The rlib is an electric motor design automation tool for reducing the hours spent on mundane tasks. This tool depends on Scilab, FEMM, Gmsh, Altair Flux and Elmer. Design analysis and design optimization methods are also integrated in the workflow.

rlib

User Documentation for Electric Motor Design Automation

Raghunath

1 Table of Contents

Note:

The tool rlib is created by RaghuNath Kumar for his personal use, and to reduce the time spent on creating & analyzing the motor models for his work. This file (rlib_documentation.pdf) and the attached files (rlib_common.bin, rlib_odr.bin, rlib_orbldc.bin, rlib_irbldc.bin, rlib_iripm.bin, rlib_orim.bin, and rlib_irim.bin,) are a result of the efforts from mostly his personal time. These files come with no warranty about the accuracy of the results. The author does not assure compatibility of this tool with later versions of the software that it depends on. The following copyright note is based on Java Research License.

License (Based on Java Research License Version 1.6):

I. DEFINITIONS.

"Licensee" means You and any other party that has entered into and has in effect a version of this License.

"Modifications" means any change or addition to the Technology.

"RaNa" means Venkata Raghunath Kumar Rachabattuni and his successors and assignees.

"Research Use" means research, evaluation, or development for the purpose of advancing knowledge, teaching, learning, or customizing the Technology or Modifications for personal use. Research Use expressly excludes use or distribution for direct or indirect commercial (including strategic) gain or advantage.

"Technology" means the source code, documentation and object code of the technology made available by RaNa pursuant to this License.

"Technology Site" means the website designated by RaNa for accessing the Technology.

"You" means the individual executing this License or the legal entity or entities represented by the individual executing this License.

II. PURPOSE.

RaNa is licensing the Technology under this License (the "License") to promote research, education, innovation, and development using the Technology. This License is not intended to permit or enable access to the Technology for active consultation as part of creating an independent implementation of the Technology.
COMMERCIAL USE AND DISTRIBUTION OF TECHNOLOGY AND MODIFICATIONS IS PERMITTED ONLY UNDER A COMMERCIAL LICENSE.

III. RESEARCH USE RIGHTS.

A. License Grant. Subject to the conditions contained herein, RaNa grants to You a non-exclusive, non-transferable, worldwide, and royalty-free license to do the following for Your Research Use only:

1. Reproduce, create Modifications of, and use the Technology alone, or with Modifications;

2. Share source code of the Technology alone, or with Modifications, with other Licensees; and

3. Distribute object code of the Technology, alone, or with Modifications, to any third parties for Research Use only, under a license of Your choice that is consistent with this License; and publish papers and books discussing the Technology which may include relevant excerpts that do not in the aggregate constitute a significant portion of the Technology.

B. Residual Rights. If You examine the Technology after accepting this License and remember anything about it later, You are not "tainted" in a way that would prevent You from creating or contributing to an independent implementation, but this License grants You no rights to RaNa's copyrights or patents for use in such an implementation.

C. No Implied Licenses. Other than the rights granted herein, RaNa retains all rights, title, and interest in Technology, and You retain all rights, title, and interest in Your Modifications and associated specifications, subject to the terms of this License.

D. Third Party Software. Portions of the Technology may be provided with licenses or other notices from third parties that govern the use of those portions. Any licenses granted hereunder do not alter any rights and obligations you may have under such licenses,

however, the disclaimer of warranty and limitation of liability provisions in this License will apply to all Technology in this distribution.

IV. INTELLECTUAL PROPERTY REQUIREMENTS

As a condition to Your License, You agree to comply with the following restrictions and responsibilities:

A. License and Copyright Notices. You must include a copy of this License in a Readme file for any Technology or Modifications you distribute. You must also include the following statement, "Use and distribution of this technology is subject to the License included herein", (a) once prominently in the source code tree and/or specifications for your source code distributions, and (b) once in the same file as Your copyright or proprietary notices for Your binary code distributions. You must cause any files containing Your Modification to carry prominent notice stating that you changed the files. You must not remove or alter any copyright or other proprietary notices in the Technology.

B. Licensee Exchanges. Any Technology and Modifications you receive from any Licensee are governed by this License.

V. GENERAL TERMS.

A. Disclaimer Of Warranties.

THE TECHNOLOGY IS PROVIDED "AS IS", WITHOUT WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE TECHNOLOGY IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE, OR NON-INFRINGING OF THIRD PARTY RIGHTS. YOU AGREE THAT YOU BEAR THE ENTIRE RISK IN CONNECTION WITH YOUR USE AND DISTRIBUTION OF ANY AND ALL TECHNOLOGY UNDER THIS LICENSE.

B. Infringement; Limitation Of Liability.

1. If any portion of, or functionality implemented by, the Technology becomes the subject of a claim or threatened claim of infringement ("Affected Materials"), RaNa may, in its unrestricted discretion, suspend Your rights to use and distribute the Affected Materials under this License. Such suspension of rights will be effective immediately upon RaNa's posting of notice of suspension on the Technology Site.

2. IN NO EVENT WILL RaNa BE LIABLE FOR ANY DIRECT, INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF THIS LICENSE (INCLUDING, WITHOUT LIMITATION, LOSS OF PROFITS, USE, DATA, OR ECONOMIC ADVANTAGE OF ANY SORT), HOWEVER IT ARISES AND ON ANY THEORY OF LIABILITY (including negligence), WHETHER OR NOT RaNa HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. LIABILITY UNDER THIS SECTION V.B.2 SHALL BE SO LIMITED AND EXCLUDED, NOTWITHSTANDING FAILURE OF THE ESSENTIAL PURPOSE OF ANY REMEDY.

C. Termination.

1. You may terminate this License at any time by notifying RaNa in a writing addressed to raghunadh@tuta.io.

2. All Your rights will terminate under this License if you fail to comply with any of its material terms or conditions and do not cure such failure within thirty (30) days after becoming aware of such noncompliance.

3. Upon termination, you must discontinue all uses and distribution under this agreement and all provisions of this Section V ("General Terms") shall survive termination.

D. Miscellaneous.

1. Trademark. You agree to comply with RaNa's Trademark Usage Requirements, as modified from time to time. Except as expressly provided in this License, You are granted no rights in or to any RaNa's trademarks now or hereafter used or licensed by RaNa.

2. Integration. This License represents the complete agreement of the parties concerning the subject matter hereof.

3. Severability. If any provision of this License is held unenforceable, such provision shall be reformed to the extent necessary to make it enforceable unless to do so would defeat the intent of the parties, in which case, this License shall terminate.

4. Governing Law. This License is governed by the laws of India and the State of Karnataka, as applied to contracts entered into and performed in India between Indian residents. In no event shall this License be construed against the drafter.

2 About

The rlib (written in lower case) is an effort to ease the use of open source and commercial tools for electromagnetic design. This effort included creating few algorithms and re-using some from elsewhere. This library is not intended to be yet another tool for solving the electromagnetic problems targeted at motors. The rlib is neither implementing the FEA algorithms that run on MATLAB or Scilab platforms, nor does creating a fancy GUI that entice users into thinking that this is the one that can do all. The rlib is primarily a collection of "Design Automation and Design Optimization" functions. It helps me use the software that solves the following Maxwell's equations.

$$\nabla \times \vec{B} = \mu_0 \vec{J} + \mu_0 \epsilon_0 \frac{\partial \vec{E}}{\partial t}$$

$$\nabla \cdot \vec{B} = 0$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}$$

$$\nabla \cdot \vec{E} = \frac{\rho}{\epsilon_0}$$

3 Requirements

The rlib has been developed and tested in the following environment

- Windows 7 and 10, 64bit versions
- Scilab (v5.5.2), 64 bit version
- FEMM 4.2(x64, Released on Jan 12, 2016)
- Flux (v12.3.1 and v2018), 64 bit versions
- Gmsh (v3.0.6 and v3.0.7), 64 bit versions
- Elmer (v8.2), 64 bit version

4 Capabilities

The following geometries can be created

- BLDC outer rotor
- BLDC inner rotor
- IPM inner rotor
- SyRM inner rotor
- IM inner rotor
- IM outer rotor

5 Features

The rlib can also create few geometrical and procedural variations such as

- Functional and robust
- The FEMM based workflow has created and solved over 1 million valid models
    - Same geometry checks are used for Flux, Gmsh and Elmer model creation
    - Most of the errors in the code are eliminated, with a very few known bugs
- Validated and calibrated models
    - BLDC topologies have been prototyped and tested on the dynamometer and predicted characteristics such as the Peak Torque, Max Speed, corner speed, kt and others have been validated
    - Phase resistance values are calibrated from the prototyped values
- Completely parametric
    - Every geometrical parameter can be changed from the command-line or the script file
    - Many solver settings and configurations can be controlled from the script
- Individual **wire strands** placement
    - Circular and rectangular wires can be placed in the slot
    - Wire placement algorithm was developed
        - supports tilting the rectangular wires to match the tooth wedge angles
        - supports placing individual wires in top half and bottom half of the slots for multi-throw winding
- Creating **air pockets** in the Stator or Rotor BackIron regions
    - These air pockets can be created specific to the geometry and be customized
    - These pocket entries can be created through the Scilab graphics plot, or manual table entry
- **Rectangular magnets**
    - Instead of the arc magnets for the surface mounted machines, rectangular magnets can be used
    - Magnet overhang is supported for rectangular magnets too
- **Consequent poles**
    - Supports for the surface mounted configurations, with arc magnets (any pole arc) and rectangular magnets
- Creating the **tooth stem angle**

    - Not just parallel slots and parallel teeth design, but any angle in between
- **Halbach array** configuration
    - Arc magnets may be configured to two variations of Halbach arrangements
        - first arrangement is radial and tangential magnetization
        - second arrangement is tangential and inclined (to radial direction, two halves) magnetization
- Rotor **eccentricity**
    - Static or dynamic eccentricity can be emulated
- **Magnet overhang**
    - Can specify the magnet overhang of the 3D models in Gmsh and Elmer. Still working on Flux 3D model creation (automatically)
- Customized **winding pattern**
    - Can include non-uniform turn number for individual slots
    - Can create and specify the winding pattern through the coil throw parameter
    - Can specify non-standard winding pattern by assigning the corresponding string to the variable
    - Can support multi-throw winding
        - The slot can be split horizontally as opposed to vertical split in the single throw winding for BLDCs.
- Automatic valid geometry creation for any pole/slot combination
    - Works for every valid pole-slot combination
    - Indicates the winding imbalance for invalid pole-slot combinations
- Inspecting the geometry
    - Checks the geometry for any spatial violation

6 Simulation features

- FEMM solution
    - Series of magneto static simulations
    - Solving the motor in FEMM for complete characterization
        - Speed-torque plots
        - Torque-Vs-Current-Vs-Rotor Position maps
        - Airgap magnetic flux density
        - Demagnetizing effects on the magnets

        - Stator/Rotor backiron saturation
        - Cogging torque
        - Back-emf waveforms
        - Line/Phase self and mutual inductances
        - Radial pull force on the rotor with/without rotor eccentricity
    - Characterization happens with parallel operation
        - multiple FEMM models are created with different rotational angles
        - resulting data is collected by reading the files and appended to the main variables
        - saves time drastically
- FEMM optimization
    - Gradient descent based and stochastically driven
    - Targets global maxima
    - Changes every possible geometrical parameter
    - Changes winding turn count, parallel strands, and wire diameter
    - Matches the set slot fill
    - Pass criteria and thresholds
    - Top/Bottom limits of chosen parameters
    - Every parameter change can be one of normal distribution, uniform distribution, or random selection from a set
    - Data visualization during the optimization process
    - saves every motor analyzed including the results as .odR files
        - saves optimization settings as .osR files
    - data files can be read and processed to csv files
        - this operation can be paralleled where batches of file are read using multiple Scilab instances
        - resulting output file is an amalgamation of individual files
- Gmsh geometry
    - Creates 2D and 3D models and unrolls them for simpler loading. A 3D model with about 5 million degrees of freedom (after meshing) can be loaded in less than 1 second.
        - 2D models are created from parameterized specifications of the motor, and boundary surfaces defined.

    - 3D models are much beyond simple extrusions
        - 3D models are extruded, additional 3D features added, surface boundaries added
- Gmsh solution
    - Creates .pro file with magnetization profiles, material characteristics and physical group numbers
    - At the moment, limited to 2D solutions only
- Elmer geometry
    - Creates mesh. files from .msh files
    - 2D workflow is stabilized, working on 3D mesh creation through import or extruding 2D mesh.
- Elmer solution
    - Creates .sif files with magnetization profiles, material characteristics and physical group numbers
    - 2D solution works fine and the data was visualized in ParaView.
    - 3D solution and multi-physics solution needs to be tried out.
- Geometry creation for simplified calculations
    - each geometry is created with the Stator and the Rotor relative positions at their maximum torque producing values according to the control scheme chosen
- Peak torque detection algorithms
    - to identify the realistic rotor position that produces actual peak torque including the geometry variations, reluctance effects and cogging effects
    - has control the number of points needed to be sampled on either side of the peak value (of torque vs rotor angle plot) to reduce the number of FEA simulations
    - also useful for peak cogging torque estimation

7 Present status

- Geometry creation
    - FEMM {.fem file is created through scifemm interface} - supports IR/OR BLDC and IRIPM
    - Gmsh {2D & 3D including boundary conditions}({geo file is creation through Scilab scripts} - supports OR BLDC so far
    - Elmer {2D and 3D including boundary conditions}{mesh.__ files are created with elmergrid using the .msh files} - supports OR BLDC so far
    - FLUX {2D geometry creation so far.} - supports IR/OR BLDC and IRIPM

- Magnetostatic simulations
    - FEMM (built-in solver) - worked on 2D geometry
    - Gmsh (with GetDP)(.pro file is created from Scilab scripts) - worked on 2D so far, supports 3D as well
    - Elmer (with 2D MagnetoStatic solver)(.sif file is created from Scilab scripts)- worked on 2D so far, supports 3D as well
    - FLUX (built in solver) - worked on 2D so far, supports 3D as well
- Transient Magnetic
    - Gmsh (with GetDP) - worked on 2D so far, supports 3D as well
    - FLUX (built in solver) - worked on 2D so far, supports 3D as well
- Optimization routines
    - FEMM - Scilab : Scilab controls FEMM actions and model parameters. It also creates new models based on previous simulations to derive parito front and the best possible solution.
        - Routine is based on modified gradient descent algorithm that searches for global maximum
        - Can work with value-constrained and step-constrained multi-variate environment
        - Can also do sensitivity analysis
        - Works with the set thresholds on chosen variables
- Motor characterization
    - FEMM - Scilab : Can characterize the motor for the following
        - Speed-Torque curves
        - Torque-Vs-Theta plot
        - Cogging torque-Vs-Theta plot
        - Torque-Vs-Current-Vs-Theta plot
        - Inductance-Vs-Current plot
        - Induced voltages Vs. Theta plots
        - Radial pull force with/without rotor offset
    - FLUX : Can characterize the motor for the following
        - Speed-Torque curves
        - Torque-Vs-Theta plot
        - Cogging torque-Vs-Theta plot

        - Torque-Vs-Current-Vs-Theta plot
        - Airgap Flux Density and the frequency spectrum
        - Induced voltages Vs. Theta plots
- GUI for editing
    - Can edit the motor and optimization variables from GUI - however, better control exists from commandprompt
- Hand-coded scripts
    - rlib is composed of common, data and geometry files for all the motor topologies

8 Getting started

Install all the required tools and make sure they are working independently. User can provide the corresponding paths during the scripting.

The Scilab version 5.5.2 has been tested all through the development of this library. Scilab 6.x has issues with FEMM calls where the session crashes when the commands are sent through scilab. GMSH and Flux work flawlessly with any version of the scilab.

9 Special functionalities

The library has been intended for use with multiple motor design tools. Some of the menus and fields are specific to one and may not have relevance to other software.

Install all the required tools and make sure they are working independently. User can provide the corresponding paths during the scripting.

10 Working with the choice of software

The rlib has an option to choose between FEMM, FLUX, Gmsh (and GetDP).

One of the ways to choose the target software is as below.

Script to activate the libraries is below.

```
motor.buildmodelin='femm';// or FLUX or gmsh or onelab
// capitalization does not matter
```

All the three tools can work with both electromagnetic and thermal models.

- FEMM
  - Magneto static, emilates dynamics with time-stepping
  - Thermal steadystate
- Flux
  - Magneto static
  - Electromagnetic Steady state
  - Electromagnetic Transient
  - Thermal steadystate
  - Thermal transient
  - Electromagnetic-Thermal CoSimulaiton with convergence
- Gmsh/GetDP
  - Magneto static, emulates dynamics with time-stepping
  - Thermal steady state
  - Thermal transient
  - Co-simulation of electromagnetic and thermal – with time stepping

11 Working with FEMM

The FEMM by Dr. Meeker is a tool used for magneto static simulations, in addition to electro statics, heatflow (steady state) and current flow problems. The tool has lua script support, in addition to ActiveX support to interface to MATLAB or VB based software. Another interface is through the DLL route to interface with Scilab like software or any C/C++ based tools.

FEMM supports creating points, lines and curves as a part of geometry construction. FEMM supports including some materials from the libraries, or allows custom definition of materials through script and the GUI. FEMM allows the creation of open boundary (multi layered with dirichlet or Neumann type). FEMM allows the definition of periodic boundary conditions. FEMM supports construction of planar or axi-symmetric models. FEMM allows circuit definitions and allows the current to be defined through each of the circuit regions independently. We use this feature to define parallel paths and multiple strands in the final models.

Rlib takes the user inputs for the required geometry and generates the point and line definitions to be created and sends a sequence of commands to FEMM to construct the model in a

sequential manner. Multiple geometries are supported for this construction. Later, the material specification is also defined for each region of the geometry including the magnets' direction. In addition to this, analyses are conducted on such constructed models with complete definition of physics to analyse multiple parameters from the FEA. Rlib sends commands to the model to rotate the rotor to required angles to conduct time-stepped analyses for limited transient simulations. Some of the analyses that can be done are listed in the lists above.

12 Working with Gmsh

The Gmsh by Dr. Geuzaine is a tool for creating meshes for 2D and 3D geometries. The geometry definitions are based on the point, line, curve, surface and volume definitions from the *.geo files. The geo files will also have the physical regions defined for a group of surfaces/volumes, which are useful in assigning material properties and circuit properties.

The GetDP is the solver by Dr. Geuzaine that can be clubbed with Gmsh to solve for multi physical problems. In the present context, this is used for solving for magneto static, transient thermal and magneto dynamic problems. The set of files with *.pro extension define material properties for physical regions (defined by numbers, same numbers defined in the *.geo files as a group of surfaces/volumes). The physical regions are grouped and assigned either static or dynamically varying property functions. Gmsh's UI also allows few text input boxes, sliders to allow users to choose few properties and other conditions to customize the model dynamically. Circuit definitions are also possible in the pro files which allows the custom definition of the circuit operation based on the existing simulation conditions. The pro file has elaborate definitions of jacobians, integrations, and constraints (dirichlet or Neumann or operatonal conditions such as currents or speeds). The pro files have function space definitions (such as Hilbert spaces) and the basis functions. The formulations are defined with the help of gelarkins. The gelarkin equations are very similar to how one would write the equations in weak formulation representation of the physical system to be solved. The resolution section helps with the flow of equations to be solved, and iterations (if any). Post processing section handles the data to find the simulation results as required.

The rlib implements 2D solutions with Gmsh/GetDP combination so far. It gathers the user inputs and generates *.geo file that represents the final geometry in terms of point locations, and line locations. The surfaces formed are defined by rlib by tracing the boundary points to identify full surfaces and the holes in the surfaces as well. The rlib further creates the physical region assignments. The resulting geo file is static in nature and has no script in it. This helps the model to be loaded in gmsh very quickly. The rlib further creates a pro file that has the gelarkin based solver. This file too, is static, and does not depend on the motor model to be solved. A single folder created by the rlib can be loaded in to gmsh for solving multiple any number of times, without requiring the rlib tool.

The rlib also includes a custom solution feature for Gmsh for the model post processing. Once the A field is solved from the generated pro file, the B fiel dis computed. Later, these B values are used over the full electrical revolution of the rotor, to identify the iron losses using the bertotti loss model. This loss model is included in the custom built gmsh copy

with the changes to source code to include the new loss plugin. This source code change indicates and allows the customization possibility of gmsh for massive analyses that paid tools cannot possibly handle.

13 Working with Flux

The Flux by Altair (originally from Cedrat, France) is a standard commercial tool that has a wide range of features including several custom macros that are based on python scripts. The python engine built within the Flux allows us to create custom scripts to be executed.

The rlib gathers the model details from the user and generates a static python file and a set of data tables (for point, line creation and for material definitions). Flux comes with few macros to construct known motor geometries, but the rlib tool doesn't use them. Complete geometry is created from points and lines. This allows full customization of the model to be created. Once the model construction is done, internal function of Flux is used for creating regions. All other operations are done by the rlib tool. Once the model is constructed in flux, the model is static and fully configured including the solution scenarios (one full electrical rotation, loss modeling, efficiency maps, etc.).

The rlib triggers the functions of flux for solution, and also for post-process analyses. The resulting data will be saved and used for creating full report of the motor. For example, the efficiency map generation map from rlib allows the scenario for efficiency map to be run (run the solution for peak torque at different speeds and currents, run the macros for bertotti losses at each speed, current values, and then calculating different components of losses, using weight factors for each of these components). Once complete, the complete data will be saved in a csv file that can be used for further analyses or plotting the efficiency map from the rlib tool.

14 Working with OR PM BLDC

The rlib has many functions which are split according to their use, into individual binary files. Loading the binary files also loads the functions into the Scilab's working memory. Additionally, FEMM provides a set of linking functions that interface Scilab with FEMM through a dll file and corresponding scilab functions.

The common_odr file has functions that are essential to handle the data. The common_motor file has functions that are essential for all motor topologies. The geom_orbldc file has functions that are useful for creating the motor variable and the optimization variable for the orpmbldc topology. In addition, this file also has functions to build the motor model in FEMM from the motor variable.

Script to activate the libraries is below.

```
Exec('D:\femm4.2\scifemm\scifemm.sci', -1);
Load('Y:\MOTOR\scripts\r_lib\rlib_odr.bin');
Load('Y:\MOTOR\scripts\r_lib\rlib_common.bin');
Load('Y:\MOTOR\scripts\r_lib\rlib_orbldc.bin');
```

14.1 Build from default template

The geometry from the motor variable is checked for any errors and geometrical feasibility. If the motor cannot be constructed for any reason, the reason is printed out and the model is not built. Script to build the model with default template is below.

```
motor=r_buildmotor(12,18) // poles, slots
```

or alternatively,

```
motor=r_buildvar(12,18) // poles, slots
motor=r_buildmotor(motor)
```

The motor variable can be constructed with as little as 2 inputs and as many as 8 inputs.

```
motor=r_buildvar(12,18) // poles, slots
motor=r_buildvar(12,18,50) // poles, slots, outer radius
motor=r_buildvar(12,18,50,15) // poles, slots, outer radius, stack depth
motor=r_buildvar(12,18,50,15,0.5) // poles, slots, outer radius, stack depth, air gap
motor=r_buildvar(12,18,50,15,0.5,'star') // poles, slots, outer radius, stack depth, air gap, winding type
motor=r_buildvar(12,18,50,15,0.5,'star',48) // poles, slots, outer radius, stack depth, air gap, winding type, voltage range
motor=r_buildvar(12,18,50,15,0.5,'star',48,80) // poles, slots, outer radius, stack depth, air gap, winding type, voltage range, current limit
// each of the above generates a valid motor variable, assuming the default values for rest of the parameters wherever they are not provided.
```

Or, the simpler notation as below can be used to specify only those parameters that are of interest, while others are assumed to be same as the in the default template.

```
motor=r_buildvar('poles',12,'slots',18,'throw',2)
motor=r_buildmotor(motor)
// or
motor=r_buildmotor('poles',12,'slots',18,'throw',2)
```

Special calling references: poles/pole, slots/slot, outerrad/rad, outerdia/dia, stacklen/len, airgap/gap, winding/wind/windtype, vdcrated/vdc, idcmax/idc, and coilthrow/throw.

The r_buildmotor function call with the same inputs as r_buildvar will build the motor with the parameters, effectively reducing the command count by 1. Similarly, the r_solve after r_buildvar will also build the motor.

14.2 Configurations

14.2.1 Poles & Slots

The poles and slots need to be selected to incorporate a balanced winding pattern. A balanced winding is the one in which all the three phase vectors on the 2-D surface of the motor winding are separated by 120 electrical degrees. Only a few pole slot combinations are valid. The rlib automatically chooses the rest of the parameters such as the tooth width, magnet arc length, etc.

```
motor.rotor.poles=12
//poles have to be a multiple of 2
motor.stator.slots=18
//slots have to be a multiple of 3
```

Poles and slots combination alters few other geometric parameters on the stator, rotor as well.

14.2.2 Outer radius and Stack depth

These two are independent parameters and are global values. The rlib chooses rest of the parameters of the motor to fit within the outer_radius parameter. When motor variable is created using the r_buildVAR or r_buildMOTOR, with only the outer radius, the default template will come up with rest of the parameters scaled appropriately to result in a valid motor geometry. The stack depth is the length of the motor in the axial direction. These two parameters can be set with the following script

```
motor.rotor.outer_rad=50
motor.stackdepth=15
```

14.2.3 Airgap

Air gap of the motor is the distance between the stator and the rotor when the motor is assembled. The rlib changes the stator geometry and retains the rotor geometry when the air gap is changed, while the template variable is being created.

```
motor.airgap=0.5
```

14.2.4 Tooth wedge angle

The tooth wedge angle is the angle of the tooth edge with respect to the line through its center. This angle can be customized in rlib resulting in many possibilities.

```
motor.stator.toothwedgeangle=-5
```

14.2.5 Magnet pole arc

Pole arc is the fill of each pole in one electrical angle span. This alue is in the range (0,180]. Script to change the magnet's pole arc

```
motor=r_buildvar(12,18)
motor.winding.turns=8
motor.stator.forcedparallelslot=1
motor.rotor.polearc=120
motor=r_buildmotor(motor)
```

14.2.6 Magnet thickness

Magnet thickness is the thickness of the hard magnetic material in radial direction. Any air gaps between magnets and the back-iron is not assumed. All the magnets are modeled as arc magnetic elements. No chamfer or a fillet addition to the edge is modeled.

```
motor.rotor.magnet_thickness=5.5
```

14.2.7 Stator slot depth

```
motor.stator.slotdepth=20
```

Slot depth is a measurement taken from the slot bottom point to the edge of air-gap with the stator outer radius. Slot depth cannot also change infinitely, because it affects several other parameters such as tooth width etc. too.

14.2.8 Stator slot opening

```
motor.stator.slotopening=2
```

Slot opening is the gap between the consecutive tooth shoe fins in tangential direction. More slot opening lets the winding slip in easily, but increases flux leakage within the stator.

14.2.9 Stator slot open angle

```
motor.stator.slotopenangle=25
```

The slot opening angle is the angle cast by the line joining stator tooth and the tooth shoe's inner edge(away from the airgap), with the tangential line at the tooth shoe's inner edge.

14.2.10 Stator tooth shoe thickness

```
motor.stator.tgd=1.1
```

Tooth shoe is the construction at the end of the tooth stem towards the air gap, usually wider than the tooth width.

