar1
End
Body 14
  Name = Bar2
End
Body 15
  Name = Bar3
End
Body 16
  Name = Bar4
End
Body 17
  Name = Bar5
End
Body 18
  Name = Bar6
End
!------ Skeleton for boundary section -----
Boundary Condition 1
  Name = OuterStator
End
Boundary Condition 2
  Name = StatorRight
End
Boundary Condition 3
  Name = StatorLeft
End
Boundary Condition 4
  Name = Sliding_Stator
End
Boundary Condition 5
  Name = Sliding_Rotor
End
Boundary Condition 6
  Name = InnerRotor
End
Boundary Condition 7
  Name = RotorRight
End
Boundary Condition 8
  Name = RotorLeft
End
```

Let us copy this file to the parent directory as `model.sif`, and augment it with other sections required for a simplest magneto-static simulation.

4.2 Header and Simulation sections

Let us add next lines to the beginning of the model.sif

```
Header
  Mesh DB "im"
  Results Directory "results"
  Include Path "materials"
End

Simulation
  Max Output Level = 3
  Coordinate System = Cartesian
  Coordinate Scaling = 0.001

  Steady State Max Iterations = 1
  Simulation Type = Steady

!  Mesh Levels = 2
!  Mesh Keep = 1
```

```
  Use Mesh Names = Logical True
End
```

Header section defines directories with which model works. Mesh is located in the directory `im`. Directory `results` will be used to store the results of the simulation. Additional folder `materials` will be used by the ElmerSolver to look for additional files and parameters. This folder now contains description of the BH curves for 2 materials.

Next section is simulation section dedicated to describe general simulation parameters which are not specific to a particular involved physical model.

`Max Output Level` defines the verbosity of the ElmerSolver, determines how much information is printed by each solver to the console.

`Coordinate Scaling` is required in order to scale mesh units (mm) to SI units (m) utilized by Elmer.

`Simulation Type` selects the type of simulation among transient (time dependent) and steady. There are other possible simulation types as well.

`Use Mesh Names` enables usage of labels of physical entities to address separate bodies within the SIF.

`Mesh Levels` and `Mesh Keep` are options used for mesh refinement. They are commented using **!** and are not seen by the ElmerSolver. If uncommented, the mesh density will be increased automatically without the need for external re-meshing.

4.3 Material and Body Force sections

In order to describe the material properties of the regions, a separate Material section is created for each material. To perform simple simulations, only 2 materials are required – air and electrical steel.

The materials are indexed. These indexes will be used in the bodies sections to assign materials to the physical bodies.

```
! Air
Material 1
  Relative Permeability = 1
End

! lamination steel
Material 2
  Name = "Iron"
  INCLUDE el_steel_M800_65A
End
```

Command `INCLUDE` inserts in the SIF contents of an external file. In this case this is the file with the properties of M800-65A laminated electrical steel located in the folder `materials`. This file contains BH curve information required for the nonlinear material modelling:

```
  Electric Conductivity = 0
!  (name is H-B, but values are in B-H format)
  H-B Curve = Variable coupled iter
    Real Monotone Cubic
      0       0
      0.10    74.7
      0.20    97.5
```

```
  0.30    110
  0.40    120
  0.50    130
  0.60    140
  0.70    150
  0.80    162
  0.90    175
  1.00    190
  1.10    208
  1.20    227
  1.30    265
  1.40    366
  1.50    633
  1.60    1490
  1.70    3670
  1.80   7420
  1.90   13000
  2.0    21000
  2.1    34000
  2.2    55000
  2.3    88000
  2.4    140000
  2.5    220000
  2.6    349000
  2.7    550000
  2.8    860000
  2.9    1350000
  3.0     2200000
  4.0     60000000
End
```

Next lines describe external forces acting on the mesh bodies:

```
Body Force 1
  Mesh Rotate 3 = Real MATC "10"
End

! for U+
Body Force 2
  Current Density = 3e6
End

! for V+
Body Force 3
  Current Density = -1.5e6
End

! for W-
Body Force 4
  Current Density = 1.5e6
End
```

`Body Force 1` is used to verify that boundary between stator and rotor is resolved correctly. It contains a command to rotate by 10 degrees along axis Z (3) all the affected bodies. All the rotor bodies will be affected by this Body Force.

Body Forces 2, 3, and 4 are imposing current densities in the slots of 3 phases. The phase peak current is or 14.7 A rms. Slot contains 32 wires and has area of about 1.4E-4 m^2 . The resultant peak current density in the slot is, therefore, about 3E6 A/(m^2).

With this data the Body section can be filled.

```
Body 1
  Name = U_plus
  Equation = 1
  Material = 1
  Body Force = 2
End

Body 2
  Name = W_minus
  Equation = 1
  Material = 1
```

```
  Body Force = 4
End

Body 3
  Name = V_plus
  Equation = 1
  Material = 1
  Body Force = 3
End

Body 4
  Name = StatorIron
  Equation = 1
  Material = 2
End

Body 5
  Name = StatorWedges
  Equation = 1
  Material = 1
End

Body 6
  Name = StatorAirgap
  Equation = 1
  Material = 1
End

Body 7
  Name = Bar7
  Equation = 1
  Material = 1
  Body Force = 1
End
…
```

Each body is assigned its corresponding material and body force. Additionally, `Equation=1` is put into each body to instruct ElmerSolver that solvers within Equation 1 (described below) are active in the body.

4.4 Solvers

Next lines are put to the `model.sif` to define solvers

```
Equation 1 :: Active Solvers(2) = 2 3

! Rotation of the rotor at the beginning of each time step
Solver 1
  Exec Solver = Before simulation
  Equation = MeshDeform
  Procedure = "RigidMeshMapper" "RigidMeshMapper"
End

Solver 2
  Exec Solver = Always
  Equation = MgDyn2D
  Variable = A
  Procedure = "MagnetoDynamics2D" "MagnetoDynamics2D"

  Nonlinear System Convergence Tolerance = 1.0e-4
  Nonlinear System Max Iterations = 30

  Linear System Solver = Direct
  Linear System Iterative Method = MUMPS
End

Solver 3
  Exec Solver = Always
  Equation = CalcFields
  Potential Variable = "A"
  Procedure = "MagnetoDynamics" "MagnetoDynamicsCalcFields"

  Calculate Current Density = Logical True
```

```
  Calculate Magnetic Vector Potential = Logical True

  Linear System Solver = Direct
  Linear System Iterative Method = MUMPS
End

Solver 4
  Exec Solver = after simulation
  Equation = "ResultOutput"
  Procedure = "ResultOutputSolve" "ResultOutputSolver"
  Output File Name = case
  Output Directory = results
  Save Geometry Ids = Logical True
  Vtu Format = Logical True

  Vector Field 1 = "magnetic flux density e"
  Vector Field 2 = "current density e"
  Scalar Field 1 = "a"

  ! for nicer visualization only (interferes with connectivity filter
  ! in ParaView and changes the field variables a bit)
!  Discontinuous Bodies = Logical True
End
```

Equation sections defines which solvers will be active. In this case solver 2 and 3 will compute and change field variables over the mesh.

Solver 1 is used to make rotor mesh spinning. In this simple model it is used to verify that sliding surface and mortar projectors are defined correctly in the model.

Solver 2 is the main solver where magnetic vector potential A is calculated in 2D. Direct linear system solver MUMPS is used here as the problem is 2D and relatively small. Nonlinear system convergence tolerance is set to 1E-4 (the linear system convergence tolerance, in case of iterative solvers, should be always smaller than the non-linear convergence tolerance). If higher accuracy and higher details need to be resolved by the simulation, then this tolerance could be decreased further for higher precision. In the Simulation section lines

```
!  Mesh Levels = 2
!  Mesh Keep = 1
```

could be uncommented in order to increase the mesh density and study the problem with higher resolution (though round sections become faceted).

If MUMPS is not available at your Elmer installation, then all lines with the linear system in solvers 2 and 3 can be replaced by iterative solver

```
!  Linear System Solver = Iterative
!  Linear System Iterative Method = BicgstabL
!  Linear System Symmetric = True
!  Linear System Max Iterations = 200
!  Linear System Preconditioning = ILU2
!  Linear System Convergence Tolerance = 1e-8
!  Linear System Residual Output = 50
```

Different solvers (and preconditioners) have different performance dealing with different problems. From experience e.g. `bicgstabl` can be fast with small 2D electrical machine models; `GCR` can be more robust and fast with larger 2D and 3D models. For more options considering solution of resultant linear and non-linear algebraic systems of equation refer to ElmerSolver Manual [5].

Solver 3 is used to compute flux density and current density fields from the A-field solution provided by the previous solver. Next keywords are used to compute required values

```
Calculate Current Density = Logical True
Calculate Magnetic Vector Potential = Logical True
```

Solver 4 is used to output the resultant vector and scalar fields. Keyword

```
Discontinuous Bodies = Logical True
```

is used to render better looking field distribution within the bodies without discontinuities.

4.5 Boundary conditions

Outer stator and inner rotor boundaries assume Dirichlet boundary condition of zero magnetic flux flowing through the boundary. In terms of MVP this condition results in A = 0 at those boundaries.

```
Boundary Condition 1
  Name = OuterStator
  A = Real 0
End

Boundary Condition 6
  Name = InnerRotor
  A = Real 0
End
```

The sides of the modelled machine sector represented anti-periodic boundary condition (if 2 poles are modelled, then this would be a periodic boundary). Next keywords define this boundary condition for stator and rotor using mortar elements (see [5]):

```
Boundary Condition 2
  Name = StatorRight
  Mortar BC = Integer 3
  Mortar BC Static = Logical True
  Anti Radial Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 3
  Name = StatorLeft
End

Boundary Condition 7
  Name = RotorRight
  Mortar BC = Integer 8
  Mortar BC Static = Logical True
  Anti Radial Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 8
  Name = RotorLeft
End
```

Sliding surface of one modelled pole can be represented by Anti Rotational Projector as it is shown here:

```
Boundary Condition 4
  Name = Sliding_Stator
  Mortar BC = Integer 5
  Anti Rotational Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 5
  Name = Sliding_Rotor
End
```

4.6 Running simulation in series

Now, the sif is fully defined and is ready to be solved. Run ElmerSolver in the console by

```
>ElmerSolver model.sif
```

The output will look like this:

```
ELMER SOLVER (v 8.2) STARTED AT: 2016/10/19 18:29:59
ParCommInit:  Initialize #PEs:           1
MAIN:
MAIN: =============================================================
MAIN: ElmerSolver finite element software, Welcome!
MAIN: This program is free software licensed under (L)GPL
MAIN: Copyright 1st April 1995 - , CSC - IT Center for Science Ltd.
MAIN: Webpage http://www.csc.fi/elmer, Email elmeradm@csc.fi
MAIN: Version: 8.2 (Rev: 59349d6, Compiled: 2016-10-19)
MAIN:  HYPRE library linked in.
MAIN:  MUMPS library linked in.
MAIN: =============================================================
MAIN:
MAIN:
MAIN: -------------------------------------
MAIN: Reading Model: model.sif
Model Input:  Unlisted keyword: [a] in section: [boundary condition 1]
Model Input:  Unlisted keyword: [a] in section: [boundary condition 6]
MAIN:
MAIN: -------------------------------------
MAIN:  Steady state iteration:           1
MAIN: -------------------------------------
MAIN:
ComputeChange: NS (ITER=1) (NRM,RELC): (  39593.595      2.0000000     ) :: mgdyn2d
ComputeChange: NS (ITER=2) (NRM,RELC): (  37584.792     0.52056124E-01 ) :: mgdyn2d
ComputeChange: NS (ITER=3) (NRM,RELC): (  33855.008     0.10441753     ) :: mgdyn2d
ComputeChange: NS (ITER=4) (NRM,RELC): (  31429.451     0.74307346E-01 ) :: mgdyn2d
ComputeChange: NS (ITER=5) (NRM,RELC): (  30608.944     0.26451557E-01 ) :: mgdyn2d
ComputeChange: NS (ITER=6) (NRM,RELC): (  30446.590     0.53182506E-02 ) :: mgdyn2d
ComputeChange: NS (ITER=7) (NRM,RELC): (  30421.016     0.84030851E-03 ) :: mgdyn2d
ComputeChange: NS (ITER=8) (NRM,RELC): (  30434.536     0.44431369E-03 ) :: mgdyn2d
ComputeChange: NS (ITER=9) (NRM,RELC): (  30419.731     0.48656189E-03 ) :: mgdyn2d
ComputeChange: NS (ITER=10) (NRM,RELC): (  30422.990     0.10713186E-03 ) :: mgdyn2d
ComputeChange: NS (ITER=11) (NRM,RELC): (  30423.554     0.18545610E-04 ) :: mgdyn2d
ComputeChange: SS (ITER=1) (NRM,RELC): ( 0.23353012E-01  2.0000000     ) :: mgdyn2d
WARNING:: GetPermittivity: Permittivity not defined in material, defaulting to that of vacuum
ComputeChange: NS (ITER=1) (NRM,RELC): ( 0.85493988      2.0000000     ) :: calcfields
ComputeChange: NS (ITER=2) (NRM,RELC): ( 0.65223768     0.26898250     ) :: calcfields
ComputeChange: NS (ITER=6) (NRM,RELC): ( 0.23353012E-01  2.0000000     ) :: calcfields
ComputeChange: NS (ITER=9) (NRM,RELC): (  547030.49      2.0000000     ) :: calcfields
ComputeChange: SS (ITER=1) (NRM,RELC): (  547030.49      2.0000000     ) :: calcfields
ElmerSolver: *** Elmer Solver: ALL DONE ***
ElmerSolver: The end
SOLVER TOTAL TIME(CPU,REAL):         1.08        1.13
ELMER SOLVER FINISHED AT: 2016/10/19 18:30:00
```

From this listing it is seen that ElmerSolver was run in series. The version of Elmer is 8.2 correspondent to github commit with id 59349d6. HYPRE and MUMPS libraries can be used together with Elmer. Two unlisted keywords were found (this is rather expected behavior as the variable name was used as a keyword in the boundary conditions). Nonlinear system tolerance was reached in 11 iterations for the MVP solver. WARNING about permittivity is also an expected behavior at this point, as the derived fields solver needs this value. Default value of the vacuum is suitable in this case. The whole solution process took 1.13 seconds.

The resultant file case0001.vtu is saved in the folder `results`. The magnetic flux density field and contours of the MVP (flux lines) are visualized using ParaView (some basic examples of ParaView usage can be found in Section 5.4 and also in [7]).

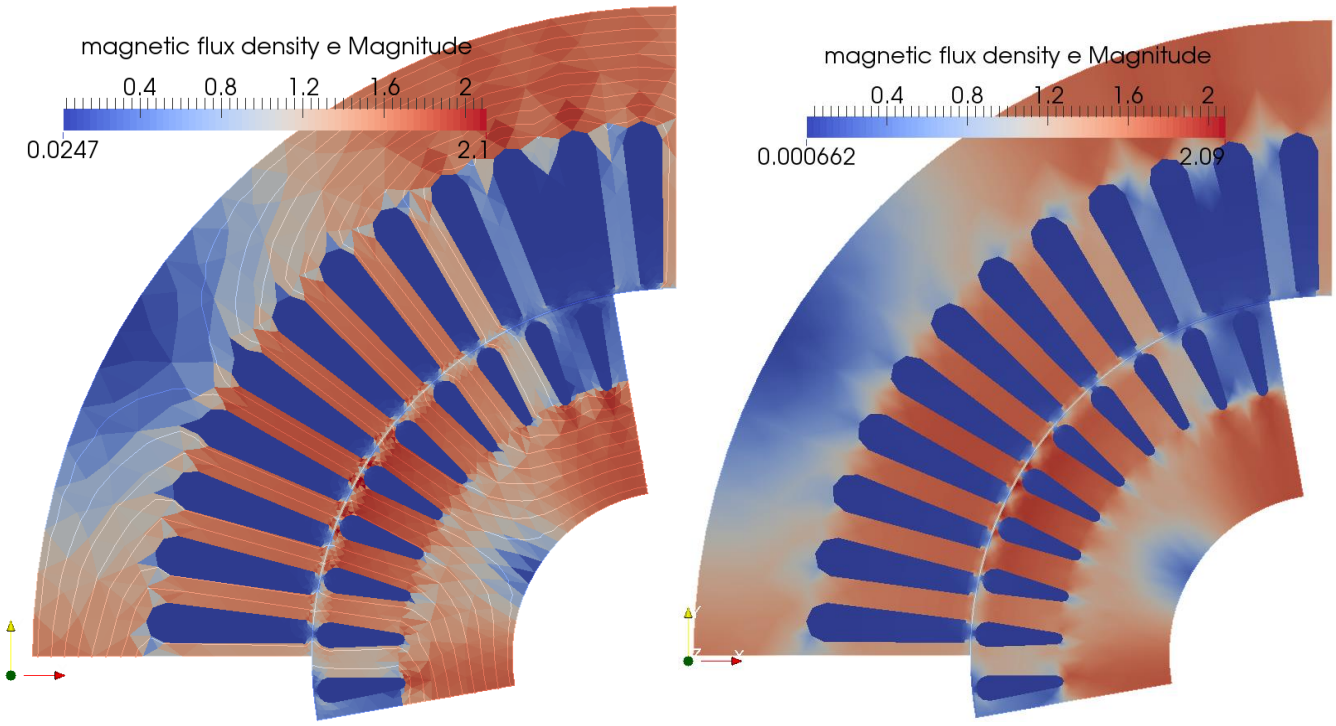


Figure 20. Static solution. Flux density and flux lines are visualized using ParaView.

Figure 20 shows the magnetic flux distribution in one pole of the machine. The flux lines are continuous over the sliding surface and tangential at the inner and outer boundaries. Periodicity and symmetry are also resolved correctly. The flux density values are within expected range. Therefore, it can be concluded, that the model is set correctly.

The flux density is somewhat faceted as the flux density is averaged over element in ParaView for visualization. Solving with parameter in the ResultOutput solver section

```
Discontinuous Bodies = Logical True
```

produces a smoothed field visualization. Notice that field values are slightly distorted by this filter. The resultant field after `Discontinuous bodies = True` should not be used for any further calculations, but rather only for nicer visualizations.

4.7 Running simulation in parallel

This relatively small problem can be also solved in parallel. At first the mesh must be decomposed to several sub-meshes so that each sub-mesh could be processed by its own processor core.

ElmerGrid can convert Elmer mesh files to parallel Elmer mesh files by

```
>ElmerGrid 2 2 im -partdual -metis 4 3 -connect 2 3 4 5 7 8
```

Here

- `2` and `2` are the input and output formats -- in this case the Elmer mesh format
- `im` is the name of the folder containing mesh files
- `-partdual` is needed for parallelization of the mesh,

- `-metis` is the partitioning utility; parameter **4** is the number of partitions; parameter **3** is the algorithm of the Metis library (see Metis documentation)
- `-connect` makes boundaries (with ids `2 3 4 5 7 8`) to be stored within one partition (sub-mesh). This option is required by mortar boundary conditions, such as sliding surface and periodicity, as those can be correctly computed only being within 1 common partition. The numerical parameters here correspond to the boundary ids in the mesh (in case of named boundaries those could be checked in the file `mesh.names`).

The output of the ElmerGrid contains next lines:

```
Distribution of elements, nodes and shared nodes
     partition  elements   nodes      shared
     1          1100       673        193
     2          1256       635        79
     3          1332       669        78
     4          1292       656        80
Average number of elements in partition 6.582e+02
Maximum deviation in ownership 38
Average deviation in ownership 1.482e+01
Average relative deviation 2.25 %
Checking for problematic sharings
Optimizing sharing for 4 partitions
There shouldn't be any problematic sharings, knock, knock...
Partitioning was not altered

Elmergrid saving data with method 2:
-------------------------------------
Saving Elmer mesh in partitioned format
Number of boundary nodes at the boundary: 282
Created mesh directory: im
Created subdirectory: partitioning.4
Saving mesh in parallel ElmerSolver format to directory im/partitioning.4.
Nodes belong to 3 partitions in maximum
Saving mesh for 4 partitions
   part  elements   nodes      shared   bc elems orphan   indirect
   1     1100       673        193      242      0        0
   2     1256       635        79       12       0        0
   3     1332       669        78       12       0        0
   4     1292       656        80       16       0        0
Nodes needed in maximum 2 boundary elements
-----------------------------------------------------------------------------------------------
   ave   1245.0     658.2      107.5    70.5     0.0      0.0
Writing of partitioned mesh finished

Thank you for using Elmergrid!
Send bug reports and feature wishes to elmeradm@csc.fi
```

From this listing it is seen that 4 partitions are created for 4 parallel processes. Each partition has about 1245 elements to process. The sharing of elements is almost even – this optimizes the workload by the processor cores. One partition has 242 bc elements – this is the partition obtained from uniting of the boundaries `2 3 4 5 7 8` by `-connect` keyword.

Partitioning can be visually studied in ParaView by applying Connectivity filter as it is shown in Figure 21. Regions 0 and 1 belong to the same partition obtained by -connect command. Also regions 5 and 4 also belong to the same partition as it can be understood comparing the sizes of the partitions. The reason why those regions are belonging to the same partition is in mesh connectivity divided by the moving air gap.

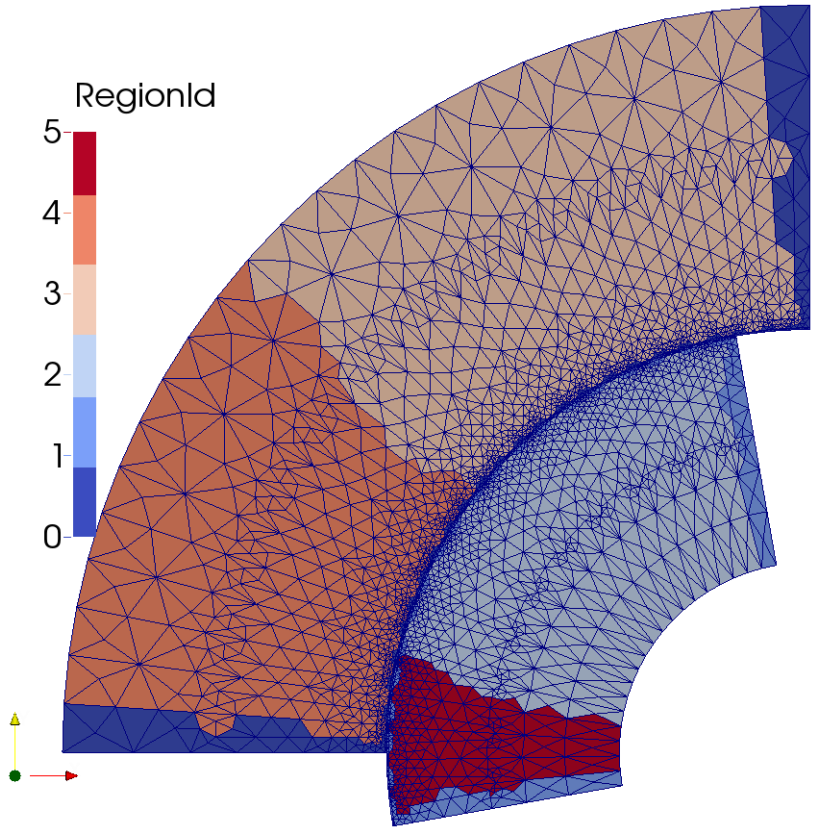


Figure 21. Solved VTU mesh with Connectivity filter applied.

Solver of the algebraic system must also be capable of dealing with parallel solutions. Most of the Elmer's iterative solvers can handle parallel execution. Direct solver MUMPS also can be used effectively in small problems.

The solution can be executed using `mpirun` command:

```
>mpirun -np 4 ElmerSolver_mpi
```

Also working directory must contain file `ELMERSOLVER_STARTINFO` in which only the name of the SIF file should be written. This is required to provide of each of the parallel processes `ElmerSolver_mpi` with information about SIF file – in this case `model.sif`.

The following listing shows that 4 parallel processes were launched and completed.

```
ELMER SOLVER (v 8.2) STARTED AT: 2016/10/21 10:53:44
ELMER SOLVER (v 8.2) STARTED AT: 2016/10/21 10:53:44
ELMER SOLVER (v 8.2) STARTED AT: 2016/10/21 10:53:44
ELMER SOLVER (v 8.2) STARTED AT: 2016/10/21 10:53:44
ParCommInit:  Initialize #PEs:            4
ParCommInit:  Initialize #PEs:            4
ParCommInit:  Initialize #PEs:            4
MAIN:
MAIN: =============================================================
MAIN: ElmerSolver finite element software, Welcome!
MAIN: This program is free software licensed under (L)GPL
MAIN: Copyright 1st April 1995 - , CSC - IT Center for Science Ltd.
MAIN: Webpage http://www.csc.fi/elmer, Email elmeradm@csc.fi
MAIN: Version: 8.2 (Rev: 59349d6, Compiled: 2016-10-19)
MAIN:  Running in parallel using 4 tasks.
MAIN:  HYPRE library linked in.
MAIN:  MUMPS library linked in.
MAIN: =============================================================
ParCommInit:  Initialize #PEs:            4
MAIN:
MAIN:
```

```
MAIN: -------------------------------------
MAIN: Reading Model: model.sif
Model Input:  Unlisted keyword: [a] in section: [boundary condition 1]
Model Input:  Unlisted keyword: [a] in section: [boundary condition 6]
MAIN:
MAIN: -------------------------------------
MAIN:  Steady state iteration:            1
MAIN: -------------------------------------
MAIN:
ComputeChange: NS (ITER=1) (NRM,RELC): (  36832.310      2.0000000     ) :: mgdyn2d
ComputeChange: NS (ITER=2) (NRM,RELC): (  35063.727     0.49198342E-01 ) :: mgdyn2d
ComputeChange: NS (ITER=3) (NRM,RELC): (  31472.314     0.10795392     ) :: mgdyn2d
ComputeChange: NS (ITER=4) (NRM,RELC): (  29237.241     0.73631676E-01 ) :: mgdyn2d
ComputeChange: NS (ITER=5) (NRM,RELC): (  28503.019     0.25431904E-01 ) :: mgdyn2d
ComputeChange: NS (ITER=6) (NRM,RELC): (  28333.743     0.59565512E-02 ) :: mgdyn2d
ComputeChange: NS (ITER=7) (NRM,RELC): (  28293.452     0.14230450E-02 ) :: mgdyn2d
ComputeChange: NS (ITER=8) (NRM,RELC): (  28314.808     0.75454761E-03 ) :: mgdyn2d
ComputeChange: NS (ITER=9) (NRM,RELC): (  28302.230     0.44433735E-03 ) :: mgdyn2d
ComputeChange: NS (ITER=10) (NRM,RELC): (  28304.266     0.71955576E-04 ) :: mgdyn2d
ComputeChange: SS (ITER=1) (NRM,RELC): ( 0.23428722E-01  2.0000000     ) :: mgdyn2d
WARNING:: GetPermittivity: Permittivity not defined in material, defaulting to that of vacuum
WARNING:: GetPermittivity: Permittivity not defined in material, defaulting to that of vacuum
WARNING:: GetPermittivity: Permittivity not defined in material, defaulting to that of vacuum
WARNING:: GetPermittivity: Permittivity not defined in material, defaulting to that of vacuum
ComputeChange: NS (ITER=1) (NRM,RELC): ( 0.85118323      2.0000000     ) :: calcfields
ComputeChange: NS (ITER=2) (NRM,RELC): ( 0.66767901     0.24163379     ) :: calcfields
ComputeChange: NS (ITER=6) (NRM,RELC): ( 0.23428722E-01  2.0000000     ) :: calcfields
ComputeChange: NS (ITER=9) (NRM,RELC): (  520996.30      2.0000000     ) :: calcfields
ComputeChange: SS (ITER=1) (NRM,RELC): (  520996.30      2.0000000     ) :: calcfields
ElmerSolver: *** Elmer Solver: ALL DONE ***
ElmerSolver: The end
SOLVER TOTAL TIME(CPU,REAL):         0.45        1.49
ELMER SOLVER FINISHED AT: 2016/10/21 10:53:45
```

The CPU time has decreased significantly in comparison with serial case. However, the REAL execution time has increased due to additional time spent on launching several processes and on communications between processes. The convergence in parallel solution is typically somewhat lower than in serial case, however it is compensated by the time savings due to smaller problem size solved by each processor core.

In this case no real time savings were obtained due to very small problem size and parallelization overhead. However, for higher mesh density or for bigger problems the parallelization can bring significant time savings.

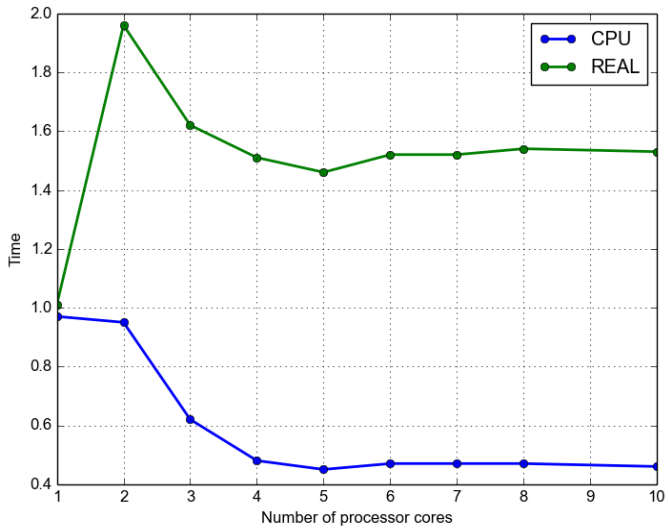


Figure 22. Dependence of execution time in seconds from number of parallel processes. Adverse influence of parallelization and execution overhead on small problems is clearly seen.

In Figure 22 it is seen that for this particular small steady-state model executed on a particular PC the parallelization does not make sense as the parallelization and execution overhead is much higher than the CPU savings. It is also can be seen that increase in number of processor cores above 4 is not feasible as the processing time does not decrease anymore. This is due to the fixed minimal size of the partition obtained by connecting boundaries. This partition cannot be any smaller. Even though the other partitions are smaller and are solved faster with higher number of processes, the parallel MPI-based FEM algorithm waits all the processes to complete before next linear system solution iteration. This makes already finished processes working with smaller partitions to stay idle while computations over larger partitions finish.

5. Time Transient Electro-Magnetic Simulation

In this section a description of the non-linear time-transient simulation with circuit equations is given. The model is augmented with circuit definitions. The model is driven now by voltage sources. Complete `transient.sif` file can be found in the Appendix.

The motor is simulated at constant rotor rated speed. The goal is to analyze rated motor performance.

5.1 Transient Simulation

The Simulation section is changed in order to facilitate time stepping:

```
Simulation Type = Transient
Timestepping Method = BDF
BDF Order = 2

Output Intervals = 1
Timestep Sizes = $ 1/f/15
Timestep Intervals = $ 25*15
```

`Simulation Type` is changed to `Transient` and `Timestepping Method` is `BDF` (see [5]). `Timestep Size` is chosen as `1/f/15` which means that there are 15 time steps over one electrical period. This value is selected lower than the recommended minimal 20 time steps over one sinusoidal period in order to decrease simulation time. When all the parameters of the simulation are set and tested, it is recommended to increase number of steps over the period to 40. And if also slotting harmonics need to be resolved, then practical value starts at 150. Here we select 15 as the most interested parameter is average torque, which could be precisely enough simulated also neglecting the slotting harmonics. And there are 25 electrical periods simulated as it is determined by `Timestep Intervals`.

Variable `f` is defined in the file `transient_params.dat`. The contents of that dat file are included in the sif by the command `include transient_params.dat` in the very beginning of the sif.

The materials section is updated with two more materials for stator copper windings and for aluminum cage rotor winding.

```
! Aluminum
Material 3
  Relative Permeability = 1
  Electric Conductivity = 24e6 ! [1/(Ohm*m)] hot rotor
End

! Copper
Material 4
  Relative Permeability = 1
  Electric Conductivity = 48e6
End
```

Body force section is updated to take into account rotor motion.

```
Body Force 7
  Mesh Rotate 3 = Variable time, timestep size
    Real MATC "180/pi*f*(1-slip)*2*pi/pp*(tx(0)-tx(1))"
End
```

The `slip` variable is defined in the `transient_params.dat` file as well.

5.2 Stator Circuit Definition

The electrical circuit coupling has been added to Elmer after version 8.2. Discretized circuit equations are attached to the main matrix of the field solution [8] by the CircuitsAndDynamics solver using keyword in the MagnetoDynamics2D solver section:

```
Export Lagrange Multiplier = Logical True
```

Two additional solvers are added to initialize and process the circuit degrees of freedom:

```
Solver 2
  Exec Solver = Always
  Equation = Circuits
  Variable = X
  Procedure = "CircuitsAndDynamics" "CircuitsAndDynamics"
  No Matrix = Logical True
End
 . . .
Solver 5
  Exec Solver = Always
  Equation = CircOutput
  Procedure = "CircuitsAndDynamics" "CircuitsOutput"
End
```

In the case of 2D Cartesian simulation circuit solver also needs information about axial length of the simulated domain in order to calculate induced voltages in the conductors. This information is included by adding keyword to the Simulation section:

```
Circuit Model Depth = Real $ l
```

where variable `l` is defined in the included file with parameters.

Defining proper coupling of the multi-phase electrical circuits with the electromagnetic FEM domain can be one of the most challenging parts of the simulation process. Here, it is very easy to mix the sign conventions for the current and flux directions. In order to reduce the amount of mistakes it is advised to model this coupling step by step:

- *Define circuit equations for one phase if possible (delta connection and unbalanced systems should be modelled as a whole, e.g. eccentricity, PWM supply, faults simulations)*
- *Debug it using simple voltage step response*

  ```
  U_u = Variable time
        Real MATC "if(tx>=0.001) {200} else {0}"
  ```

  Model other phases and rotor excitation as air. Make sure that the flux lines in the magnetic domain are physical and as expected.
- *Make sure, that the step response converges to the value I=U/R. Make sure that the time constant derived from the current response from the voltage step excitation corresponds approximately to the phase self-inductance obtained from the analytical design (tau = L/R).*
- *When the single phase circuit is debugged, use those circuit equations as a model for other phases.*
- *Run stator circuit simulation with sinusoidal supply. Make sure that the multiphase system creates rotating magnetic field in the FEM domain with expected poles number. Make sure that the direction of the magnetic field rotation corresponds to the direction of the rotor motion.*

The sources of the electrical circuits must be defined in the Body Force 1. The nominal line-to-line voltage is 400 V. The 3-phase star winding connection is assumed to be symmetrical. Hence, the peak phase voltages are defined as:

```
Body Force 1
  Name = "Circuit"

  ! U-phase voltage 400/sqrt(3) Vrms at 0 degrees
  U_u = Variable time
    Real MATC "400*sqrt(2)/sqrt(3)*sin(tx(0)*2*pi*f)"

  ! V-phase voltage 400/sqrt(3) Vrms at -120 degrees
  U_v = Variable time
    Real MATC "400*sqrt(2)/sqrt(3)*sin(tx(0)*2*pi*f-2*pi/3)"

  ! W-phase voltage 400/sqrt(3) Vrms at +120 degrees
  U_w = Variable time
    Real MATC "400*sqrt(2)/sqrt(3)*sin(tx(0)*2*pi*f+2*pi/3)"
End
```

The FEM bodies are associated with the circuits using new Component sections:

```
  Component 11
    Name = String Uplus
    Body = Integer 11
    Coil Type = String Stranded
    Number of Turns = Real $ Nph/2
    Resistance = Real $ Rs
  End

  Component 12
    Name = String Vplus
    Body = Integer 13
    Coil Type = String Stranded
    Number of Turns = Real $ Nph/2
    Resistance = Real $ Rs
  End

  Component 13
    Name = String Wminus
    Body = Integer 12
    Coil Type = String Stranded
    Number of Turns = Real $ Nph/2
    Resistance = Real $ Rs
  End
```

The Component numbering starts from 11, as first 10 components are reserved for the rotor bars. Each component has its own associated Body index. All the components are stranded coils which means that eddy-current effect is neglected in the coil wires, and in this body the current flows only along machine axis.

Each component has its resistance and number of turns. The resistance of the modelled domain is equal to the total phase resistance as phase is constructed as double star and only one half of a one star-branch is modelled (see Figure 8).

The stator circuit equations are defined in the `transient_params.dat`. There are 3 identical circuits representing 3 phases. Here only circuit equations for phase U are described. Complete file with circuits can be found in the Appendix.

The modelled circuit diagram is shown in the next figure. It consists of the source voltage, FEM component of the winding coil, and the end winding inductance. It results in an electrical circuit with 1 contour and 3 nodes between components.

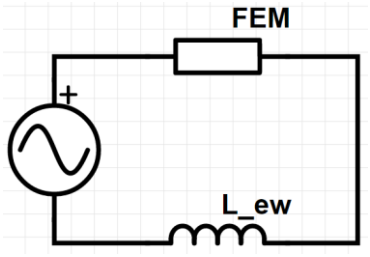


Figure 23. Modelled phase circuit diagram.

The circuit definition (in the file `transient_params.dat`, which has been included in the beginning of the SIF) starts with the declaration of the number of the modelled circuits (obligatory variable):

```
$ Circuits = 4
```

Here 4 circuits are modelled – 3 stator circuits and 1 rotor circuit. This is done in assumption that the 3-phase star winding connection is symmetrical and supplied by symmetrical sinusoidal voltages. If it is not so (e.g. PWM-supply, harmonics are presented in the supply, a magnetic disbalance, rotor eccentricity), then the stator circuit should not be divided into 3 single-phase separate circuits, but rather simulated as a more complex 3-phase circuit.

If the model is built from a scratch, it is advisable to verify the model after each step: first simulate just stator circuits and simulate rotor bars as air and verify that the stator-created field is as expected in order to avoid complex debugging of the complete model.

Then goes initialization of the variables and matrices in the `transient_params.dat` file:

```
!-----------------------------------------------------------
! Phase U_plus
!-----------------------------------------------------------

! init matrices phase U
! init matrices of Ax' + Bx = Source
$ C.1.variables = 6         ! I and V for each circuit element
$ C.1.perm = zeros(6) ! Permutation vector (possibility to reduce bandwidth of the matrix)
$ C.1.A = zeros(6, 6)       ! Mass matrix
$ C.1.B = zeros(6, 6)       ! Stiffness matrix
$ C.1.Mre = zeros(6, 6)     ! Can be used in harmonic case, zeros for transient
$ C.1.Mim = zeros(6, 6)     ! Can be used in harmonic case, zeros for transient
```

The circuit is described by a first-order linear differential system of equations with constant coefficients Ax’ + Bx = F(t), where

- x is the vector of circuit variables
- A is the coefficient matrix of the time-derivative component (mass matrix)
- B is the proportional coefficient matrix (stiffness matrix)
- F is the source part which is set in the Body Force 1 section (force vector)

A and B matrices at the moment are filled by hand using Kirchhoff I and Kirchhoff II circuit laws, and element equations. Ax’ + Bx = F(t) is then added to the main field system, and both equations are solved together as a single system.

The modelled phase circuit has 3 elements connected in series (see Figure 23). For each element 2 circuit variables are defined – current and voltage. 1 and 2 for voltage source; 3 and 4 for FEM connected winding; 5 and 6 for the end winding outside the FEM domain.

```
! define circuit variables (x)
$ C.1.name.1 = "i_su"
$ C.1.name.2 = "v_su"
```

```
$ C.1.name.3 = "i_component(11)"     !the number of the field component
$ C.1.name.4 = "v_component(11)"          !the number of the field component
$ C.1.name.5 = "i_ewu"
$ C.1.name.6 = "v_ewu"
```

The matrices are filled considering each circuit law and element separately. The source term must be defined in the `Body Force 1` sif section.

```
! Define sources:
!------------------
$ C.1.B(0,1) = 1
$ C.1.source.1 = "U_u"
```

KCL (Kirchhoff's current law, Kirchhoff's point rule) – sum of the currents in a node is zero. Multiplier 0.5 is used because only 1 star from the parallel-star winding is modelled. This way variable `i_su` will contain nominal motor current in the output scalars file.

```
! Circuit equations:
!-------------------------
! Kirchoff current law
$ C.1.B(2,0) = -0.5
$ C.1.B(2,2) = 1
$ C.1.B(4,0) = -0.5
$ C.1.B(4,4) = 1
```

KVL (Kirchhoff's voltage law, Kirchhoff's loop rule) – sum of the voltages in a loop/contour is zero.

```
! Kirchoff voltage law
$ C.1.B(1,1) = -0.5
$ C.1.B(1,3) = 1
$ C.1.B(1,5) = 1
```

Elemental equations consider resistive and inductive voltage drops. Factor ¼ comes from the Figure 8.

```
!Elemental equations
$ C.1.A(5,4) = L_ew/4
$ C.1.B(5,5) = -1
```

Here, an example of a star phase connection is given. It is possible to construct more complicated phase interconnections using Kirchhoff rules and elemental equations. See, e.g. how equations are derived for the cage winding in the Appendix.

5.3 Cage Winding

Figure 24 shows the equivalent circuit of one pole for the rotor. The antiperiodic boundary conditions are set by cross-connecting boundary bars 1 and 10. Here, R_{er} and L_{er} are resistance and inductance of the rotor end rings sections between two neighboring bars.

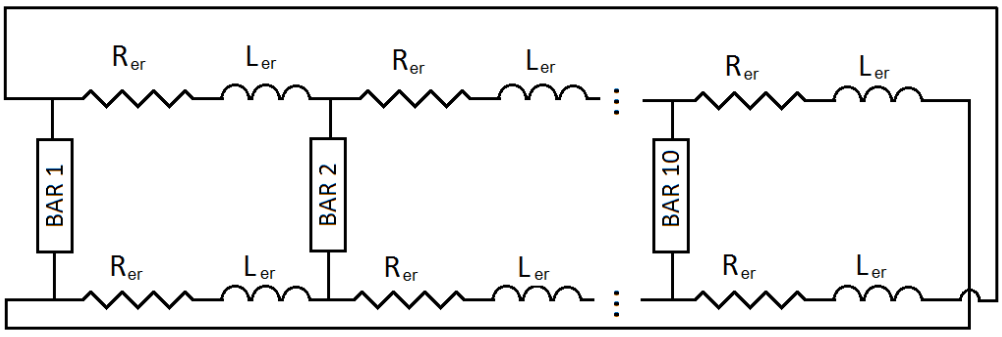


Figure 24. Single pole rotor cage winding equivalent circuit.

The number of elements in this rotor circuit is 50. The R_{er} and L_{er} can be united within one element, this gives 30 elements, which results in 60 circuit variables. This is too much for building the circuit by hand as the probability for a mistake increases significantly. Therefore, a special python script `cage_generator.py` has been written for generating circuit equation for cage windings. The input parameters include number of bars, symmetry conditions, number of slices (used for multi-slice 2.5D models), and type of conductors. The listing can be found in the Appendix.

The result of running `cage_generator.py` is a file `cage.definitions` with rotor circuit definitions.

The parameters R_{er} and L_{er} can be estimated analytically using e.g. [9] and Figure 4–Figure 6. The values for the hot rotor are correspondingly 1.6e-6 Ohm and 6.3e-9 H. They are defined in the `transient_params.dat` file as `R_er` and `L_er`. However, the formula does not take into account proximity of the rotor and stator stacks – due to this, the estimate for the end ring section inductance is increased to 12e-9 H.

5.4 Simulation Results

The model was run using