14.2.11 Stator tooth width

```
motor.stator.tws=6.8
```

Stator tooth width is the width of the tooth towards the airgap. This width may change gradually as we move towards the center of the motor, if the toothwedgeangle is not equal to zero.

14.2.12 Stator slot bottom fillet radius

```
motor.stator.sbfillet_rad=1.1
```

This is the fillet radius at the bottom of the slot.

14.2.13 Stator slot opening fillet radius

```
motor.stator.sofillet_rad=1.1
```

This is the fillet radius of the slot's top corner.

14.2.14 Magnet Overhang

```
motor.rotor.magnetoverhang=0
```

This is a parameter that is not used for any electromagnetic calculations, and is purely used for calculating the mass of magnets in the solution parameters.

14.2.15 Forced Parallel Slots

```
motor.stator.forcedparallelslot=0
```

This is a parameter that is used to create parallel slots, irrespective of pole & slot combination. This parameter overrides toothwedgeangle and always creates the parallel slots with the required toothwedgeangle.

14.2.16 Slot depth thresholds

```
motor.stator.placeholetill=18.1
```

This is a parameter that is used to create thresholds on slot bottom. This is required to limit the slot bottom during the optimization runs so that the mounting screws may be arranged in the space at the center. This parameter has no role in the geometry, but only effects the placement of slots towards the center. A slot may not come into the zone as specified by the radius in this parameter.

14.3 Topological variations

14.3.1 Halbach motor type 1

This type of motor has radial and lateral magnet arrangement. To achieve this effect, use the following code.

```
motor.rotor.halbach_type=1
motor.rotor.halbach_fraction=0.25
```

14.3.2 Halbach motor type 2

This type of motor has inclined and lateral magnet arrangement. To achieve this effect, use the following code.

```
motor.rotor.halbach_type=2
```

14.3.3 Rectangular magnets

This arrangement will create a realistic representation of the magnet arrangement on the rotor when the rectangular magnets are used. The airgap parameter is used to create shortest distance between the magnets and the stator.To have this construction done, the halback type needs to be set to zero, and the rotor's polearc needs to be set to 180, else this parameter is reset to 0. To achieve this effect, use the following code.

```
motor.rotor.rectangularmagnet=1
motor.rotor.rectangularmaggluethick=0.05 // has to be at least 0.01 mm
motor.rotor.rectangularmagspacing=0.05 // has to be at least 0.01 mm
```

14.4 Constructional Parameters

14.4.1 Winding

```
motor.winding.pattern='abcabcabcabcabcabc' // calculated automatically. may be overwritten.
motor.winding.type='star' // can be of type 'star' or 'delta'
```

```
motor.winding.marking_slot_tooth='tooth' // can be of type 'tooth' or 'slot'.

// tooth winding assumes concentrated winding, and the pattern in motor.winding.pattern refers to how the phase wires are would around individual tooth. capital letter means that the winding is counter-clockwise when seen from the air gap.

// when tooth winding pattern is given, a '-' symbol in the motor.winding.pattern indicates a tooth left unwound. alternate '-' symbols result in a single layer winding. the length of motor.winding.pattern has to be same as the number of slots, for the winding pattern to be valid.

// slot winding can be used to represent any type of winding, be it concentric, concentrated or distributed, and the pattern in motor.winding.pattern refers to how the phase wires go in or come out of the slots. capital letter means that the positive current comes out of the screen (positive winding turns).

// when slot winding pattern is given, a pattern with equal length as the number of slots result in single layer winding. a pattern with length equal to double the slot count result in double layer winding.

// at the moment, transposed winding is not implemented in the geometry. winding space for two halves in the double layer winding is divided along the slot depth.

// winding pattern is implemented in the geometry from a horizontal line and progresses clockwise (in the direction of the phases)


motor.winding.turns=10

motor.winding.nstrands=1

motor.winding.ppaths=1

//the above three define how the winding is done. the nstrands and ppaths are used while calculating phase resistances. winding.type is used while calculating line resistance. nstrands is also used to plot the total conductors when individual conductors are plotted in the geometry.

motor.winding.wire.rectangular=0

//the above setting is the default. if it is 0, circular cross section of the wire is assumed from the below variable.

motor.winding.wiredia=1.016

//if the rectangular property is set to 1, then the following two variables decide the geometry of the wire.

motor.winding.wire.len=1

motor.winding.wire.wid=2
```

```
//for representing the individual conductors in the geometry, set the following variable to 1 as shown
motor.winding.drawwiresemag=1
// wire seperation and insulation thickness of the wires in the geometry can be changed by using the following variables
motor.winding.wiregap=0.1
motor.winding.insulthickness=0.05
//if resistance estimation is wrong, either because the winding is concentric or distributed type, or because of other reasons, the following variable can be used to force set the resistance to a predetermined value
motor.winding.projectedrph=0
//setting the variable to 0 means that the value calculated by rlib is to be used.
//additionally, a correction factor for the resistance calculation can be included, by the following variable.
motor.winding.correctionrph=1
//the resistance used in the rlib calculations is the estimated resistance multiplied by the above factor. a value of 1 indicates no change in the estimated resistance.
motor.winding.throw=2; // coil throw represented in the number of slot pitches.
//
```

The rlib checks for winding feasibility and generates the winding pattern automatically for the BLDC configuration of single-throw / concentrated winding.

The multi-throw winding will give rise to a configuration in which the slots are split horizontally.

14.4.2 Magnet

```
motor.rotor.magnettype='ferrite'
```

The magnet may be chosen as Ferrite or NdFeB or any other material from the material choice available from the FEMM material library. Magnet's pole arc can be changed too.

14.5 Control parameters

14.5.1 Battery voltage

```
motor.controller.vdcrated=12 // or 48
motor.controller.cellsinseries=1 // or more
```

This is a very important parameter that decides the maximum speed of the motor. This parameter does not directly decide the operating voltage. This parameter is implemented in two variants of the batteries. When this value is chosen to be less than 15, Lead-Acid is activated. Any value higher than this will indicate Li-ion cells. When the r_buildVAR is used with the Vdcrated parameter, the SoC_V table is automatically populated with the corresponding cell's Soc vs Voltage and cell resistance value. For a 12 V system, it is assumed that each cell corresponds to 12v nominal voltage. To build a 48v operating voltage, adjust the Motor.Controller.CellsInSeries to 4. For a Li-ion cell system, default number of cells resulting in 48V is 13. Change this number accordingly to adjust the operating voltage. CellsInParallel only effects the pack resistance.

14.5.2 State of Change

```
motor.controller.soc=50
```

The state of change for a pack affects the pack voltage as per the SoC_V table. This table is useful for evaluating the performance of the motor at worst case conditions.

14.5.3 Winding temperature

```
motor.winding.temperaturec=30
```

The temperature of the winding is a condition that affects the stator resistance seen by the power source, which will reduce the corner rpm on the speed-torque plot for the motor.

14.5.4 Motor Current

```
motor.controller.idcmax=80
// Forced currents into the winding when solving the model can be set using the following variable.
motor.winding.cur=[-80 80 0]// these represent the phase currents. in this example, these correspond to star winding currents.
```

The IdcMax parameter will set the absolute maximum DC current allowed by the controller. The Cur field in the winding is the current that is passed into the winding for each phase. This cur field is set automatically while solving for the motor model based on the resistance of the winding and the rest of the path, available voltage, and the controller dc current limit. However, a known value may also be given to the model for solving for performance.

14.5.5 Controller resistance

```
motor.controller.rpath=0.015
```

The controller path resistance is the PCB resistance that is added to the whole resistance calculation. The complete resistance faced by the available battery voltage decides the corner rpm and the maximum current that the motor can accept for a given voltage.

14.6 Material parameters

The field Motor.Prop has all the custom materials defined with their properties. Any materials specified for the magnets from the materials library will assume default values.

14.7 Wire placement

14.7.1 Circular conductors

Script to create the model with individual circular conductors

```
motor=r_buildvar(12,18)
motor.winding.drawwiresemag=1
motor=r_buildmotor(motor)
```

14.7.2 Rectangular conductors

Script to create the model with individual rectangular conductors

```
motor=r_buildvar(12,18)
motor.winding.drawwiresemag=1
motor.winding.wire.rectangular=1
motor.winding.wire.wid=1.5
motor.winding.turns=8
motor=r_buildmotor(motor)
```

14.7.3 Parallel slots

Script to create parallel slots, instead of parallel tooth, with rectangular conductors

```
motor=r_buildvar(12,18)
motor.winding.drawwiresemag=1
motor.winding.wire.rectangular=1
motor.winding.wire.wid=3
motor.winding.turns=8
motor.stator.forcedparallelslot=1
motor=r_buildmotor(motor)
```

14.7.4 Rotor assembly - radial offset emulation

Script to re-create the assembly failure - rotor offset in radial direction

```
motor=r_buildvar(12,18)
```

```
motor.rotor.offset.x=0.25
motor.rotor.offset.y=0
motor=r_buildmotor(motor)
```

The following variable can let us select the type of offset - static (the offset that does not rotate when the rotor rotates) when set to 1, and dynamic (the offset that rotates around along with the rotor) when set to 0.

```
motor.rotor.offset.rotateaboutneworigin
```

14.8 Running the FEA

14.8.1 Finding the solution

```
motor=r_solve(motor)
motor=r_solve(motor,[0,0,0])
```

The above command will solve the motor based on the current constraints and the voltage input provided in Motor.controller.

FEMM gives pretty standard outputs for B, H, J across the cross-sectional area.

This function requires a motor structure as an input and optional phase currents [i1,i2,i3] to solve using FEA. If the phase currents are not provided, the geometry is used to calculate resistance, and the controller settings are used to calculate the currents in each phase.

This function requires the FEMM model to have been created. This function returns a motor structure with OUTPUT values updated based on the solution. Then the Motor.RUN.dAngle_md is set to non-zero values (or when Motor.Solve.ToOUTPUT's torque line has no starting and ending and incrementa angles), the perturbation is not done to calculate the kb and NLrpm.

The torque after rotating by the angle Motor.RUN.dAngle_md is calculated, along with few other parameters. OUTPUT is populated based on Motor.Solve.ToOUTPUT parameters.

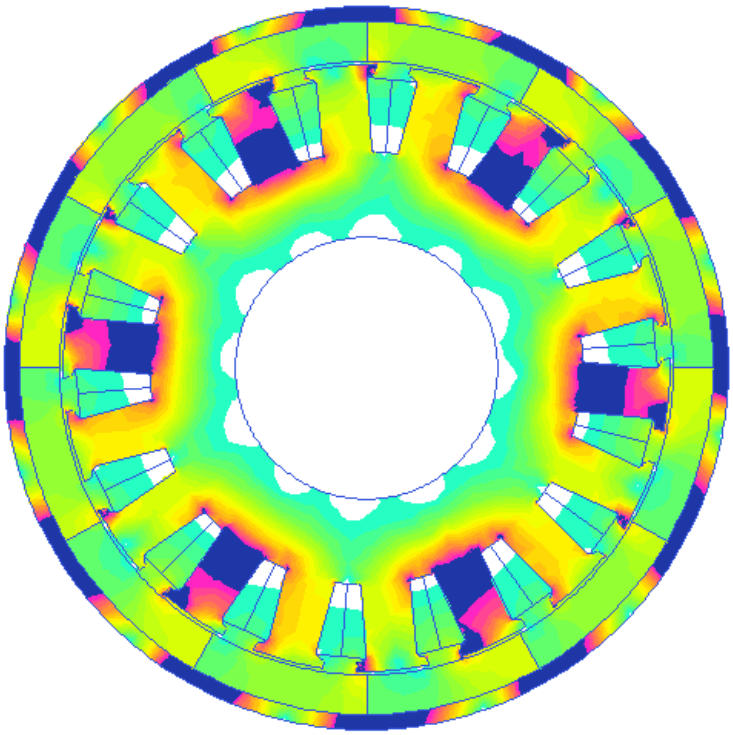
The syntax is B/H: name,starting radius, ending radius, number of points between radii(including), number of points on each circle, B/H. fill factor, slot area: name, x coordinate, y coordinate,,,fill/area. circuit properties, volume, i2r: CircProp/volume/i2r,,,,,circprop/volume/i2r. Torque: Torq,startAngle,EndAngle,IncrementalAngle,,torque.

When the start/end/increments are given as empty, instantanious fea is done for the single rotor position. When the Motor.RUN.dAngle_md is given to be other than 0, rotor is rotated to that position and solved once.

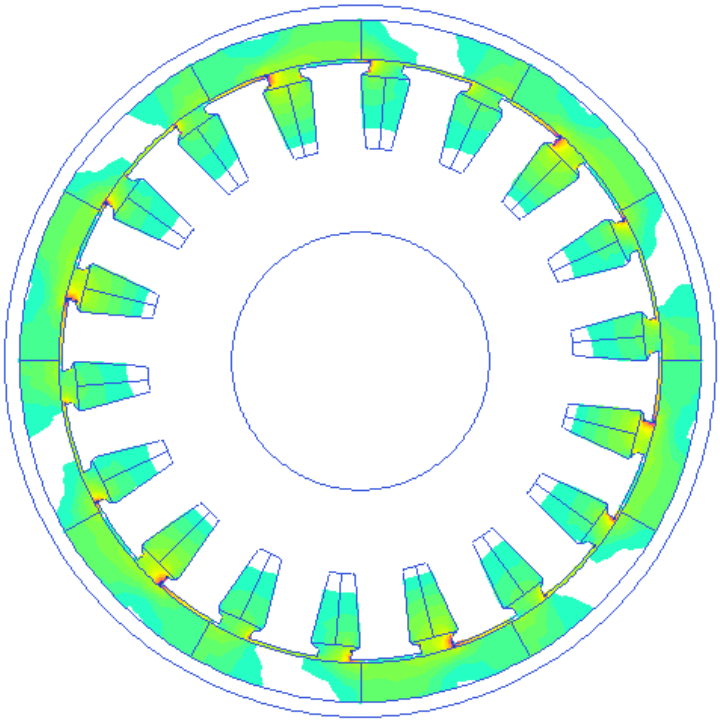
To solve for no load speed, backenmf constant, keep the torque fields active, and provide a zero Motor.RUN.dAngle_md. Once solved, the rotor is placed at initial position.

14.9 FEMM's in-built graphical plots

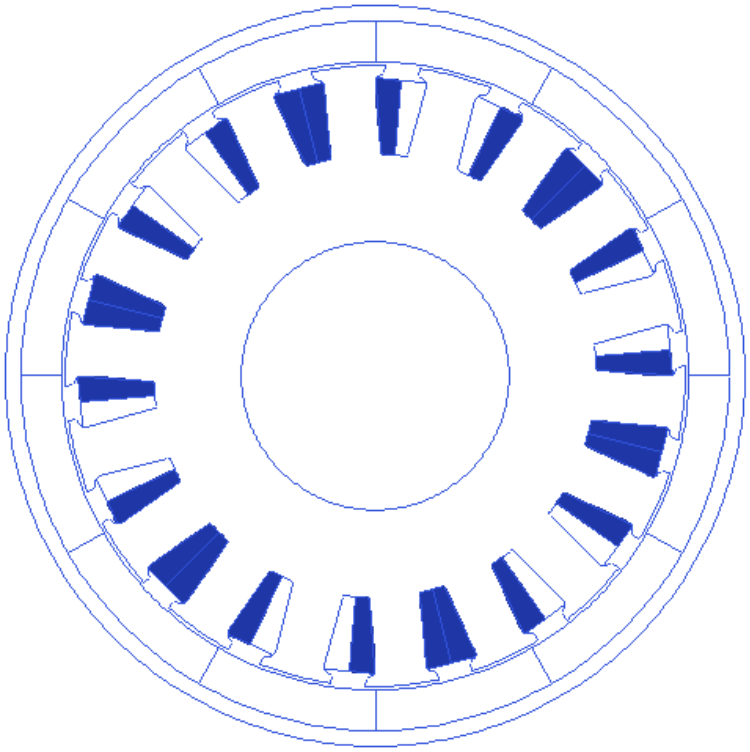
14.9.1 Magnetic flux density (B field)



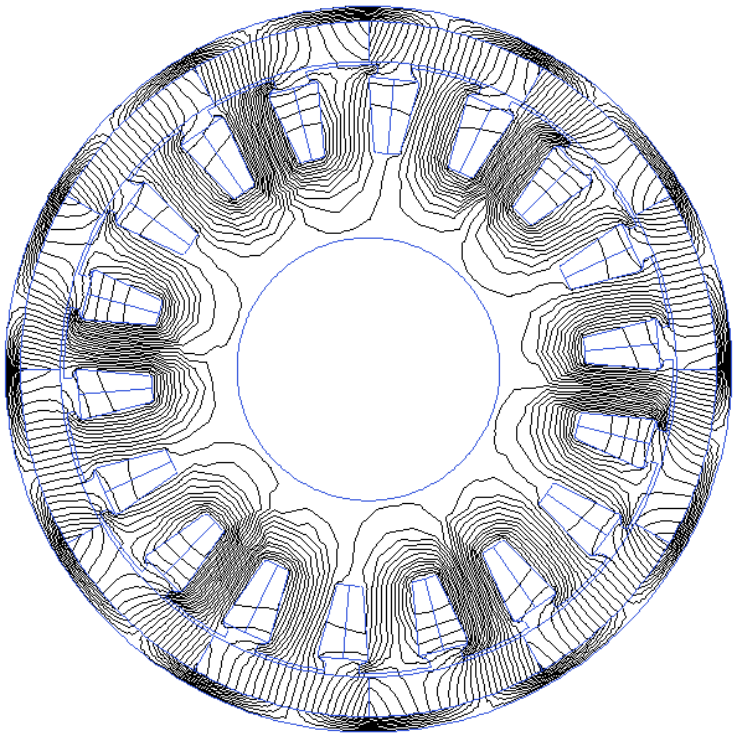
14.9.2 Magnetic field strength (H Field)



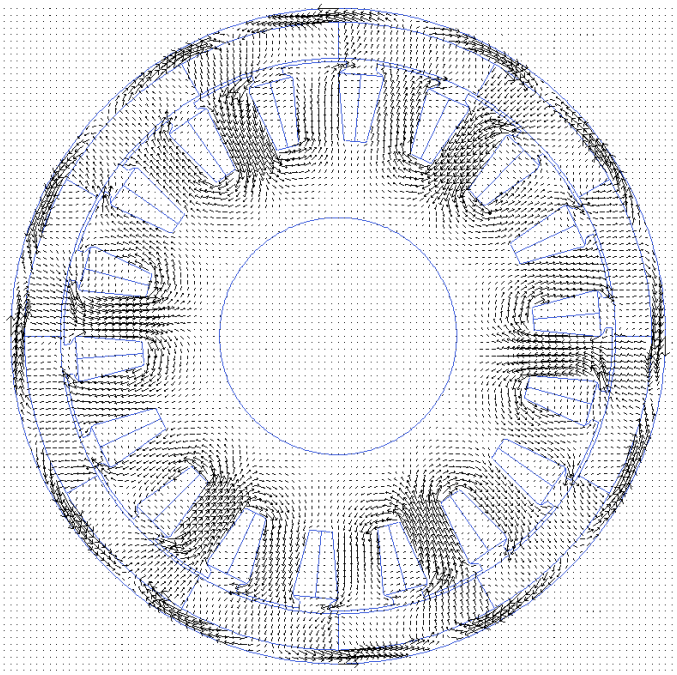
14.9.3 Current density distribution



14.9.4 ISO field lines distribution



14.9.5 Arrow plot for B field

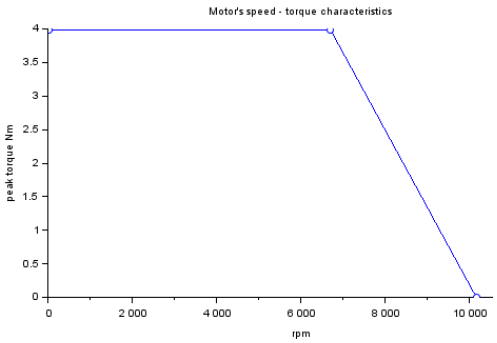


14.10 Quantified parameters - with rlib

In addition to the default plots from the FEMM, rlib calculates and plots additional parameters and provides them for the optimization routine.

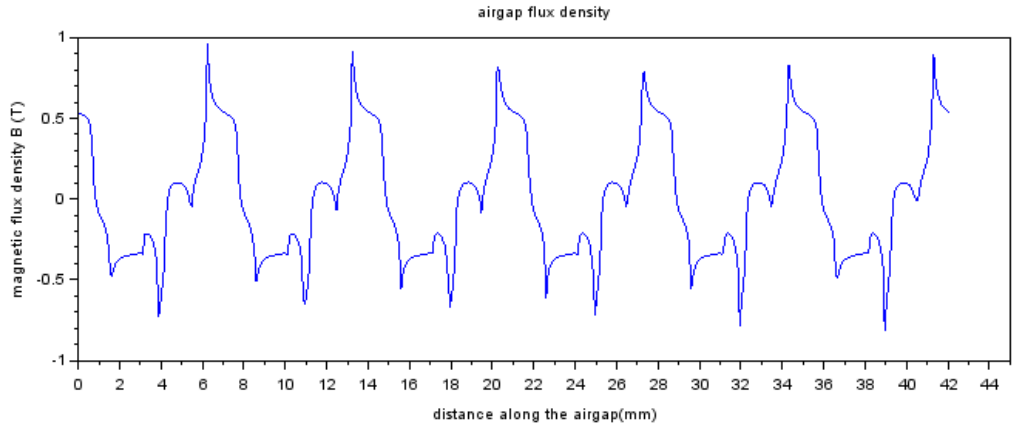
14.10.1 Speed- Torque characteristics

```
plot(motor.output.rpm_nm(:,1),motor.output.rpm_nm(:,2),'.-o')
xlabel('rpm')
ylabel('peak torque nm')
title('motor''s speed - torque characteristics')
x=gca()
x.box='off'
```



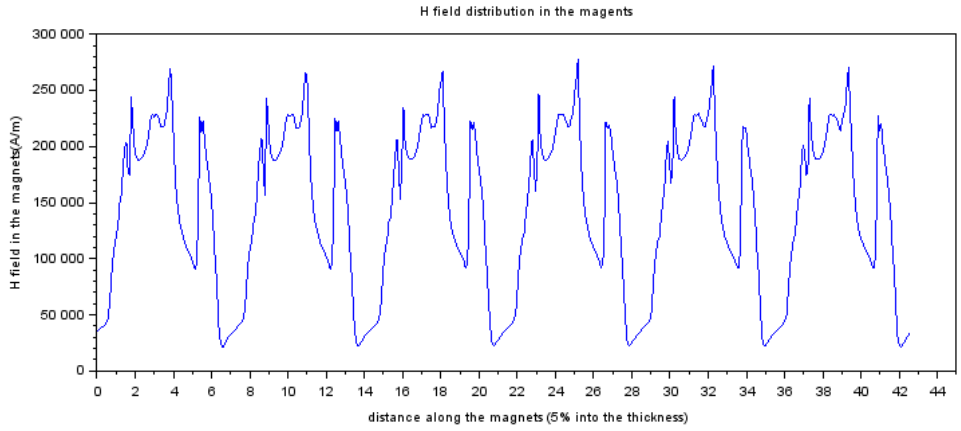
14.10.2 Airgap flux density

```
x=(0:499)*motor.points.rad_airgap/500
a=motor.output.bgap
plot(x,a)
xlabel('distance along the airgap(mm)')
ylabel('magnetic flux density b (t)')
title('airgap flux density')
```



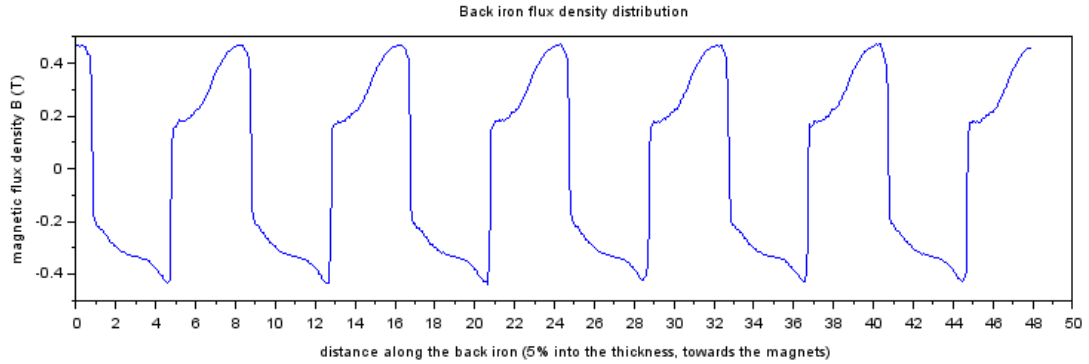
14.10.3 H field inside the magnets

```
x=(0:499)/500;x=x*(motor.rotor.outer_rad-motor.rotor.bi_thickness-
      motor.rotor.magnet_thickness*0.95)
plot(x,motor.output.hmag)
xlabel('distance along the magnets (5% into the thickness)')
ylabel('h field in the magnets(a/m)')
title('h field distribution in the magents')
```



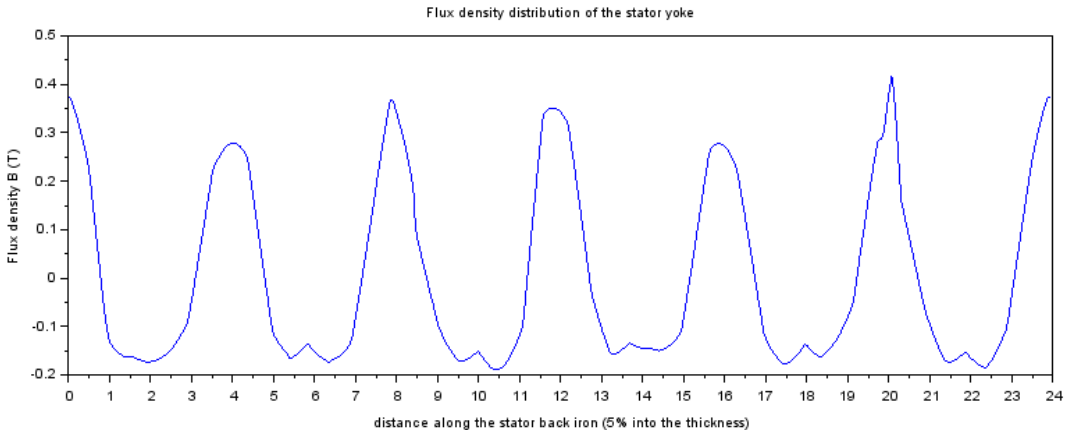
14.10.4 Rotor backiron satiration

```
x=(0:499)/500;x=x*(motor.rotor.outer_rad-motor.rotor.bi_thickness*0.95)
plot(x,motor.output.bbi)
xlabel('distance along the back iron (5% into the thickness, towards the magnets)')
ylabel('magnetic flux density b (t)')
title('back iron flux density distribution')
```