```
>ElmerSolver transient.sif
```

The solving process ended within 506 seconds. The field solutions for 375 timesteps are saved in `vtu` output files in the folder `results`. Those files can be viewed in e.g. ParaView by running the app and opening those files using standard open dialog. Then Apply button should be pressed.

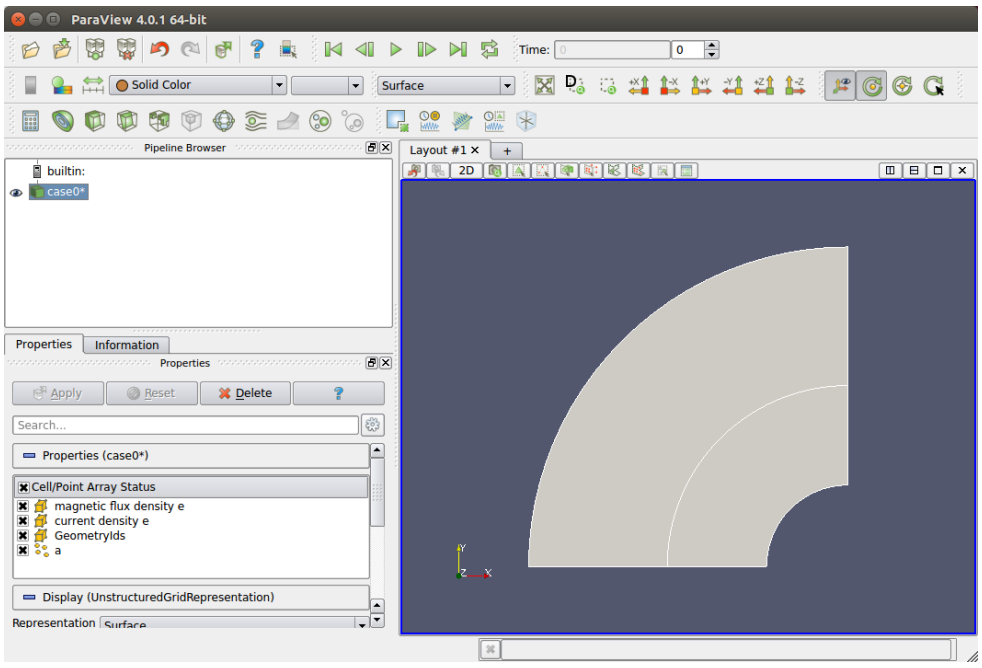


Figure 25. Paraview.

From the dropdown menu (where “Solid Color” is labeled in Figure 25) one can select which field solution to show. The green arrows at the top of the window allow to select the time

step. Next figures demonstrate magnetic vector potential A, flux density B, and current density J – the fields saved by the Solver 6 (ResultOutputSolver). Refer to [6] to check what fields are available to save in a specific Elmer model solver. Some simple examples of post-processing with ParaView (e.g. flux density in the airgap) can be found also in [7].

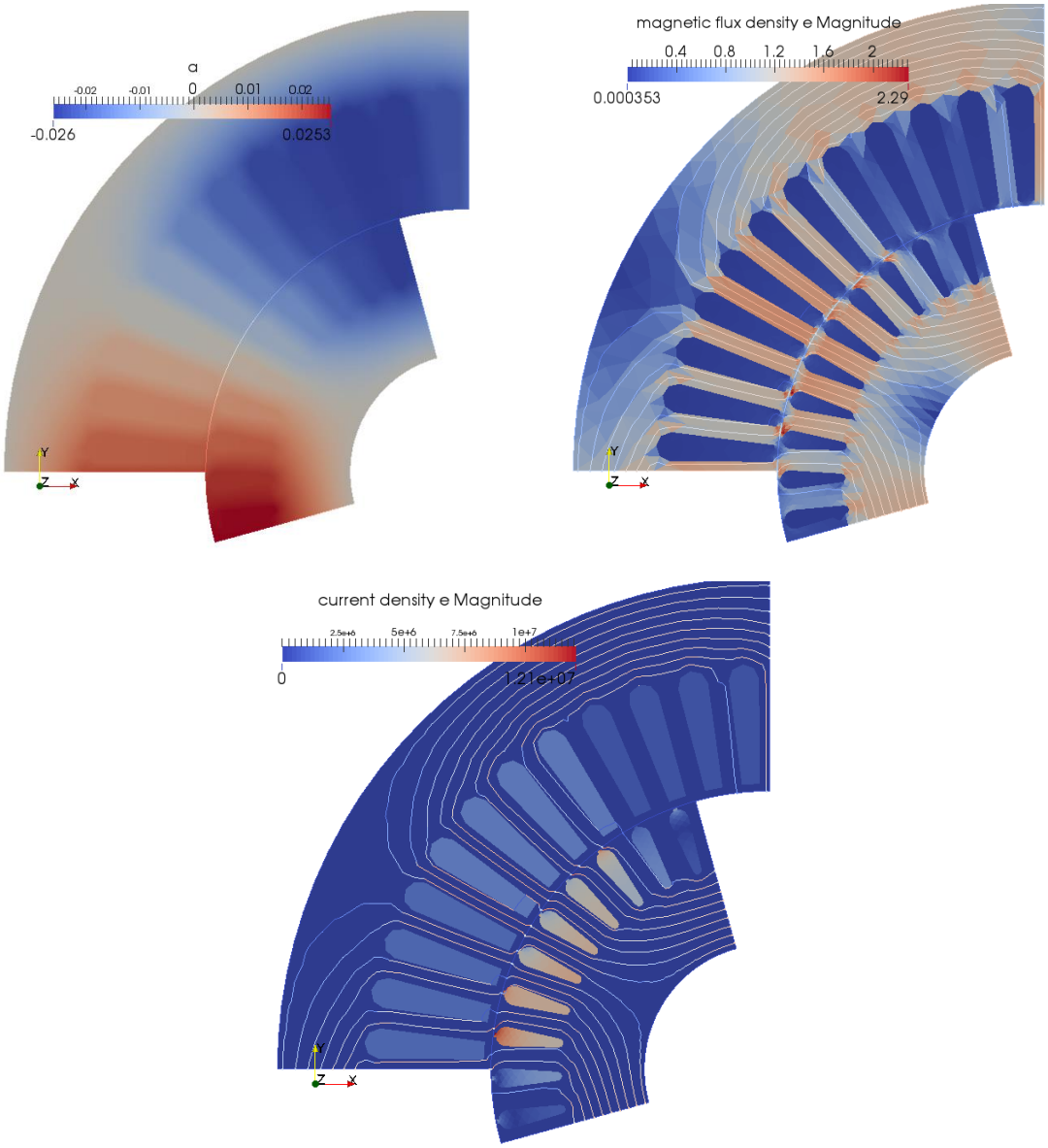


Figure 26. Field solutions of magnetic vector potential, flux density with equipotential lines, and current density visualized with ParaView.

The scalar data (such as circuit variables, torque, timesteps) is saved by Solver 7 (SaveScalars) in the file `transient_results.dat` in the folder results. The format is ASCII txt format. Each time-step is saved in its own row. The names of the saved scalar variables are listed in the file `transient_results.dat.names`. The scalar results can be visualized with Matlab, Octave, Excell, or any other visualization toolkit able to import numerical data from ASCII format. Here, python script `post.py` is listed in the Appendix to visualize the scalars.

The scalar plots of the transient simulation from zero initial conditions are presented in the next figures.

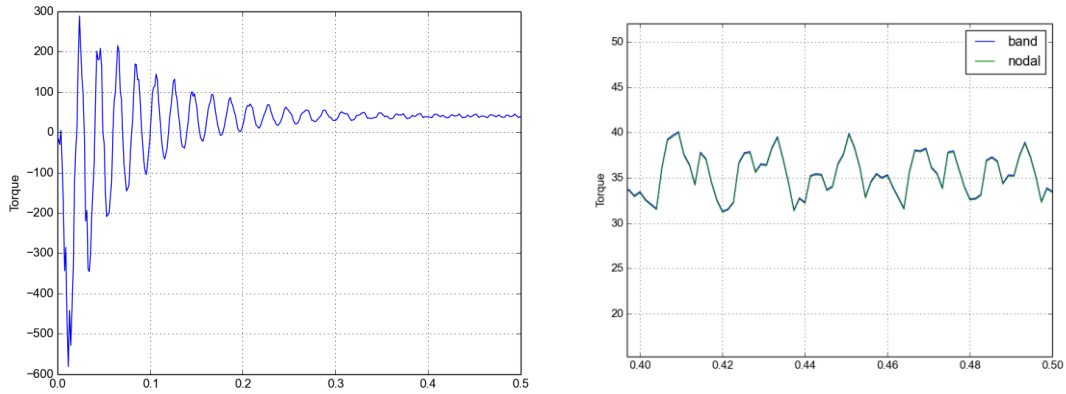


Figure 27. Torque transient.

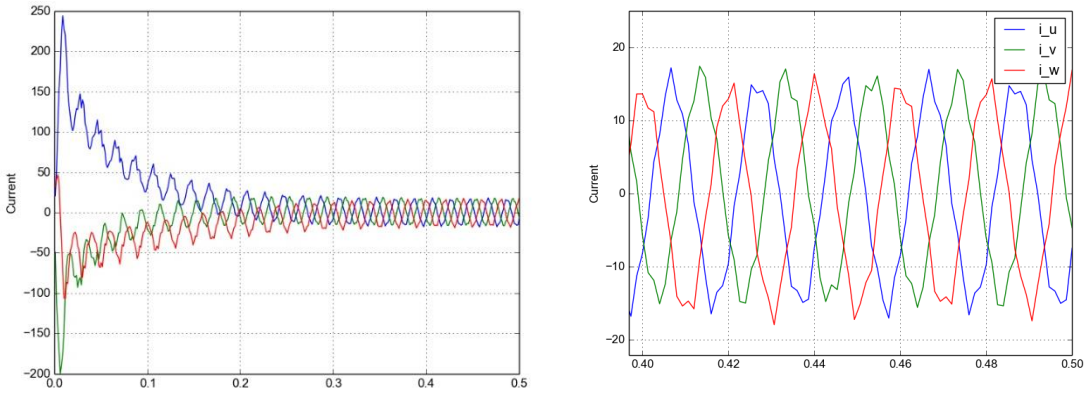


Figure 28. Stator currents transient.

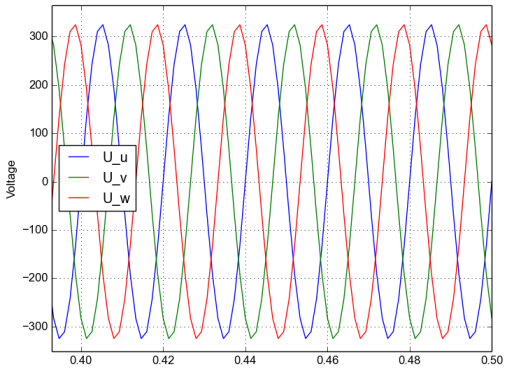


Figure 29. Stator voltages.

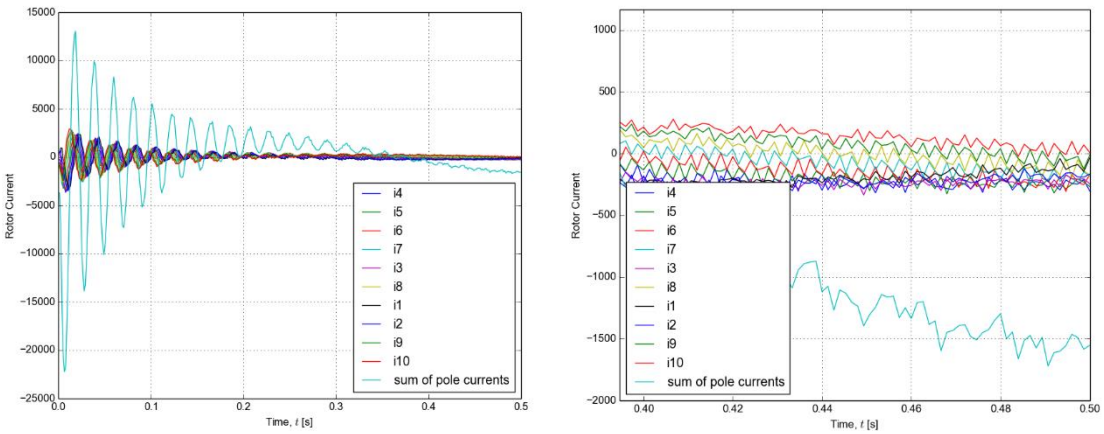


Figure 30. Stator voltages.

The curves are very noisy as the time resolution (15 steps per electrical period) is adequate only to estimate average values. The slotting effect is totally ignored. Having 20 slots in the rotor and 24 slots in the stator over 2 poles, the adequate number of time-steps per electrical period to capture main slotting harmonics is about 400.

From the torque and currents curves it can be seen that steady state is reached after about 0.4 s. The average torque is about 36 Nm and peak phase current is about 17.5 A (12.4 A rms). Those values are somewhat higher than the nominal values 32.9 Nm and 10.4 A rms. The main reason is that the skew is not taken into account in this 2D model and losses are not subtracted from the mechanical power (iron losses, windage losses, mechanical losses, and eddy-current losses). However, some idea of the performance of the simulated machine still can be captured from this 2D model. The no load current is 6 A, which is a bit lower than 6.5 A design value.

Next figures show torque and current dependence vs slip for this 2D model obtained by spline-approximation of FEM-calculated points. The steady state was assumed after 40 electrical periods, however, for high slip values the steady state needs longer time to reach.

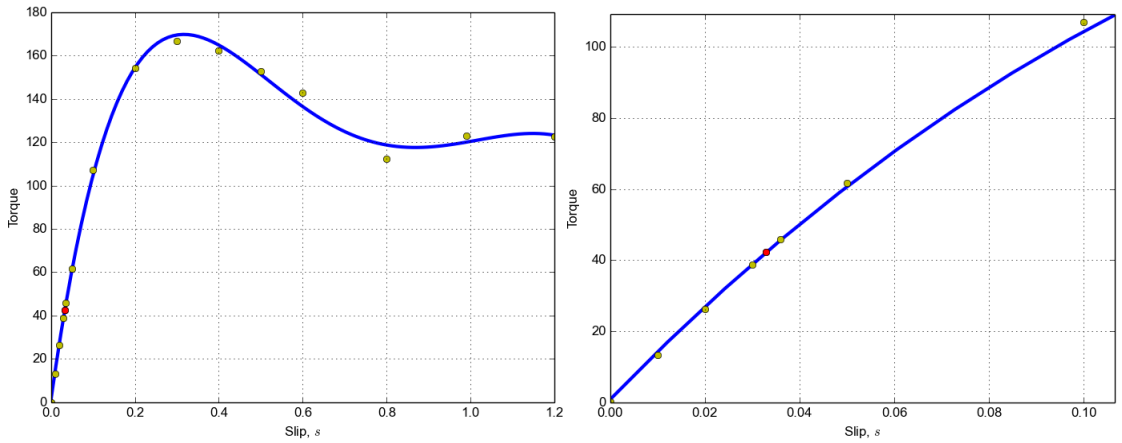


Figure 31. Torque vs slip.

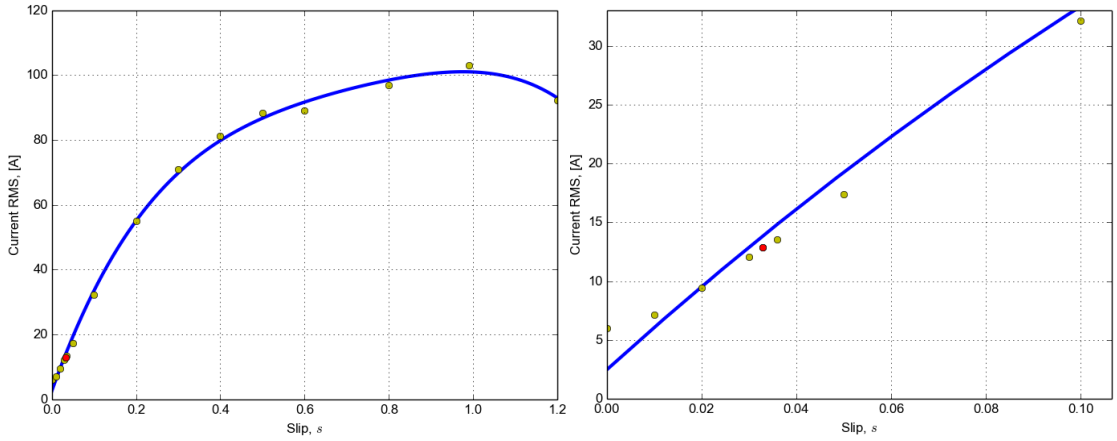


Figure 32. Stator current vs slip.

Figure 33 shows loci of the flux density in the stator yoke, in the middle of the tooth, and in the tooth tip during one electrical period. The data was saved to the scalars output dat file by the SaveScalars solver using next parameters:

```
Save Coordinates(3,2) = -0.07 0.07 -0.053 0.053 -0.0428 0.046
```

The elemental values are averaged over all nodal values for the element. The loci have many loops due to the slotting effect. A multi-slice model should be used for iron losses estimation due to the attenuation of the current ripple due to skewing.

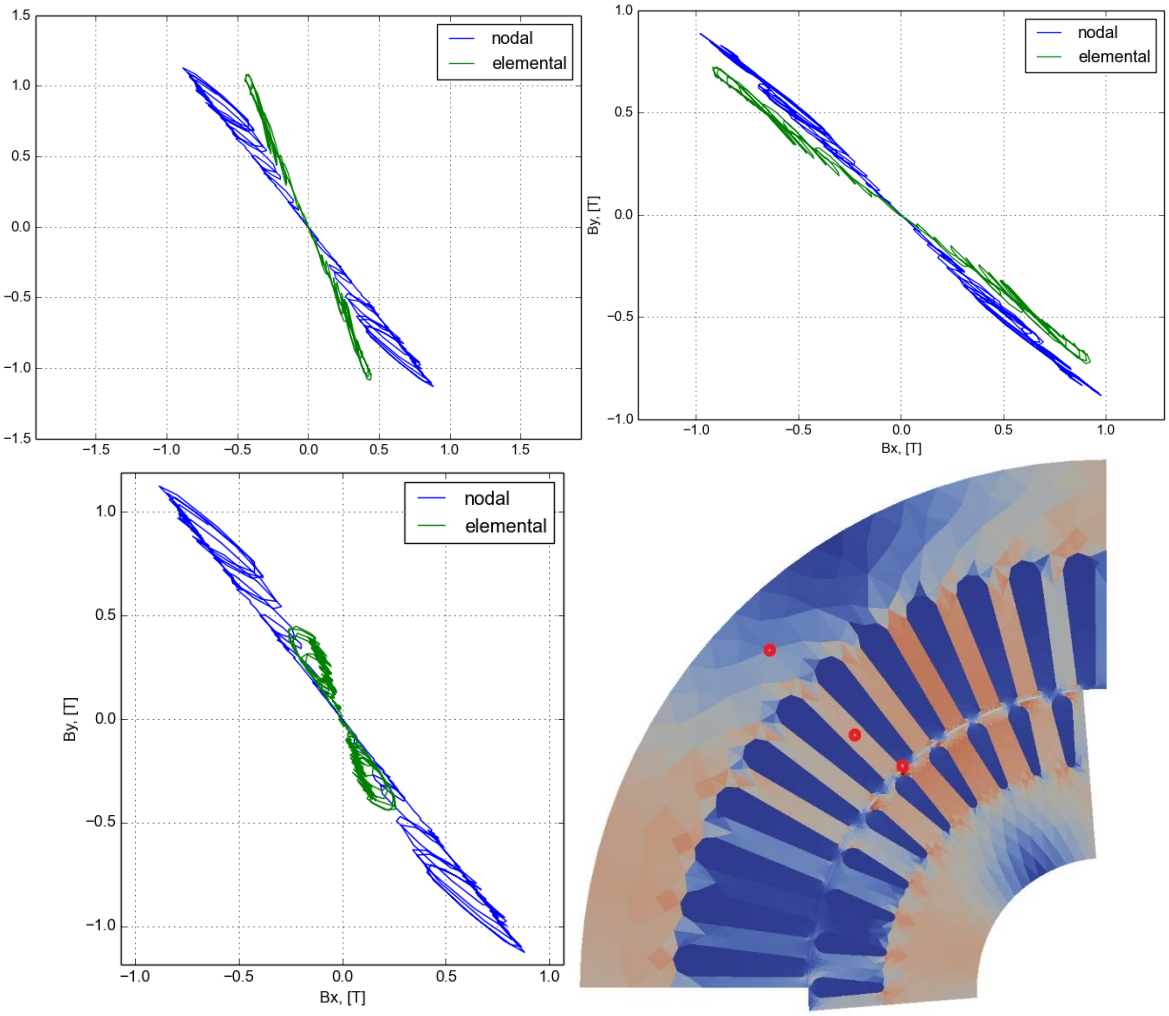


Figure 33. Flux density loci (no hysteresis) in the stator yoke (top left), in the middle of the tooth (top right), and in the tooth tip (bottom left) during one electrical period.

5.5 Parallel Performance

375 timesteps of Transient.sif simulation from previous section took 506 seconds. The solution time can be further decreased by selecting optimal solver for a problem and by tuning the solver parameters. Next lines in the Solver 3 section significantly decrease computational time for this case – from 506 second to 383 seconds (This was obtained for the direct solver MUMPS; for iterative solvers the difference is less significant). The results are practically the same.

```
!!!!!!!!!!!!!!!!!!  These are for better convergence
  Stabilize = False
  Partition Local Constraints = Logical True
  Nonlinear System Compute Change in Scaled System = Logical True
  Nonlinear System Convergence Measure = Residual
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Optimal selection of solvers, preconditioners, and tolerances, as well as solver settings is not very trivial task and strongly depends from the particular problem. Good initial suggestions can be found in test and example cases. For more solver settings refer to [5,6].

The computational time can be further decreased by utilizing parallelization capabilities.

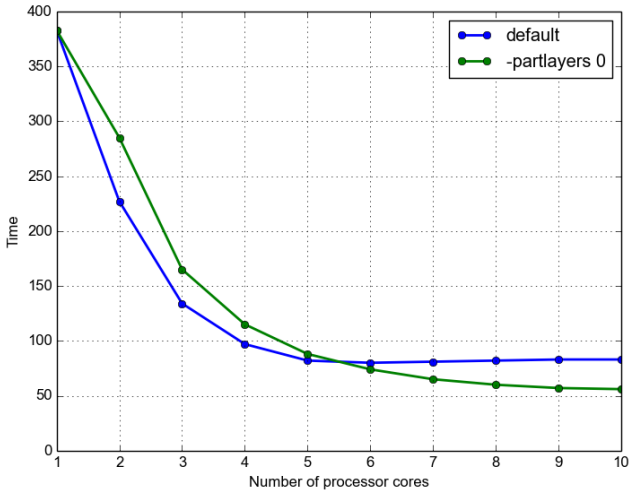


Figure 34. Parallel performance. Time is wall clock time.

As can be seen from Figure 34 with default partitioning using

```
>ElmerGrid 2 2 im -partdual -metis 10 3 -connect 2 3 4 5 7 8
```

the scalability saturates after 5 cores. The reason for this is the partition with mortar boundaries obtained by `-connect 2 3 4 5 7 8`, which has fixed size of 1096 elements. With more than 5 partitions, that fixed-sized partition becomes the largest, and other processor cores are often finished earlier and idling while computations over that partition proceed.

The size of that connected partition can be minimized from 1096 to 238 elements by using

```
>ElmerGrid 2 2 im -partdual -partlayers 0 -metis 10 3 -connect 2 3 4 5 7 8
```

This will result in larger misbalance of partition sizes at lower number of cores, however, it will enable the scalability for larger number of cores as can be seen in the Figure 34.

To make a small conclusion – less than a minute for FEM non-linear simulation of a time transient in an IM using 12 cores office PC is a good result.

Parallelization allows for refined simulations within acceptable time. To capture all the details of the IM 2D model behavior, the mesh density and time-stepping resolution was increased, and numerical tolerances were tightened. Figure 35 shows torque and stator and rotor currents. In the stator current waveform, the effect of slotting is clearly seen. In order to reduce this slotting effect, the skewing is applied in this machine. The effect of skewing cannot be precisely modelled in 2D domain.

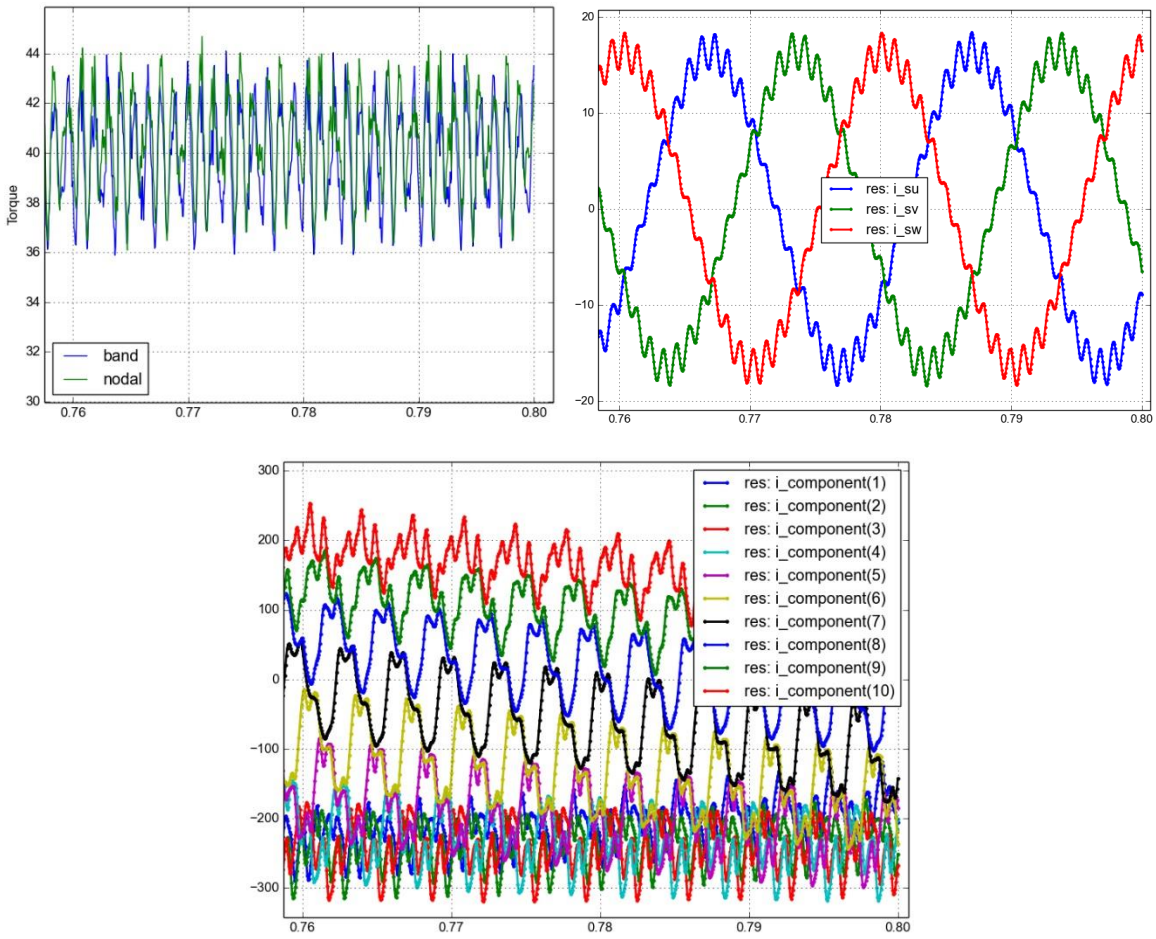


Figure 35. Results of refined simulation (600 time-steps per electrical period and doubled mesh density): torque, stator and rotor currents.

6. Time-Harmonic Simulation

An approximation of the steady state torque of an IM can be obtained even faster than was described in the previous section. Harmonic steady state solver can be used in assumption that all the variables are changing sinusoidally and iron is described as a linear material. Elmer's MagnetoDynamics2DHarmonic solver solves eddy-current problems in frequency domain.

The harmonic analysis allows for fast calculation of the following characteristics of the IM:

- Rated load torque and current
- Starting torque and current (slip=1)
- Torque versus slip curve

Being able to model these cases and knowing iron losses, one can estimate fast the equivalent circuit parameters (see e.g. [12, 13]).

6.1 Harmonic Case SIF

The SIF for harmonic simulations slightly differs from the time-transient SIF. The simulation type is steady and excitation angular frequency is defined in the Simulation section:

```
Simulation Type = Steady
Angular Frequency = Real $ 2*pi*f
```

Solvers must be changed to harmonic:

```
 Solver 2
   Exec Solver = Always
   Equation = Circ
   Variable = X
   Procedure = "CircuitsAndDynamics" "CircuitsAndDynamicsHarmonic"
 End
 Solver 3
   Exec Solver = Always
   Equation = MgDyn2D
   Variable = A[A re:1 A im:1]
   Procedure = "MagnetoDynamics2D" "MagnetoDynamics2DHarmonic"
```

The material properties must be changed to linear for the stator and rotor iron (the equivalent BH curve also can be used here to include non-linearity into account):

```
Relative Permeability = 300  ! Typical value is in 50-350
```

Results that are more reliable can be obtained if the iron domain would be divided in several regions characterized by the average permeability. E.g. rotor and stator yokes usually exhibit higher permeability than teeth due to the saturation.