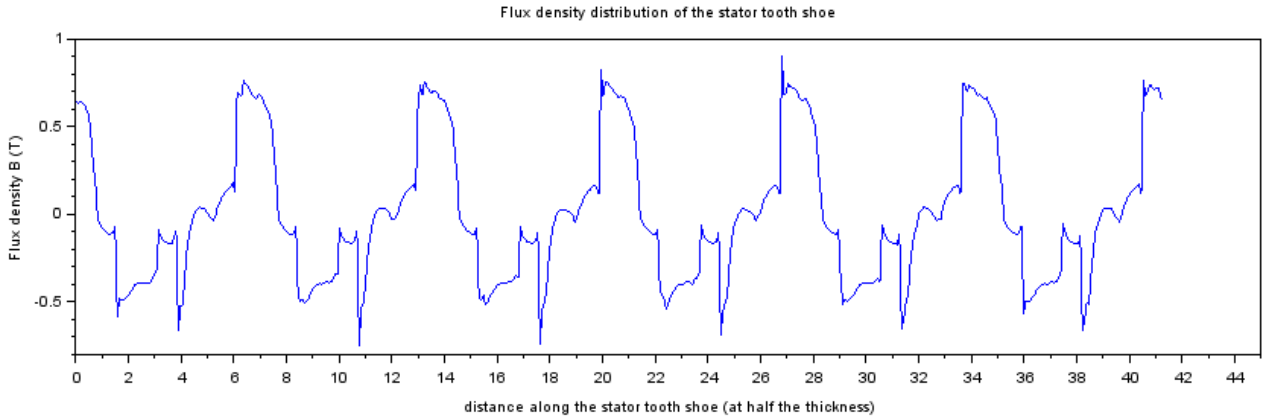
14.10.5 Stator yoke saturation

```
x=(0:499)/500;x=x*((motor.stator.shaft_rad +
    mydiag(motor.points.toothsteminner.x,motor.points.toothsteminner.y))/2)
plot(x,motor.output.bcore.innermass)
xlabel('distance along the stator back iron (5% into the thickness)')
ylabel('flux density b (t)')
title('flux density distribution of the stator yoke')
```



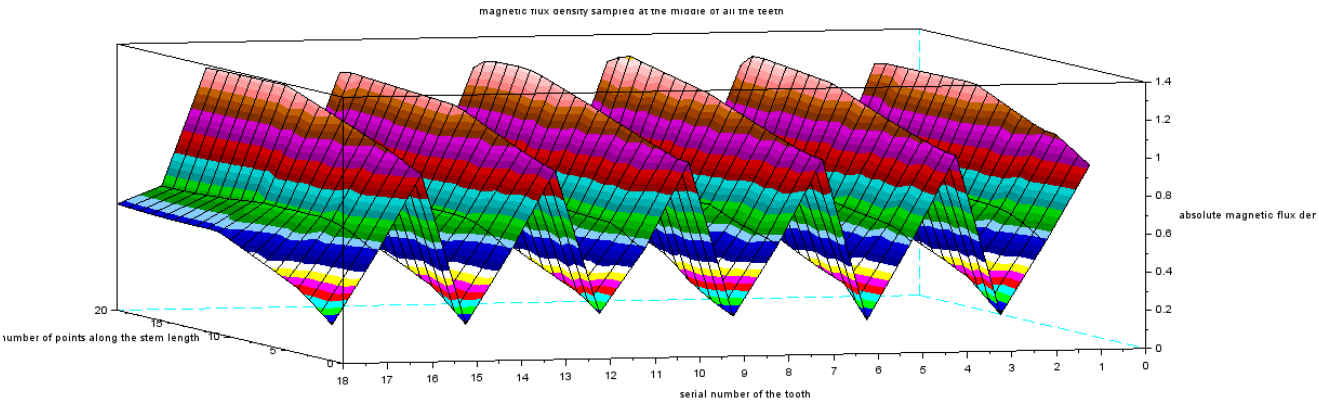
14.10.6 Stator teeth shoe saturation

```
x=(0:499)/500;x=x*((mydiag(motor.points.toothfinouter.x,motor.points.toothfinouter.y) +
    mydiag(motor.points.toothfininner.x,motor.points.toothfininner.y))/2)
plot(x,motor.output.bcore.fins)
xlabel('distance along the stator tooth shoe (at half the thickness)')
ylabel('flux density b (t)')
title('flux density distribution of the stator tooth shoe')
```



14.10.7 Stator teeth stem saturation

```
surf(motor.output.bcore.teeth,'facecol','interp')
xlabel('number of points along the stem length')
ylabel('serial number of the tooth')
zlabel('absolute magnetic flux density')
title('magnetic flux density sampled at the middle of all the teeth')
```




```
surf(motor.output.bcore.teeth,'facecol','interp')
```

```
xlabel('number of points along the stem length')
ylabel('serial number of the tooth')
zlabel('absolute magnetic flux density')
title('magnetic flux density sampled at the middle of all the teeth')
//test code in progress for polar representation of the above
a=motor.output.bcore.teeth
amin=min(a);a=a-amin;
amax=max(a);a=a/amax;a=a*128;a=[a;a(1,:)];
theta=2*%pi*(0:(motor.stator.slots))/(motor.stator.slots)
theta=theta+0*d2r(motor.points.stator_rotation)+d2r(0);
scf;f=gcf();f.color_map= hotcolormap(128);clf();graypolarplot(theta,20:-1:1,a);colorbar(0,128)
scf();f1=gcf();f1.color_map= hotcolormap(128);clf();grayplot(theta,20:-1:1,a);colorbar(0,128)
```

14.11 OUTPUT quantities

14.11.1 Torque

The following stores the peak or instantaneous torque value

```
motor.output.torq
```

The following stores the average torque value, for non-instantaneous solves

```
motor.output.torqavg
```

The following stores the cogging torque value, for non-instantaneous solves

```
motor.output.torqcogging
```

The following stores the average torque value, for non-instantaneous solves and full solves

```
motor.output.torqavg
```

14.11.2 RPM

The following stores the maximum rpm of the machine

```
motor.output.nlrpm
```

The following stores the corner rpm of the machine

```
motor.output.cornerrpm
```

Corner RPM of the machine is that rpm at which the motor's back-emf becomes sufficient to limit the current into the motor at exactly the controller's current limit. Any increase in the motor's rpm will reduce the current into the motor below the controller's threshold, effectively reducing the torque production.

14.11.3 Back-emf constant

The following gives the back-emf constant

```
motor.output.kb
```

14.11.4 Resistance

The resistance of the motor is calculated from the expected copper length and its effective cross section. The effective copper length is calculated including the end-winding length, by considering the average path length for the winding. Any of the following two commands will give the phase resistance.

```
r1_findrphase(motor)
motor.output.rph
```

The following will give the line-line resistance

```
motor.output.rll
```

14.11.5 Operating voltage & currents

The following gives the operating battery voltage

```
motor.output.bat.voltage
```

The following gives the operating currents into the individual phases

```
motor.output.circprop(:,1)
```

14.11.6 Weight

The following gives the weight of copper

```
motor.output.kgcu
```

The following gives the weight of the Stator laminations

```
motor.output.kgstatorstack
```

The following gives the weight of the rotor backiron

```
motor.output.kgbi
```

The following gives the weight of the magnets

```
motor.output.kgmagnets
```

The following gives an estimated weight of the full motor, excluding shaft, bearings, end-covers, etc.

```
motor.output.kgmotor
```

14.11.7 Slot fill

The following gives the slot's fill factor. A number less than 40% is practically possible to be wound with hand for most motors.

```
motor.output.fill
```

14.11.8 Current density

The following gives the slot current density

```
motor.output.jslot
```

The following gives the copper current density

```
motor.output.jslot/motor.output.fill
```

14.11.9 Losses

The following gives the Joule's losses

```
motor.output.i2r
```

14.11.10 Radial forces

The following give the x, and y components of the radial pull force on the rotor.

```
motor.output.rotorxpull
motor.output.rotorypull
```

14.11.11 Inductances

The following give the D axis and Q axis inductances

```
motor.output.ld
motor.output.lq
```

14.12 Solver settings

14.12.1 Mesh settings

FEMM uses Triangle for mesh generation.

```
motor.solve.meshminangle=15 // mesh triangles' minimum angle. small angles reduce mesh node count
motor.solve.masseginarc=1 // minimum degrees of angle for each segment in an arc. smaller angle results in smoother arc
motor.solve.wiremeshsize=0.2 // mesh size in the wires
motor.solve.airmeshsize=0.4 // mesh size in the air. higher value results in coarse mesh nodes.
```

14.12.2 Peak torque evaluation

```
motor.solve.valuesbelowpeak=2
```

This variable is a key to track peak torque of a motor. When the motor is being solved, one of the parameter is peak torque. The motor is aligned to a position which results in almost the peak torque, as per the design of rlib. However, due to some reluctance effects, the peak occurrence may shift to either left or right of the zero position. The peak detection algorithm rotates the rotor counterclockwise till it sees 2 (or the value from the variable above) values lower than the peak. Then it rotates clockwise to see 2 values lower than the peak, i.e. the peak should be surrounded by at least 2 values lower than itself. This is a critical parameter to eliminate local peak detection, and avoids the effects of ripples caused by cogging torque. More count results in more FEA runs and hence more time for the peak torque detection. The same variable is used for cogging torque detection also.

14.12.3 solutions in one electrical cycle

```
motor.solve.ptsin60degree=10
```

When the motor is being characterized, or being solved, the step rotation defines the smoothness of the generated curve. A total of 6*ptsin60degree points will be evaluated in one electrical cycle for characterization. A step size corresponding to 60/ptsin60degree electrical degrees will be used for rotating the rotor for each solution evaluation for peak torque detection or any such analyses.

14.12.4 Instantaneous solution

```
motor.solve.instantaneous=0
```

Setting the above variable to 1 results in a snapshot evaluation of the motor, when the r_solve(Motor) is called.

14.12.5 mini mode

```
motor.solve.mini=0
```

Setting the above variable to 1 results in truncation of solution parameter space, such as Valuesbelowpeak and back-emf evaluation. This setting exists only to reduce the motor solution time, when all the parameters are not required for comparison.

14.12.6 Values to Output

```
motor.solve.tooutput
```

The above variable is a string table that holds the variables to solve the motor for. This table also includes some settings to obtain the parameters such as coordinates, repetitions, and angle offsets. Editing this table from the GUI is relatively easy. To update the values from the command line, use strings for all the value updations at the corresponding positions marked by coordinates in the array, eg:Motor.Solve.ToOUTPUT(2,1)='0'.

14.12.7 Characterizing the motor

Motor may be run in parallel mode to run faster characterizations.

```
motor.agents=6
motor.par=1
motor.femmpath= "d:\femm4.2\bin\femm.exe" // or any valid path
//
    r1_findmotorchar(motor,c/m/m_sw/g/ind/ind_a/ind_a_vs_i/ind_sw/ind_sw_vs_i/m_vs_i,[imax,imin,steps],erev)
```

Setting the parallel mode to 1 and setting appropriate agent count will execute parallel FEMM sessions to get the characterization results faster.

This function plots the motor's torque or back emf waveforms or inductance for one full electrical cycle. Use second argument 'c' or 'm' or 'g' or 'm_sw' or 'ind' or 'ind_A' or 'ind_A_vs_i' or 'm_vs_i' for only cogging mode or motoring mode or generation mode plots, motoring mode with switching, phase inductances, phase A inductance, and inductance of phase A with current.

This function consumes considerable time. m mode considers same switch combinations all around, while m_sw considers switches for the corresponidng rotor position. ind or ind_A accept an optional third argument of current at which inductance needs to be calculated. Exclusion will result in calculation at IDCmax.

ind_A_vs_i accepts 3 inputs that are max current, min current and steps. exclusion of any of these values will cause : max to be set at idcmax, min to be set at -max, and 11 steps.

ind_A_vs_i will take considetable time to finish.

ind_sw will calculate the cumulative inductance of as seen by the controller for one switching instance for a given dc bus current limit, ind_sw_vs_i will range the currents and plot. If the pc supports and when you want to run parallel agents for faster characterization for ind, ind_A_vs_i and ind_sw_vs_i; use Motor.par=1 and Motor.FEMMpath='yourfemmexecutablepath\femm.exe'.

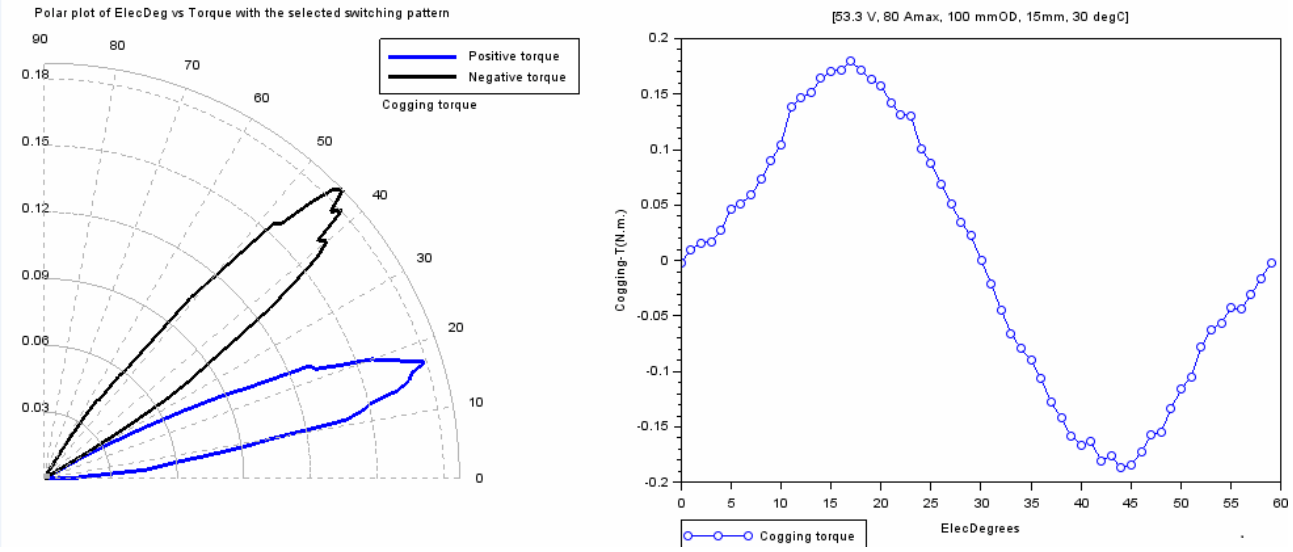
Select Motor.agents=#of concurrent parallel sessions to be run. These are not native fields, and so you have to create. To disable parallelization, set par=0. It is disabled by default when the Motor structure is created. ind executes all three inductances in one go, ind_A_vs_i and ind_sw_vs_i execute all current variations in one go.

Position increments are taken as anoher parallel run. m, g and c run for multiple points across the electrical angle. m_sw implementation is buggy.

The radial pull force on the rotor is also calculated for every evaluation in m_vs_i mode.

14.12.8 Cogging torque vs rotor angle

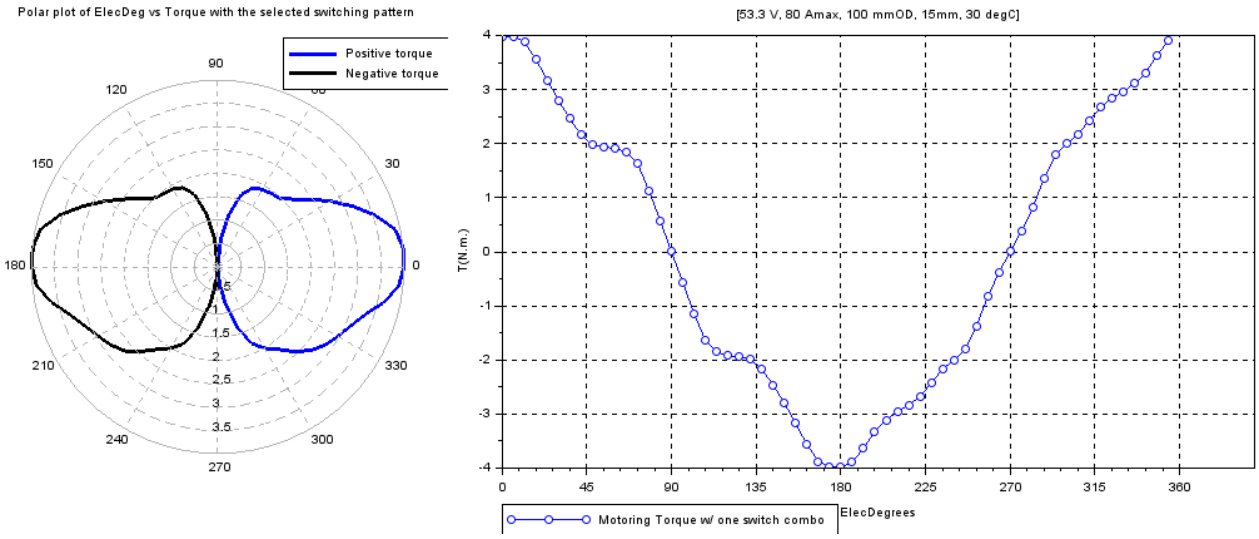
```
char=r1_findmotorchar(motor,'c')
```



14.12.9 Torque vs rotor angle

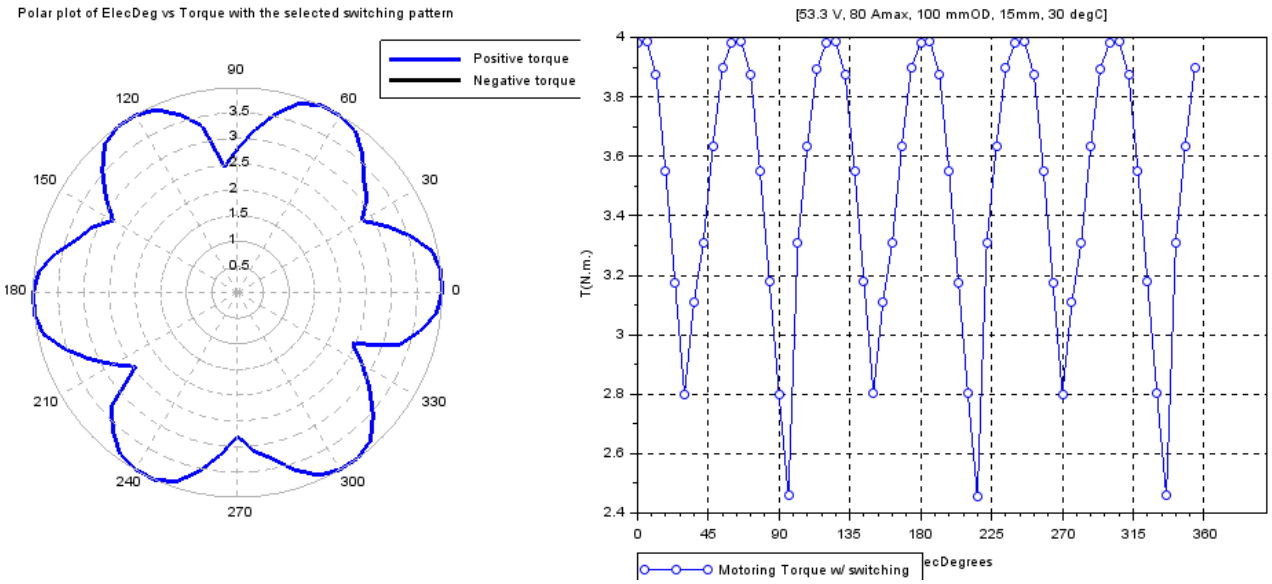
With single switch combo turned-on as the rotor rotates for 1 electrical revolution

```
char=r1_findmotorchar(motor,'m')
```



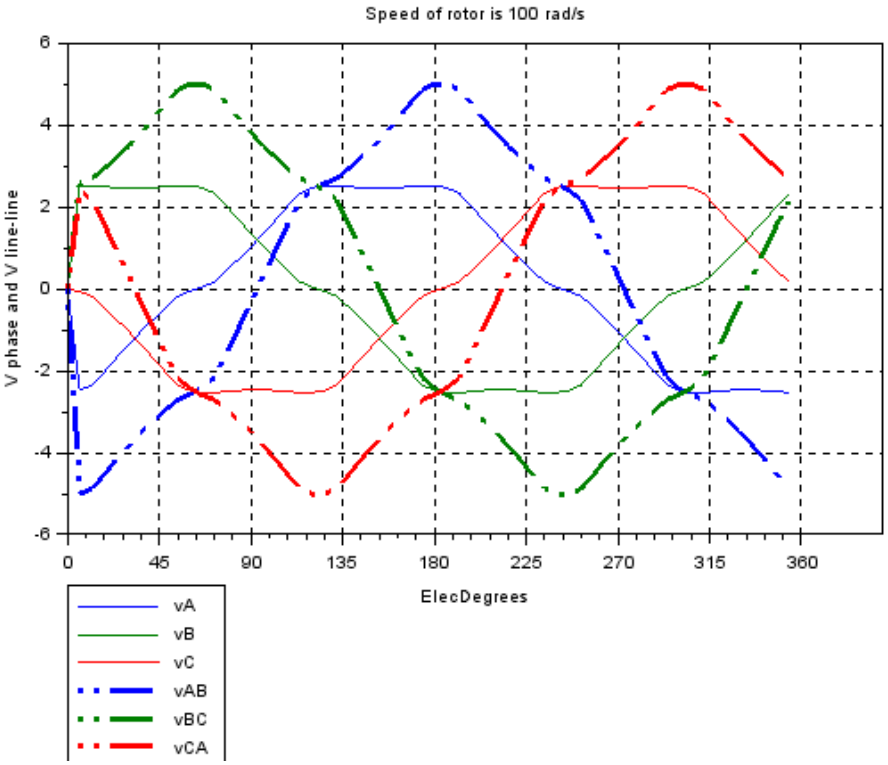
With appropriate switch combo turned-on as the rotor rotates for 1 electrical revolution

char=r1_findmotorchar(motor,'m_sw')



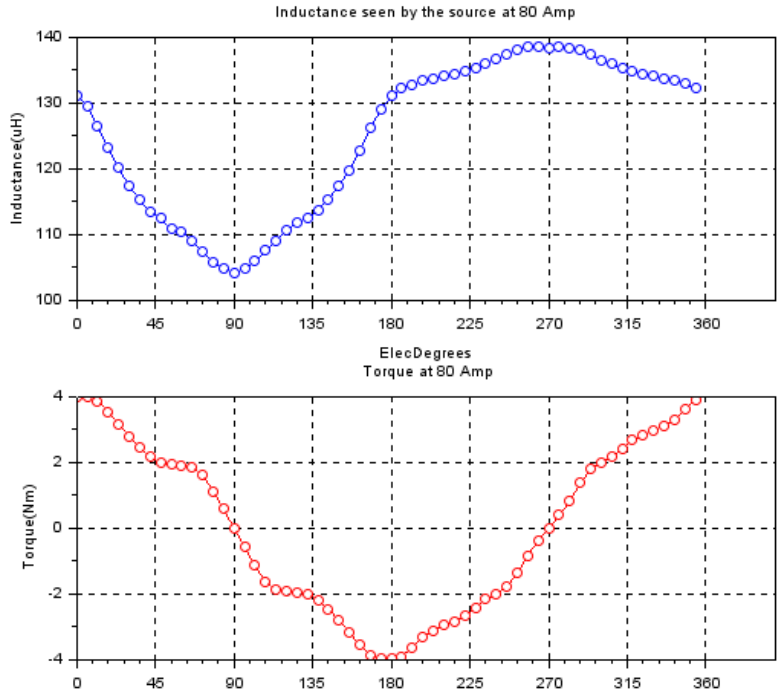
14.12.10 Induced voltages vs rotor angle

char=r1_findmotorchar(motor,'g')



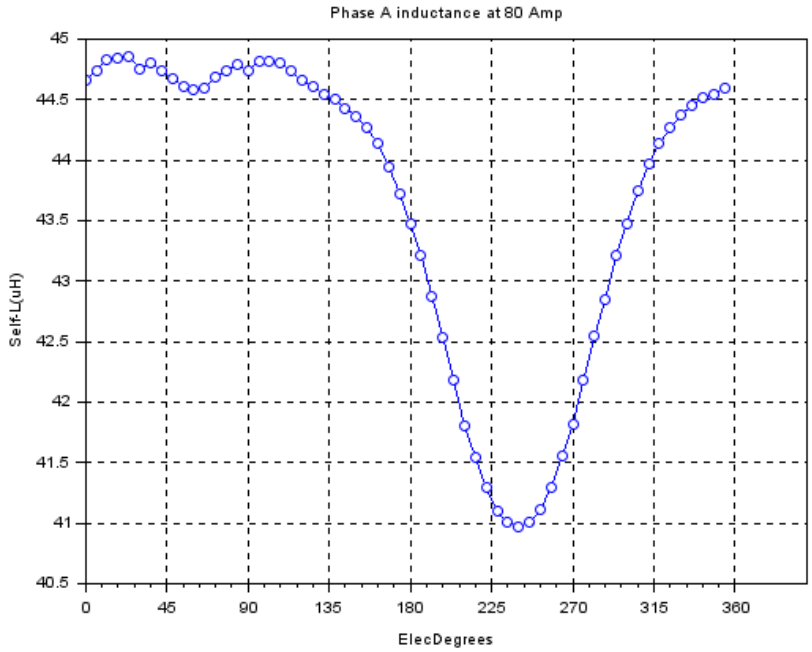
14.12.11 Line inductance & Torque vs Rotor angle

```
char=r1_findmotorchar(motor,'ind_sw')
```



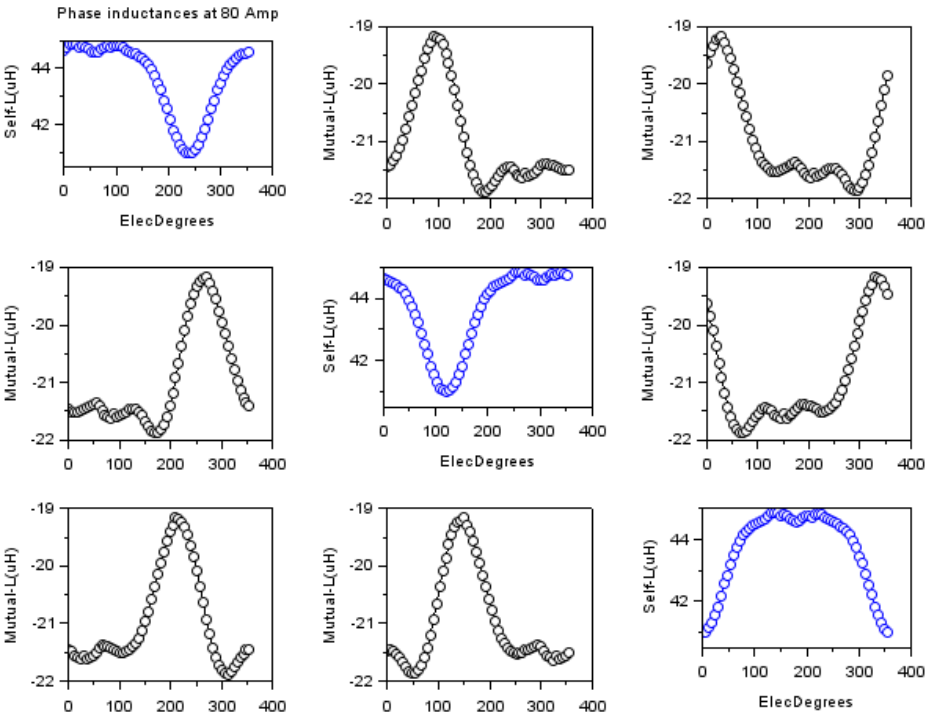
14.12.12 Phase inductance vs Rotor angle

char=r1_findmotorchar(motor,'ind_a')



14.12.13 Self and Mutual inductances vs rotor angle

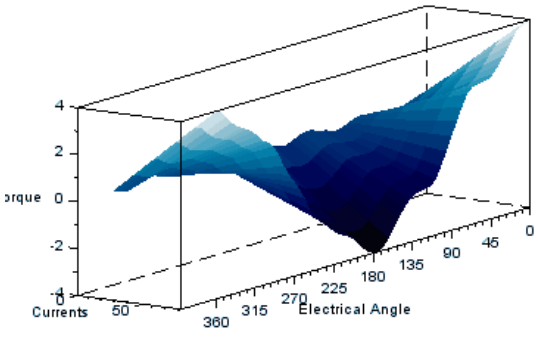
char=r1_findmotorchar(motor,'ind')

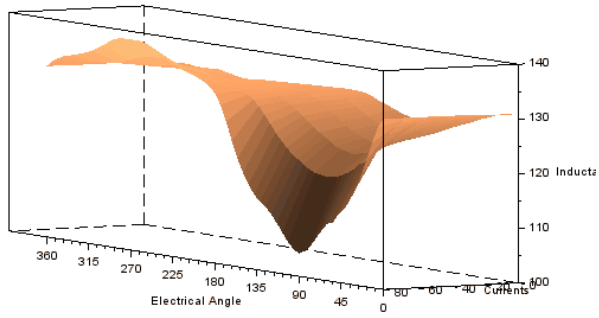


14.12.14 (Torque and Inductance) vs Current Vs rotor position

```
char=r1_findmotorchar(motor,'ind_sw_vs_i')
```

Some images from the 3d plot resulting from the characterization is presented below.