The voltage supply is defined as

```
Body Force 1
    Name = "Circuit"
    ! U-phase voltage peak phase voltage at 0 degrees
    U_u Re = Real $ sqrt(2)/sqrt(3)*400*cos(0)
    U_u Im = Real $ sqrt(2)/sqrt(3)*400*sin(0)
    ! V-phase voltage 400/sqrt(3) Vrms at -120 degrees
    U_v Re = Real $ sqrt(2)/sqrt(3)*400*cos(-120*pi/180)
    U_v Im = Real $ sqrt(2)/sqrt(3)*400*sin(-120*pi/180)
    ! W-phase voltage 400/sqrt(3) Vrms at +120 degrees
    U_w Re = Real $ sqrt(2)/sqrt(3)*400*cos(120*pi/180)
    U_w Im = Real $ sqrt(2)/sqrt(3)*400*sin(120*pi/180)
End
```

Due to the presence of different electrical frequencies in the stator and in the rotor, several additional assumptions are introduced during the modelling as the harmonic model works with just one frequency. The rotor conductivities and resistances are modelled as slip dependent:

```
Electric Conductivity = Real $ 24000000*slip
$R_er = 0.0000016/slip
```

This assumption allows usage of the same electrical circuit definition for time-stepping and for time-harmonic cases.

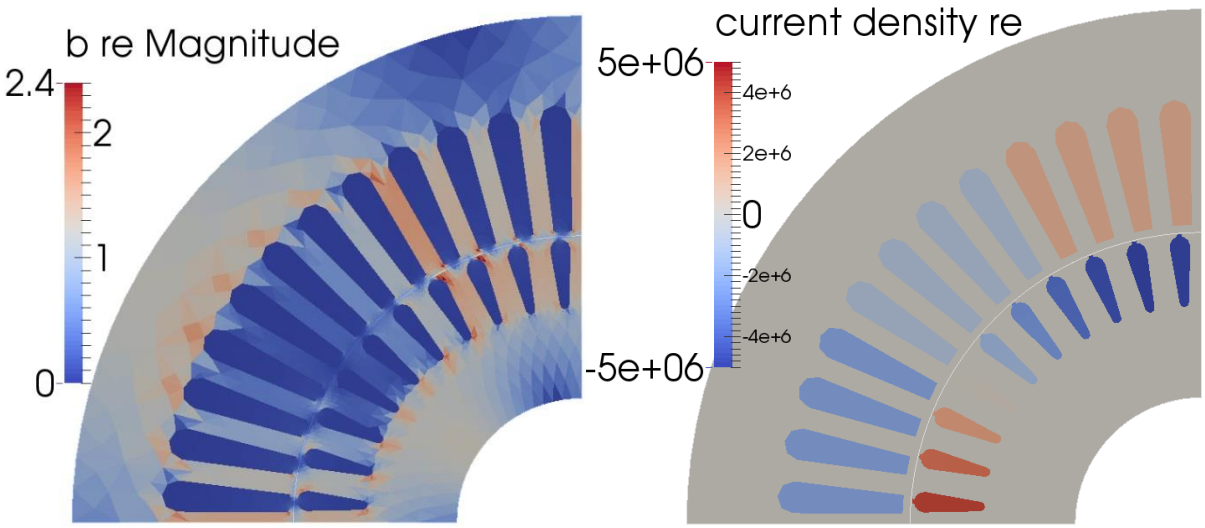


Figure 36. Real parts of B-field and J-field for harmonic solution.

The time-stepping simulations of Figure 37 took 12 hours using 10 cores in parallel (doubled mesh density, increased number of time-steps to capture steady state). Time-harmonic simulations took just 1 minute. The results show very good agreement.

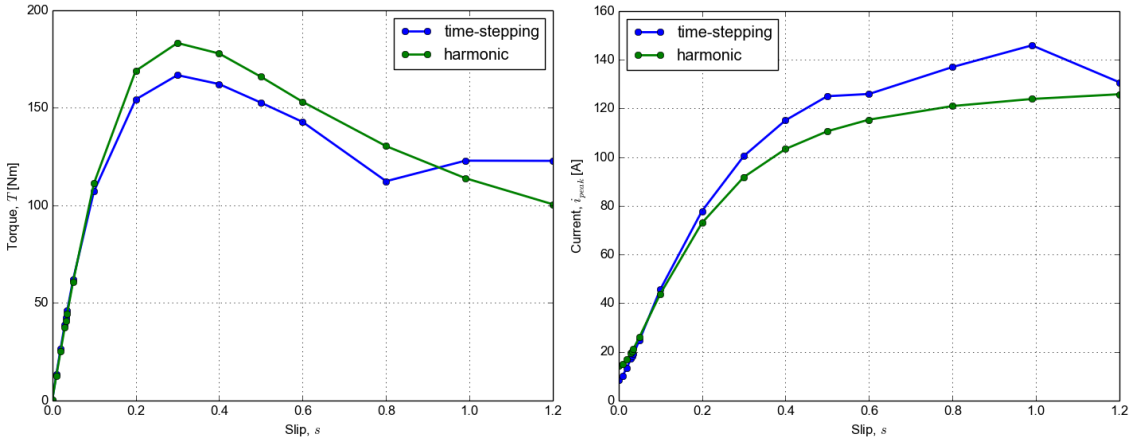


Figure 37. Comparison of time-stepping and harmonic results.

The time-harmonic analysis gives fast estimate of the IM characteristics without including the rotor motion. Even though the time-harmonic simulation cannot capture precisely non-linear phenomena, it can still be used for fast evaluation of the motor performance.

6.2 Rotor position influence

Harmonic steady state solution is strongly dependent on the rotor position. As can be seen from Figure 38, in this particular model the variation is within 5% from the nominal values.

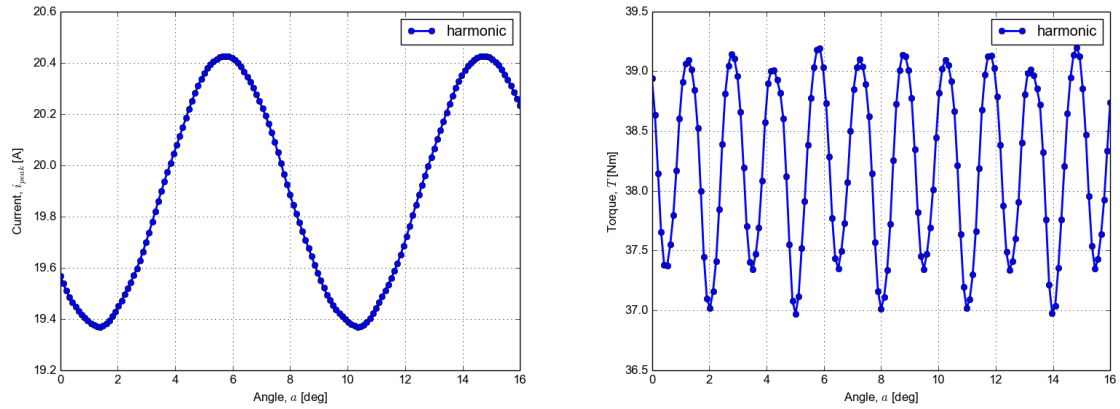


Figure 38. Rotor-position dependence of the harmonic solution.

The torque period is 90/(lcm(12,10)) = 1.5 degrees due to stator and rotor slotting.

For the harmonic analysis it is advisable to select rotor angle with average values of torque and current to get better quality averaged characteristics of the IM.

7. Multi-Slice Simulation

The waveforms in the previous two chapters demonstrate significant ripple due to the slotting effect. In real machines the slotting effect is eliminated by skewing – angular displacement of stator/rotor laminations along the machine axis in radial flux machines. This shifts the air gap harmonics along the axial features of the rotor, cancelling them when they are summed up along the whole axial length. The motor under the study has 8.5 degrees skew along the rotor length 160 mm.

Naturally, the skewing is a 3D phenomenon and can be precisely simulated using only 3D FEM. However, it is possible to dissect the machine to several slices along the axis and solve the magnetic fields in those slices in 2D. The slices are connected only via the electrical circuits. [10]

7.1 Mesh Preparation

The mesh for the multi-slice model is created by copying a 2D single slice mesh several times with spatial shift along the Z-axis. The command sequence for the mesh preparation is given here:

```
#!/bin/sh
ns=4
slen=0.04

# make rotor and stator elmer meshes from geo files
gmsh gmsh/im_stator.geo -2 -o im_stator.msh
ElmerGrid 14 2 im_stator.msh -2d -autoclean -names

gmsh gmsh/im_rotor.geo -2 -o im_rotor.msh
ElmerGrid 14 2 im_rotor.msh -2d -autoclean -names

# unite and scale to SI
ElmerGrid 2 2 im_stator -in im_rotor -unite -autoclean -names -out im
ElmerGrid 2 2 im -scale 0.001 0.001 0.001

# create slices
ElmerGrid 2 2 im -out imms -clone 1 1 $ns -clonesize 0 0 $slen
ElmerGrid 2 2 imms -partition 1 1 $ns
cp im/mesh.names imms/mesh.name
```

In the beginning, the number of slices `ns` and the distance between slices `slen` are defined. Then, rotor and stator meshes are converted from the `msh` format and united within 1 mesh `im`. Scaling is performed in order to switch to SI units (mm -> m). Then goes ElmerGrid command `-clone`, which copies the 2D mesh `im` along the Z-direction `$ns` times with the distance between the copies `$slen`. The last command partitions the resultant mesh to $ns equal partitions along the Z-axis. This keeps each slice in its own partition.

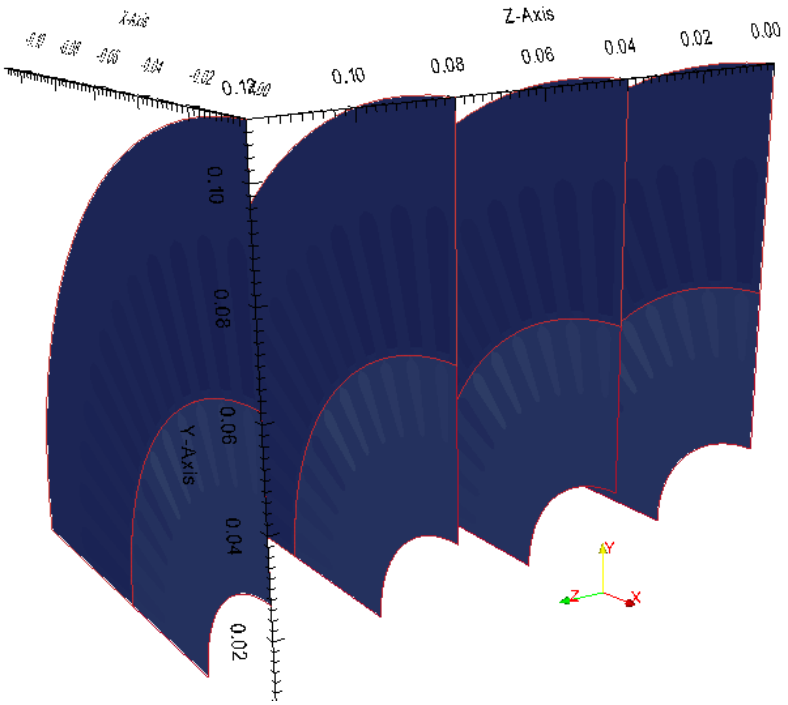


Figure 39. Four slices spatially distributed along the machine axis.

The last ElmerGrid command `-partition 1 1 $ns` creates 1 partition per slice (you can check the result of slicing in ParaView by using ElmerGrid with the second parameter 5). If further parallelization is feasible (in terms of available cores and large mesh size), then one may use e.g.:

```
>ElmerGrid 2 2 im -partdual -metis 12 3 -connect 2 3 4 5 7 8 -partconnect 4
```

where each slice will have 3 partitions.

The last command in the sequence copies `mesh.names` from serial mesh to partitioned mesh.

This partitioning scheme creates slices where ids and names of the physical bodies are the same in all the slices. If unique body names and indices are desired in every slice (e.g. complex winding interconnects in the possible cooling channels), then one may use `-cloneinds` command together with `-clone`.

Notice that the model has no skew at this point, just identical copies of rotor and stator without angular displacement along the axis. The skew will be realized within the SIF using analytical dependence of the skew angle from the Z-coordinate within the `RigidMeshMapper` solver.

7.2 SIF and Circuits

In the SIF the Simulation section now has no scaling command as the scaling to SI has been performed during the mesh conversion in ElmerGrid.

The skewing is embedded in the mesh rotation equation in the `Body Force 7` section, where additional dependence from `Coordinate 3` is added:

```
Body Force 7
  Mesh Rotate 3 = Variable time, timestep size, coordinate 3
    Real MATC "180/pi*f*(1-slip)*2*pi/pp*(tx(0)-tx(1))+skew*tx(2)/l"
End
```

All other sections and circuit equations remain the same. The scaling factor `ns` is used for the circuit model depth and for the number of turns in each component as multi-slice model is represented by the bodies which are distributed spatially between slices and virtually the component should reflect this cloning:

```
Simulation section:
            Circuit Model Depth = Real $ l/ns

Components sections:
            Number of Turns = Real $ ns*Nph/2
```

The rotor circuits are automatically constructed by using cage_generator.py script with the following settings in the beginning (the ns influences only the scaling for the number of conductors for the bar definition):

```
# Settings:
ns = 4              # number of slices (assuming equal slice lengths)
nob = 10            # number of rotor bars simulated
boffset = 1         # body number of the first bar (assuming consequent numbering)
antiperiodic = 1    # periodic or antiperiodic boundary
cn = 4              # circuit number which describes the rotor bars
ctype = "Stranded" # Stranded for equal current sharing within the same body number
OUTFILE = 'cage.definitions'
```

Here, `ns` denotes the number of slices. The rotor circuits are modelled using "`Stranded`" coil type in order to ensure the same current density between all the slices in the same physical body of the conductors. If effects in solid conductors are important to capture, then the bodies in each slice should have unique body indices (`-cloneinds` in ElmerGrid) and circuits will be somewhat more complex to construct, but still possible.

Hence, in this model, the number of slices appears only in:

- Creating slices in the mesh grid
- Scaling length of the 2D domain, and the number of turns in the whole domain for each component
- In the cage winding generator for the number of turns for the components of the rotor bars.

7.3 Results

Here, the results for 4 slices are presented. The computational time with 4 partitions using 4 cores has been slightly bigger than for a simple 2D serial computation as the convergence time has increased. Comparison of multi-slice 2.5D modelling results with 2D, 3D, and measurements can be found e.g. in [10, 11].

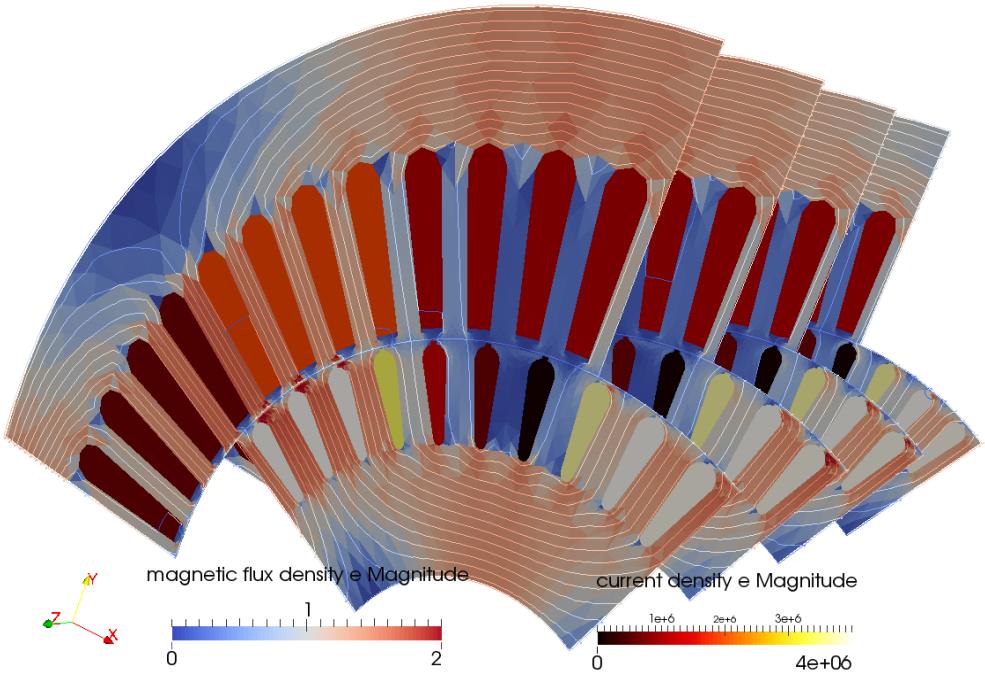


Figure 40. Fields in the multi-slice model (video: https://www.youtube.com/watch?v=WBSNSm4CB78).

Figure 41 shows the stator and rotor currents waveforms. The stator currents exhibit now significant ripple reduction in comparison with 2D results (Figure 35) – this is due to the skewing effect, which cancels the slotting harmonics. The average torque is also somewhat lower due to the skew. The nominal torque of the real machine is 32.9 Nm. This results in discrepancy of ~3.3 Nm, which corresponds to about 10% of the power, which is lost due to the losses in the machine (typical efficiency of industrial IMs is in the range 86–94%).

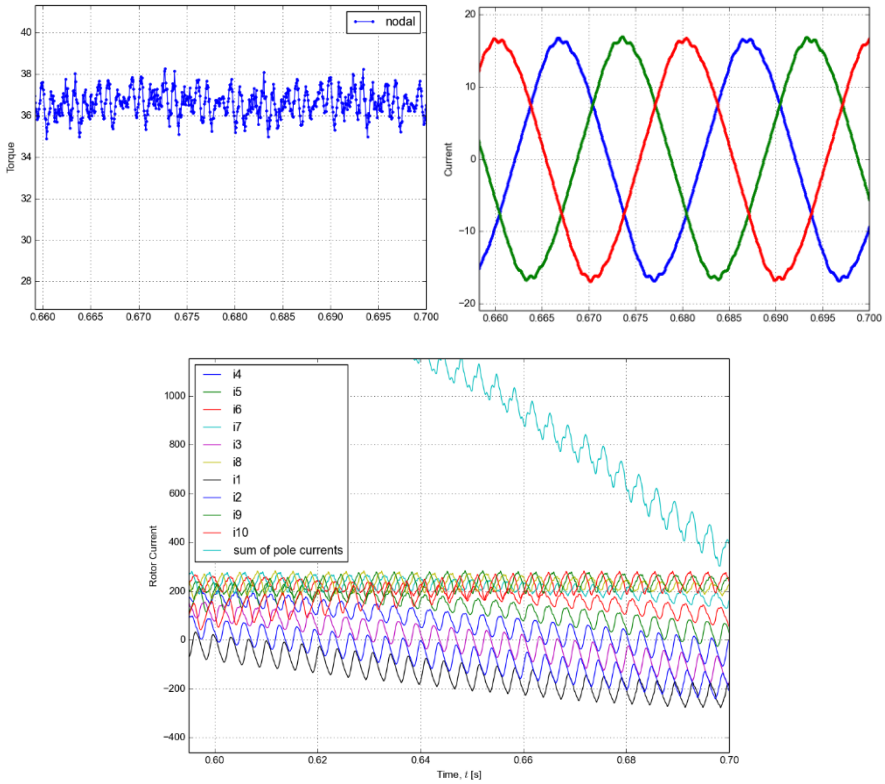


Figure 41. Currents and torque waveforms of the multi-slice model (4 slices).

Next figure shows measured nominal phase current waveform and simulated scaled by measured losses (0.93 to account for iron losses and rotor losses) current waveform from a 7-slice simulation. Both waveforms contain noise – due to measurement errors, numerical approximations, discretization, and uncertainties in parameter values. The rotor bars of the 2.5D model were modelled as stranded conductors resulting in neglection of the skin effect in the rotor conductors, which has slightly changed the harmonic spectrum of the resultant simulated waveform.

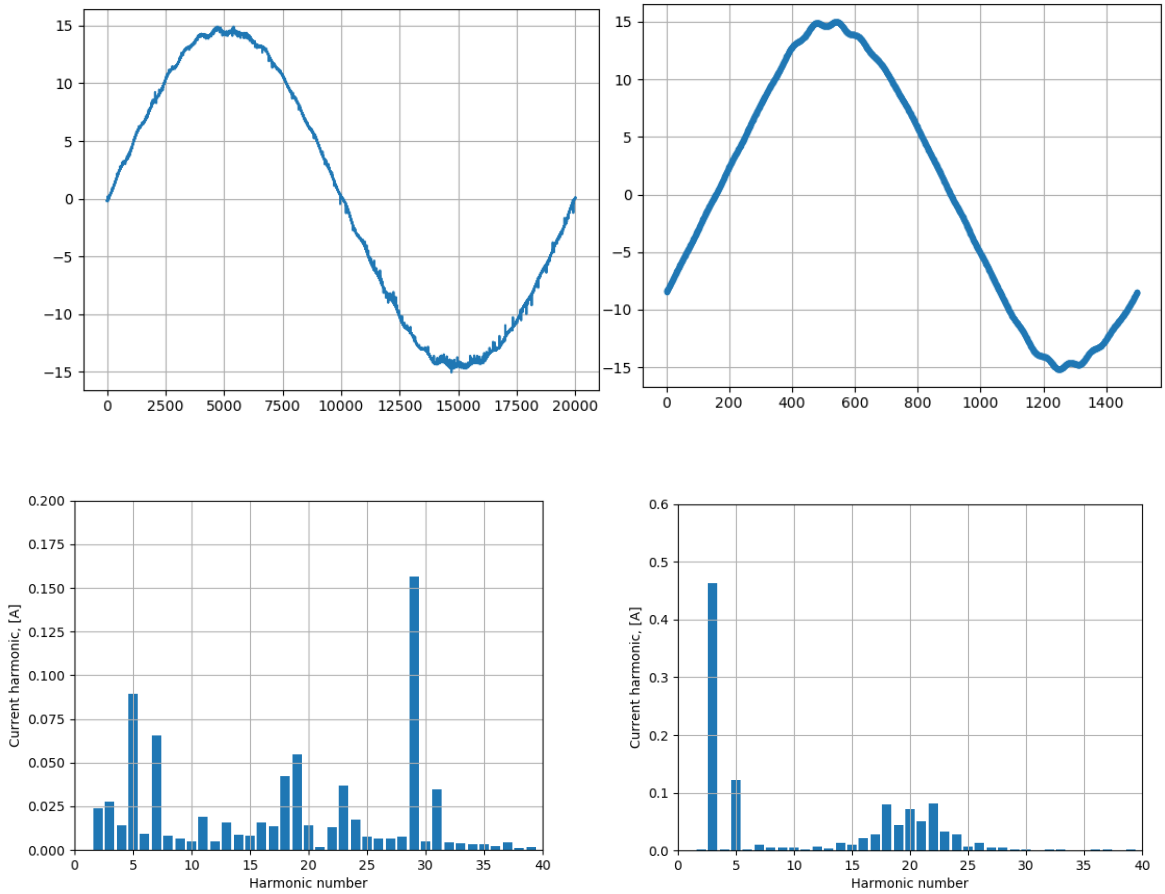


Figure 42. Measured (left) and simulated (right, 7 slices, scaled by 0.93).

The spectrum of the measured current contains 29th and 31st harmonics due to the presence of 30th harmonic in the voltage supply. The 3^{d} and 5th harmonics are mainly due to the saturation. Usage of a generic BH curve explains differences in those harmonics.

This figure validates effectiveness of usage of multi-slice 2.5D models in the analysis of skewed radial flux rotating induction electrical machines. The 2.5D model requires comparable solution time (with appropriate parallel environment) to 2D model, but additionally allows inclusion of heavy-to-compute 3D features of skewed machines.

8. Iron-Losses

The losses estimation is a must-have step within iterative multi-objective optimization for electrical machines. Iron losses are an important losses component, and quite often it can be the dominant component. Therefore, an accurate prediction of iron losses from FEA is essential for design of optimal machines.