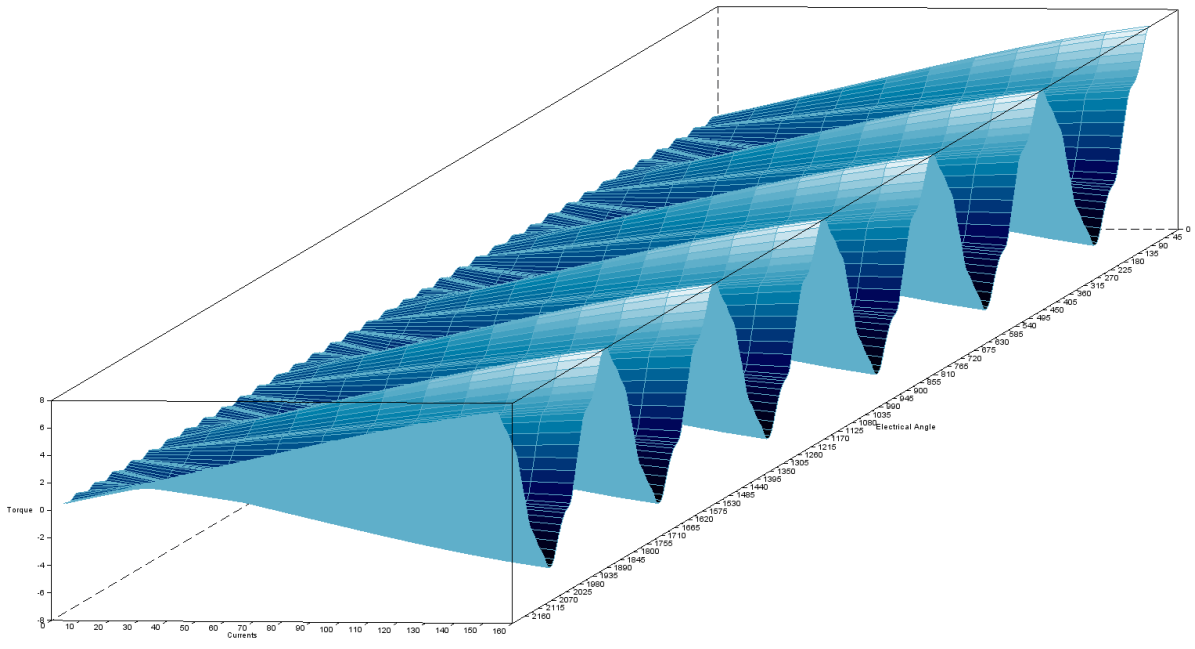


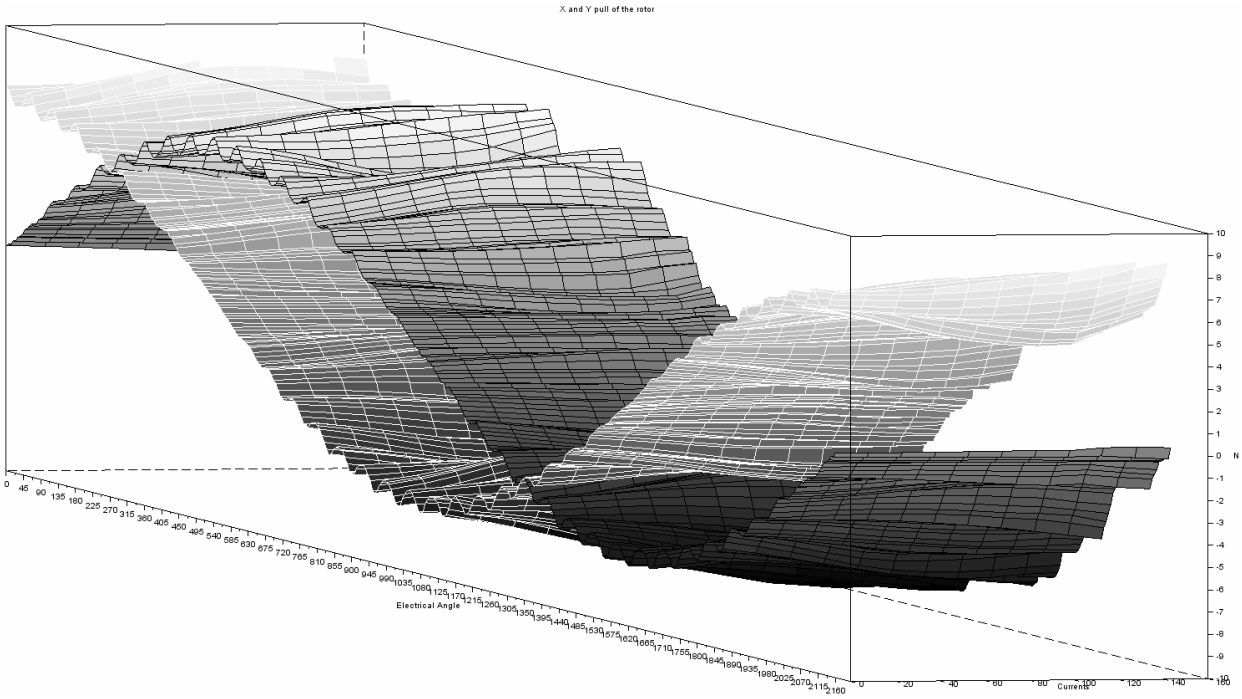


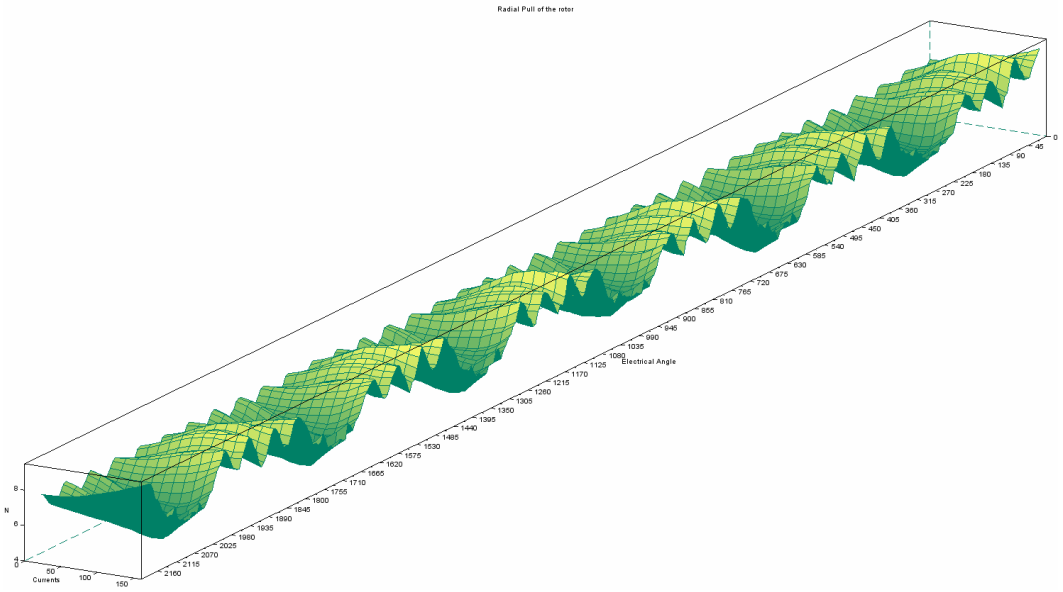
14.12.15 Radial pull

```
char=r1_findmotorchar(motor,'m_vs_i')
char=r1_findmotorchar(motor,'m_vs_i',150,0,11,6)
```

This characterization gives torque-spread, in addition to the radial pull. The above characterization with a radial offset error of 0.2mm in x direction is run for 6 electrical revolutions (one full mechanical revolution) to give full view of the radial force as below.



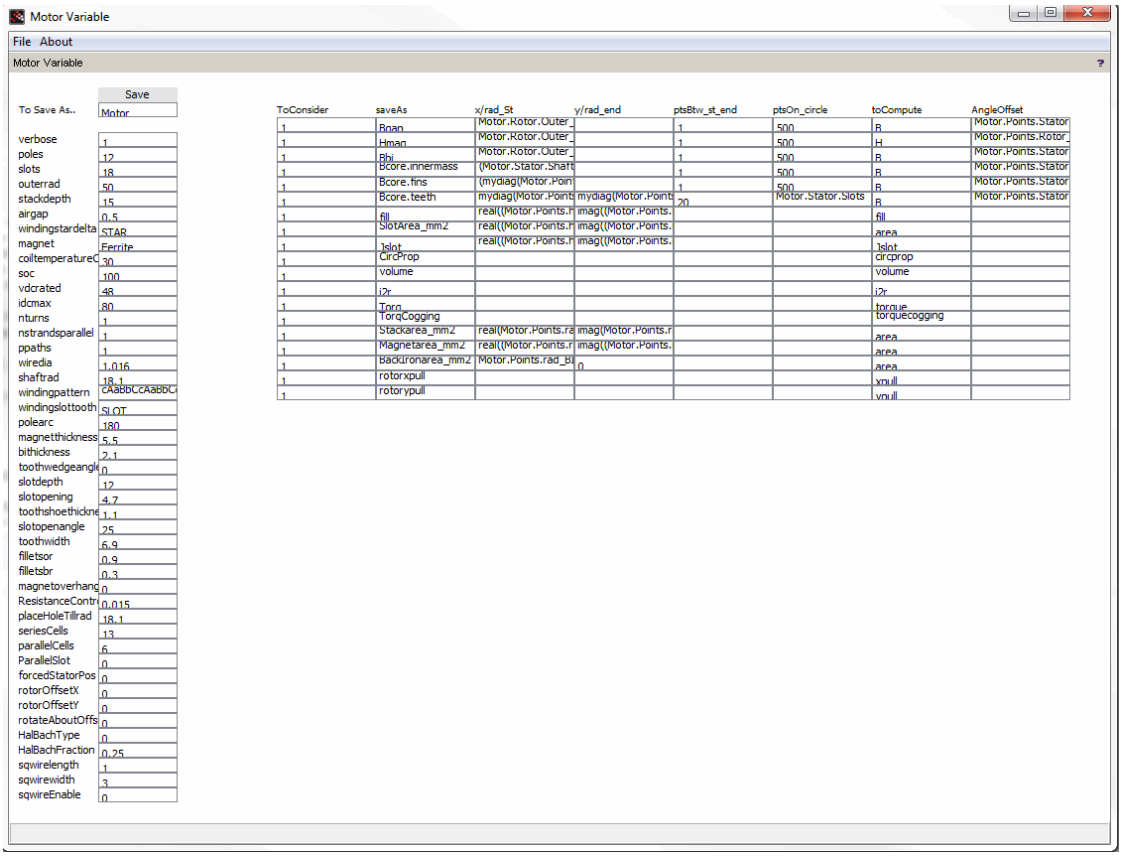




14.13 GUI

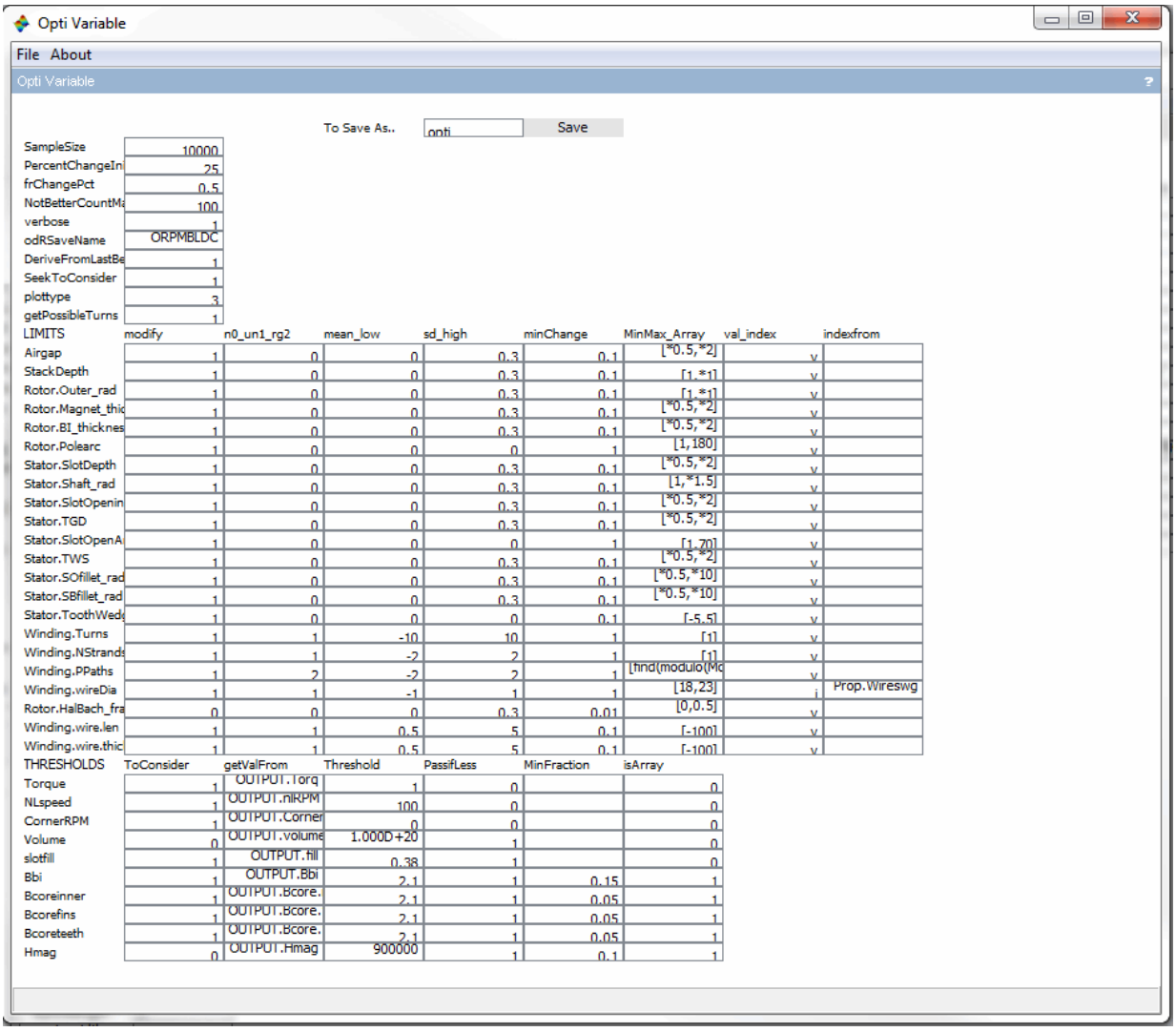
14.13.1 Motor

All the script based operations to change the motor parameters, and saving the model from command prompt can be done from the GUI too. The settings can be saved using the save button. Don't use space or numbers at the beginning of the string for the name.



14.13.2 Optimization

All the script based operations to change the optimization parameters, and saving the settings from command prompt can be done from the GUI too. The settings can be saved using the save button. Don't use space or numbers at the beginning of the string for the name.



15 Working with IR PM BLDC

15.1 Activate the libraries

The rlib has many functions which are split according to their use, into individual binary files. Loading the binary files also loads the functions into the Scilab's working memory. Additionally, FEMM provides a set of linking functions that interface Scilab with FEMM through a dll file and corresponding scilab functions.

The common_odr file has functions that are essential to handle the data. The common_motor file has functions that are essential for all motor topologies. The geom_irbldc file has functions that are useful for creating the motor variable and the optimization variable for the irpmbldc topology. In addition, this file also has functions to build the motor model in FEMM from the motor variable.

Script to activate the libraries is below.

```
exec('d:\femm4.2\scifemm\scifemm.sci', -1);
load('y:\motor\scripts\r_lib\rlib_odr.bin');
load('y:\motor\scripts\r_lib\rlib_common.bin');
load('y:\motor\scripts\r_lib\rlib_irbldc.bin');
```

15.2 Build the model from the default template

The geometry from the motor variable is checked for any errors and geometrical feasibility. If the motor cannot be constructed for any reason, the reason is printed out and the model is not built. Script to build the model with default template is below.

```
motor=r_buildmotor(12,18) // poles, slots
```

or alternatively,

```
motor=r_buildvar(12,18) // poles, slots
motor=r_buildmotor(motor)
```

The motor variable can be constructed with as little as 2 inputs and as many as 8 inputs.

```
motor=r_buildvar(12,18) // poles, slots
motor=r_buildvar(12,18,50) // poles, slots, outer radius
motor=r_buildvar(12,18,50,15) // poles, slots, outer radius, stack depth
motor=r_buildvar(12,18,50,15,0.5) // poles, slots, outer radius, stack depth, air gap
motor=r_buildvar(12,18,50,15,0.5,'star') // poles, slots, outer radius, stack depth, air gap, winding type
```

```
motor=r_buildvar(12,18,50,15,0.5,'star',48) // poles, slots, outer radius, stack depth, air gap, winding type, voltage range
motor=r_buildvar(12,18,50,15,0.5,'star',48,80) // poles, slots, outer radius, stack depth, air gap, winding type, voltage range, current limit
// each of the above generates a valid motor variable, assuming the default values for rest of the parameters wherever they are not provided.
```

Or, the simpler notation as below can be used to specify only those parameters that are of interest, while others are assumed to be same as the in the default template.

```
motor=r_buildvar('poles',12,'slots',18,'throw',2)
motor=r_buildmotor(motor)
// or
motor=r_buildmotor('poles',12,'slots',18,'throw',2)
```

Special calling references: poles/pole, slots/slot, outerrad/rad, outerdia/dia, stacklen/len, airgap/gap, winding/wind/windtype, vdcrated/vdc, idcmax/idc, and coilthrow/throw.

The r_buildMOTOR function call with the same inputs as r_buildVAR will build the motor with the parameters, effectively reducing the command count by 1. Similarly, the r_solve after r_buildVAR will also build the motor.

15.3 Configuration

15.3.1 Poles and Slots

The poles and slots need to be selected to incorporate a balanced winding pattern. A balanced winding is the one in which all the three phase vectors on the 2-D surface of the motor winding are separated by 120 electrical degrees. Only a few pole slot combinations are valid. The rlib automatically chooses the rest of the parameters such as the tooth width, magnet arc length, etc.

```
motor.rotor.poles=12
//poles have to be a multiple of 2
motor.stator.slots=18
//slots have to be a multiple of 3
```

Poles and slots combination alters few other geometric parameters on the stator, rotor as well.

15.3.2 Outer radius and stack depth

These two are independent parameters and are global values. The rlib chooses rest of the parameters of the motor to fit within the outer_radius parameter. When motor variable is created using the r_buildVAR or r_buildMOTOR, with only the outer radius, the default template will come up with rest of the parameters scaled appropriately to result in a valid motor geometry. The stack depth is the length of the motor in the axial direction. These two parameters can be set with the following script

```
motor.stator.outer_rad=50
motor.stackdepth=15
```

15.3.3 Air gap

Air gap of the motor is the distance between the stator and the rotor when the motor is assembled. The rlib changes the stator geometry and retains the rotor geometry when the air gap is changed, while the template variable is being created.

```
motor.airgap=0.5
```

15.3.4 Tooth wedge angle

The tooth wedge angle is the angle of the tooth edge with respect to the line through its center. This angle can be customized in rlib resulting in many possibilities. It is necessary to observe that the toothwedgeangle for ORPMBLDC is opposite in sign to that of this motor type, due to the notation used during the consutrction.

```
motor.stator.toothwedgeangle=-5
```

15.3.5 Magnet PoleArc

Pole arc is the fill of each pole in one electrical angle span. This alue is in the range (0,180]. Script to change the magnet's pole arc

```
motor=r_buildvar(12,18)
motor.winding.turns=8
motor.stator.forcedparallelslot=1
motor.rotor.polearc=120
motor=r_buildmotor(motor)
```

15.3.6 Magnet thickness

Magnet thickness is the thickness of the hard magnetic material in radial direction. Any air gaps between magnets and the back-iron is not assumed. All the magnets are modeled as arc magnetic elements. No chamfer or a fillet addition to the edge is modeled.

```
motor.rotor.magnet_thickness=5.5
```

15.3.7 Stator back iron thickness

Stator back iron is the supporting structure for the stator teeth, and also the conductive path for the magnetic field lines through the stator teeth.

```
motor.stator.bi_thickness=2.1
```

15.3.8 Stator slot depth

```
motor.stator.slotdepth=20
```

Slot depth is a measurement taken from the slot bottom point to the edge of air-gap. depth can not also change infinitely, because it affects several other parameters such as tooth width etc. too.

15.3.9 Stator slot opening

```
motor.stator.slotopening=2
```

Slot opening is the gap between the consecutive tooth shoe fins in tangential direction. More slot opening lets the winding slip in easily, but increases flux leakage within the stator.

15.3.10 Stator slot open angle

```
motor.stator.slotopenangle=25
```

The slot opening angle is the angle cast by the line joining stator tooth and the tooth shoe's inner edge(away from the airgap), with the tangential line at the tooth shoe's inner edge.

15.3.11 Stator tooth shoe thickness

```
motor.stator.tgd=1.1
```

Tooth shoe is the construction at the end of the tooth stem towards the air gap, usually wider than the tooth width.

15.3.12 Stator tooth width

```
motor.stator.tws=6.8
```

Stator tooth width is the width of the tooth towards the airgap. This width may change gradually as we move towards the center of the motor, if the toothwedgeangle is not equal to zero.

15.3.13 Stator Slot bottom fillet radius

```
motor.stator.sbfillet_rad=1.1
```

This is the fillet radius at the bottom of the slot.

15.3.14 Stator Slot opening fillet radius

```
motor.stator.sofillet_rad=1.1
```

This is the fillet radius of the slot's top corner.

15.4Additional Geometric parameters

15.4.1 Magnet overhang

```
motor.rotor.magnetoverhang=0
```

This is a parameter that is not used for any electromagnetic calculations, and is purely used for calculating the mass of magnets in the solution parameters.

15.4.2 Forced Parallel Slots

```
motor.stator.forcedparallelslot=0
```

This is a parameter that is used to create parallel slots, irrespective of pole & slot combination. This parameter overrides toothwedgeangle and always creates the parallel slots with the required toothwedgeangle.

15.5 Topological variations

15.5.1 Halbach motor type 1

This type of motor has radial and lateral magnet arrangement. To achieve this effect, use the following code.

```
motor.rotor.halbach_type=1
motor.rotor.halbach_fraction=0.25
```

15.5.2 Halbach motor type 2

This type of motor has inclined and lateral magnet arrangement. To achieve this effect, use the following code.

```
motor.rotor.halbach_type=2
motor.rotor.halbach_fraction=0.25
```

15.5.3 Rectangular Magnets

This arrangement will create a realistic representation of the magnet arrangement on the rotor when the rectangular magnets are used. The airgap parameter is used to create shortest distance between the magnets and the stator.To have this construction done, the halback type needs to be set to zero, and the rotor's polearc needs to be set to 180, else this parameter is reset to 0. To achieve this effect, use the following code.

```
motor.rotor.rectangularmagnet=1
motor.rotor.rectangularmaggluethick=0.05 // has to be at least 0.01 mm
motor.rotor.rectangularmagspacing=0.05 // has to be at least 0.01 mm
```

15.6 Constructional Parameters

15.6.1 Winding

```
motor.winding.pattern='abcabcabcabcabcabc' // calculated automatically. may be overwritten.
motor.winding.type='star' // can be of type 'star' or 'delta'
motor.winding.marking_slot_tooth='tooth' // can be of type 'tooth' or 'slot'.
```

```
// tooth winding assumes concentrated winding, and the pattern in motor.winding.pattern refers to how the phase wires are would around individual tooth. capital letter means that the winding is counter-clockwise when seen from the air gap.

// when tooth winding pattern is given, a '-' symbol in the motor.winding.pattern indicates a tooth left unwound. alternate '-' symbols result in a single layer winding. the length of motor.winding.pattern has to be same as the number of slots, for the winding pattern to be valid.

// slot winding can be used to represent any type of winding, be it concentric, concentrated or distributed, and the pattern in motor.winding.pattern refers to how the phase wires go in or come out of the slots. capital letter means that the positive current comes out of the screen (positive winding turns).

// when slot winding pattern is given, a pattern with equal length as the number of slots result in single layer winding. a pattern with length equal to double the slot count result in double layer winding.

// at the moment, transposed winding is not implemented in the geometry. winding space for two halves in the double layer winding is divided along the slot depth.

// winding pattern is implemented in the geometry from a horizontal line and progresses clockwise (in the direction of the phases)


motor.winding.turns=10

motor.winding.nstrands=1

motor.winding.ppaths=1

//the above three define how the winding is done. the nstrands and ppaths are used while calculating phase resistances. winding.type is used while calculating line resistance. nstrands is also used to plot the total conductors when individual conductors are plotted in the geometry.

motor.winding.wire.rectangular=0

//the above setting is the default. if it is 0, circular cross section of the wire is assumed from the below variable.

motor.winding.wiredia=1.016

//if the rectangular property is set to 1, then the following two variables decide the geometry of the wire.

motor.winding.wire.len=1

motor.winding.wire.wid=2

//for representing the individual conductors in the geometry, set the following variable to 1 as shown
```

```
motor.winding.drawwiresemag=1
// wire seperation and insulation thickness of the wires in the geometry can be changed by using the following variables
motor.winding.wiregap=0.1
motor.winding.insulthickness=0.05
//if resistance estimation is wrong, either because the winding is concentric or distributed type, or because of other reasons, the following variable can be used to force set the resistance to a predetermined value
motor.winding.projectedrph=0
//setting the variable to 0 means that the value calculated by rlib is to be used.
//additionally, a correction factor for the resistance calculation can be included, by the following variable.
motor.winding.correctionrph=1
//the resistance used in the rlib calculations is the estimated resistance multiplied by the above factor. a value of 1 indicates no change in the estimated resistance.
motor.winding.throw=2; // coil throw represented in the number of slot pitches.


//
//
```

The rlib checks for winding feasibility and generates the winding pattern automatically for the BLDC configuration of single-throw / concentrated winding.