Many different models are presented in the literature [14,15,18]. Iron losses can be estimated using FourierLossSolver (*this solver has been in active development at the moment of writing this report; refer to the Elmer documentation for the actual parameters and functionality*). This solver utilizes Steinmetz approach (different models can be simulated e.g. extended Pry and Bean equation, Bertotti equation) of type:

$$P = \mathrm{sum}(P_k) = \mathrm{sum}(C_1(f) \cdot f_k^{a1} \cdot B_k^{b1} + C_2(f) \cdot f_k^{a2} \cdot B_k^{b2} + \ldots)$$

where

- P_k is a harmonic power losses component associated with k^{th} B harmonic
- B_k is the peak flux density value of k^{th} B harmonic
- f_k is the k^{th} harmonic frequency
- C is the loss coefficient (can be a function of f)
- a is the harmonic loss frequency exponent
- b is the harmonic loss field exponent

The sum may contain 2 terms for Pry and Bean equation, or 3 terms for Bertotti equation, and even more terms.

Parameters `a`, `b`, and `C(f)` can be set in the sif. Number of harmonics `k` is defined by the `Fourier Series Components` keyword. This number should correlate with the number of time-steps per electrical period. In order to take into account also slotting effect, the rule of thumb would be: the time-stepping resolution should be fine enough and the number of `Fourier Series Components` should correspond to the doubled slotting frequency (2·Qs/pp). In this case the number of samples per electrical period should be at least 4· Qs/pp according to the Nyquist–Shannon sampling theorem.

Parameters `a`, `b`, and `C(f)` can be obtained from curve fitting of the measured data for various frequencies and flux densities. At the moment of writing this document, exact coefficients for the material M800_65A were not available. Here, only indicative values for parameters C are given.

The solver section for the losses calculation is listed next. The base frequency for the Fourier analysis is selected to be supply frequency; all the harmonics are computed in reference to that frequency. The fourier decomposition will be averaged over 30 electric periods, which roughly corresponds to 1 rotor electric period.

```
Equation 1 :: Active Solvers(3) = 2 3 4
Equation 2 :: Active Solvers(4) = 2 3 4 8
Solver 8
  Equation = "Fourier"
  Exec Solver = Always
  Procedure = "FourierLoss" "FourierLossSolver"
  Target Variable = "A"

  !Base frequency for Fourier transform
  Frequency = Real $50.0

!  Fourier Start Cycles = Integer 1
  Fourier Start Time = Real 0.05
```

```
  Fourier Integrate Cycles = Integer 30

  Fourier Series Components = Integer 25

  Fourier Loss Filename = File "Loss.dat"
  Harmonic Loss Field Exponent(3) = Real 2.0 2.0 1.5     !b1 b2 b3 …
  Harmonic Loss Frequency Exponent(3) = Real 1.0 2.0 1.5 !a1 a2 a3 …
End
```

All the iron bodies should have `Equation = 2` instruction in their sections to enable losses calculation over those bodies. Iron material section should be augmented with C coefficients:

```
! lamination steel
Material 2
  Name = "Iron"
  INCLUDE el_steel_M800_65A
  Electric Conductivity = 0

  ! C1
  Harmonic Loss Coefficient 1 = Variable "Frequency"
    Real
      0.0       $231.9
      4000.0     $231.9
    End

  ! C2
  Harmonic Loss Coefficient 2 = Variable "Frequency"
    Real
      0.0       $0.97
      10000.0   $0.97
    End

  ! C3
  Harmonic Loss Coefficient 3 = Variable "Frequency"
    Real
      0.0       $6.4
      10000.0   $6.4
    End

End
```

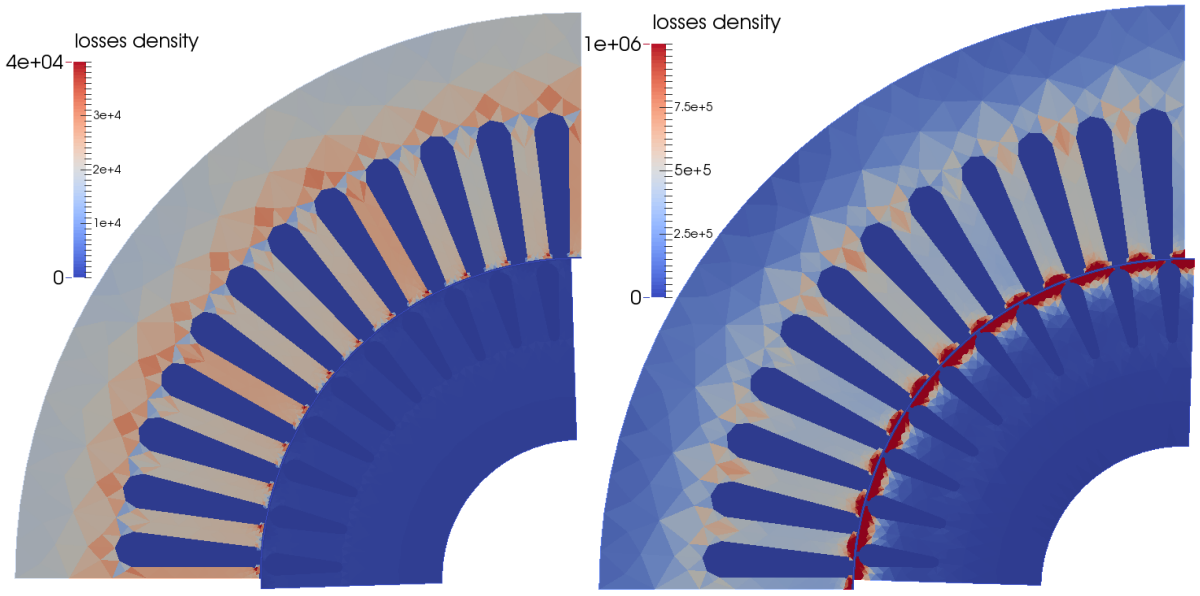


Figure 43. Left: Iron losses density calculated with 3 stator frequency harmonics (90 rotor harmonics, the slotting effect is therefore excluded). Right: Iron losses density calculated with 25 stator frequency harmonics (750 rotor harmonics, the slotting effect is therefore included).

Current solver returns several losses density fields – "`fourier loss 1`" and "`fourier loss 2`", etc, and one loss density field where components are summed up "`fourier`

`loss total`" in W/m^3. From the figure it is seen that stator tooth tips and rotor surface exhibit largest losses density as expected, since they carry high frequency harmonic fluxes due to the slotting and rotation.

Output file `Loss.dat` contains averaged losses density (W/m) for 2 bodies – rotor and stator (all the bodies which active for `Equation = 2`). A 2.5D model can give better estimate of iron losses taking into account attenuation of slotting harmonics by the skew, provided that the C coefficients are measured for high frequencies and high flux densities for the utilised magnetic material.

The iron losses for this machine were measured using calorimetric measurement method with losses separation [17]. The measured iron losses are 205 W at the nominal point.

9. DOL Startup

In this section a direct online startup of the IM is simulated starting from zero speed and accelerating to nominal speed with addition of nominal torque in the end. The model takes into account rotor inertia. It is possible to include also friction terms and other loading terms as well.

```
Jr * w' = Te - Tload
```

In the current version of Elmer, electromagnetic solvers MagnetoDynamics and MagnetoDynamics2D do not have the functionality to solve this scalar kinematic equation (perhaps it is not feasible to include dedicated rotating machine design equation to the general purpose electromagnetics modules). Therefore, a separate simple solver is required here for this scalar equation.

The complete commented listing of this solver Kinematics.F90 can be found in the Appendix. At initialization it creates global scalar variables for speed `Rotor Velo` and angular position `Rotor Angle`.

At the first time-step it initializes required internal variables and reads values of the moment of inertia of the rotor `Jr` and `Torque scaling factor` from the Simulation section of the SIF. The torque scaling factor is needed here as MagnetoDynamics2D returns torque in SI units scaled to the unit axial length, and also possible periodicities are not taken into account.

```
! Variables for the kinematics solver
Jr = Real 0.028
Torque Scaling Factor = Real $ 4*0.161
```

The torque is read from the scalars list "`res: group 1 torque`" which was obtained from the nodal forces computation in `torque group 1` in the `MagnetoDynamicsCalcFields` procedure.

Then, knowing speed at the previous timestep, electromagnetic torque, and load torque with the inertia, the solver calculates new speed and rotor displacement. After this, it updates global variables `Rotor Velo` and `Rotor Angle` with new values.

This additional solver is compiled using command:

```
>elmerf90 -O -o Kinematics Kinematics.f90
```

The correspondent solver section in the SIF looks like this:

```
Solver 6
  Exec Solver = after timestep
  Equation = Kinem
  Procedure = "Kinematics" "Kinematics"
  No Matrix = Logical True
End
```

The body force for RigidMeshMapper is updated to use Rotor angle global variable:

```
Body Force 7
!  Mesh Rotate 3 = Variable time, timestep size, rotor velo
!    Real MATC "180/pi*tx(2)*(tx(0)-tx(1))"
  Mesh Rotate 3 = Variable "Rotor angle"
     Real MATC "180*tx/pi"
End
```

The complete listing for the startup transients `transient_kinematics.sif` can be found in the Appendix. Here are some results (video https://www.youtube.com/watch?v=HI6ICoWC4B8):

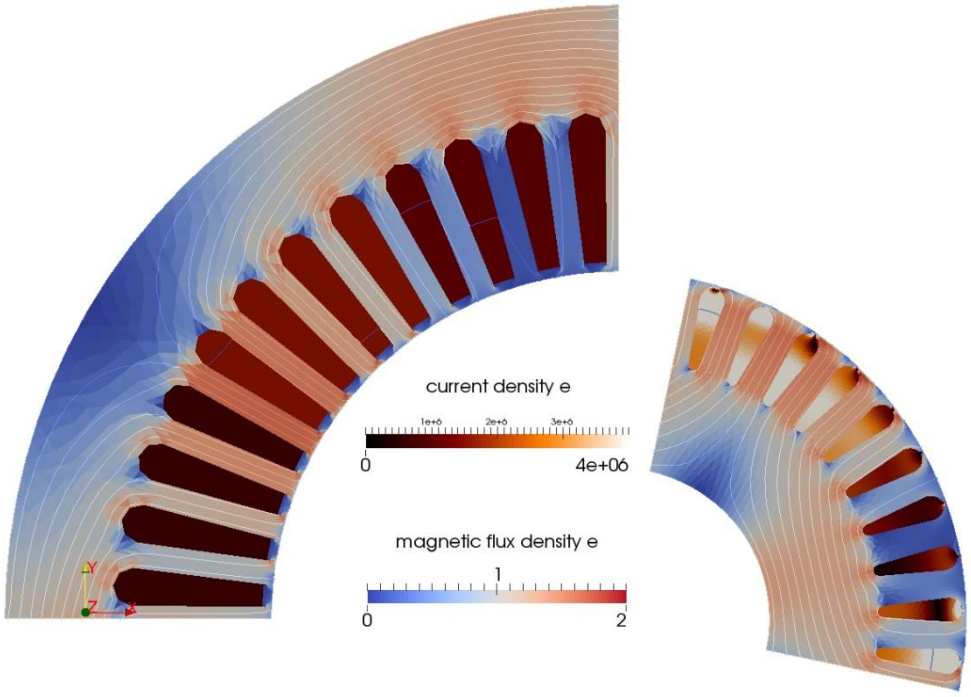


Figure 44. Current and flux densities at loaded steady state after ~0.7s.

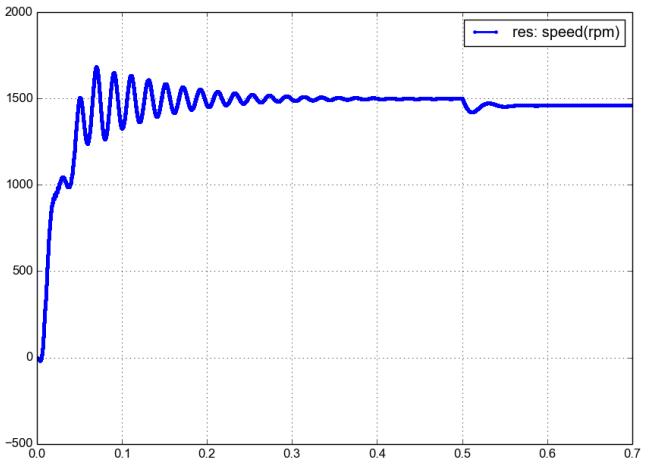


Figure 45. Speed transients during startup with no-load, and then application of the nominal load at 0.5s.

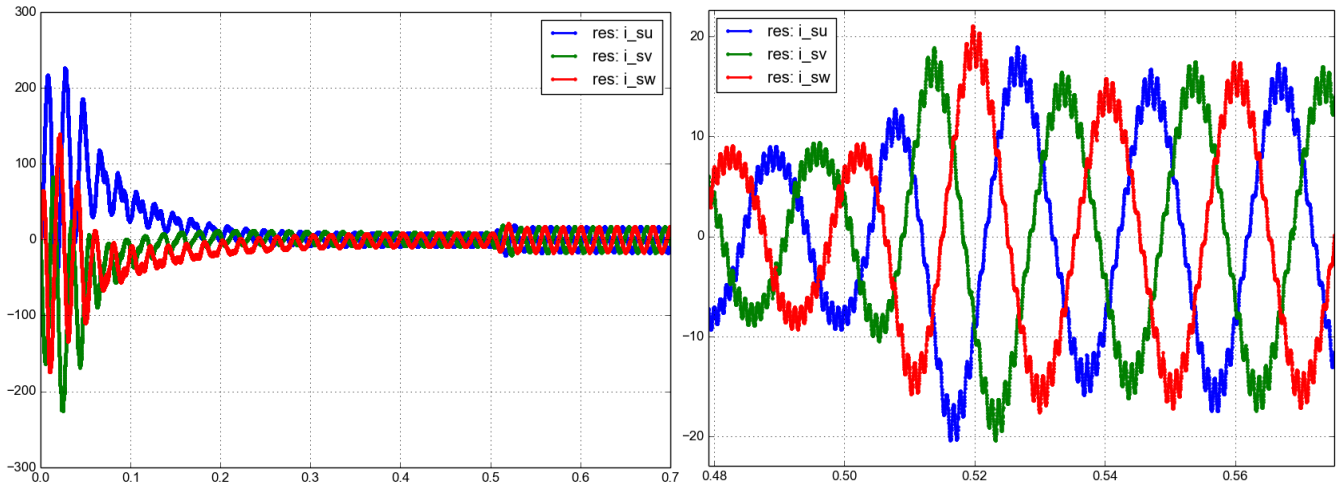


Figure 46. Stator currents during transient.

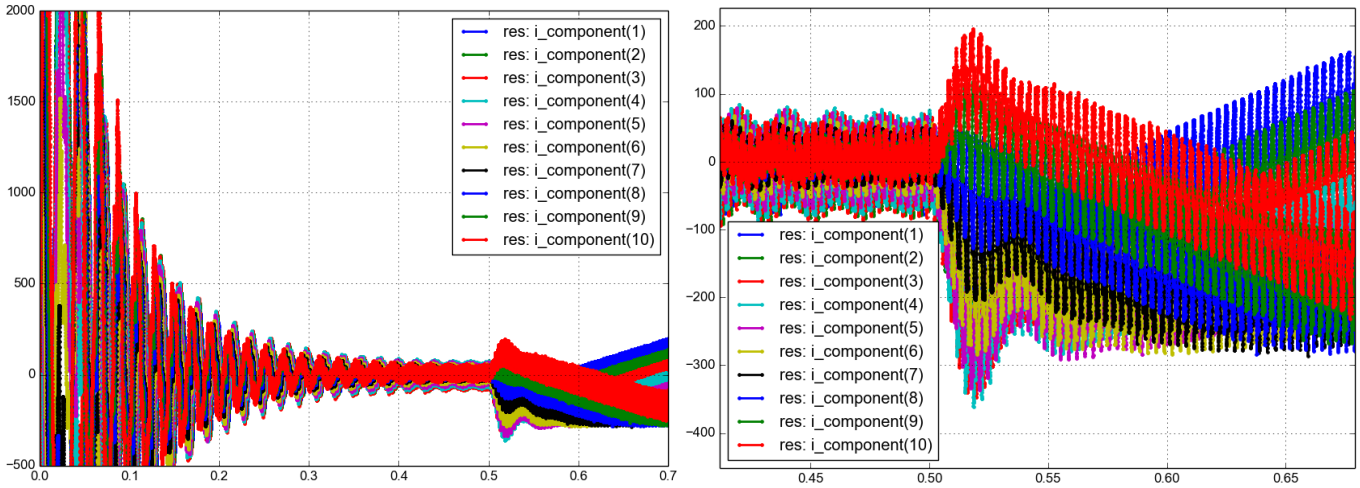


Figure 47. Currents of one rotor pole during transients.

The waveforms exhibit noticeable ripple as the model is only 2D and skewing is not included, hence, the slotting effect influences a lot.

10. Conclusions

This report/tutorial presents capabilities of an open-source toolchain for 2D and 2.5D (multi-slice) modelling of electromagnetic devices. The toolchain comprises GMSH as pre-processor, ParaView and Python as post-processing tools, and Elmer as the core FEM engine. Typical workflows with example models are also demonstrated allowing for fast adaptation of this toolchain for end-users. Parallelization capabilities are highlighted as well, and demonstrated allowing for fast solution of larger models.

References

1. C. Geuzaine and J.-F. Remacle; “Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities,” *International Journal for Numerical Methods in Engineering* 79(11), pp. 1309-1331, 2009.
2. http://gmsh.info/doc/texinfo/gmsh.html
3. ElmerGrid Manual. ftp://nic.funet.fi/index/elmer/doc/ElmerGridManual.pdf
4. Overview of Elmer. ftp://nic.funet.fi/index/elmer/doc/ElmerOverview.pdf
5. Elmer Solver Manual.
6. Elmer Models Manual.
7. P. Ponomarev; “Fem modeling of PMSMs using Elmer”, May 2015, DOI:10.13140/RG.2.1.1721.4886.
8. Eelis Takala, Evren Yurtesen, Jan Westerholm, Juha Ruokolainen, and Peter Råback. Parallel simulations of inductive components with elmer finite element software in cluster environments. Electromagnetics, 36(3):167-185, April 2016.
9. P. Lombard and F. Zidat, "Determining end ring resistance and inductance of squirrel cage for induction motor with 2D and 3D computations," *2016 XXII International Conference on Electrical Machines (ICEM)*, Lausanne, 2016, pp. 266-271.
10. J. Keranen; P. Ponomarev; J. Pippuri; P. Raback; M. Lyly; J. Westerlund, "Parallel Performance of Multi-Slice Finite Element Modelling for Skewed Electrical Machines," in *IEEE Transactions on Magnetics* , vol.PP, no.99, pp.1-1 doi: 10.1109/TMAG.2017.2653421
11. P. Ponomarev, J. Keränen, M. Lyly, J. Westerlund and P. Råback, "Multi-slice 2.5D modelling and validation of skewed electrical machines using open-source tools," *2016 IEEE Conference on Electromagnetic Field Computation (CEFC)*, Miami, FL, 2016, pp. 1-1. doi: 10.1109/CEFC.2016.7815918
12. J Pyrhonen, T Jokinen, V Hrabovcova, “Design of rotating electrical machines”, 2nd edition, 612p, 2013.
13. A. Stermecki, O. Biro, K. Preis, S. Rainer, K. Krischan and G. Ofner, "Calculation of load-dependent equivalent circuit parameters of squirrel cage induction motors using time-harmonic FEM," 2008 18th International Conference on Electrical Machines, Vilamoura, 2008, pp. 1-6.
14. D. Eggers, S. Steentjes and K. Hameyer, "Advanced Iron-Loss Estimation for Nonlinear Material Behavior," in IEEE Transactions on Magnetics, vol. 48, no. 11, pp. 3021-3024, Nov. 2012
15. P. A. Hargreaves, B. C. Mecrow and R. Hall, "Calculation of Iron Loss in Electrical Generators Using Finite-Element Analysis," in *IEEE Transactions on Industry Applications*, vol. 48, no. 5, pp. 1460-1466, Sept.-Oct. 2012.
16. D. M. Ionel, M. Popescu, S. J. Dellinger, T. J. E. Miller, M. I. McGilp and R. J. Heideman, "Factors Affecting the Accurate Prediction of Iron Losses in Electrical Machines," *IEEE International Conference on Electric Machines and Drives, 2005.*, San Antonio, TX, 2005, pp. 1625-1632.
17. L. Aarniovuori, P. Rasilo, M. Niemelä and J. J. Pyrhönen, "Analysis of 37-kW Converter-Fed Induction Motor Losses," in *IEEE Transactions on Industrial Electronics*, vol. 63, no. 9, pp. 5357-5365, Sept. 2016.
18. A. Krings and J. Soulard, ”Overview and Comparison of Iron Loss Models for Electrical Machines”, presented at the 5th International Conference and Exhibition on Ecological Vehicles and Renewable Energies (EVER 10), Monte-Carlo, MONACO, MAR 25-28, 2010

Appendix A IM_rotor.geo

```
// May 2013 - Authors: J. Gyselinck, R.V. Sabariego
// Modified July 2016 -- P. Ponomarev

Geometry.AutoCoherence = 0;
Mesh.Light = 0;

Qr = 40; // number of rotor teeth
N_rs = 10;

//Main rotor parameters
R_rin = 28;     // inner rotor radius
R_rout = 62;      // outer stator radius
R_g = 62.25;       // outer radius of moving band

//Slot dimensions
b_1 = 1;
h_1 = 0.4;
b_4 = 4.4;
b_5 = 2;
h_5 = 12;

//offset for the curvature of the slot opening in the gap
s = R_rout-Sqrt((R_rout)^2-(b_1/2)^2);

//ofsett of the top arc curvature at the tip
s2 = b_4/2-Sqrt((b_4/2)^2-(b_1/2)^2);

//Mesh density
m_coarse = 10;
m_normal = 8;
m_gap = 0.5;
m_sl_bot = 4;
m_sl_top = 1;
m_r_in = 10;

//build rotor slots
For i In {0:N_rs-1}

  //build two halfs
  For half In {0:1}
           //points of one half
    dP=newp;
    Point(dP+0)  = {0,0,0,m_coarse};
           //right toothtip points
    Point(dP+1)  = {b_1/2, R_rout-s, 0, 1.4*m_gap};
    Point(dP+2)  = {b_1/2, R_rout-h_1-s2, 0, m_sl_top};
           //center of the bottom circle
           Point(dP+3)  = {0, R_rout-h_1-b_4/2-h_5, 0, m_sl_bot};
           //b4 top circle side
    Point(dP+4)  = {b_4/2, R_rout-h_1-b_4/2, 0, 2*m_sl_top};
           //b5 bottom circle side
    Point(dP+5)  = {b_5/2, R_rout-h_1-b_4/2-h_5, 0, m_sl_bot};
           // central point of top circle
    Point(dP+6)  = {0, R_rout-h_1-b_4/2, 0, m_sl_top};
           // bottom slot point
    Point(dP+7)  = {0, R_rout-h_1-b_4/2-h_5-b_5/2, 0, m_sl_bot};
           // inner rotor sector
    Point(dP+8) = {R_rin*Sin(Pi/Qr), R_rin*Cos(Pi/Qr), 0, m_r_in};
           // inner rotor center
    Point(dP+9) = {0, R_rin, 0, m_r_in};
           // outer rotor sector
    Point(dP+10) = {R_rout*Sin(Pi/Qr), R_rout*Cos(Pi/Qr), 0, 2.5*m_gap};
           // sliding sector
    Point(dP+11) = {R_g*Sin(Pi/Qr), R_g*Cos(Pi/Qr), 0, 2.8*m_gap};
           //outer rotor center
    Point(dP+12) = {0, R_rout, 0, 1.5*m_gap};
           //sliding center
    Point(dP+13) = {0, R_g, 0, 1.4*m_gap};

           // rotate the built points to the i-th slot position
    For t In {dP+0:dP+13}
      Rotate {{0,0,1},{0,0,0}, 2*Pi*i/Qr+2*Pi/Qr/2} {Point{t};}
    EndFor

    If (half==1) //second half
      For t In {dP+0:dP+13}
        Symmetry {Cos(2*Pi*i/Qr+2*Pi/Qr/2),Sin(2*Pi*i/Qr+2*Pi/Qr/2),0,0} { Point{t}; }
      EndFor
    EndIf

    dR=newl-1;
           //vertical tooth tip line
    Line(dR+1) = {dP+1,dP+2};

           //top arc
           Circle(dR+2) = {dP+2,dP+6,dP+4};

           //vertical slot side
    Line(dR+3) = {dP+4,dP+5};

           //arc slot bottom
    Circle(dR+4) = {dP+5,dP+3,dP+7};

           //sliding
           Circle(dR+5) = {dP+11,dP+0,dP+13};

           //tooth top arc
    Circle(dR+6) = {dP+10,dP+0,dP+1};

           //arc slot opening
    Circle(dR+7) = {dP+1,dP+0,dP+12};
```

```
          //inner rotor
    Circle(dR+8) = {dP+8,dP+0,dP+9};

          //vertical central slot opening
    Line(dR+9) = {dP+12,dP+7};

          //vertical from bot of the slot to the rotor inner
    Line(dR+10) = {dP+7,dP+9};

          //sector border via steel
    Line(dR+11) = {dP+10,dP+8};

          //sector border via airgap
    Line(dR+12) = {dP+11,dP+10};

          //vertical sector center via gap
    Line(dR+13) = {dP+13,dP+12};

          //filling the lists for boundaries
    InnerRotor_[] += dR+8;

    RotorBoundary_[] += {dR+8,dR+4,dR+3,dR+2,dR+1,dR+6};

          SlidingR_[] += {dR+5};

          //Periodic boundary
    If (Qr != N_rs)
            //right boundary
      If (i==0 && half==0)
        RotorPeriod_Right_[] = {dR+11,dR+12};
      EndIf
            //left boundary
      If (i == N_rs-1  && half==1)
        RotorPeriod_Left_[] = {dR+11,dR+12};
      EndIf
    EndIf

          //if mirrorred, then the lines order is reversed
          //direction is important defining the Line Loops
    rev = (half ? -1 : 1);

          //surface of the slot conductors
    Line Loop(newll) = {dR+9,-dR-4,-dR-3,-dR-2,-dR-1,dR+7};
    dH = news; Plane Surface(news) = -rev*{newll-1};
    RotorConductor_[] += dH;

          //surface of the stator iron
    Line Loop(newll) = {dR+1,dR+2,dR+3,dR+4,dR+10,-dR-8,-dR-11,dR+6};
    dH = news; Plane Surface(news) = -rev*{newll-1};
    RotorIron_[] += dH;

          //airgap stator
    Line Loop(newll) = {-dR-13,-dR-5,dR+12,dR+6,dR+7};
    dH = news; Plane Surface(news) = rev*{newll-1};
    RotorAirgapLayer_[] += dH;

  EndFor
EndFor

//-------------------------------------------------------------------------------------------
// Physical regions
//-------------------------------------------------------------------------------------------
Color Yellow {Surface{RotorConductor_[]};}

Physical Surface("Bar1") = {RotorConductor_[{0,1}]};
Physical Surface("Bar2") = {RotorConductor_[{2,3}]};
Physical Surface("Bar3") = {RotorConductor_[{4,5}]};
Physical Surface("Bar4") = {RotorConductor_[{6,7}]};
Physical Surface("Bar5") = {RotorConductor_[{8,9}]};
Physical Surface("Bar6") = {RotorConductor_[{10,11}]};
Physical Surface("Bar7") = {RotorConductor_[{12,13}]};
Physical Surface("Bar8") = {RotorConductor_[{14,15}]};
Physical Surface("Bar9") = {RotorConductor_[{16,17}]};
Physical Surface("Bar10") = {RotorConductor_[{18,19}]};

Physical Surface("RotorIron") = {RotorIron_[]};
Physical Surface("RotorAirgap") = {RotorAirgapLayer_[]};

Color SteelBlue {Surface{RotorIron_[]};}
Color SkyBlue {Surface{RotorAirgapLayer_[]};}

Physical Line("InnerRotor") = {InnerRotor_[]};
Physical Line("RotorRight") = {RotorPeriod_Right_[]};
Physical Line("RotorLeft") = {RotorPeriod_Left_[]};
Physical Line("Sliding_Rotor") = {SlidingR_[]};

Coherence;

show_rotor[] = CombinedBoundary{Surface{RotorIron_[]};};
Hide{ Point{Point '*'};}
Hide{ Line{Line '*'};}
Show{ Line{show_rotor[]};}

Mesh 2;
```

Appendix B IM_stator.geo

```
// May 2013 - Authors: J. Gyselinck, R.V. Sabariego
// Modified July 2016 -- P. Ponomarev

Geometry.AutoCoherence = 0;

//Stator
Qs = 48; // number of stator teeth
N_ss = 12;

//Main Stator parameters
R_sin = 62.5;     // inner stator radius
R_sout = 110;      // outer stator radius
R_g = 62.25;       // middle of the air gap radius

//Slot dimensions
b_5 = 6.8;
b_1 = 2.8;
h_1 = 0.7;
h_2 = 0.2;
b_4 = 4;
h_5 = 24;

//Mesh density
m_coarse = 20;
m_normal = 12;
m_gap = 0.55;
m_sl_top = 7;
m_sl_bot = 2;
m_s_out = 12;

//offset for the curvature of the slot opening in the gap
s = R_sin-Sqrt(R_sin^2-(b_1/2)^2);

//build stator slots
For i In {0:N_ss-1}

  //build two halfs
  For half In {0:1}

            //points of one half
    dP=newp;
    Point(dP+0)  = {0,0,0,m_coarse};

            //right toothtip points
    Point(dP+1)  = {b_1/2, R_sin-s, 0, 2*m_gap};
    Point(dP+2)  = {b_1/2, R_sin-s+h_1, 0, m_sl_bot};

            //center of the top circle
            Point(dP+3)  = {0, R_sin-s+h_1+h_2+h_5, 0, m_sl_top};

            //h2-b4 bottom teeth point
    Point(dP+4)  = {b_4/2, R_sin-s+h_1+h_2, 0, m_sl_bot};

            //h5-b5 top teeth point
    Point(dP+5)  = {b_5/2, R_sin-s+h_1+h_2+h_5, 0, m_sl_top};

            // point of lower winding
    Point(dP+6)  = {b_4/2, R_sin-s+h_1+h_2+0.5, 0, m_sl_bot/2};

            // central point of lower winding
    Point(dP+7)  = {0, R_sin-s+h_1+h_2+0.5, 0, m_sl_bot};

            // top slot point
    Point(dP+8)  = {0, R_sin-s+h_1+h_2+h_5+b_5/2, 0, m_sl_top};

            // outer stator sector
    Point(dP+9) = {R_sout*Sin(Pi/Qs), R_sout*Cos(Pi/Qs), 0, m_s_out};

            // outer stator center
    Point(dP+10) = {0, R_sout, 0, m_s_out};

            // inner stator sector
    Point(dP+11) = {R_sin*Sin(Pi/Qs), R_sin*Cos(Pi/Qs), 0, 2.5*m_gap};

            // sliding sector
    Point(dP+12) = {R_g*Sin(Pi/Qs), R_g*Cos(Pi/Qs), 0, 2.4*m_gap};

            //inner stator center
    Point(dP+13) = {0, R_sin, 0, 1.5*m_gap};

            //sliding center
    Point(dP+14) = {0, R_g, 0, 1.5*m_gap};

    // rotate the built points to the i-th slot position
    For t In {dP+0:dP+14}
      Rotate {{0,0,1},{0,0,0}, 2*Pi*i/Qs+2*Pi/Qs/2} {Point{t};}
    EndFor

    If (half==1) //second half
      For t In {dP+0:dP+14}
        Symmetry {Cos(2*Pi*i/Qs+2*Pi/Qs/2),Sin(2*Pi*i/Qs+2*Pi/Qs/2),0,0} { Point{t}; }
      EndFor
    EndIf

    dR=newl-1;
            //vertical tooth tip line
    Line(dR+1) = {dP+1,dP+2};

            //line from tooth tip arc to start of the winding
    Line(dR+2) = {dP+4,dP+6};
```

```
        //vertical slot side
    Line(dR+3) = {dP+6,dP+5};

        //horizontal winding bottom
    Line(dR+4) = {dP+6,dP+7};

        //arc toothtip - side
        Line(dR+5) = {dP+2,dP+4};

        //arc top of the slot-side
    Circle(dR+6) = {dP+5,dP+3,dP+8};

        //sliding
        Circle(dR+7) = {dP+12,dP+0,dP+14};

        //slot opening arc
    Circle(dR+8) = {dP+11,dP+0,dP+1};

        //arc inner teeth surface
    Circle(dR+9) = {dP+1,dP+0,dP+13};

        //outer stator
    Circle(dR+10) = {dP+9,dP+0,dP+10};

        //vertical central slot opening
    Line(dR+11) = {dP+13,dP+7};

        //vertical in the center of the slot
    Line(dR+12) = {dP+7,dP+8};

        //vertical from top of the slot to the stator outer
    Line(dR+13) = {dP+8,dP+10};

        //sector border via steel
    Line(dR+14) = {dP+11,dP+9};

        //sector border via airgap
    Line(dR+15) = {dP+12,dP+11};

        //vertical sector center via gap
    Line(dR+16) = {dP+14,dP+13};

    //filling the lists for boundaries
    OuterStator_[] += dR+10;
    StatorBoundary_[] += {dR+10,dR+6,dR+3,dR+2,dR+5,dR+1,dR+8};
        Sliding_[] += {dR+7};

        //Periodic boundary
    If (Qs != N_ss)
          //right boundary
      If (i==0 && half==0)
        StatorPeriod_Right_[] = {dR+14,dR+15};
      EndIf
          //left boundary
      If (i == N_ss-1  && half==1)
        StatorPeriod_Left_[] = {dR+14,dR+15};
      EndIf
    EndIf

        //if mirrorred, then the lines order is reversed
        //direction is important defining the Line Loops
    rev = (half ? -1 : 1);

        //surface of the slot conductors
    Line Loop(newll) = {dR+12,-dR-6,-dR-3,dR+4};
    dH = news; Plane Surface(news) = -rev*{newll-1};
    StatorConductor_[] += dH;

        //surface of the stator iron
    Line Loop(newll) = {dR+1,dR+5,dR+2,dR+3,dR+6,dR+13,-dR-10,-dR-14,dR+8};
    dH = news; Plane Surface(news) = -rev*{newll-1};
    StatorIron_[] += dH;

        //wedges
    Line Loop(newll) = {dR+11,-dR-4,-dR-2,-dR-5,-dR-1,dR+9};
    dH = news; Plane Surface(news) = -rev*{newll-1};
    StatorWedge_[] += dH;

        //airgap stator
    Line Loop(newll) = {-dR-16,-dR-7,dR+15,dR+8,dR+9};
    dH = news; Plane Surface(news) = rev*{newll-1};
    StatorAirgapLayer_[] += dH;

  EndFor
EndFor

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Physical regions
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//Assigning StatorConductors to phase Regions
//number of slots in a belt
qq=4;

//for each phase side
For f In {0:5}
  Con[]={};
  For i In {0:N_ss/qq-1}
    If (Fmod(i,6) == f)
      For j In {0:qq-1}
        Con[] += StatorConductor_[{2*i*qq+2*j, 2*i*qq+2*j+1}];
      EndFor
    EndIf
  EndFor
```

```
  //color the phase and assign to a physical region
  If (#Con[] > 0)
    If (f == 0)
                          Color Red {Surface{Con[]};}
                          Physical Surface("U_plus") = {Con[]};
    EndIf
    If (f == 1)
                          Color SpringGreen {Surface{Con[]};}
                          Physical Surface("W_minus") = {Con[]};
    EndIf
    If (f == 2)
                          Color Gold {Surface{Con[]};}
                          Physical Surface("V_plus") = {Con[]};
    EndIf
    If (f == 3)
                          Color Pink {Surface{Con[]};}
                          Physical Surface("U_minus") = {Con[]};
    EndIf
    If (f == 4)
                          Color ForestGreen {Surface{Con[]};}
                          Physical Surface("W_plus") = {Con[]};
    EndIf
    If (f == 5)
                          Color PaleGoldenrod {Surface{Con[]};}
                          Physical Surface("V_minus") = {Con[]};
    EndIf
  EndIf
EndFor

Physical Surface("StatorIron") = {StatorIron_[]};
Physical Surface("StatorWedges") = {StatorWedge_[]};
Physical Surface("StatorAirgap") = {StatorAirgapLayer_[]};

Color SteelBlue {Surface{StatorIron_[]};}
Color Black {Surface{StatorWedge_[]};}
Color SkyBlue {Surface{StatorAirgapLayer_[]};}

Physical Line("OuterStator") = {OuterStator_[]};

Physical Line("StatorRight") = {StatorPeriod_Right_[]};
Physical Line("StatorLeft") = {StatorPeriod_Left_[]};

Physical Line("Sliding_Stator") = {Sliding_[]};

Coherence;

show_stator[] = CombinedBoundary{Surface{StatorIron_[]};};
Hide{ Point{Point '*'};}
Hide{ Line{Line '*'};}
Show{ Line{show_stator[]};}

Mesh 2;
```

Appendix C model.sif

```
Header
  Mesh DB "im"
  Results Directory "results"
  Include Path "materials"
End

Simulation
  Max Output Level = 3
  Coordinate System = Cartesian
  Coordinate Scaling = 0.001

  Steady State Max Iterations = 1
  Simulation Type = Steady

!  Mesh Levels = 2
!  Mesh Keep = 1
  Use Mesh Names = Logical True
End

!!##############################################################################
!!################################### Materials ################################
!!##############################################################################

! Air
Material 1
  Relative Permeability = 1
End

! lamination steel
Material 2
  Name = "Iron"
  INCLUDE el_steel_M800_65A
End

!!##############################################################################
!!#########################------ Body Force Section ----#######################
!!##############################################################################

Body Force 1
  Mesh Rotate 3 = Real MATC "10"
End

! for U+
Body Force 2
  Current Density = 2e6
End

! for V+
Body Force 3
  Current Density = -1e6
End

! for W-
Body Force 4
  Current Density = 1e6
End


!------ Skeleton for body section -----
Body 1
  Name = U_plus
  Equation = 1
  Material = 1
  Body Force = 2
End

Body 2
  Name = W_minus
  Equation = 1
  Material = 1
  Body Force = 4
End

Body 3
  Name = V_plus
  Equation = 1
  Material = 1
  Body Force = 3
End

Body 4
  Name = StatorIron
  Equation = 1
  Material = 2
End

Body 5
  Name = StatorWedges
  Equation = 1
  Material = 1
End

Body 6
  Name = StatorAirgap
  Equation = 1
  Material = 1
End

Body 7
```

```
  Name = Bar7
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 8
  Name = Bar8
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 9
  Name = Bar9
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 10
  Name = Bar10
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 11
  Name = RotorIron
  Equation = 1
  Material = 2
  Body Force = 1
End

Body 12
  Name = RotorAirgap
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 13
  Name = Bar1
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 14
  Name = Bar2
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 15
  Name = Bar3
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 16
  Name = Bar4
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 17
  Name = Bar5
  Equation = 1
  Material = 1
  Body Force = 1
End

Body 18
  Name = Bar6
  Equation = 1
  Material = 1
  Body Force = 1
End

!###############################################################################
!#######################------ Solver section -----#######################!
!###############################################################################

Equation 1 :: Active Solvers(2) = 2 3

! Rotation of the rotor at the beginning of each time step
Solver 1
  Exec Solver = Before simulation
  Equation = MeshDeform
  Procedure = "RigidMeshMapper" "RigidMeshMapper"
End

Solver 2
  Exec Solver = Always
  Equation = MgDyn2D
  Variable = A

  Procedure = "MagnetoDynamics2D" "MagnetoDynamics2D"

  Nonlinear System Convergence Tolerance = 1.0e-4
  Nonlinear System Max Iterations = 30
```

```
  Linear System Solver = Direct
  Linear System Iterative Method = MUMPS

!  Linear System Solver = Iterative
!  Linear System Iterative Method = BicgstabL
!  Linear System Symmetric = True
!  Linear System Max Iterations = 200
!  Linear System Preconditioning = ILU2
!  Linear System Convergence Tolerance = 1e-8
!  Linear System Residual Output = 50
End

Solver 3
  Exec Solver = Always
  Equation = CalcFields
  Potential Variable = "A"
  Procedure = "MagnetoDynamics" "MagnetoDynamicsCalcFields"

  Calculate Current Density = Logical True
  Calculate Magnetic Vector Potential = Logical True

  Linear System Solver = Direct
  Linear System Iterative Method = MUMPS

!  Linear System Solver = Iterative
!  Linear System Iterative Method = BicgstabL
!  Linear System Symmetric = True
!  Linear System Max Iterations = 200
!  Linear System Preconditioning = ILU2
!  Linear System Convergence Tolerance = 1e-8
!  Linear System Residual Output = 50
End


Solver 4
  Exec Solver = after simulation
  Equation = "ResultOutput"
  Procedure = "ResultOutputSolve" "ResultOutputSolver"
  Output File Name = case
  Output Directory = results
  Save Geometry Ids = Logical True
  Vtu Format = Logical True

  Vector Field 1 = "magnetic flux density e"
  Vector Field 2 = "current density e"
  Scalar Field 1 = "a"

  ! better visualization of disconinuous fields (interfers with connectivity filter)
!  Discontinuous Bodies = Logical True
End


!------ Skeleton for boundary section -----
Boundary Condition 1
  Name = OuterStator
  A = Real 0
End

Boundary Condition 2
  Name = StatorRight
  Mortar BC = Integer 3
  Mortar BC Static = Logical True
  Anti Radial Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 3
  Name = StatorLeft
End

Boundary Condition 4
  Name = Sliding_Stator
  Mortar BC = Integer 5
  Anti Rotational Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 5
  Name = Sliding_Rotor
End

Boundary Condition 6
  Name = InnerRotor
  A = Real 0
End

Boundary Condition 7
  Name = RotorRight
  Mortar BC = Integer 8
  Mortar BC Static = Logical True
  Anti Radial Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 8
  Name = RotorLeft
End
```

Appendix D transient.sif

```
include transient_params.dat

Header
  Mesh DB "im"
  Results Directory "results"
  Include Path "materials"
End

Simulation
  Max Output Level = 3
  Coordinate System = Cartesian 2D
  Coordinate Scaling = 0.001

  Circuit Model Depth = Real $ l

  Simulation Type = Transient
  Timestepping Method = BDF
  BDF Order = 2

  Output Intervals = 1
  Timestep Sizes = $ 1/f/25
  Timestep Intervals = $ 26*25

!  Mesh Levels = 2
!  Mesh Keep = 1
  Use Mesh Names = Logical True
End

!###############################################################################
!################################### Materials ##################################
!###############################################################################

! Air
Material 1
  Relative Permeability = 1
  Electric Conductivity = 0
End

! lamination steel
Material 2
  Name = "Iron"
  INCLUDE el_steel_M800_65A
  Electric Conductivity = 0
End

! Aluminum
Material 3
  Relative Permeability = 1
!  Electric Conductivity = 36e6 ! [1/(Ohm*m)] cold rotor
  Electric Conductivity = 24e6 ! [1/(Ohm*m)] hot rotor
!  Electric Conductivity = 0
End

! Copper
Material 4
  Relative Permeability = 1
  Electric Conductivity = 48e6
End

!###############################################################################
!########################------ Body Force Section ----#########################
!###############################################################################

Body Force 1
  Name = "Circuit"

  ! U-phase voltage 400/sqrt(3) Vrms at 0 degrees
  U_u = Variable time
    Real MATC "400*sqrt(2)/sqrt(3)*sin(tx(0)*2*pi*f)"

  ! V-phase voltage 400/sqrt(3) Vrms at -120 degrees
  U_v = Variable time
    Real MATC "400*sqrt(2)/sqrt(3)*sin(tx(0)*2*pi*f-2*pi/3)"

  ! W-phase voltage 400/sqrt(3) Vrms at +120 degrees
  U_w = Variable time
    Real MATC "400*sqrt(2)/sqrt(3)*sin(tx(0)*2*pi*f+2*pi/3)"
End

Body Force 7
  Mesh Rotate 3 = Variable time
    Real MATC "180/pi*2*pi*f/pp*(1-slip)*tx"
End

!###############################################################################
!################################### Bodies #####################################
!###############################################################################

Body 1
  Name = Bar1
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End

Body 2
  Name = Bar2
  Equation = 1
  Material = 3
  Body Force = 7
```

```
  Torque Groups = Integer 1
End

Body 3
  Name = Bar3
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End

Body 4
  Name = Bar4
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End

Body 5
  Name = Bar5
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End

Body 6
  Name = Bar6
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End

Body 7
  Name = Bar7
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End

Body 8
  Name = Bar8
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End

Body 9
  Name = Bar9
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End

Body 10
  Name = Bar10
  Equation = 1
  Material = 3
  Body Force = 7
  Torque Groups = Integer 1
End


Body 11
  Name = U_plus
  Equation = 1
  Material = 4
End

Body 12
  Name = W_minus
  Equation = 1
  Material = 4
End

Body 13
  Name = V_plus
  Equation = 1
  Material = 4
End

Body 14
  Name = StatorIron
  Equation = 1
  Material = 2
End

Body 15
  Name = StatorWedges
  Equation = 1
  Material = 1
End

Body 16
  Name = StatorAirgap
  Equation = 1
  Material = 1
End

Body 17
```

```
  Name = RotorIron
  Equation = 1
  Material = 2
  Body Force = 7
  Torque Groups = Integer 1
End

Body 18
  Name = RotorAirgap
  Equation = 1
  Material = 1
  Body Force = 7
  R Inner = Real 0.0620
  R Outer = Real 0.0625
End


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Components !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

include "cage/cage.definitions"

Component 11
  Name = String Uplus
  Body = Integer 11
  Coil Type = String Stranded
  Number of Turns = Real $ Nph/2
  Resistance = Real $ Rs
End

Component 12
  Name = String Vplus
  Body = Integer 13
  Coil Type = String Stranded
  Number of Turns = Real $ Nph/2
  Resistance = Real $ Rs
End

Component 13
  Name = String Wminus
  Body = Integer 12
  Coil Type = String Stranded
  Number of Turns = Real $ Nph/2
  Resistance = Real $ Rs
End

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!------ Solver section -----#####################!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Equation 1 :: Active Solvers(3) = 2 3 4

! Rotation of the rotor at the beginning of each time step
Solver 1
  Exec Solver = Before timestep
  Equation = MeshDeform
  Procedure = "RigidMeshMapper" "RigidMeshMapper"
End

Solver 2
  Exec Solver = Always
  Equation = Circuits
  Variable = X
  Procedure = "CircuitsAndDynamics" "CircuitsAndDynamics"
  No Matrix = Logical True
End

Solver 3
  Exec Solver = Always
  Equation = MgDyn2D
  Variable = A

  Procedure = "MagnetoDynamics2D" "MagnetoDynamics2D"

!!!!!!!!!!!!!!!!!!!  These are for better convergence for parallel
!!!!!!!!!!!!!!!!!!! Might be not optimal
  Stabilize = False
  Partition Local Constraints = Logical True
  Nonlinear System Compute Change in Scaled System = Logical True
  Nonlinear System Convergence Measure = Residual
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  Nonlinear System Convergence Tolerance = 1e-6
  Nonlinear System Max Iterations = 20
  Nonlinear System Min Iterations = 1
  Nonlinear System Relaxation Factor = 1.0
  Nonlinear System Newton After Iterations = 7

  Export Lagrange Multiplier = Logical True

  Linear System Abort Not Converged = False
  Linear System Solver = Iterative
  Linear System Iterative Method = GCR ! GCR !

  Linear System GCR Restart = 500
  Bicgstabl Polynomial Degree = 4
  Linear System Preconditioning = ILU2
  Linear System Max Iterations = 1500
  Linear System Residual Output = Integer 20
  Linear System Convergence Tolerance = 1e-7 ! 2.0e-6

  Mortar BCs Additive = Logical True
!  Linear System Symmetric = True
End
```

```
Solver 4
  Exec Solver = Always
  Equation = CalcFields
  Potential Variable = "A"
  Procedure = "MagnetoDynamics" "MagnetoDynamicsCalcFields"

  ! GNF Torque calculation
  Calculate Nodal Forces = Logical True
  Calculate Magnetic Torque = Logical True

  Calculate Current Density = Logical True
!  Calculate Electric Field = Logical True
!  Calculate Magnetic Field Strength = Logical True
  Calculate Magnetic Vector Potential = Logical True
End

Solver 5
  Exec Solver = Always
  Equation = CircOutput
  Procedure = "CircuitsAndDynamics" "CircuitsOutput"
End

Solver 6
  Exec Solver = after timestep
  Equation = "ResultOutput"
  Procedure = "ResultOutputSolve" "ResultOutputSolver"
  Output File Name = case
  Output Directory = results
  Save Geometry Ids = Logical True
  Vtu Format = Logical True
  Binary Output = Logical True
  Single Precision = Logical True

!  Scalar Field 1 = "magnetic flux density e"
!  Scalar Field 2 = "current density e"
!  Scalar Field 3 = "a"

  ! better visualization of disconinuous fields (interfers with connectivity ParaView filter)
!  Discontinuous Bodies = Logical True
End

Solver 7
  Exec Solver = After timestep
  Equation = scalars
  Procedure = "SaveData" "SaveScalars"
  Filename = transient_results.dat

  Save Coordinates(3,2) =  -0.053  0.053  -0.0428 0.046  -0.07   0.07
!  Exact Coordinates = True
End

!##############################################################################
!########################------ Boundary section -----########################!
!##############################################################################

Boundary Condition 1
  Name = OuterStator
  A = Real 0
End

Boundary Condition 2
  Name = StatorRight
  Mortar BC = Integer 3
  Mortar BC Static = Logical True
  Anti Radial Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 3
  Name = StatorLeft
End

Boundary Condition 4
  Name = Sliding_Stator
  Mortar BC = Integer 8
  Anti Rotational Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 5
  Name = InnerRotor
  A = Real 0
End

Boundary Condition 6
  Name = RotorRight
  Mortar BC = Integer 7
  Mortar BC Static = Logical True
  Anti Radial Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 7
  Name = RotorLeft
End

Boundary Condition 8
  Name = Sliding_Rotor
End
```