The multi-throw winding will give rise to a configuration in which the slots are split horizontally.

15.6.2 Magnet

```
motor.rotor.magnettype='ferrite'
```

The magnet may be chosen as Ferrite or NdFeB or any other material from the material choice available from the FEMM material library. Magnet's pole arc can be changed too.

15.7Control parameters

15.7.1 Battery voltage

```
motor.controller.vdcrated=12 // or 48
motor.controller.cellsinseries=1 // or more
```

This is a very important parameter that decides the maximum speed of the motor. This parameter does not directly decide the operating voltage. This parameter is implemented in two variants of the batteries. When this value is chosen to be less than 15, Lead-Acid is activated. Any value higher than this will indicate Li-ion cells. When the r_buildVAR is used with the Vdcrated parameter, the SoC_V table is automatically populated with the corresponding cell's Soc vs Voltage and cell resistance value. For a 12 V system, it is assumed that each cell corresponds to 12v nominal voltage. To build a 48v operating voltage, adjust the Motor.Controller.CellsInSeries to 4. For a Li-ion cell system, default number of cells resulting in 48V is 13. Change this number accordingly to adjust the operating voltage. CellsInParallel only effects the pack resistance.

15.7.2 State of Change

```
motor.controller.soc=50
```

The state of change for a pack affects the pack voltage as per the SoC_V table. This table is useful for evaluating the performance of the motor at worst case conditions.

15.7.3 Winding temperature

```
motor.winding.temperaturec=30
```

The temperature of the winding is a condition that affects the stator resistance seen by the power source, which will reduce the corner rpm on the speed-torque plot for the motor.

15.7.4 Motor Current

```
motor.controller.idcmax=80
// forced currents into the winding when solving the model can be set using the following variable.
motor.winding.cur=[-80 80 0]// these represent the phase currents. in this example, these correspond to star winding currents.
```

The IdcMax parameter will set the absolute maximum DC current allowed by the controller. The Cur field in the winding is the current that is passed into the winding for each phase. This cur field is set automatically while solving for the motor model based on the resistance of the winding and the rest of the path, available voltage, and the controller dc current limit. However, a known value may also be given to the model for solving for performance.

15.7.5 Controller resistance

```
motor.controller.rpath=0.015
```

The controller path resistance is the PCB resistance that is added to the whole resistance calculation. The complete resistance faced by the available battery voltage decides the corner rpm and the maximum current that the motor can accept for a given voltage.

15.8 Material parameters

The field Motor.Prop has all the custom materials defined with their properties. Any materials specified for the magnets from the materials library will assume default values.

15.9 Wire placement

15.9.1 Circular conductors

Script to create the model with individual circular conductors

```
motor=r_buildvar(12,18)
motor.winding.drawwiresemag=1
motor=r_buildmotor(motor)
```

15.9.2 Rectangular conductors

Script to create the model with individual rectangular conductors

```
motor=r_buildvar(12,18)
motor.winding.drawwiresemag=1
motor.winding.wire.rectangular=1
motor.winding.wire.wid=1.5
motor.winding.turns=8
motor=r_buildmotor(motor)
```

15.9.3 Parallel slots

Script to create parallel slots, instead of parallel tooth, with rectangular conductors

```
motor=r_buildvar(12,18)
motor.winding.drawwiresemag=1
motor.winding.wire.rectangular=1
motor.winding.wire.wid=3
motor.winding.turns=8
motor.stator.forcedparallelslot=1
motor=r_buildmotor(motor)
```

15.10 Fault Simulation

15.10.1 Rotor assembly - radial offset emulation

Script to re-create the assembly failure - rotor offset in radial direction

```
motor=r_buildvar(12,18)
motor.rotor.offset.x=0.25
motor.rotor.offset.y=0
motor=r_buildmotor(motor)
```

```
motor.rotor.offset.rotateaboutneworigin
```

The above variable can let us select the type of offset - static (the offset that does not rotate when the rotor rotates) when set to 1, and dynamic (the offset that rotates around along with the rotor) when set to 0.

15.11 Running the FEA

15.11.1 Finding the solution

```
motor=r_solve(motor)
motor=r_solve(motor,[0,0,0])
```

The above command will solve the motor based on the current constraints and the voltage input provided in Motor.Controller.

FEMM gives pretty standard outputs for B, H, J across the cross-sectional area.

This function requires a motor structure as an input and optional phase currents [i1,i2,i3] to solve using FEA. If the phase currents are not provided, the geometry is used to calculate resistance, and the controller settings are used to calculate the currents in each phase.

This functions requires the FEMM model to have been created.

This function returns a motor structure with OUTPUT values updated based on the solution. Then the Motor.RUN.dAngle_md is set to non zero values (or when Motor.Solve.ToOUTPUT's torque line has no starting and ending and incrementa angles), the perturbation is not done to calculate the kb and NLrpm.

The torque after rotating by the angle Motor.RUN.dAngle_md is calculated, along with few other parameters.

OUTPUT is populated based on Motor.Solve.ToOUTPUT parameters.

The syntax is B/H: name,starting radius, ending radius, number of points between radii(including), number of points on each circle, B/H.

fill factor, slot area: name, x coordinate, y coordinate,,,fill/area.

circuit properties, volume, i2r: CircProp/volume/i2r,,,,,circprop/volume/i2r.

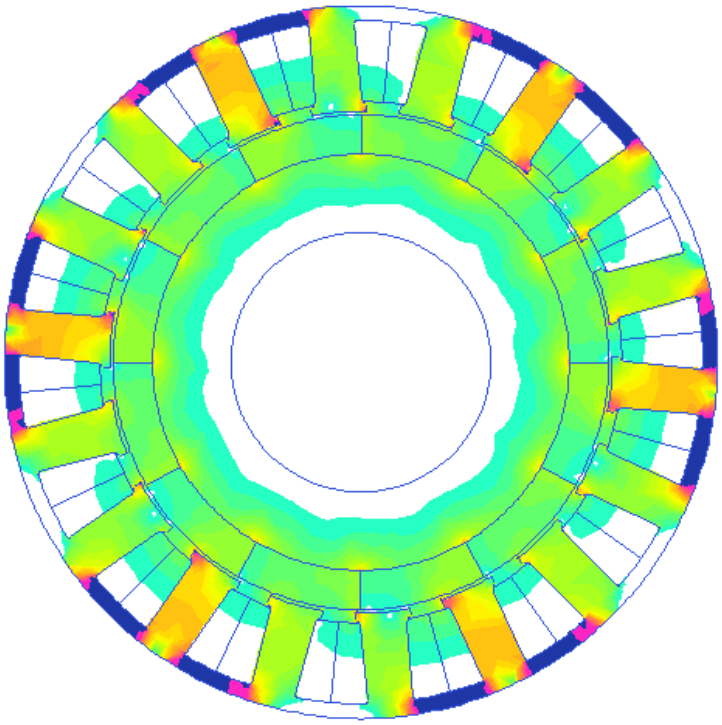
Torque: Torq,startAngle,EndAngle,IncrementalAngle,,torque.

When the start/end/increments are given as empty, instantanious fea is done for the single rotor position. When the Motor.RUN.dAngle_md is given to be other than 0, rotor is rotated to that position and solved once.

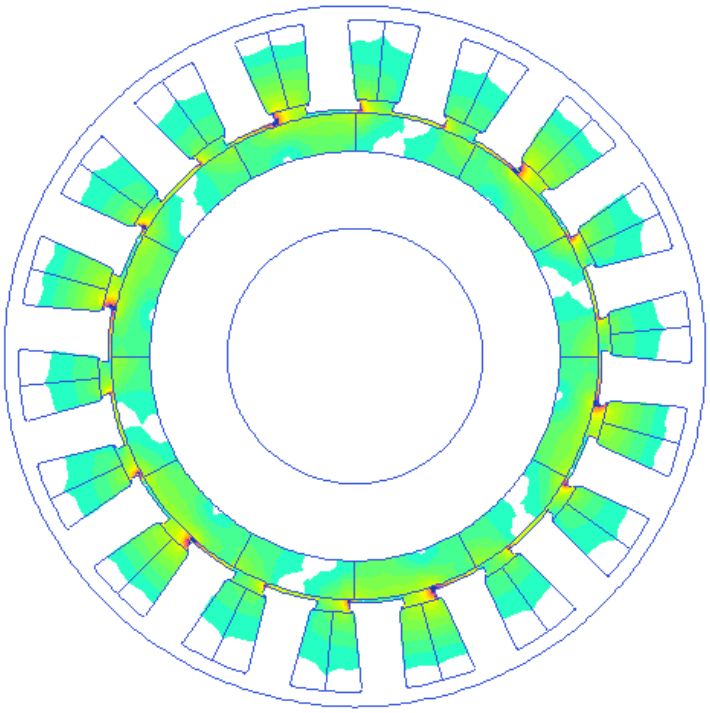
To solve for no load speed, backenmf constant, keep the torque fields active, and provide a zero Motor.RUN.dAngle_md. Once solved, the rotor is placed at initial position.

15.12 FEMM's built-in graphical plots

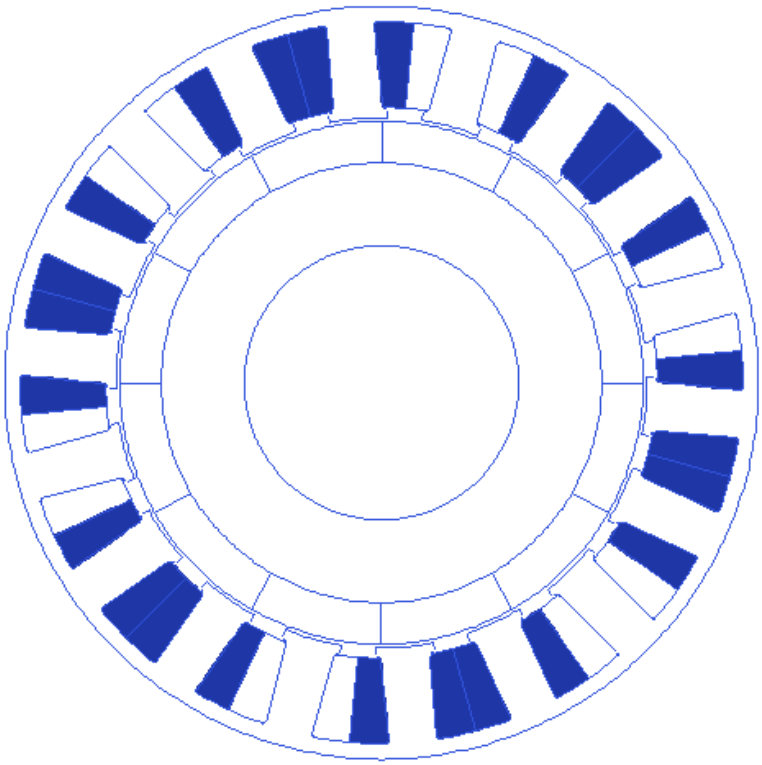
15.12.1 Magnetic flux density (B field)



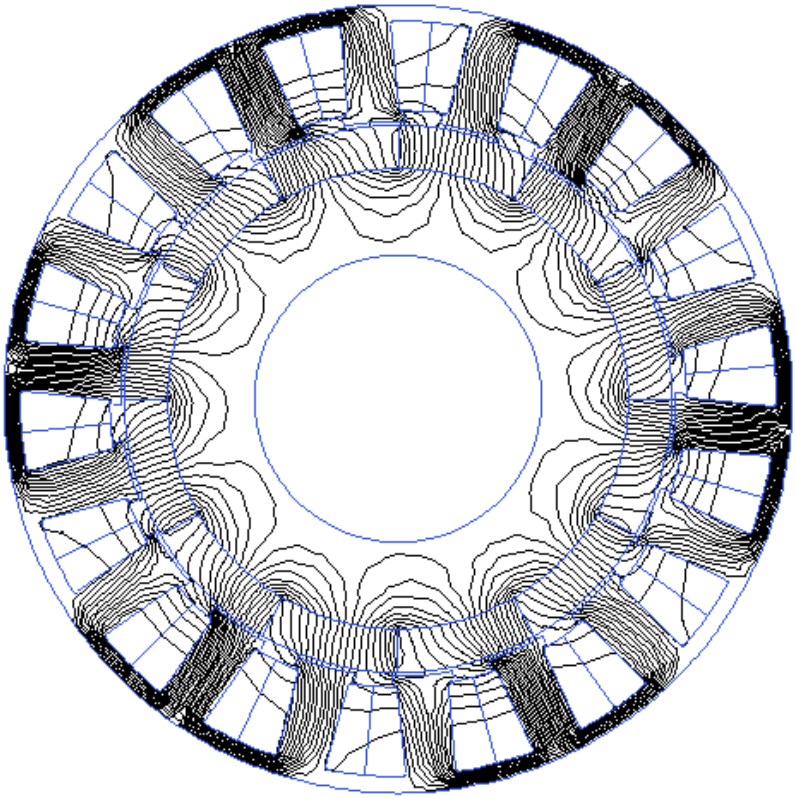
15.12.2 Magnetic field strength (H Field)



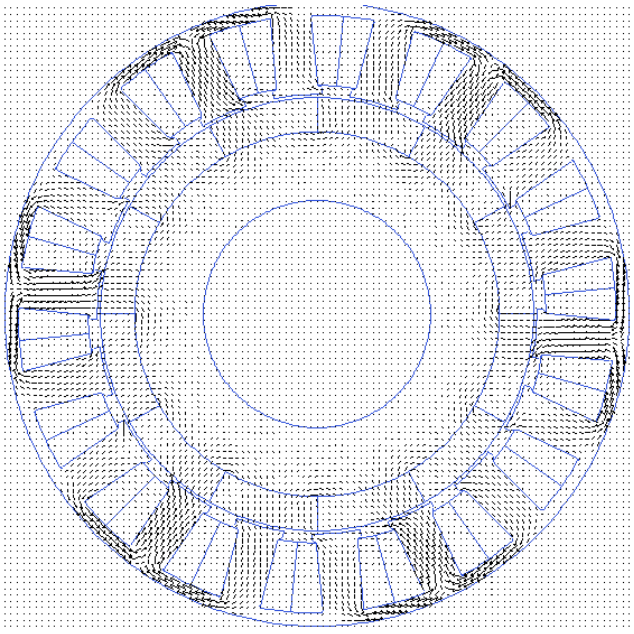
15.12.3 Current density distribution



15.12.4 Iso field lines distribution



15.12.5 Arrow plot for B field

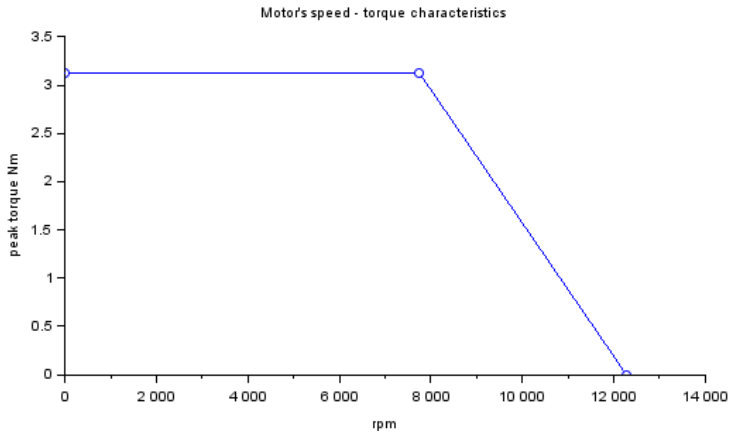


15.13 Quantified parameters - with rlib

In addition to the default plots from the FEMM, rlib calculates and plots additional parameters and provides them for the optimization routine.

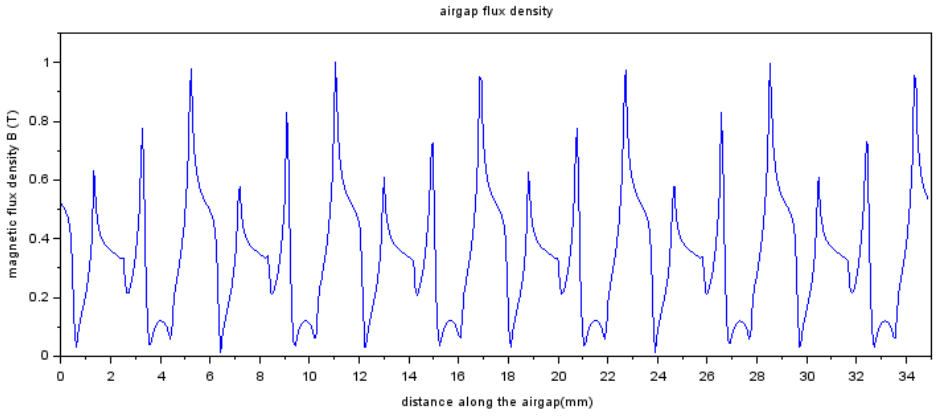
15.13.1 Speed- Torque characteristics

```
plot(motor.output.rpm_nm(:,1),motor.output.rpm_nm(:,2),'.-o')
xlabel('rpm')
ylabel('peak torque nm')
title('motor''s speed - torque characteristics')
x=gca()
x.box='off'
```



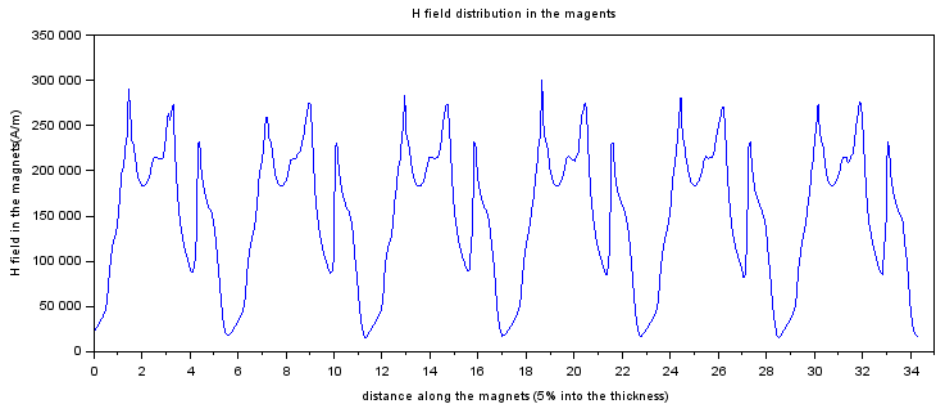
15.13.2 Airgap flux density

```
x=(0:499)*motor.points.rad_airgap/500
a=motor.output.bgap
plot(x,a)
xlabel('distance along the airgap(mm)')
ylabel('magnetic flux density b (t)')
title('airgap flux density')
```



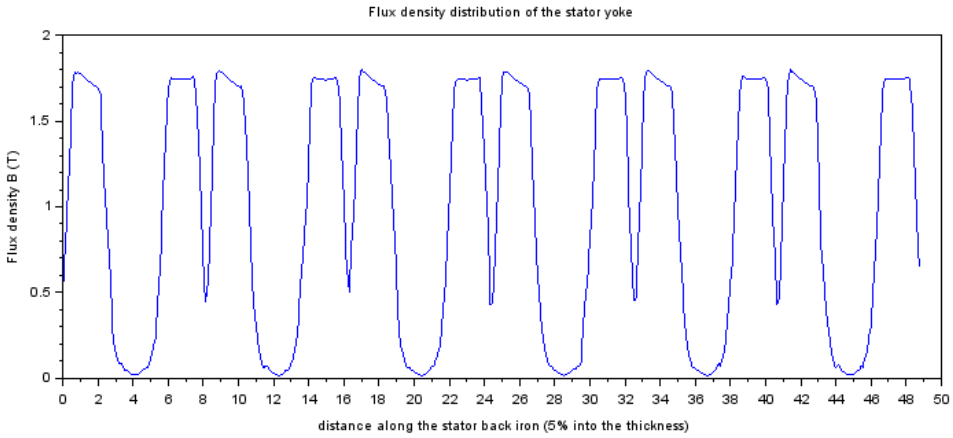
15.13.3 H field inside the magnets

```
x=(0:499)/500;x=x*(motor.stator.outer_rad-motor.stator.bi_thickness- motor.stator.slotdepth-
    motor.airgap - motor.rotor.magnet_thickness*0.05)
plot(x,motor.output.hmag)
xlabel('distance along the magnets (5% into the thickness)')
ylabel('h field in the magnets(a/m)')
title('h field distribution in the magents')
```



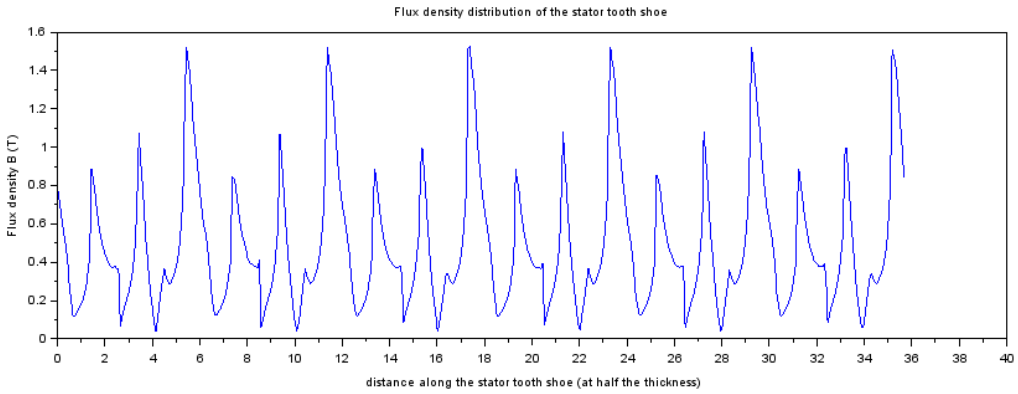
15.13.4 Stator backiron satiration

```
x=(0:499)/500;x=x*(motor.stator.outer_rad-motor.stator.bi_thickness*0.5)
plot(x,motor.output.bbi)
xlabel('distance along the back iron (5% into the thickness, towards the magnets)')
ylabel('magnetic flux density b (t)')
title('back iron flux density distribution')
```



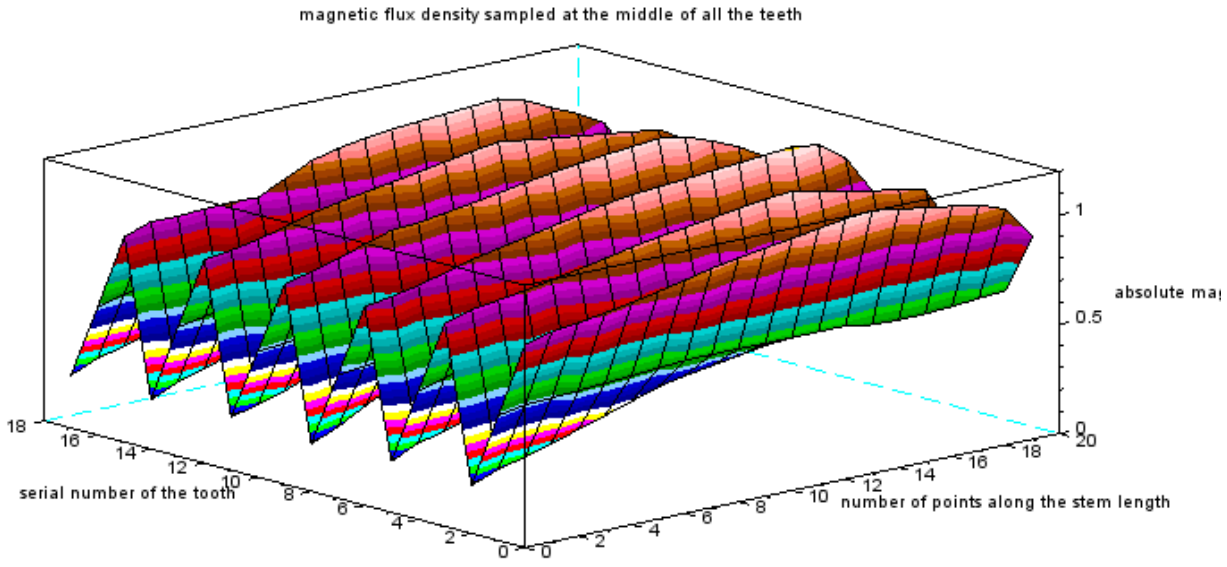
15.13.5 Stator teeth shoe saturation

```
x=(0:499)/500;x=x*((mydiag(motor.points.toothfinouter.x,motor.points.toothfinouter.y) +
    mydiag(motor.points.toothfininner.x,motor.points.toothfininner.y))/2)
plot(x,motor.output.bcore.fins)
xlabel('distance along the stator tooth shoe (at half the thickness)')
ylabel('flux density b (t)')
title('flux density distribution of the stator tooth shoe')
```



15.13.6 Stator teeth stem saturation

```
surf(motor.output.bcore.teeth,'facecol','interp')
xlabel('number of points along the stem length')
ylabel('serial number of the tooth')
zlabel('absolute magnetic flux density')
title('magnetic flux density sampled at the middle of all the teeth')
```



15.14 OUTPUT quantities

15.14.1 Torque

The following stores the peak or instantaneous torque value

```
motor.output.torq
```

The following stores the average torque value, for non-instantaneous solves

```
motor.output.torqavg
```

The following stores the cogging torque value, for non-instantaneous solves

```
motor.output.torqcogging
```

The following stores the average torque value, for non-instantaneous solves and full solves

```
motor.output.torqavg
```

15.14.2 RPM

The following stores the maximum rpm of the machine

```
motor.output.nlrpm
```

The following stores the corner rpm of the machine

```
motor.output.cornerrpm
```

Corner RPM of the machine is that rpm at which the motor's back-emf becomes sufficient to limit the current into the motor at exactly the controller's current limit. Any increase in the

motor's rpm will reduce the current into the motor below the controller's threshold, effectively reducing the torque production.