Appendix E transient_params.dat

```
$ slip = 0.0333

$ f = 50                            ! [Hz] electrical frequency
$ U = 400       ! [V] rated voltage
$ pp = 2        ! number of polepairs
$ n = 1450            ! [rpm] rated speed

$ l = 0.160     ! [m] stator stack length
$ Rs = 0.94/2     ! [Ohm] stator phase resistance
$ Nph = 256            ! Number of phase turns
$ w_syn = 1500/60*2*pi                              ! [rad/s] mech frequency

!analytically estimated values:
$ L_ew = 0.0062    ! [H] end winding phase inductance
$ R_er = 0.0000016   ! [Ohm] end ring rotor resistance of a section betwen 2 bars
$ L_er = 0.000000012  ! [H] end ring rotor inductance of a section between 2 bars

!-------------------------------------------------------------
! Stator voltage driven circuits for 3 phases
!-------------------------------------------------------------
$ Circuits = 4
!-------------------------------------------------------------
! Phase U_plus
!-------------------------------------------------------------

! init matrices phase U
! init matrices of Ax' + Bx = Source
$ C.1.variables = 6
$ C.1.perm = zeros(C.1.variables)
$ C.1.A = zeros(C.1.variables, C.1.variables)
$ C.1.B = zeros(C.1.variables, C.1.variables)
$ C.1.Mre = zeros(C.1.variables, C.1.variables)
$ C.1.Mim = zeros(C.1.variables, C.1.variables)

! define circuit variables
$ C.1.name.1 = "i_su"
$ C.1.name.2 = "v_su"
$ C.1.name.3 = "i_component(11)"
$ C.1.name.4 = "v_component(11)"
$ C.1.name.5 = "i_ewu"
$ C.1.name.6 = "v_ewu"

! Define sources:
!-----------------
$ C.1.B(0,1) = 1
$ C.1.source.1 = "U_u"

! Circuit equations:
!------------------------
! Kirchoff current law
$ C.1.B(2,0) = -0.5
$ C.1.B(2,2) = 1
$ C.1.B(4,0) = -0.5
$ C.1.B(4,4) = 1


! Kirchoff voltage law
$ C.1.B(1,1) = -0.5
$ C.1.B(1,3) = 1
$ C.1.B(1,5) = 1

!Elemental equations
$ C.1.A(5,4) = L_ew/4
$ C.1.B(5,5) = -1

!-------------------------------------------------------------
! Phase V_plus
!-------------------------------------------------------------

! init matrices of Ax' + Bx = Source
$ C.2.variables = 6
$ C.2.perm = zeros(C.2.variables)
$ C.2.A = zeros(C.2.variables, C.2.variables)
$ C.2.B = zeros(C.2.variables, C.2.variables)
$ C.2.Mre = zeros(C.2.variables, C.2.variables)
$ C.2.Mim = zeros(C.2.variables, C.2.variables)

! define circuit variables
$ C.2.name.1 = "i_sv"
$ C.2.name.2 = "v_sv"
$ C.2.name.3 = "i_component(12)"
$ C.2.name.4 = "v_component(12)"
$ C.2.name.5 = "i_ewv"
$ C.2.name.6 = "v_ewv"

! Define sources:
!-----------------
$ C.2.B(0,1) = 1
$ C.2.source.1 = "U_v"

! Circuit equations:
!------------------------
! Kirchoff current law
$ C.2.B(2,0) = -0.5
$ C.2.B(2,2) = 1
$ C.2.B(4,0) = -0.5
$ C.2.B(4,4) = 1


! Kirchoff voltage law
$ C.2.B(1,1) = -0.5
```

```
$ C.2.B(1,3) = 1
$ C.2.B(1,5) = 1

!Elemental equations
$ C.2.A(5,4) = L_ew/4
$ C.2.B(5,5) = -1

!-------------------------------------------------------------
! Phase W_minus
!-------------------------------------------------------------

! init matrices of Ax' + Bx = Source
$ C.3.variables = 6
$ C.3.perm = zeros(C.3.variables)
$ C.3.A = zeros(C.3.variables, C.3.variables)
$ C.3.B = zeros(C.3.variables, C.3.variables)
$ C.3.Mre = zeros(C.3.variables, C.3.variables)
$ C.3.Mim = zeros(C.3.variables, C.3.variables)

! define circuit variables
$ C.3.name.1 = "i_sw"
$ C.3.name.2 = "v_sw"
$ C.3.name.3 = "i_component(13)"
$ C.3.name.4 = "v_component(13)"
$ C.3.name.5 = "i_eww"
$ C.3.name.6 = "v_eww"

! Define sources:
!-----------------
$ C.3.B(0,1) = 1
$ C.3.source.1 = "U_w"

! Circuit equations:
!------------------------
! Kirchoff current law
$ C.3.B(2,0) = 0.5
$ C.3.B(2,2) = 1
$ C.3.B(4,0) = -0.5
$ C.3.B(4,4) = 1

! Kirchoff voltage law
$ C.3.B(1,1) = -0.5
$ C.3.B(1,3) = -1
$ C.3.B(1,5) = 1

!Elemental equations
$ C.3.A(5,4) = L_ew/4
$ C.3.B(5,5) = -1
```

Appendix F cage_generator.py

```
# Elmer circuit equations generator for a cage winding taking into account periodicity
# Author: P.Ponomarev
# July 2016
# changelog:
# version 1.3 (03.2017) by PP:
#       - added offset of the body numbers boffset
#
from __future__ import print_function

# Settings:
ns = 1
nob = 10             # number of rotor bars simulated
boffset = 1         #number of the first bar body
antiperiodic = 1     # periodic or antiperiodic boundary
cn = 4               # circuit number which describes the rotor bars
ctype = "Stranded"   # Coil type Massive/Stranded
OUTFILE = 'cage.definitions'

# Rotor circuit
# Bar 1 to Bar N are FEM components of the modelled domain
# L_N and R_N are bar-to-bar inductance and resistance of the rotor end rings
# For 1-pole model (antiperiodic):
#    terminal 1 is connected to 2' and 2 is connected to 1'
# For 2-pole model (periodic):
#    terminal 1 is connected to 1' and 2 is connected to 2'
#                  (i,v)
#     1'           (0,1)           2'
#     O       + _________  -  I_bar O
#     |_______|  Bar 1  |____\____|
#   + |       |_________|    /    |
#     C                           C +
#     C  L_1l   _____________     C  L_1r
#     C        |   U_loop    |    C
#      | (4,5) |             |     |   (2,3)
#     <        |            \|/    <
#     <  R_1l  |___          V     <  R_1r
#     <                           <
#   - |       + _________  -      |-
#     |_______|  Bar 2  |_________|
#     |       |_________|         |
#     C             (6,7)         C
#     C  L_2l                     C  L_2r
#     C                           C
#      |   (10,11)                 |      (8,9)
#     <                           <
#     <  R_2l                     <  R_2r
#     <                           <
#      |                           |
#
#     ...
#                  _________
#      |_______|  Bar N  |_________|
#      |       |_________|         |
#     C                           C
#     C  L_Nl                     C  L_Nr
#     C                           C
#      |                           |
#     <                           <
#     <  R_Nl                     <  R_Nr
#     <                           <
#      |                           |
#      O                          O
#      1                          2

barstxt = ""

###############################################################################
### Filling components section
###############################################################################
# Coil Type can be Massive or Stranded
# assuming that rotor bar bodies are numbered
# consequently starting from 1 onwards to N, where 1 and N are closest to
# periodic boundaries bars:
for nbar in range(1,nob+1):
    s = "Component " + str(nbar) + "\n" + \
        "  Name = String RB" + str(nbar) + "\n" + \
        "  Body = Integer " + str(nbar+boffset-1) + "\n" + \
        "  Coil Type = String "+ ctype + "\n" + \
        "  Number of Turns = Real 1" + "\n" + \
        "End" + "\n\n"
    barstxt = barstxt + s
###############################################################################
### Declare variables
###############################################################################
# first, the dimensions of the variable arrays are declared
s = "!------------------------------------------------------------\n" + \
    "! Equations for " + str(nob) + " rotor bars\n" + \
    "!------------------------------------------------------------\n\n" + \
    "$ C." + str(cn) + ".source.1 = 0\n\n" + \
    "! init matrices of Ax' + Bx = Source\n" + \
    "$ C." + str(cn) + ".variables = " + str(nob*3*2)+ "\n" + \
    "$ C." + str(cn) + ".perm = zeros(" + str(nob*3*2)+ ")\n" + \
    "$ C." + str(cn) + ".A = zeros(" + str(nob*3*2) + ", " + str(nob*3*2) + ")\n" + \
    "$ C." + str(cn) + ".B = zeros(" + str(nob*3*2) + ", " + str(nob*3*2) + ")\n" + \
    "$ C." + str(cn) + ".Mre = zeros(" + str(nob*3*2) + ", " + str(nob*3*2) + ")\n" + \
    "$ C." + str(cn) + ".Mim = zeros(" + str(nob*3*2) + ", " + str(nob*3*2) + ")\n" + \
    "! define circuit variables\n\n"

barstxt = barstxt + s

# then, each variable receives its unique name
```

```
# each component and element is described by 2 circuit variables - "u" and "i"
# each bar is associated with 2 sections of the end ring - left (l) and right (r)
# each section is described by one single element of the circuit possesing R and L.
for nbar in range(0,nob):
    s = "$ C." + str(cn) + ".name." + str(nbar*6 + 1) + " = \"i_component(" + str(nbar+1) + ")\"\n" + \
        "$ C." + str(cn) + ".name." + str(nbar*6 + 2) + " = \"v_component(" + str(nbar+1) + ")\"\n" + \
        "$ C." + str(cn) + ".name." + str(nbar*6 + 3) + " = \"i_r" + str(nbar+1) + "\"\n" + \
        "$ C." + str(cn) + ".name." + str(nbar*6 + 4) + " = \"v_r" + str(nbar+1) + "\"\n" + \
        "$ C." + str(cn) + ".name." + str(nbar*6 + 5) + " = \"i_l" + str(nbar+1) + "\"\n" + \
        "$ C." + str(cn) + ".name." + str(nbar*6 + 6) + " = \"v_l" + str(nbar+1) + "\"\n\n\n"
    barstxt = barstxt + s

################################################################################
### Kirchoff voltage law
################################################################################
# describes voltages in each loop between two bars. Hence, each circuit segment
# contains 4 components(elements)
# loops directed clockwise
s = "! Kirchoff voltage law\n\n"
barstxt = barstxt + s

for nbar in range(0,nob-1):
    s = "!Bar" + str(nbar+1) + "\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+2) + "," + str(nbar*6+1) + ") = 1/" + str(ns) + "\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+2) + "," + str(nbar*6+3) + ") = 1\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+2) + "," + str(nbar*6+5) + ") = -1\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+2) + "," + str(nbar*6+7) + ") = -1/" + str(ns) + "\n\n"
    barstxt = barstxt + s

# last bar includes periodicity definition
s = "!Bar" + str(nob) + "\n" + \
    "$ C." + str(cn) + ".B(" + str((nob-1)*6+2) + "," + str((nob-1)*6+1) + ") = 1/" + str(ns) + "\n" + \
    "$ C." + str(cn) + ".B(" + str((nob-1)*6+2) + "," + str((nob-1)*6+3) + ") = 1\n" + \
    "$ C." + str(cn) + ".B(" + str((nob-1)*6+2) + "," + str((nob-1)*6+5) + ") = -1\n" + \
    "$ C." + str(cn) + ".B(" + str((nob-1)*6+2) + "," + str(1) + ") = " + str(1 if antiperiodic==1 else -1) +"/" + str(ns) +
"\n\n\n"
barstxt = barstxt + s

################################################################################
### Kirchoff current law
################################################################################
# each bar is connected to two knots -- left and right
s = "! Kirchoff current law\n\n"
barstxt = barstxt + s

# bar 1 knots contain periodicity information

s = "!Bar" + str(1) + " right knot\n" + \
    "$ C." + str(cn) + ".B(" + str(0+0) + "," + str(0+0) + ") = 1\n" + \
    "$ C." + str(cn) + ".B(" + str(0+0) + "," + str(nob*6-(2 if antiperiodic == 1 else 4)) + ") = 1\n" + \
    "$ C." + str(cn) + ".B(" + str(0+0) + "," + str(0+2) + ") = -1\n" + \
    "!Bar" + str(1) + " left knot\n" + \
    "$ C." + str(cn) + ".B(" + str(0+4) + "," + str(0+4) + ") = -1\n" + \
    "$ C." + str(cn) + ".B(" + str(0+4) + "," + str(nob*6-(4 if antiperiodic == 1 else 2)) + ") = 1\n" + \
    "$ C." + str(cn) + ".B(" + str(0+4) + "," + str(0+0) + ") = -1\n\n"
barstxt = barstxt + s

# other bars are composed similarly
for nbar in range(1,nob):
    s = "!Bar" + str(nbar+1) + " right knot\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+0) + "," + str(nbar*6+0) + ") = 1\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+0) + "," + str(nbar*6-4) + ") = 1\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+0) + "," + str(nbar*6+2) + ") = -1\n" + \
        "!Bar" + str(nbar+1) + " left knot\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+4) + "," + str(nbar*6+4) + ") = -1\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+4) + "," + str(nbar*6-2) + ") = 1\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+4) + "," + str(nbar*6+0) + ") = -1\n\n"
    barstxt = barstxt + s

################################################################################
### Elemental equations
################################################################################

# these equations describe R and L elements in the circuit
# v = vr+vl
# v -iR - Li' = 0

s = "! Elemental equations\n\n"
barstxt = barstxt + s

for nbar in range(0,nob):
    s = "$ C." + str(cn) + ".B(" + str(nbar*6+3) + "," + str(nbar*6+3) + ") = -1\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+3) + "," + str(nbar*6+2) + ") = R_er\n" + \
        "$ C." + str(cn) + ".A(" + str(nbar*6+3) + "," + str(nbar*6+2) + ") = L_er\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+5) + "," + str(nbar*6+5) + ") = -1\n" + \
        "$ C." + str(cn) + ".B(" + str(nbar*6+5) + "," + str(nbar*6+4) + ") = R_er\n" + \
        "$ C." + str(cn) + ".A(" + str(nbar*6+5) + "," + str(nbar*6+4) + ") = L_er\n\n"

    barstxt = barstxt + s

with open(OUTFILE, 'w+') as f:
    f.write(barstxt)

print('Cage circuit equations for circuit number', cn,
        'with', ns, 'slices',
        'for', nob, 'bars with',
        'antiperiodic' if antiperiodic == 1 else 'periodic',
        'boundary conditions are saved to', OUTFILE)
```

Appendix G cage.definitions

```
Component 1
  Name = String RB1
  Body = Integer 1
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 2
  Name = String RB2
  Body = Integer 2
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 3
  Name = String RB3
  Body = Integer 3
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 4
  Name = String RB4
  Body = Integer 4
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 5
  Name = String RB5
  Body = Integer 5
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 6
  Name = String RB6
  Body = Integer 6
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 7
  Name = String RB7
  Body = Integer 7
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 8
  Name = String RB8
  Body = Integer 8
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 9
  Name = String RB9
  Body = Integer 9
  Coil Type = String Stranded
  Number of Turns = Real 1
End

Component 10
  Name = String RB10
  Body = Integer 10
  Coil Type = String Stranded
  Number of Turns = Real 1
End

!------------------------------------------------------------
! Equations for 10 rotor bars
!------------------------------------------------------------

$ C.4.source.1 = 0

! init matrices of Ax' + Bx = Source
$ C.4.variables = 60
$ C.4.perm = zeros(60)
$ C.4.A = zeros(60, 60)
$ C.4.B = zeros(60, 60)
$ C.4.Mre = zeros(60, 60)
$ C.4.Mim = zeros(60, 60)
! define circuit variables

$ C.4.name.1 = "i_component(1)"
$ C.4.name.2 = "v_component(1)"
$ C.4.name.3 = "i_r1"
$ C.4.name.4 = "v_r1"
$ C.4.name.5 = "i_l1"
$ C.4.name.6 = "v_l1"


$ C.4.name.7 = "i_component(2)"
$ C.4.name.8 = "v_component(2)"
$ C.4.name.9 = "i_r2"
$ C.4.name.10 = "v_r2"
$ C.4.name.11 = "i_l2"
$ C.4.name.12 = "v_l2"
```

```
$ C.4.name.13 = "i_component(3)"
$ C.4.name.14 = "v_component(3)"
$ C.4.name.15 = "i_r3"
$ C.4.name.16 = "v_r3"
$ C.4.name.17 = "i_l3"
$ C.4.name.18 = "v_l3"


$ C.4.name.19 = "i_component(4)"
$ C.4.name.20 = "v_component(4)"
$ C.4.name.21 = "i_r4"
$ C.4.name.22 = "v_r4"
$ C.4.name.23 = "i_l4"
$ C.4.name.24 = "v_l4"


$ C.4.name.25 = "i_component(5)"
$ C.4.name.26 = "v_component(5)"
$ C.4.name.27 = "i_r5"
$ C.4.name.28 = "v_r5"
$ C.4.name.29 = "i_l5"
$ C.4.name.30 = "v_l5"


$ C.4.name.31 = "i_component(6)"
$ C.4.name.32 = "v_component(6)"
$ C.4.name.33 = "i_r6"
$ C.4.name.34 = "v_r6"
$ C.4.name.35 = "i_l6"
$ C.4.name.36 = "v_l6"


$ C.4.name.37 = "i_component(7)"
$ C.4.name.38 = "v_component(7)"
$ C.4.name.39 = "i_r7"
$ C.4.name.40 = "v_r7"
$ C.4.name.41 = "i_l7"
$ C.4.name.42 = "v_l7"


$ C.4.name.43 = "i_component(8)"
$ C.4.name.44 = "v_component(8)"
$ C.4.name.45 = "i_r8"
$ C.4.name.46 = "v_r8"
$ C.4.name.47 = "i_l8"
$ C.4.name.48 = "v_l8"


$ C.4.name.49 = "i_component(9)"
$ C.4.name.50 = "v_component(9)"
$ C.4.name.51 = "i_r9"
$ C.4.name.52 = "v_r9"
$ C.4.name.53 = "i_l9"
$ C.4.name.54 = "v_l9"


$ C.4.name.55 = "i_component(10)"
$ C.4.name.56 = "v_component(10)"
$ C.4.name.57 = "i_r10"
$ C.4.name.58 = "v_r10"
$ C.4.name.59 = "i_l10"
$ C.4.name.60 = "v_l10"


! Kirchoff voltage law

!Bar1
$ C.4.B(2,1) = 1/1
$ C.4.B(2,3) = 1
$ C.4.B(2,5) = -1
$ C.4.B(2,7) = -1/1

!Bar2
$ C.4.B(8,7) = 1/1
$ C.4.B(8,9) = 1
$ C.4.B(8,11) = -1
$ C.4.B(8,13) = -1/1

!Bar3
$ C.4.B(14,13) = 1/1
$ C.4.B(14,15) = 1
$ C.4.B(14,17) = -1
$ C.4.B(14,19) = -1/1

!Bar4
$ C.4.B(20,19) = 1/1
$ C.4.B(20,21) = 1
$ C.4.B(20,23) = -1
$ C.4.B(20,25) = -1/1

!Bar5
$ C.4.B(26,25) = 1/1
$ C.4.B(26,27) = 1
$ C.4.B(26,29) = -1
$ C.4.B(26,31) = -1/1

!Bar6
$ C.4.B(32,31) = 1/1
$ C.4.B(32,33) = 1
$ C.4.B(32,35) = -1
$ C.4.B(32,37) = -1/1

!Bar7
$ C.4.B(38,37) = 1/1
$ C.4.B(38,39) = 1
```

```
$ C.4.B(38,41) = -1
$ C.4.B(38,43) = -1/1

!Bar8
$ C.4.B(44,43) = 1/1
$ C.4.B(44,45) = 1
$ C.4.B(44,47) = -1
$ C.4.B(44,49) = -1/1

!Bar9
$ C.4.B(50,49) = 1/1
$ C.4.B(50,51) = 1
$ C.4.B(50,53) = -1
$ C.4.B(50,55) = -1/1

!Bar10
$ C.4.B(56,55) = 1/1
$ C.4.B(56,57) = 1
$ C.4.B(56,59) = -1
$ C.4.B(56,1) = 1/1


! Kirchoff current law

!Bar1 right knot
$ C.4.B(0,0) = 1
$ C.4.B(0,58) = 1
$ C.4.B(0,2) = -1
!Bar1 left knot
$ C.4.B(4,4) = -1
$ C.4.B(4,56) = 1
$ C.4.B(4,0) = -1

!Bar2 right knot
$ C.4.B(6,6) = 1
$ C.4.B(6,2) = 1
$ C.4.B(6,8) = -1
!Bar2 left knot
$ C.4.B(10,10) = -1
$ C.4.B(10,4) = 1
$ C.4.B(10,6) = -1

!Bar3 right knot
$ C.4.B(12,12) = 1
$ C.4.B(12,8) = 1
$ C.4.B(12,14) = -1
!Bar3 left knot
$ C.4.B(16,16) = -1
$ C.4.B(16,10) = 1
$ C.4.B(16,12) = -1

!Bar4 right knot
$ C.4.B(18,18) = 1
$ C.4.B(18,14) = 1
$ C.4.B(18,20) = -1
!Bar4 left knot
$ C.4.B(22,22) = -1
$ C.4.B(22,16) = 1
$ C.4.B(22,18) = -1

!Bar5 right knot
$ C.4.B(24,24) = 1
$ C.4.B(24,20) = 1
$ C.4.B(24,26) = -1
!Bar5 left knot
$ C.4.B(28,28) = -1
$ C.4.B(28,22) = 1
$ C.4.B(28,24) = -1

!Bar6 right knot
$ C.4.B(30,30) = 1
$ C.4.B(30,26) = 1
$ C.4.B(30,32) = -1
!Bar6 left knot
$ C.4.B(34,34) = -1
$ C.4.B(34,28) = 1
$ C.4.B(34,30) = -1

!Bar7 right knot
$ C.4.B(36,36) = 1
$ C.4.B(36,32) = 1
$ C.4.B(36,38) = -1
!Bar7 left knot
$ C.4.B(40,40) = -1
$ C.4.B(40,34) = 1
$ C.4.B(40,36) = -1

!Bar8 right knot
$ C.4.B(42,42) = 1
$ C.4.B(42,38) = 1
$ C.4.B(42,44) = -1
!Bar8 left knot
$ C.4.B(46,46) = -1
$ C.4.B(46,40) = 1
$ C.4.B(46,42) = -1

!Bar9 right knot
$ C.4.B(48,48) = 1
$ C.4.B(48,44) = 1
$ C.4.B(48,50) = -1
!Bar9 left knot
$ C.4.B(52,52) = -1
$ C.4.B(52,46) = 1
$ C.4.B(52,48) = -1
```

```
!Bar10 right knot
$ C.4.B(54,54) = 1
$ C.4.B(54,50) = 1
$ C.4.B(54,56) = -1
!Bar10 left knot
$ C.4.B(58,58) = -1
$ C.4.B(58,52) = 1
$ C.4.B(58,54) = -1

! Elemental equations

$ C.4.B(3,3) = -1
$ C.4.B(3,2) = R_er
$ C.4.A(3,2) = L_er
$ C.4.B(5,5) = -1
$ C.4.B(5,4) = R_er
$ C.4.A(5,4) = L_er

$ C.4.B(9,9) = -1
$ C.4.B(9,8) = R_er
$ C.4.A(9,8) = L_er
$ C.4.B(11,11) = -1
$ C.4.B(11,10) = R_er
$ C.4.A(11,10) = L_er

$ C.4.B(15,15) = -1
$ C.4.B(15,14) = R_er
$ C.4.A(15,14) = L_er
$ C.4.B(17,17) = -1
$ C.4.B(17,16) = R_er
$ C.4.A(17,16) = L_er

$ C.4.B(21,21) = -1
$ C.4.B(21,20) = R_er
$ C.4.A(21,20) = L_er
$ C.4.B(23,23) = -1
$ C.4.B(23,22) = R_er
$ C.4.A(23,22) = L_er

$ C.4.B(27,27) = -1
$ C.4.B(27,26) = R_er
$ C.4.A(27,26) = L_er
$ C.4.B(29,29) = -1
$ C.4.B(29,28) = R_er
$ C.4.A(29,28) = L_er

$ C.4.B(33,33) = -1
$ C.4.B(33,32) = R_er
$ C.4.A(33,32) = L_er
$ C.4.B(35,35) = -1
$ C.4.B(35,34) = R_er
$ C.4.A(35,34) = L_er

$ C.4.B(39,39) = -1
$ C.4.B(39,38) = R_er
$ C.4.A(39,38) = L_er
$ C.4.B(41,41) = -1
$ C.4.B(41,40) = R_er
$ C.4.A(41,40) = L_er

$ C.4.B(45,45) = -1
$ C.4.B(45,44) = R_er
$ C.4.A(45,44) = L_er
$ C.4.B(47,47) = -1
$ C.4.B(47,46) = R_er
$ C.4.A(47,46) = L_er

$ C.4.B(51,51) = -1
$ C.4.B(51,50) = R_er
$ C.4.A(51,50) = L_er
$ C.4.B(53,53) = -1
$ C.4.B(53,52) = R_er
$ C.4.A(53,52) = L_er

$ C.4.B(57,57) = -1
$ C.4.B(57,56) = R_er
$ C.4.A(57,56) = L_er
$ C.4.B(59,59) = -1
$ C.4.B(59,58) = R_er
$ C.4.A(59,58) = L_er
```