15.14.3 Back-emf constant

The following gives the back-emf constant

```
motor.output.kb
```

15.14.4 Resistance

The resistance of the motor is calculated from the expected copper length and its effective cross section. The effective copper length is calculated including the end-winding length, by considering the average path length for the winding. Any of the following two commands will give the phase resistance.

```
r1_findrphase(motor)
motor.output.rph
```

The following will give the line-line resistance

```
motor.output.rll
```

15.14.5 Operating voltage & currents

The following gives the operating battery voltage

```
motor.output.bat.voltage
```

The following gives the operating currents into the individual phases

```
motor.output.circprop(:,1)
```

15.14.6 Weight

The following gives the weight of copper

```
motor.output.kgcu
```

The following gives the weight of the Stator laminations

```
motor.output.kgstatorstack
```

The following gives the weight of the rotor backiron

```
motor.output.kgbi
```

The following gives the weight of the magnets

```
motor.output.kgmagnets
```

The following gives an estimated weight of the full motor, excluding shaft, bearings, end-covers, etc.

```
motor.output.kgmotor
```

15.14.7 Slot fill

The following gives the slot's fill factor. A number less than 40% is practically possible to be wound with hand for most motors.

```
motor.output.fill
```

15.14.8 Current density

The following gives the slot current density

```
motor.output.jslot
```

The following gives the copper current density

```
motor.output.jslot/motor.output.fill
```

15.14.9 Losses

The following gives the Joule's losses

```
motor.output.i2r
```

15.14.10 Radial forces

The following give the x, and y components of the radial pull force on the rotor.

```
motor.output.rotorxpull
motor.output.rotorypull
```

15.14.11 Inductances

The following give the D axis and Q axis inductances

```
motor.output.ld
motor.output.lq
```

15.15 Solver settings

15.15.1 Mesh settings

FEMM uses Triangle for mesh generation.

```
motor.solve.meshminangle=15 // mesh triangles' minimum angle. small angles reduce mesh node count
motor.solve.masseginarc=1 // minimum degrees of angle for each segment in an arc. smaller angle results in smoother arc
motor.solve.wiremeshsize=0.2 // mesh size in the wires
motor.solve.airmeshsize=0.4 // mesh size in the air. Higher value results in coarse mesh nodes.
```

15.15.2 Peak torque evaluation

```
motor.solve.valuesbelowpeak=2
```

This variable is a key to track peak torque of a motor. When the motor is being solved, one of the parameter is peak torque. The motor is aligned to a position which results in almost the peak torque, as per the design of rlib. However, due to some reluctance effects, the peak occurrence may shift to either left or right of the zero position. The peak detection algorithm rotates the rotor counterclockwise till it sees 2 (or the value from the variable above) values lower than the peak. Then it rotates clockwise to see 2 values lower than the peak, i.e. the peak should be surrounded by at least 2 values lower than itself. This is a critical parameter to eliminate local peak detection, and avoids the effects of ripples caused by cogging torque. More count results in more FEA runs and hence more time for the peak torque detection. The same variable is used for cogging torque detection also.

15.15.3 solutions in one electrical cycle

```
motor.solve.ptsin60degree=10
```

When the motor is being characterized, or being solved, the step rotation defines the smoothness of the generated curve. A total of 6*ptsin60degree points will be evaluated in one electrical cycle for characterization. A step size corresponding to 60/ptsin60degree electrical degrees will be used for rotating the rotor for each solution evaluation for peak torque detection or any such analyses.

15.15.4 Instantaneous solution

```
motor.solve.instantaneous=0
```

Setting the above variable to 1 results in a snapshot evaluation of the motor, when the r_solve(Motor) is called.

15.15.5 mini mode

```
motor.solve.mini=0
```

Setting the above variable to 1 results in truncation of solution parameter space, such as Valuesbelowpeak and back-emf evaluation. This setting exists only to reduce the motor solution time, when all the parameters are not required for comparison.

15.15.6 Values to Output

```
motor.solve.tooutput
```

The above variable is a string table that holds the variables to solve the motor for. This table also include some settings to obtain the parameters such as coordinates, repetitions, and angle offsets. Editing this table from the GUI is relatively easy. To update the values from the command line, use strings for all the value updations at the corresponding positions marked by coordinates in the array, eg:Motor.Solve.ToOUTPUT(2,1)='0'.

15.15.7 Characterizing the motor

Motor may be run in parallel mode to run faster characterizations.

```
motor.agents=6
motor.par=1
motor.femmpath= "d:\femm4.2\bin\femm.exe" // or any valid path
//
    r1_findmotorchar(motor,c/m/m_sw/g/ind/ind_a/ind_a_vs_i/ind_sw/ind_sw_vs_i/m_vs_i,[imax,imin,steps],erev)
```

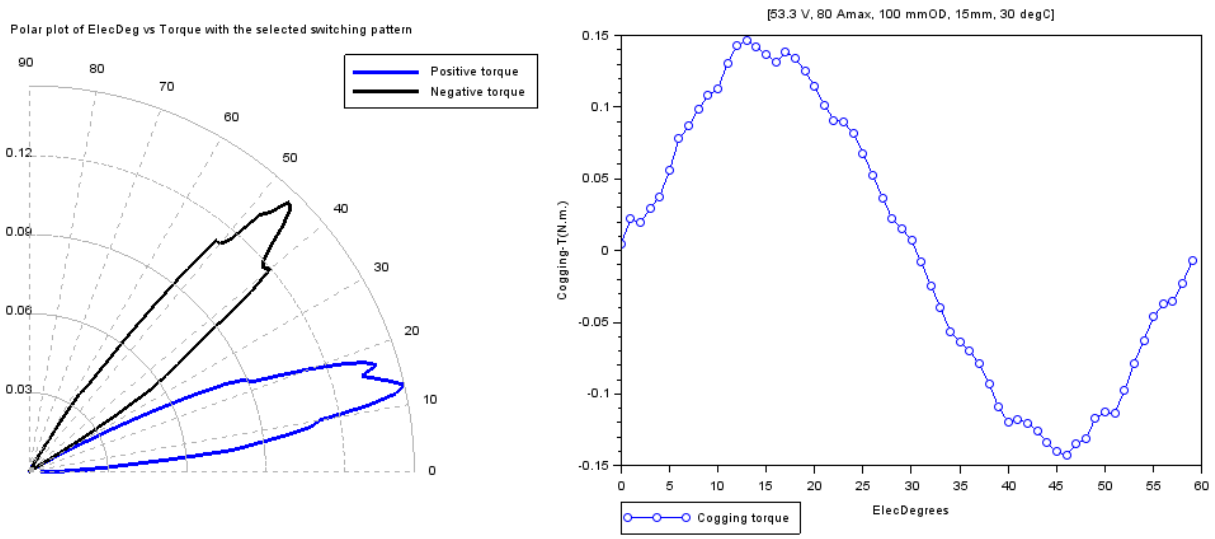
Setting the parallel mode to 1 and setting appropriate agent count will execute parallel FEMM sessions to get the characterization results faster.

This function plots the motor's torque or back emf waveforms or inductance for one full electrical cycle. Use second argument 'c' or 'm' or 'g' or 'm_sw' or 'ind' or 'ind_A' or 'ind_A_vs_i' or 'm_vs_i' for only cogging mode or motoring mode or generation mode plots, motoring mode with switching, phase inductances, phase A inductance, and inductance of phase A with current. This function consumes considerable time. m mode considers same switch combinations all around, while m_sw considers switches for the corresponidng rotor position. ind or ind_A accept an optional third argument of current at which inductance needs to be calculated. exclusion will result in calculation at IDCmax. ind_A_vs_i accepts 3 inputs that are max current, min current and steps. exclusion of any of these values will cause : max to be set at idcmax, min to be set at -max, and 11 steps. ind_A_vs_i will take considetable time to finish. ind_sw will calculate the cumulative inductance of as seen by the controller for one switching instance for a given dc bus current limit, ind_sw_vs_i will range the currents and plot. If the pc supports and when you want to run parallel agents for

faster characterization for ind, ind_A_vs_i and ind_sw_vs_i; use Motor.par=1 and Motor.FEMMpath='yourfemmexecutablepath\femm.exe'. Select Motor.agents=#of concurrent parallel sessions to be run. These are not native fields, and so you have to create. To disable parallelization, set par=0. It is disabled by default when the Motor structure is created. ind executes all three inductances in one go, ind_A_vs_i and ind_sw_vs_i execute all current variations in one go. Position increments are taken as anoher parallel run. m, g and c run for multiple points across the electrical angle. m_sw implementation is buggy. The radial pull force on the rotor is also calculated for every evaluation in m_vs_i mode.

15.15.8 Cogging torque vs rotor angle

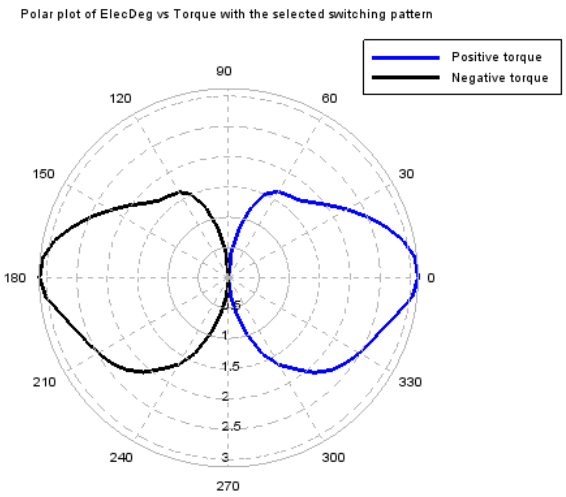
```
char=r1_findmotorchar(motor,'c')
```

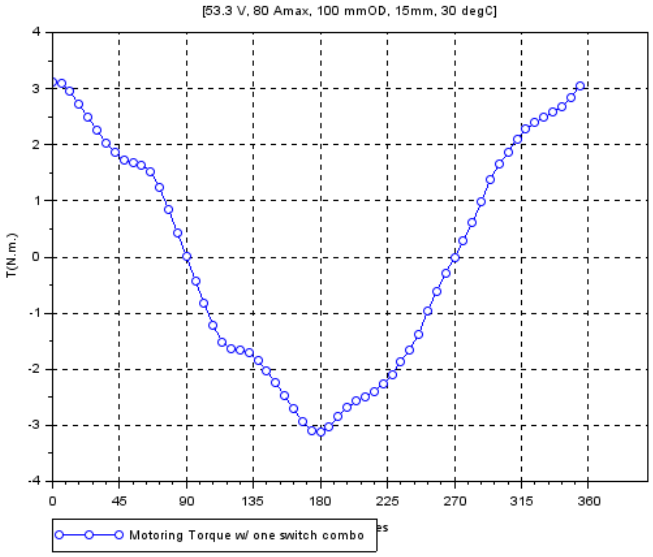


15.15.9 Torque vs rotor angle

With single switch combo turned-on as the rotor rotates for 1 electrical revolution

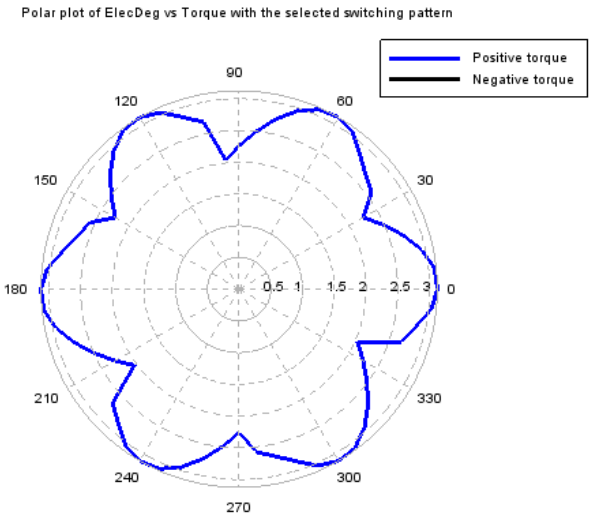
```
char=r1_findmotorchar(motor,'m')
```

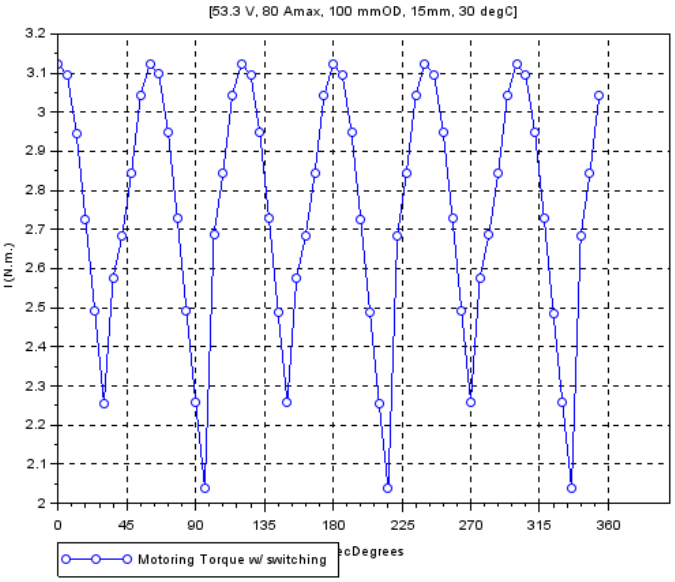




With appropriate switch combo turned-on as the rotor rotates for 1 electrical revolution

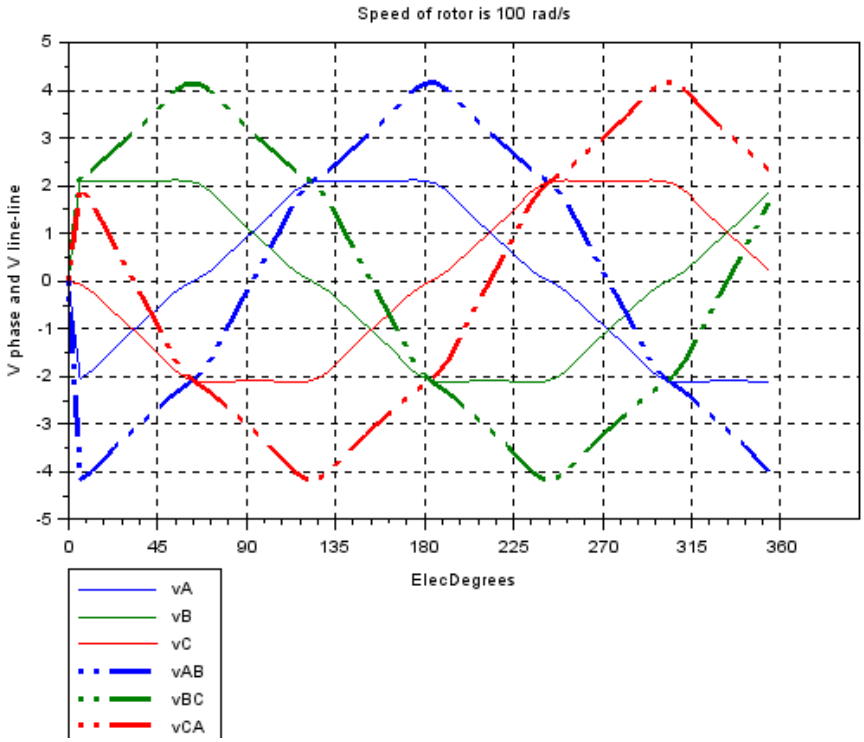
char=r1_findmotorchar(motor,'m_sw')





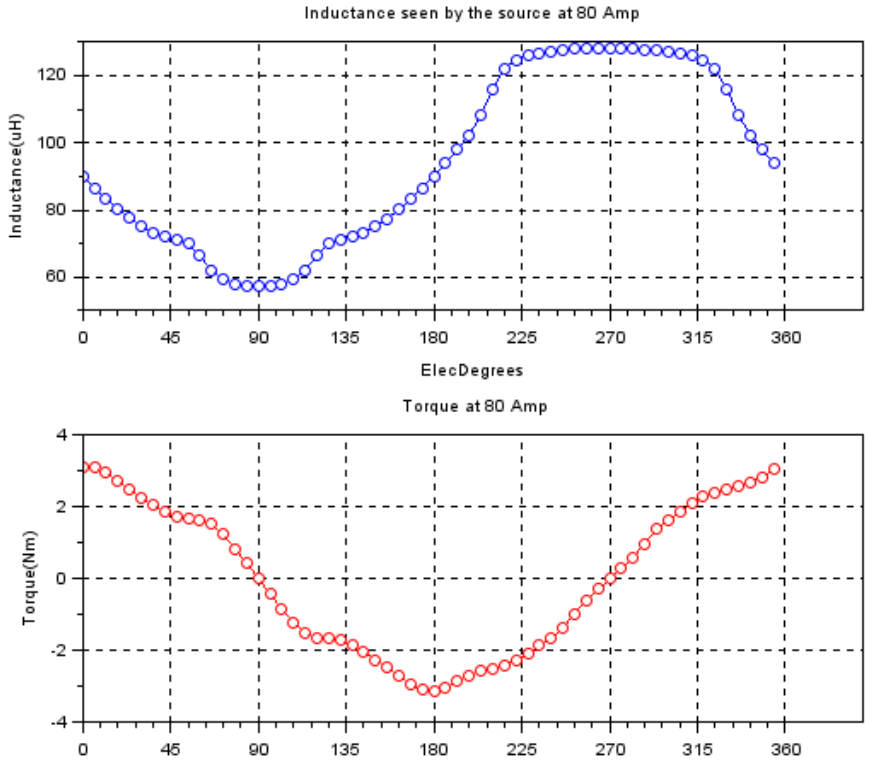
15.15.10 Induced voltages vs rotor angle

char=r1_findmotorchar(motor,'g')



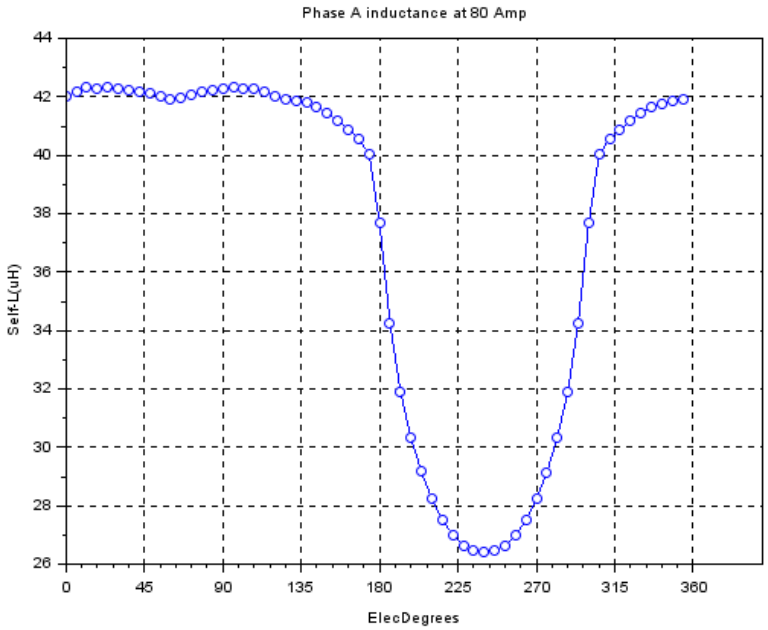
15.15.11 Line inductance & Torque vs Rotor angle

```
char=r1_findmotorchar(motor,'ind_sw')
```



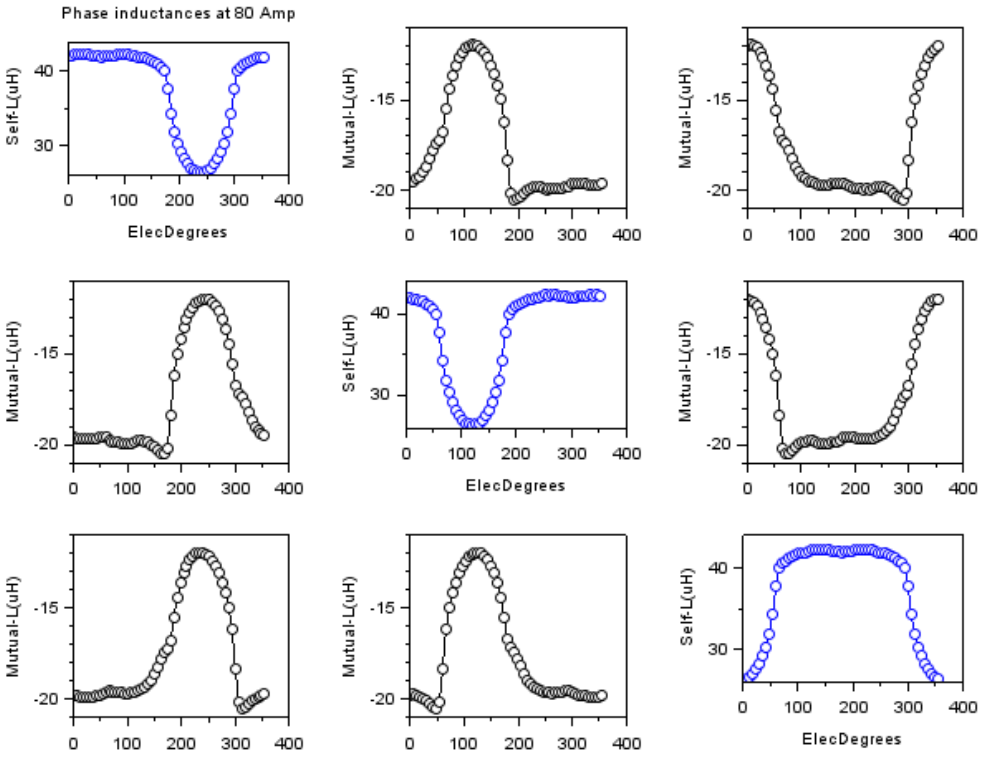
15.15.12 Phase inductance vs Rotor angle

char=r1_findmotorchar(motor,'ind_a')



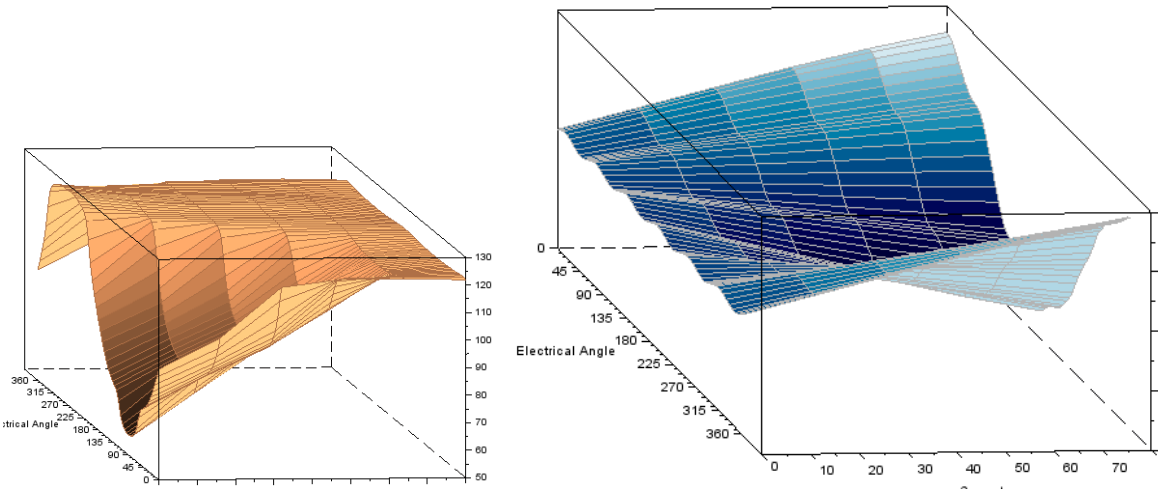
15.15.13 Self and Mutual inductances vs rotor angle

char=r1_findmotorchar(motor,'ind')



15.15.14 (Torque and Inductance) vs Current Vs rotor position

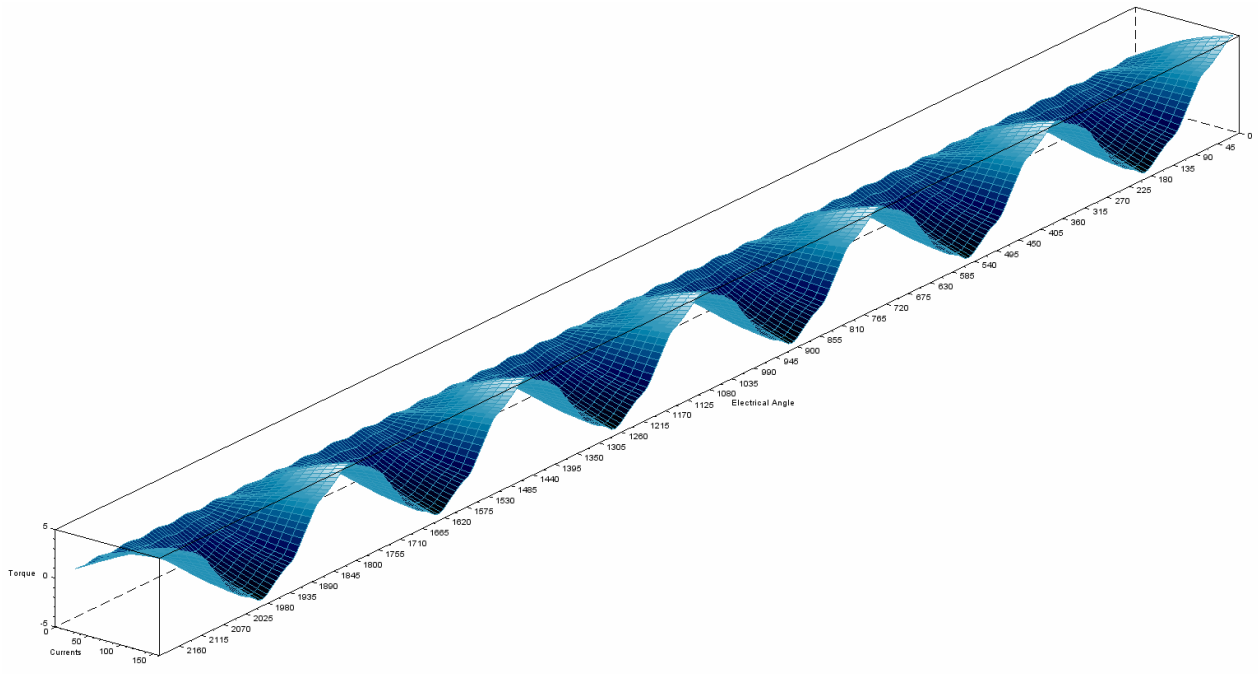
```
char=r1_findmotorchar(motor,'ind_sw_vs_i')
```

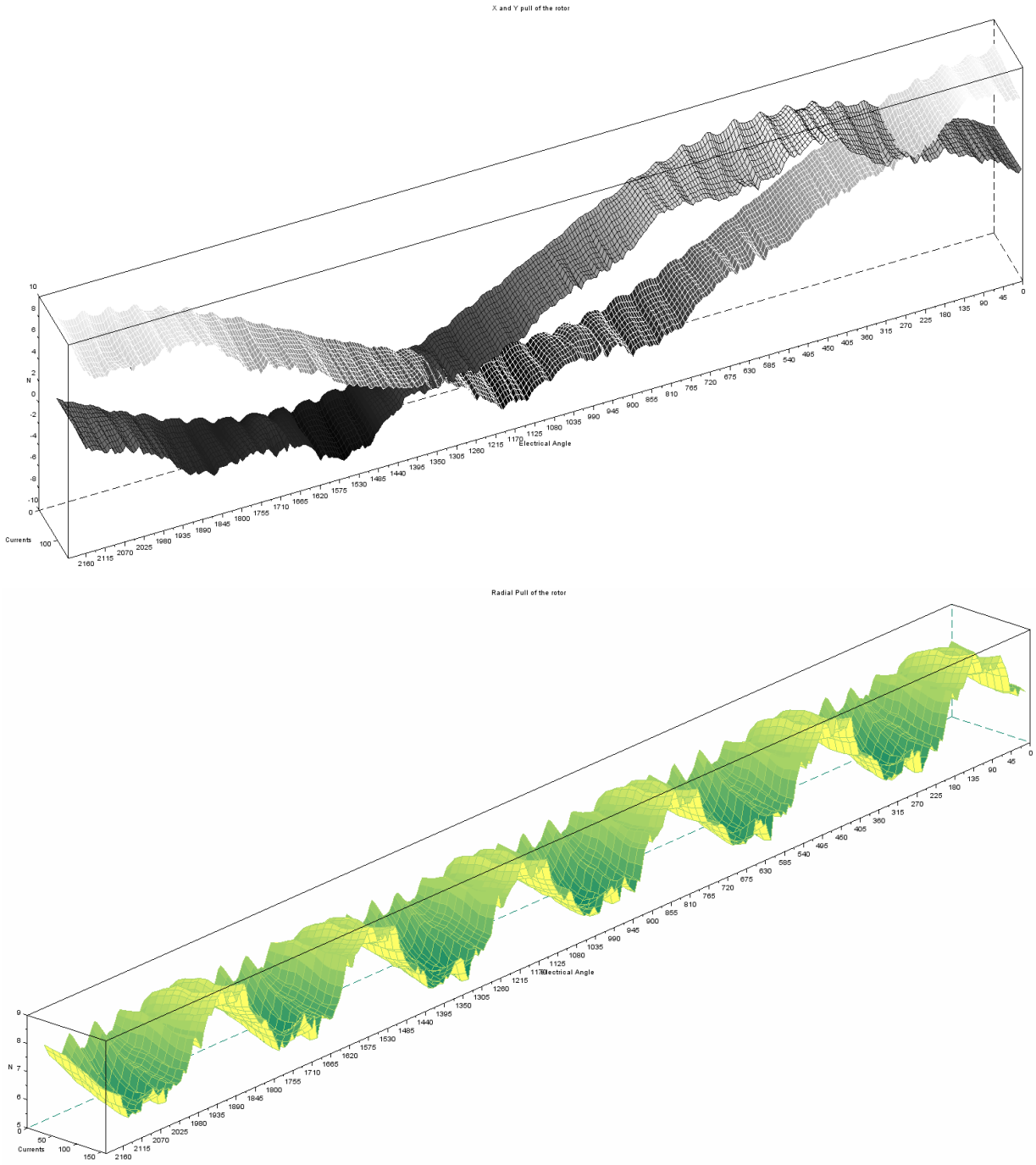


15.15.15 Radial pull

```
char=r1_findmotorchar(motor,'m_vs_i')
char=r1_findmotorchar(motor,'m_vs_i',150,0,11,6)
```

This characterization gives torque-spread, in addition to the radial pull. The above characterization with a radial offset error of 0.2mm in x direction is run for 6 electrical revolutions (one full mechanical revolution) to give full view of the radial force as below.

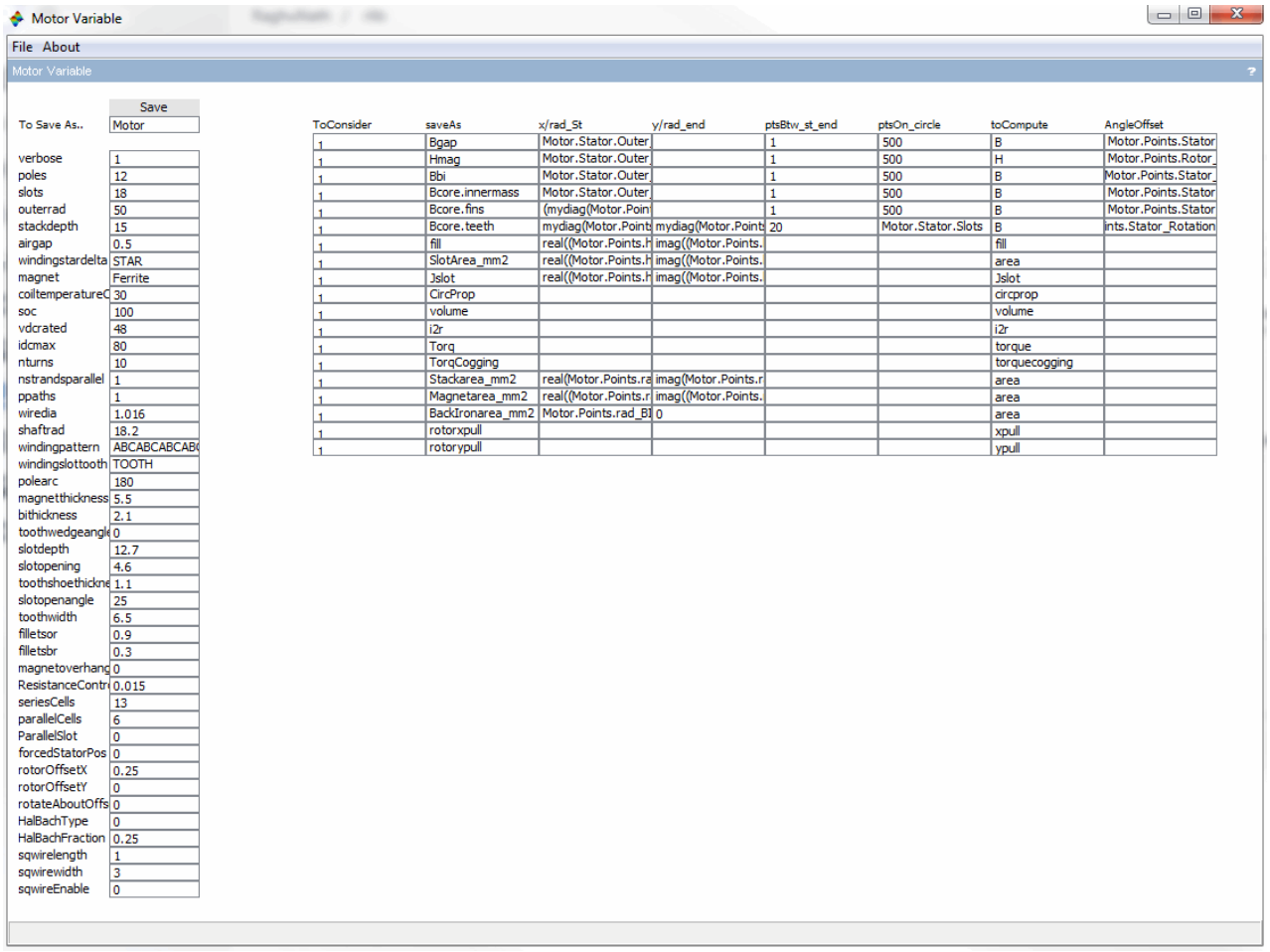




15.16 GUI

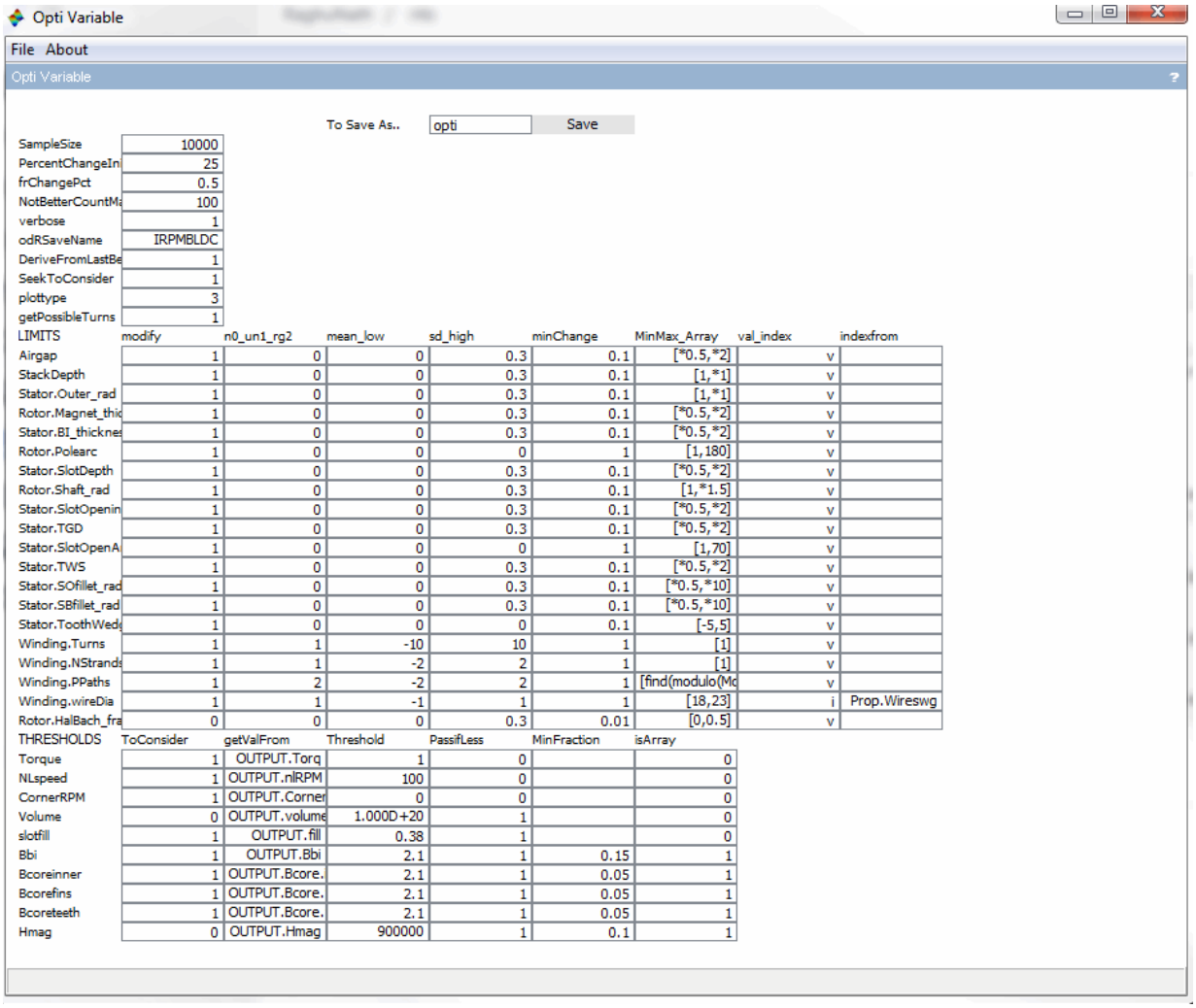
15.16.1 Motor

All the script based operations to change the motor parameters, and saving the model from command prompt can be done from the GUI too. The settings can be saved using the save button. Don't use space or numbers at the beginning of the string for the name.



15.16.2 Optimization

All the script based operations to change the optimization parameters, and saving the settings from command prompt can be done from the GUI too. The settings can be saved using the save button. Don't use space or numbers at the beginning of the string for the name.



SampleSize	10000
PercentChangeIn	25
frChangePct	0.5
NotBetterCountM	100
verbose	1
odRSaveName	IRPMBLDC
DeriveFromLastBe	1
SeekToConsider	1
plottype	3
getPossibleTurns	1

LIMITS	modify	n0_un1_rg2	mean_low	sd_high	minChange	MinMax_Array	val_index	indexfrom
Airgap	1	0	0	0.3	0.1	[*0.5,*2]	v	
StackDepth	1	0	0	0.3	0.1	[1,*1]	v	
Stator.Outer_rad	1	0	0	0.3	0.1	[1,*1]	v	
Rotor.Magnet_thic	1	0	0	0.3	0.1	[*0.5,*2]	v	
Stator.BI_thickne	1	0	0	0.3	0.1	[*0.5,*2]	v	
Rotor.Polearc	1	0	0	0	1	[1,180]	v	
Stator.SlotDepth	1	0	0	0.3	0.1	[*0.5,*2]	v	
Rotor.Shaft_rad	1	0	0	0.3	0.1	[1,*1.5]	v	
Stator.SlotOpenin	1	0	0	0.3	0.1	[*0.5,*2]	v	
Stator.TGD	1	0	0	0.3	0.1	[*0.5,*2]	v	
Stator.SlotOpenA	1	0	0	0	1	[1,70]	v	
Stator.TWS	1	0	0	0.3	0.1	[*0.5,*2]	v	
Stator.SOfillet_rad	1	0	0	0.3	0.1	[*0.5,*10]	v	
Stator.SBfillet_rad	1	0	0	0.3	0.1	[*0.5,*10]	v	
Stator.ToothWed	1	0	0	0	0.1	[-5,5]	v	
Winding.Turns	1	1	-10	10	1	[1]	v	
Winding.NStrand	1	1	-2	2	1	[1]	v	
Winding.PPaths	1	2	-2	2	1	[find(modulo(Mc	v	
Winding.wireDia	1	1	-1	1	1	[18,23]	i	Prop.Wireswg
Rotor.HalBach_fra	0	0	0	0.3	0.01	[0,0.5]	v	

THRESHOLDS	ToConsider	getValFrom	Threshold	PassifLess	MinFraction	isArray
Torque	1	OUTPUT.Torq	1	0		0
NLspeed	1	OUTPUT.nlRPM	100	0		0
CornerRPM	1	OUTPUT.Corner	0	0		0
Volume	0	OUTPUT.volume	1.000D+20	1		0
slotfill	1	OUTPUT.fill	0.38	1		0
Bbi	1	OUTPUT.Bbi	2.1	1	0.15	1
Bcoreinner	1	OUTPUT.Bcore.	2.1	1	0.05	1
Bcorefins	1	OUTPUT.Bcore.	2.1	1	0.05	1
Bcoreteeth	1	OUTPUT.Bcore.	2.1	1	0.05	1
Hmag	0	OUTPUT.Hmag	900000	1	0.1	1

16 Optimization

16.1 Initialization

All the optimization related configurations can be stored in a single variable. Default variable can be created from the following script. Every optimization execution happens in its own sandbox environment, undisturbed by the temporary variables or temporary files. Parallel Scilab sessions (or even scilab console windows) can be run as many as the PC/workstation supports.

```
opti=r_buildvar(motor)
opti=r_buildvar(0) // or any number, other than motor variable.
```

When the motor variable is provided, the optimization variable includes the provided variable. When a number is provided, the optimization variable includes the default template of the motor. The following has the motor variable, embedded inside opti variable.

```
opti.motor
```

16.2 Methodology

Motor design optimization within the constraints are multivariate optimization problems. Additionally, changing some parameters will also affect the other parameters and/or their boundaries. The optimization technique used for rlib is based on 'random walk optimization'. All the parameters set to change are changed at every possibility of creating a new model, and whenever a model satisfying the constraints better is found, it is assigned as seed model for the next generation. This is sort of an gradient descent approach, but on a multi-dimensional space.

The optimization has the primary target to maximize the torque. The thresholds for Torque and cornerrpm or nlrpm need to be used to set the lower bounds on the requirements. A Torque threshold of about 1Nm may be set to eliminate all the designs with very low peak torque but very high no load rpm. A cornerrpm or nlrpm of about 100 may be set to eliminate all the designs that may be with zero cornerrpm, or have very low constant-torque-band. The SoC of the motor bay be adjusted to run the optimization for worst voltage conditions, say at 10% SoC. The winding temperature may be adjusted to higher than room temperature, say at 80 degC, to account for increased coil temperature while the motor is in continuous operation. With the worst case conditions set, the optimization is set to result in a better selection of designs.

16.3 Settings

16.3.1 Samples

The following sets the number of samples to test before concluding the optimization. The run may be stopped in the middle if needed to.

```
opti.samplesize=10000 // usually set to large number so as not to become a limiting factor.
```

16.3.2 Change from past value

The following sets the percentage change of the parameters' value based on the conditions specified in 'Limits'.

```
opti.percentchangeinit=25 // set to a number less than 100. larger number results in drastic variations
```

When the optimization run is not able to find any better model than the current one even after a lot of trials, the percentage change is set to increase so that the spread of search can be increased. The following variable decides the spread by multiplying the percentagechange with (1+ftchangepct).

```
opti.frchangepct=0.5 // multiplies percentchangeinit with 1+frchangepct
opti.notbettercountmax=100 // the threshold iterations after which the above operation variable is used
```

16.3.3 Tweaking for improvements

The following variable fills the slot with the maximum possible number of turns when the winding turns are to be changed.

```
opti.getpossibleturns=1
```

The following variable uses the better model as the seed model, and proceeds from that model. Without this, the model is just perturbed, and is useful for dimensional tolerance analysis.

```
opti.derivefromlastbest=1
```

16.3.4 Thresholds

```
opti.thresholds
```

The optimization pass criteria are given using this table. Some important output numbers are provided with thresholds. Only if a given model passes the threshold condition, the model will be saved as odR file and populated in the corresponding plot.

Thresholds are given as numbers, with 'passifless' flag. If it is 1, then the threshold inequality changes to '<=', else the inequality stands as '>='. If the value considered is not a single value but rather an array, the 'MinFraction' gives an indication of what minimum fraction of the array values need to pass the threshold criteria. The 'isArray' field just confirms that this output is an array.

16.3.5 Limits

opti.limits

The motor parameters that may need to be changed are included in this table.

Setting the modify to 1 results in that parameter changed during optimization.

The n0_un1_rg2 is an important selection for normal distribution, uniform distribution and range. For a normal distribution, mean_low is taken as mean, and the sd_high is taken as standard deviation. For the uniform distribution, mean_low is taken as low boundary, and the sd_high is taken as the high boundary. For the range selection, minmax_array is used for selecting values from the population.

mean_low is mean for standard didsribution(if zero, variable is changed from present value, if non-zero, variable is assigned a value as the computed new value changed from mean), low for uniform distrubution(if min-max array is given only with min and is assigned -100, a random value is assigned to the variable from the low and high), NA for range.

sd_high is standard deviation for normal dsitrubution(if zero, sd is computed from min_max array bounds {min-max} /6. still depends on mean for offset from present or assign as new), high for a uniform distrubution, NA for range

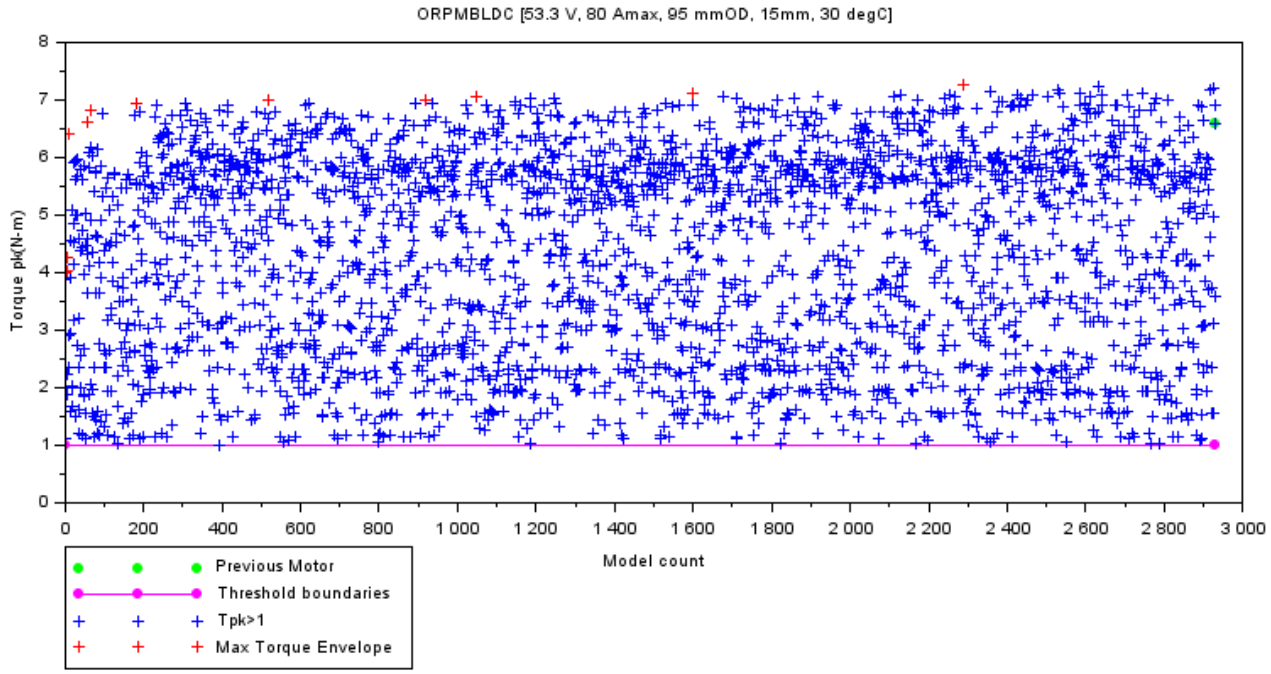
The minChange is an important parameter to include the process parameters, i.e. the minimum change in the geometry. For example, the number of turns can not be changed from 1 to 1.1. It has to be either 2 or 3.

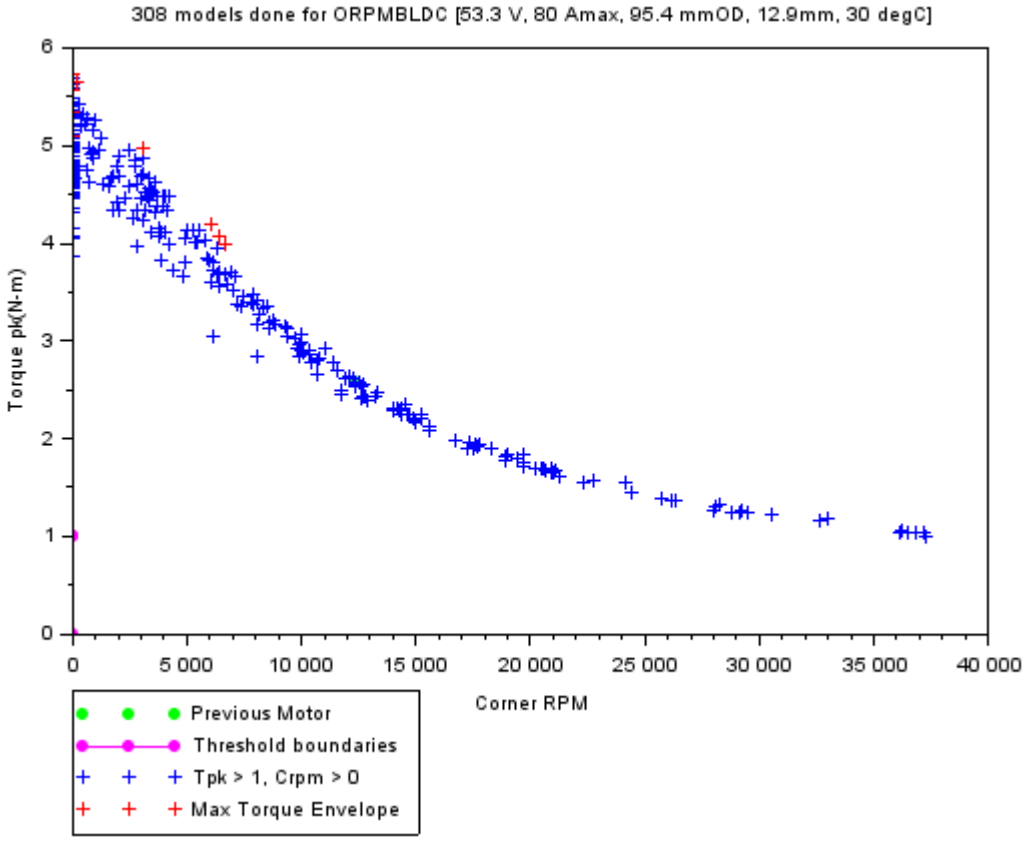
The minmax_array is used as the boundary values for the parameters after they are passed through the distribution. This value can be either a no value [] (no bounds), or single value (min only) or two values(min and max values). Each value can be an integer itself indicating an absolute value, or a value followed by a star symbol, indicating a multiplication with the present value of the parameter.

The val_index is to indicate whether the final output is a direct value or an input to the array as an index. V is to say that the value is updatd to the parameter as it is obtained, and i is to say that the obtained value is used as an index to obtain the updated parameter's value from the indexfrom field.

16.4Plots from optimization

The plots can be chosen to be either Torque-ModelCount or Torque-CornerRPM or Torque-MaxRPM. Essentially these are different representations of the same data. Since the optimization plots look similar irrespective of the geometry, I have presented the same images from the ORBLDC model's optimization run.





16.5 Data

16.5.1 Handling the Data

Motor model data is saved in .odR files (odR stands for Optimization Data Record) and the optimization settings data is saved in .osR files (osR stands for Optimization Settings Record).

16.5.2 Save the model

```
filename=r1_saverecord(motor,'mymotorname','extrastring')
filename=r1_saverecord(opti,'myoptisetting','extrastring')
```

This function requires an optimization or Motor structure and the name of the variable to be stored in .osR/.odR as the optimization/Motor variable. This function returns the filename.

Script to save the model's variable

```
r1_saverecord(motor,'testmotorname')
```

This saves the variable Motor in the file titled testMotorName_20171014111055.odR, where the number string represents the date and time of saving the variable.

16.5.3 Load the model

```
motor=r_loadrecord('.\1.odr')
opti=r_loadrecord('.\1.osr')
```

This function loads the optimization/Motor structure from the osR file. If the file has an optimization settings record, optimization structure's Motor field needs to be updated with your own Motor structure. Add optional second argument 'preserve' if you want to preserve the original fields from the file.

Script to load the individual motor variable from the saved file is below.

```
motor=r_loadrecord('testmotorname_20171014111055.odr')
```

16.5.4 odr2csv

After an optimization run, many odR files get accumulated. Reading individual files to get their contents and comparing individual files is difficult, and impractical. The following function reads all the odR files in the present working directory and stores all of their contents in a csv (comma separated variables) file. This file can later be used to sort and filter based on the requirement.

```
r_odr2csv('y:\motor\scripts\r_lib\r_common_odr.bin',50,'par')
r_odr2csv('y:\motor\scripts\r_lib\r_common_odr.bin','par')
```

```
r_odr2csv('y:\motor\scripts\r_lib\r_common_odr.bin')
```

This function can be used as a standalone scilab mode, or a parallel scilab mode. In standalone mode, as indicated by the lack of 'par' keyword in the function call, present scilab session reads each odR file in sequence and updates the csv file.

With the 'par' keyword, multiple parallel Scilab sessions are started in the background, each operating on a set of individual odR files, the number may be specified or not. If the number is not specified, 50 is taken as default. This par method is most effective if the number of odR files to read is more.

This function requires the path of compiled function file(*.bin) which has this function. Most likely it will be r_common_odr.bin. Each background session is expected to occupy 200MB to 250MB. Carefully choose the parallel sessions to leave enough space for other programs on the PC. If you have 12GB RAM installed, 7GB still un-utilized as seen from the Windows task manager, then use 5GB (about 70 percent) for scilab, i.e. you can run 20 scilab sessions, max. If you have N odR files, then call odr2csv (lib , N/20 , par). If the N/20 becomes more than 500, consider that each scilab takes more memory. It becomes iterative. Instead, use the feature without par, plain old way.

17 Some Geometrical Features

17.1 Individual wire strands

The rlib can create very fine details in the geometry including individual wire placement in the motor slots.