Appendix H harmonic.sif

```
include harmonic_params.dat

Header
  Mesh DB "im"
  Results Directory "results"
  Include Path "materials"
End

Simulation
  Max Output Level = 3
  Coordinate System = Cartesian 2D
  Coordinate Scaling = 0.001

  Simulation Type = Steady
  Steady State Max Iterations = 2

  Circuit Model Depth = Real $ l
  Angular Frequency = Real $ 2*pi*f

  Mesh Levels = 2
  Mesh Keep = 1
  Use Mesh Names = Logical True
End

!##############################################################################
!################################### Materials ################################
!##############################################################################
! Air
Material 1
  Relative Permeability = 1
  Electric Conductivity = 0
End

! lamination steel
Material 2
  Name = "Iron"
  Relative Permeability = 300  !Typical value in harmonic analysis is in 50-350
  Electric Conductivity = 0
End

! Aluminum
Material 3
  Relative Permeability = 1
  Electric Conductivity = Real $ 24000000*slip
End

! Copper
Material 4
  Relative Permeability = 1
  Electric Conductivity = 48e6
End

!##############################################################################
!------------------------ Body Force Section ----#####################
!##############################################################################
Body Force 1
  Name = "Circuit"

  ! U-phase voltage peak phase voltage at 0 degrees
  U_u Re = Real $ sqrt(2)/sqrt(3)*400*cos(0)
  U_u Im = Real $ sqrt(2)/sqrt(3)*400*sin(0)

  ! V-phase voltage 400/sqrt(3) Vrms at -120 degrees
  U_v Re = Real $ sqrt(2)/sqrt(3)*400*cos(-120*pi/180)
  U_v Im = Real $ sqrt(2)/sqrt(3)*400*sin(-120*pi/180)

  ! W-phase voltage 400/sqrt(3) Vrms at +120 degrees
  U_w Re = Real $ sqrt(2)/sqrt(3)*400*cos(120*pi/180)
  U_w Im = Real $ sqrt(2)/sqrt(3)*400*sin(120*pi/180)
End

Body Force 2
  Mesh Rotate 3 = Real $ r_angle
End

!##############################################################################
!################################### Bodies ###################################
!##############################################################################
Body 1
  Name = Bar1
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 2
  Name = Bar2
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 3
  Name = Bar3
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 4
  Name = Bar4
```

```
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 5
  Name = Bar5
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 6
  Name = Bar6
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 7
  Name = Bar7
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 8
  Name = Bar8
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 9
  Name = Bar9
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 10
  Name = Bar10
  Equation = 1
  Material = 3
  Body Force = 2
  Torque Groups = Integer 1
End
Body 11
  Name = U_plus
  Equation = 1
  Material = 4
End
Body 12
  Name = W_minus
  Equation = 1
  Material = 4
End
Body 13
  Name = V_plus
  Equation = 1
  Material = 4
End
Body 14
  Name = StatorIron
  Equation = 1
  Material = 2
End
Body 15
  Name = StatorWedges
  Equation = 1
  Material = 1
End
Body 16
  Name = StatorAirgap
  Equation = 1
  Material = 1
End
Body 17
  Name = RotorIron
  Equation = 1
  Material = 2
  Body Force = 2
  Torque Groups = Integer 1
End
Body 18
  Name = RotorAirgap
  Equation = 1
  Material = 1
  Body Force = 2
  R Inner = Real 0.0620
  R Outer = Real 0.0625
End

!#############################################################################
!################################### Components ##############################
!#############################################################################
include "cage/cage.definitions"

Component 11
  Name = String Uplus
  Body = Integer 11
  Coil Type = String Stranded
  Number of Turns = Real $ Nph/2
  Resistance = Real $ Rs
End
```

```
Component 12
  Name = String Vplus
  Body = Integer 13
  Coil Type = String Stranded
  Number of Turns = Real $ Nph/2
  Resistance = Real $ Rs
End

Component 13
  Name = String Wminus
  Body = Integer 12
  Coil Type = String Stranded
  Number of Turns = Real $ Nph/2
  Resistance = Real $ Rs
End

!###############################################################################
!########################------ Solver section -----#######################!
!###############################################################################
Equation 1
  Active Solvers(3) = 2 3 4
End

Solver 1
  Exec Solver = Before All
  Equation = MeshDeform
  Procedure = "RigidMeshMapper" "RigidMeshMapper"
End

Solver 2
  Exec Solver = Always
  Equation = Circ
  Variable = X
  Procedure = "CircuitsAndDynamics" "CircuitsAndDynamicsHarmonic"
End

Solver 3
  Exec Solver = Always
  Equation = MgDyn2D
  Variable = A[A re:1 A im:1]
  Procedure = "MagnetoDynamics2D" "MagnetoDynamics2DHarmonic"

!!!!!!!!!!!!!!!!!!!  These are for better convergence for parallel
  Stabilize = False
  Partition Local Constraints = Logical True
  Nonlinear System Compute Change in Scaled System = Logical True
  Nonlinear System Convergence Measure = Residual
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  Nonlinear System Convergence Tolerance = 1e-6
  Nonlinear System Max Iterations = 20
  Nonlinear System Min Iterations = 1
  Nonlinear System Relaxation Factor = 1.0
  Nonlinear System Newton After Iterations = 7

  Export Lagrange Multiplier = Logical True

  Linear System Abort Not Converged = False
  Linear System Solver = Iterative

  Linear System Complex = Logical True
  Linear System Iterative Method = bicgstabl
  Linear System GCR Restart = 1000
  Bicgstabl Polynomial Degree = 4

  Linear System Preconditioning = Ilu6
  Linear System Max Iterations = 1500
  Linear System Residual Output = Integer 20
  Linear System Convergence Tolerance = 1.0e-7 ! 2.0e-6

  Linear System Symmetric = True
End

Solver 4
   Equation = CompB
   Exec Solver = Always
   Variable = -nooutput temp
   Exported Variable 1 = B[B re:2 B im:2]
   Target Variable="A"
   Procedure = "MagnetoDynamics2D" "bSolver"

  ! GNF Torque calculation
  Calculate Nodal Forces = Logical True
  Calculate Magnetic Torque = Logical True

   Discontinuous Galerkin = Logical True
   Calculate Joule Heating = Logical True
   Calculate Current Density = Logical True
!   Calculate Body Current = Logical True

   Linear System Solver = Iterative
   Linear System Iterative Method = BicgstabL
   Linear System Symmetric = True
   Linear System Max Iterations = 100
   Linear System Preconditioning = None
   Linear System Convergence Tolerance = 1e-6
End

Solver 5
  Exec Solver = Always
  Equation = CircOutput
  Procedure = "CircuitsAndDynamics" "CircuitsOutput"
End
```

```
Solver 6
  Exec Solver = After simulation
  Equation = "ResultOutput"
  Procedure = "ResultOutputSolve" "ResultOutputSolver"
  Output File Name = "harmonic"
  Output Directory = results
  Save Geometry Ids = Logical True
  Vtu Format = Logical True

  ! better visualization of disconinuous fields
  !Discontinuous Bodies = Logical True
End

Solver 7
  Exec Solver = After simulation
  Equation = scalars
  Procedure = "SaveData" "SaveScalars"
  Filename = harmonic_results.dat
End

!##############################################################################
!------------------------ Boundary section ----------------------------!
!##############################################################################
Boundary Condition 1
  Name = OuterStator
  A re = Real 0
  A im = Real 0
End

Boundary Condition 2
  Name = StatorRight
  Mortar BC = Integer 3
  Mortar BC Static = Logical True
  Anti Radial Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 3
  Name = StatorLeft
End

Boundary Condition 4
  Name = Sliding_Stator
  Mortar BC = Integer 8
  Anti Rotational Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 5
  Name = InnerRotor
  A re = Real 0
  A im = Real 0
End

Boundary Condition 6
  Name = RotorRight
  Mortar BC = Integer 7
  Mortar BC Static = Logical True
  Anti Radial Projector = Logical True
  Galerkin Projector = Logical True
End

Boundary Condition 7
  Name = RotorLeft
End

Boundary Condition 8
  Name = Sliding_Rotor
End
```

Appendix I harmonic_params.dat

```
$ slip = 0.0333
$ r_angle = 0.0

$ f = 50                            ! [Hz] electrical frequency
$ U = 400       ! [V] rated voltage
$ pp = 2        ! number of polepairs
$ n = 1450            ! [rpm] rated speed
$ I = 12        ! [A] rated current
$ I0 = 6.7      ! [A] no-load current
$ l = 0.160     ! [m] stator stack length
$ Rs = 0.94/2     ! [Ohm] stator phase resistance
$ L_ew = 0.0062    ! [H] end winding phase inductance

$ R_er = 0.0000016/slip    ! [Ohm] bar-2-bar end ring rotor resistance
$ L_er = 0.000000012  ! [H] bar-2-bar end ring rotor inductance
$ Nph = 256            ! Number of phase turns

$ w_syn = 1500/60*2*pi                                ! [rad/s] mech frequency

!-------------------------------------------------------------
! Stator voltage driven circuits for 3 phases
!-------------------------------------------------------------
$ Circuits = 4
!-------------------------------------------------------------
! Phase U_plus
!-------------------------------------------------------------

! init matrices phase U
! init matrices of Ax' + Bx = Source
$ C.1.variables = 6
$ C.1.perm = zeros(C.1.variables)
$ C.1.A = zeros(C.1.variables, C.1.variables)
$ C.1.B = zeros(C.1.variables, C.1.variables)
$ C.1.Mre = zeros(C.1.variables, C.1.variables)
$ C.1.Mim = zeros(C.1.variables, C.1.variables)

! define circuit variables
$ C.1.name.1 = "i_su"
$ C.1.name.2 = "v_su"
$ C.1.name.3 = "i_component(11)"
$ C.1.name.4 = "v_component(11)"
$ C.1.name.5 = "i_ewu"
$ C.1.name.6 = "v_ewu"

! Define sources:
!-----------------
$ C.1.B(0,1) = 1
$ C.1.source.1 = "U_u"

! Circuit equations:
!------------------------
! Kirchoff current law
$ C.1.B(2,0) = -0.5
$ C.1.B(2,2) = 1
$ C.1.B(4,0) = -0.5
$ C.1.B(4,4) = 1

! Kirchoff voltage law
$ C.1.B(1,1) = -0.5
$ C.1.B(1,3) = 1
$ C.1.B(1,5) = 1

!Elemental equations
$ C.1.A(5,4) = L_ew/4
$ C.1.B(5,5) = -1

!-------------------------------------------------------------
! Phase V_plus
!-------------------------------------------------------------
! init matrices of Ax' + Bx = Source
$ C.2.variables = 6
$ C.2.perm = zeros(C.2.variables)
$ C.2.A = zeros(C.2.variables, C.2.variables)
$ C.2.B = zeros(C.2.variables, C.2.variables)
$ C.2.Mre = zeros(C.2.variables, C.2.variables)
$ C.2.Mim = zeros(C.2.variables, C.2.variables)

! define circuit variables
$ C.2.name.1 = "i_sv"
$ C.2.name.2 = "v_sv"
$ C.2.name.3 = "i_component(12)"
$ C.2.name.4 = "v_component(12)"
$ C.2.name.5 = "i_ewv"
$ C.2.name.6 = "v_ewv"

! Define sources:
!-----------------
$ C.2.B(0,1) = 1
$ C.2.source.1 = "U_v"

! Circuit equations:
!------------------------
! Kirchoff current law
$ C.2.B(2,0) = -0.5
$ C.2.B(2,2) = 1
$ C.2.B(4,0) = -0.5
$ C.2.B(4,4) = 1


! Kirchoff voltage law
$ C.2.B(1,1) = -0.5
```

```
$ C.2.B(1,3) = 1
$ C.2.B(1,5) = 1

!Elemental equations
$ C.2.A(5,4) = L_ew/4
$ C.2.B(5,5) = -1

!-------------------------------------------------------------
! Phase W_minus
!-------------------------------------------------------------

! init matrices of Ax' + Bx = Source
$ C.3.variables = 6
$ C.3.perm = zeros(C.3.variables)
$ C.3.A = zeros(C.3.variables, C.3.variables)
$ C.3.B = zeros(C.3.variables, C.3.variables)
$ C.3.Mre = zeros(C.3.variables, C.3.variables)
$ C.3.Mim = zeros(C.3.variables, C.3.variables)

! define circuit variables
$ C.3.name.1 = "i_sw"
$ C.3.name.2 = "v_sw"
$ C.3.name.3 = "i_component(13)"
$ C.3.name.4 = "v_component(13)"
$ C.3.name.5 = "i_eww"
$ C.3.name.6 = "v_eww"

! Define sources:
!-----------------
$ C.3.B(0,1) = 1
$ C.3.source.1 = "U_w"

! Circuit equations:
!------------------------
! Kirchoff current law
$ C.3.B(2,0) = 0.5
$ C.3.B(2,2) = 1
$ C.3.B(4,0) = -0.5
$ C.3.B(4,4) = 1

! Kirchoff voltage law
$ C.3.B(1,1) = -0.5
$ C.3.B(1,3) = -1
$ C.3.B(1,5) = 1

!Elemental equations
$ C.3.A(5,4) = L_ew/4
$ C.3.B(5,5) = -1
```

Appendix J Kinematics.f90

```
! This solver dedicated for single kinematic equation
! Jr * dw/dt = Te - Tload
! This functionality might be included in future to MagnetDynamics.F90 module
! using "group 1 torque" by default

SUBROUTINE Kinematics_init( Model,Solver,dt,TransientSimulation )
!------------------------------------------------------------------------------
  USE DefUtils
  IMPLICIT NONE
!------------------------------------------------------------------------------
  TYPE(Solver_t) :: Solver       !< Linear & nonlinear equation solver options
  TYPE(Model_t) :: Model         !< All model information (mesh, materials, BCs, etc...)
  REAL(KIND=dp) :: dt            !< Timestep size for time dependent simulations
  LOGICAL :: TransientSimulation !< Steady state or transient simulation
!------------------------------------------------------------------------------
  TYPE(ValueList_t), POINTER :: Params

  Params => Solver % Values

  ! When we introduce the variables in this way the variables are created
  ! so that they exist when the proper simulation cycle starts.
  ! This also keeps the command file cleaner.
  CALL ListAddString( Params,'Exported Variable 1', '-global Rotor Angle')
  CALL ListAddString( Params,'Exported Variable 2', '-global Rotor Velo')
  Solver % Values => Params

!------------------------------------------------------------------------------
END SUBROUTINE Kinematics_init
!------------------------------------------------------------------------------


!------------------------------------------------------------------------------
SUBROUTINE Kinematics( Model,Solver,dt,TransientSimulation )
!------------------------------------------------------------------------------
  USE DefUtils
  IMPLICIT NONE
!------------------------------------------------------------------------------
  TYPE(Solver_t) :: Solver       !< Linear & nonlinear equation solver options
  TYPE(Model_t) :: Model         !< All model information (mesh, materials, BCs, etc...)
  REAL(KIND=dp) :: dt            !< Timestep size for time dependent simulations
  LOGICAL :: TransientSimulation !< Steady state or transient simulation
!------------------------------------------------------------------------------
! Local variables
!------------------------------------------------------------------------------
  LOGICAL :: Found, First=.TRUE.

  TYPE(Variable_t), POINTER, SAVE :: AngVar, VeloVar
  TYPE(Variable_t), POINTER :: TimeVar
  Real(KIND=dp) :: Time

  REAL(KIND=dp):: ang0=0._dp, velo0=0._dp
  REAL(KIND=dp), TARGET :: torq,imom=0._dp,tscale=1._dp,Tload=0._dp,ang=0._dp,velo=0._dp

  integer :: i
!------------------------------------------------------------------------------

  IF (First) THEN
    First = .FALSE.

    AngVar => DefaultVariableGet( 'Rotor Angle' )
    ! Variable should alreadt exist as it was introduced in the _init section.
    IF(.NOT. ASSOCIATED( AngVar ) ) THEN
      CALL Fatal('Kinematics','Variable > Rotor Angle < does not exist!')
    END IF
    ang = AngVar % Values(1)

    VeloVar => DefaultVariableGet( 'Rotor Velo' )
    IF(.NOT. ASSOCIATED( VeloVar ) ) THEN
      CALL Fatal('Kinematics','Variable > Rotor Velo < does not exist!')
    END IF
    velo = VeloVar % Values(1)

    ! Read parameters from the SIF's Simulation section
    imom = GetConstReal( GetSimulation(),'Jr') ! interatial moment of the rotor
    tscale = GetConstReal( GetSimulation(),'Torque scaling factor') !
  END IF

  ! Save values from the previous timestep
  ang0 = AngVar % Values(1)
  velo0 = VeloVar % Values(1)

  ! Here loading can be changed
  TimeVar => VariableGet (Solver % Mesh % Variables, 'Time')
  Time = TimeVar % Values(1)
  IF (Time > 0.5) THEN
    Tload = 32.0
  END IF

  ! Time integration for the angular momentum equation
  !---------------------------------------------------
  torq = GetConstReal( GetSimulation(),'res: group 1 torque', Found)

  ! Here is the main kinematic equation
  ! It can be augumented by friction and additional loads as required
  IF(imom /= 0) THEN
    velo = velo0 + dt * (Tscale*torq-Tload) / imom
    ang  = ang0  + dt * velo
  END IF

  ! Export Kinematics variables for "SaveScalars":
  ! -----------------------------------------------------
```

```
  CALL ListAddConstReal(GetSimulation(),'res: time', GetTime())
  CALL ListAddConstReal(GetSimulation(),'res: Angle(rad)', ang)
  CALL ListAddConstReal(GetSimulation(),'res: Speed(rpm)', velo/(2._dp*pi)*60)

  !update global variables
  VeloVar % Values(1) = velo
  AngVar % Values(1) = ang
!------------------------------------------------------------------------------
END SUBROUTINE Kinematics
!------------------------------------------------------------------------------
```

Appendix K post.py

```
# post-processing routine using matplotlib

import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate, signal

# name of the file with scalar data
postfile = 'results/transient_results.dat'

t, M, iu, iv, iw, uu, uv, uw = np.loadtxt(postfile, unpack=True,
usecols=(5,2,6,13,20,8,15,22))

i1,i2,i3,i4,i5,i6,i7,i8,i9,i10 = np.loadtxt(postfile, unpack=True,
usecols=(27,43,49,55,61,67,73,79,85,91))

plt.figure()
plt.plot(t, iu)
plt.grid(True)
plt.hold(True)
plt.plot(t, iv)
plt.plot(t, iw)
plt.ylabel("Current")

plt.figure()
plt.plot(t, M*4*0.160)
plt.grid(True)
plt.ylabel("Torque")

plt.figure()
plt.plot(t, uu, label="U_u")
plt.grid(True)
plt.hold(True)
plt.plot(t, uv, label="U_v")
plt.plot(t, uw, label="U_w")
plt.ylabel("Voltage")
plt.legend(loc='best')

plt.figure()
plt.plot(t, i4, label="i4")
plt.grid(True)
plt.hold(True)
plt.plot(t, i5, label="i5")
plt.plot(t, i6, label="i6")
plt.plot(t, i7, label="i7")
plt.plot(t, i3, label="i3")
plt.plot(t, i8, label="i8")
plt.plot(t, i1, label="i1")
plt.plot(t, i2, label="i2")
plt.plot(t, i9, label="i9")
plt.plot(t, i10, label="i10")
plt.plot(t, i1+i2+i3+i4+i5+i6+i7+i8+i9+i10, label="sum of pole currents")
plt.ylabel("Rotor Current")
plt.legend(loc='best')
plt.xlabel("Time, $t$ [s]")

plt.show()
```

Appendix L run.sh

```
#!/bin/sh

rm -rf results im Kinematics ELMERSOLVER_STARTINFO

# generate roto cage winding equations and components
# parameters are in the begining of the cage_generator.py script
cd cage
python cage_generator.py
cd ..


# generate meshes for stator and rotor and then unite them
gmsh gmsh/im_stator.geo -2 -o im_stator.msh
ElmerGrid 14 2 im_stator.msh -2d -autoclean -names

gmsh gmsh/im_rotor.geo -2 -o im_rotor.msh
ElmerGrid 14 2 im_rotor.msh -2d -autoclean -names

ElmerGrid 2 2 im_stator -in im_rotor -unite -autoclean -names -out im
rm -rf im_stator im_rotor im_stator.msh im_rotor.msh

# parallel partitioning
np=4      #number of partitions and parallel processes
ElmerGrid 2 2 im -partdual -partlayers 0 -metis $np 3 -connect 2 3 4 5 7 8
#ElmerGrid 2 2 im -partdual -metis $np 4 -connect 2 3 4 5 7 8   # without partlayers 0 there is a discrepancy between NF
torque and band torque



# simple model showing that the geometry and boundary conditions are defined correctly
#ElmerSolver model.sif
# parallel execution
#echo "model.sif" > ELMERSOLVER_STARTINFO
#mpirun -np $np ElmerSolver_mpi



# Transient voltage driven simulation from zero initial conditions
#ElmerSolver transient.sif
# parallel execution
echo "transient.sif" > ELMERSOLVER_STARTINFO
mpirun -np $np ElmerSolver_mpi



# steady state AC voltage driven model with defined stator and rotor circuits
#ElmerSolver harmonic.sif
# parallel execution
#echo "harmonic.sif" > ELMERSOLVER_STARTINFO
#mpirun -np $np ElmerSolver_mpi



# Transient speed up of IM from zero speed with kinematic equation
#elmerf90 -O -o Kinematics Kinematics.f90
#ElmerSolver transient_kinematics.sif
# parallel execution
#echo "transient_kinematics.sif" > ELMERSOLVER_STARTINFO
#mpirun -np $np ElmerSolver_mpi
```