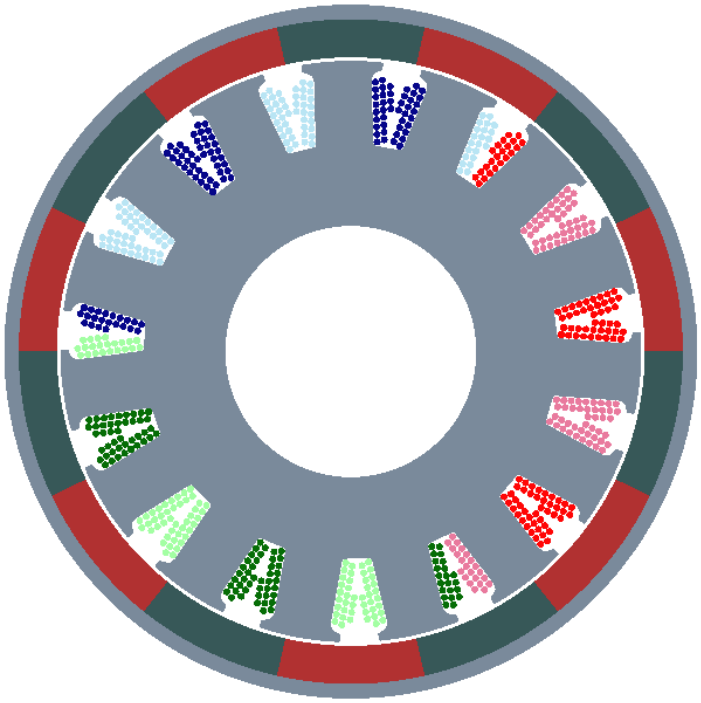
17.1.1 Positioning controls

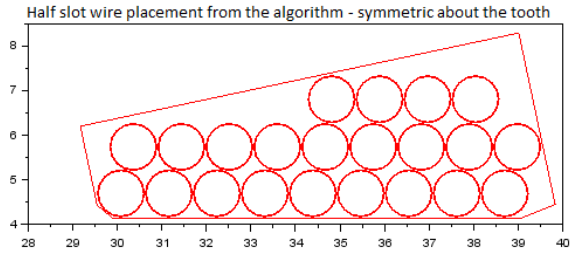
The wire placement in the individual slots in the models is controlled by the following parameters.

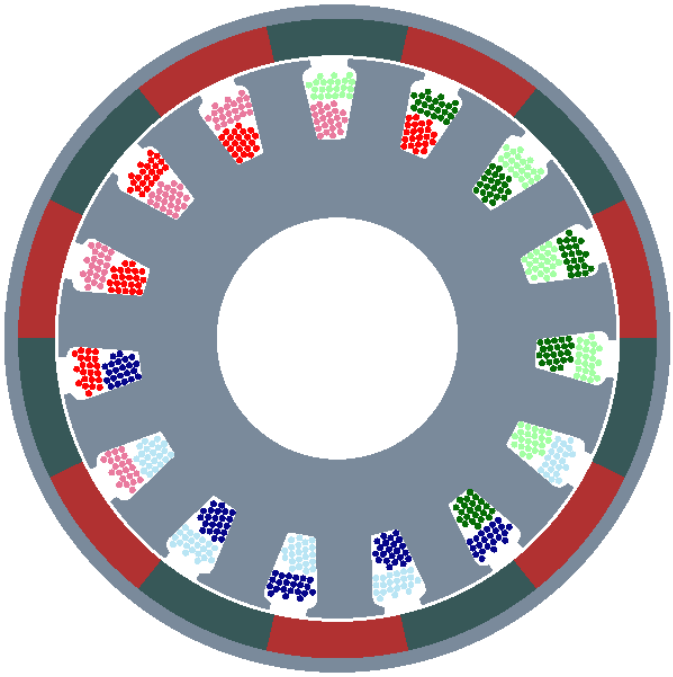
```
motor.winding.turns=6 // sets the number of turns for each occurrence of the phase letter in the winding pattern
motor.winding.nstrands=1 // sets the number of parallel strands for each winding turn
motor.winding.insulthickness=0.05 // sets the insulation thickness for the border line around the copper conductor
motor.winding.wiregap=0.1 // sets the wire spacing around the conductor from the insulation layer
motor.winding.slotlinerthickness=0.1 // sets the slot liner thickness around the slot, this includes the slot hald seperator
motor.winding.throw=1 // when more than 1, refers to multi throw winding. then the slot is split as horizontal.
motor.winding.pattern // the winding pattern. for example aaabbbbbccccccaa
motor.winding.slotsplit='vertical'// or horizontal
//either the following two lines
motor.winding.drawwiresemag=1 // this flag decides whether to draw individual wires in the electromagnetic solution mode. these are drawn by default in the thermal solution mode.
motor.winding.animate=1 // decides whether to animate the wire positioning or just place the wires in the geometry.
// or the following one line
motor.buildmodelin='slot' // sets the above two lines internally, plots the slot view only in scilab plot
// for circular wires
motor.winding.wiredia=1.016
//for rectangular wires
motor.winding.wire.rectangular=1// setting this flag will force the wires to be rectangular.
motor.winding.wire.len=1
```

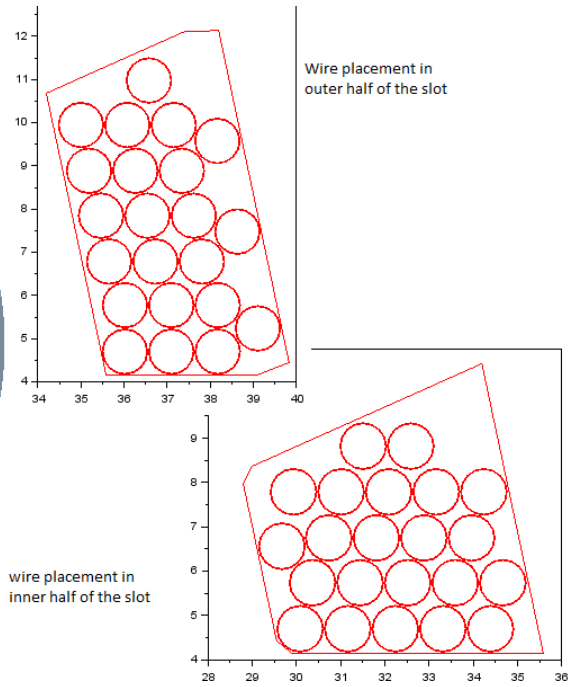
```
motor.winding.wire.thick=2
```

17.1.2 Slot Split - horizontal vs vertical









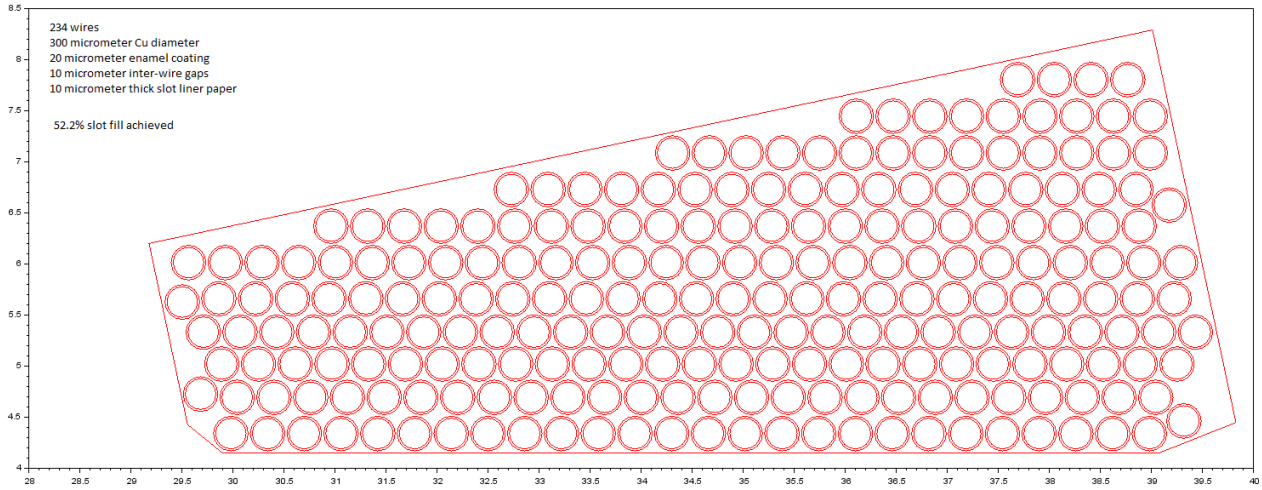
17.1.3 Wire placement algorithm

Below images show how the wire fill looks like for the given slot configuration with different angles of the bottom edge.

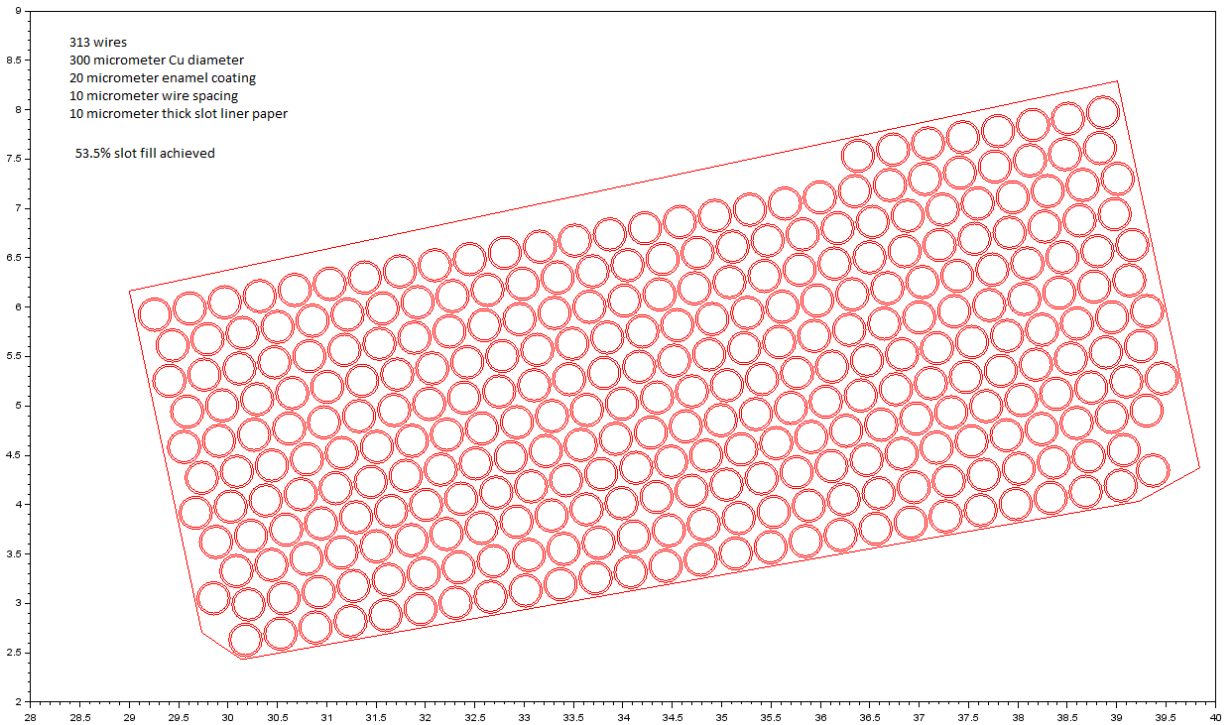
To present the working methodology of the algorithm developed by me, the geometric details are deliberately reduced to physically non-achievable levels. A realistic winding slot fill is about 40% through hand winding methods.

17.1.3.1 Wires with circular cross section

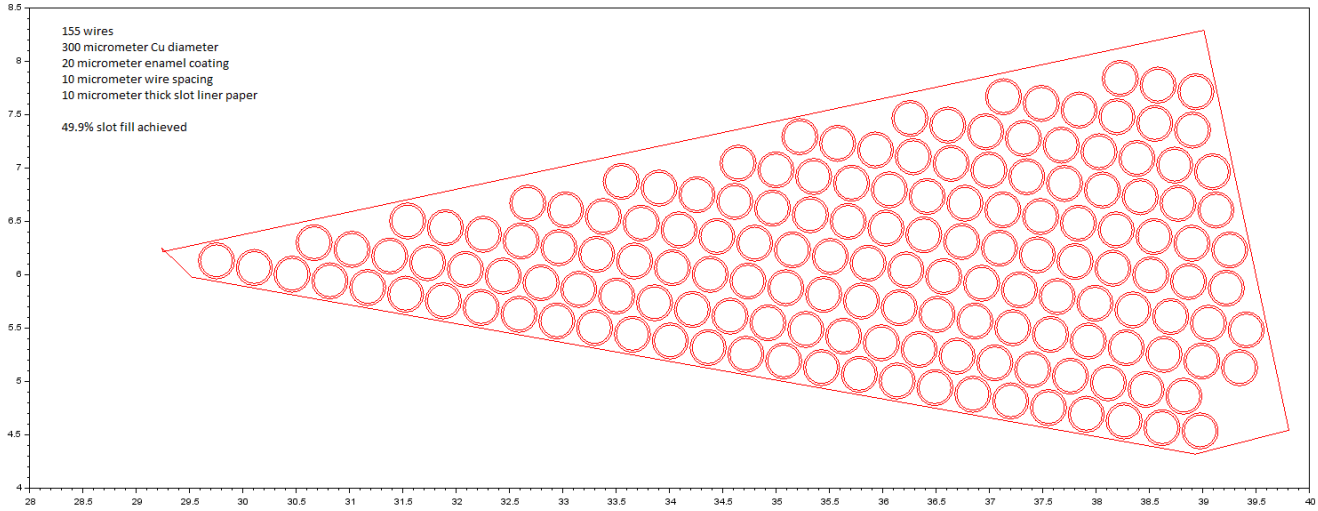
For one slot geometry



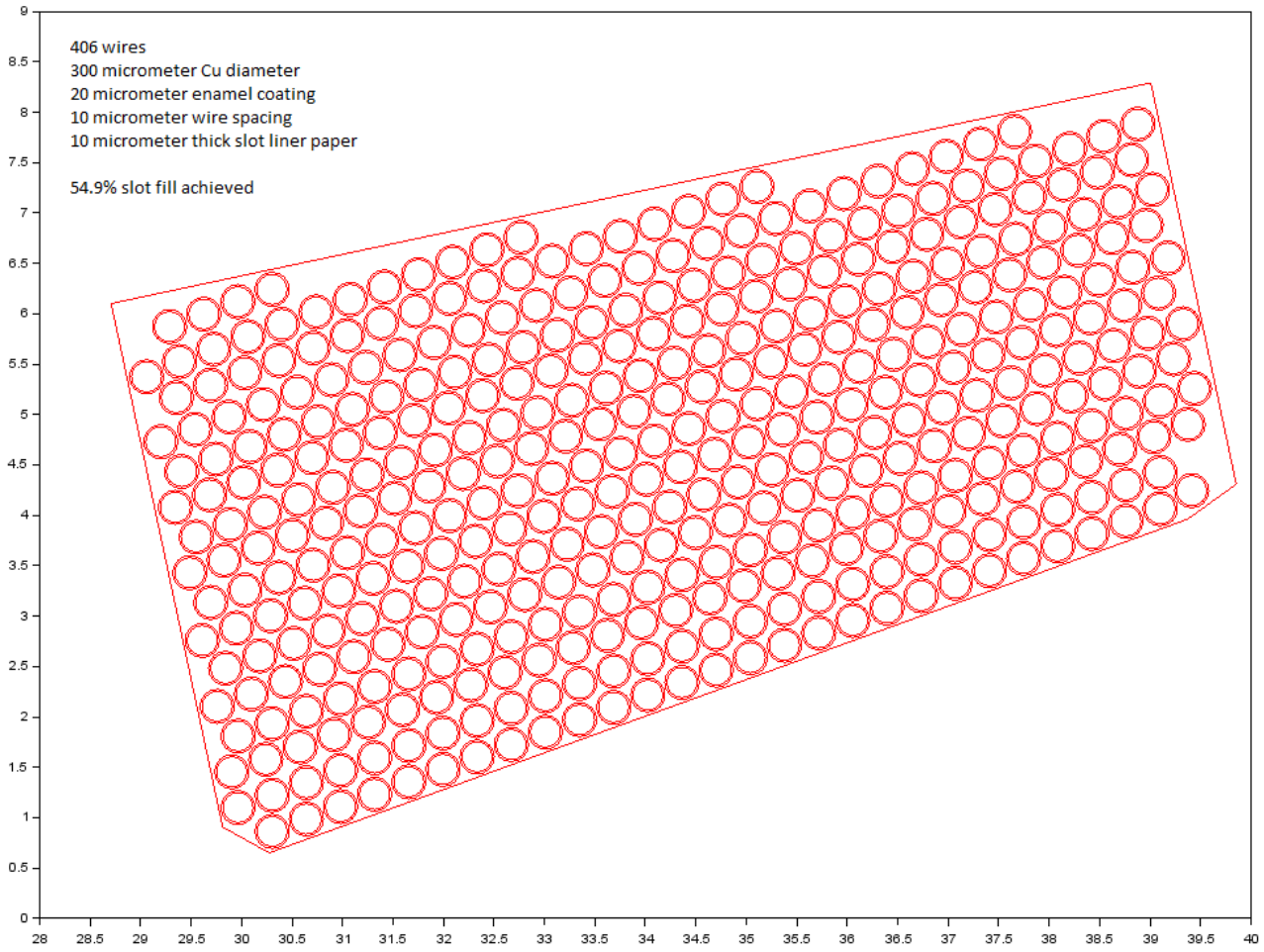
Different slot geometry



Different slot geometry

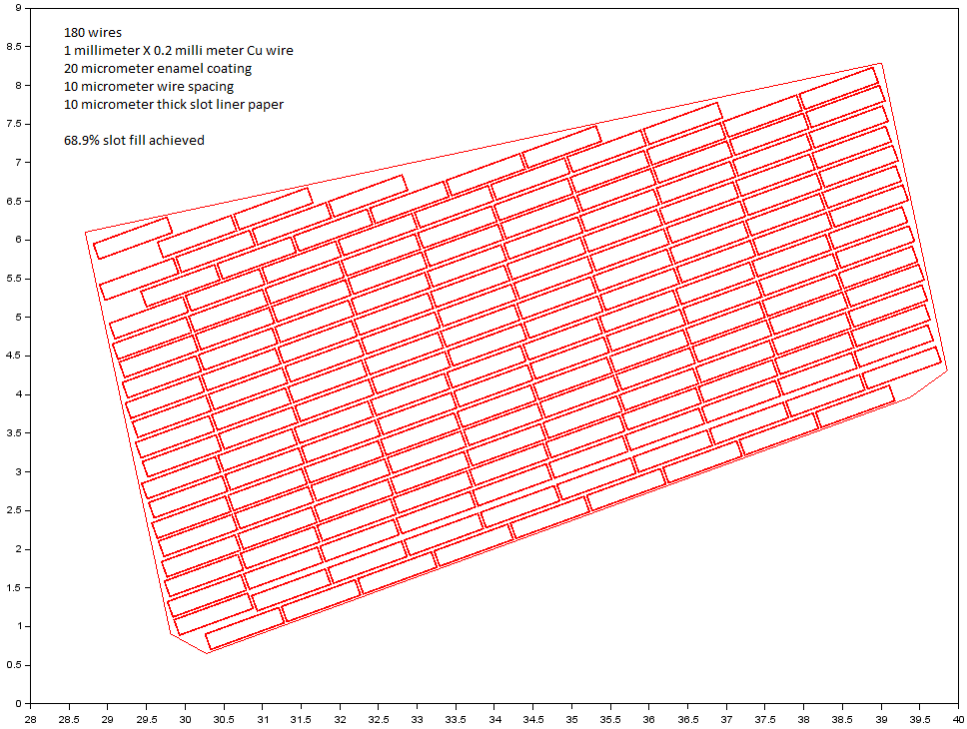


A different slot geometry

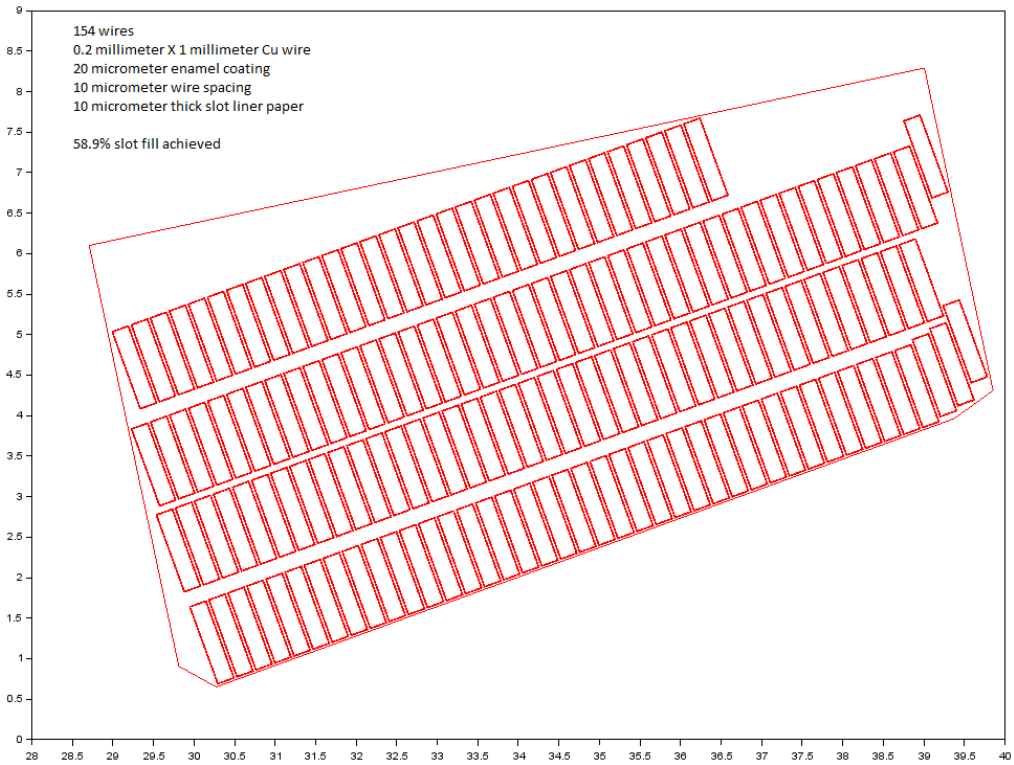


17.1.3.2 Wires with rectangular cross section

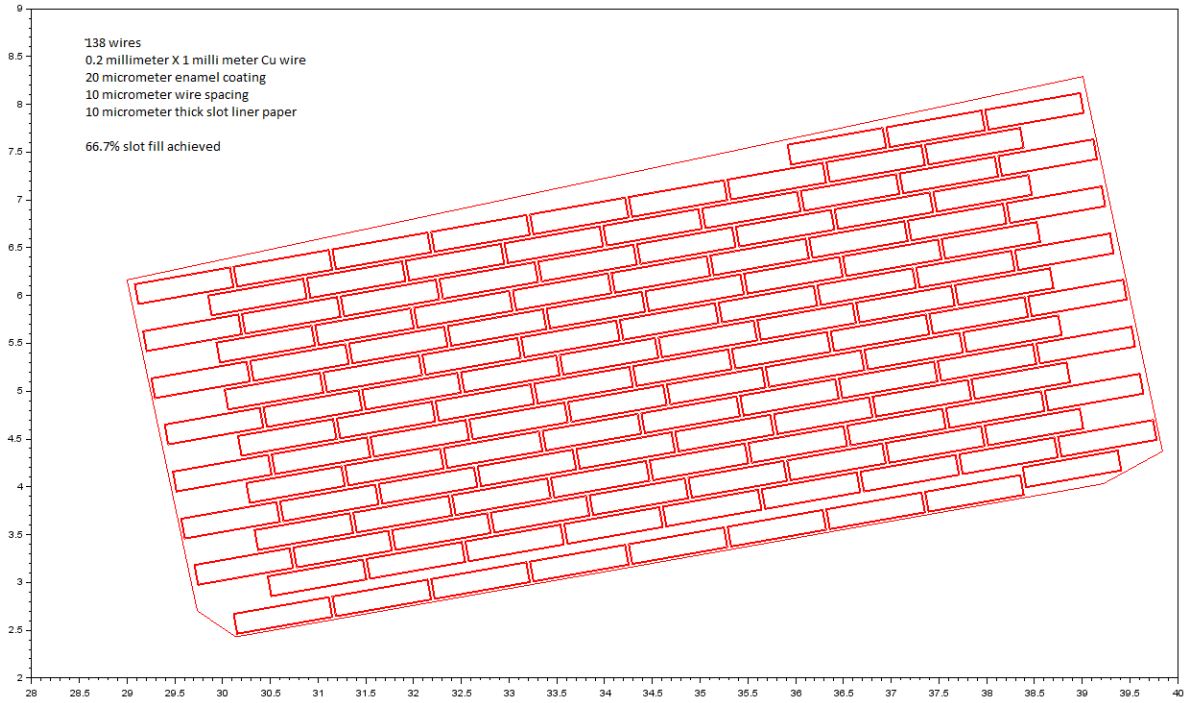
For one slot geometry



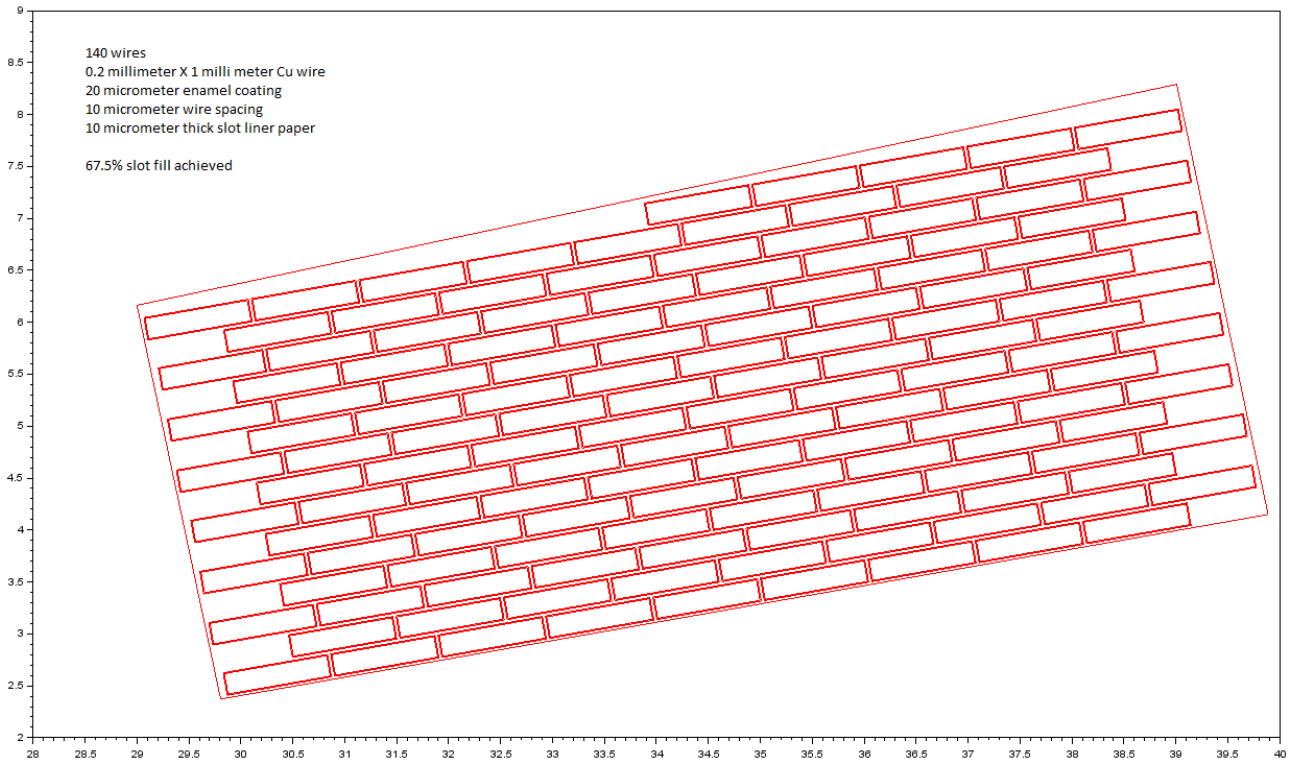
For the same slot geometry as above, but the wire rotated by 90 degrees



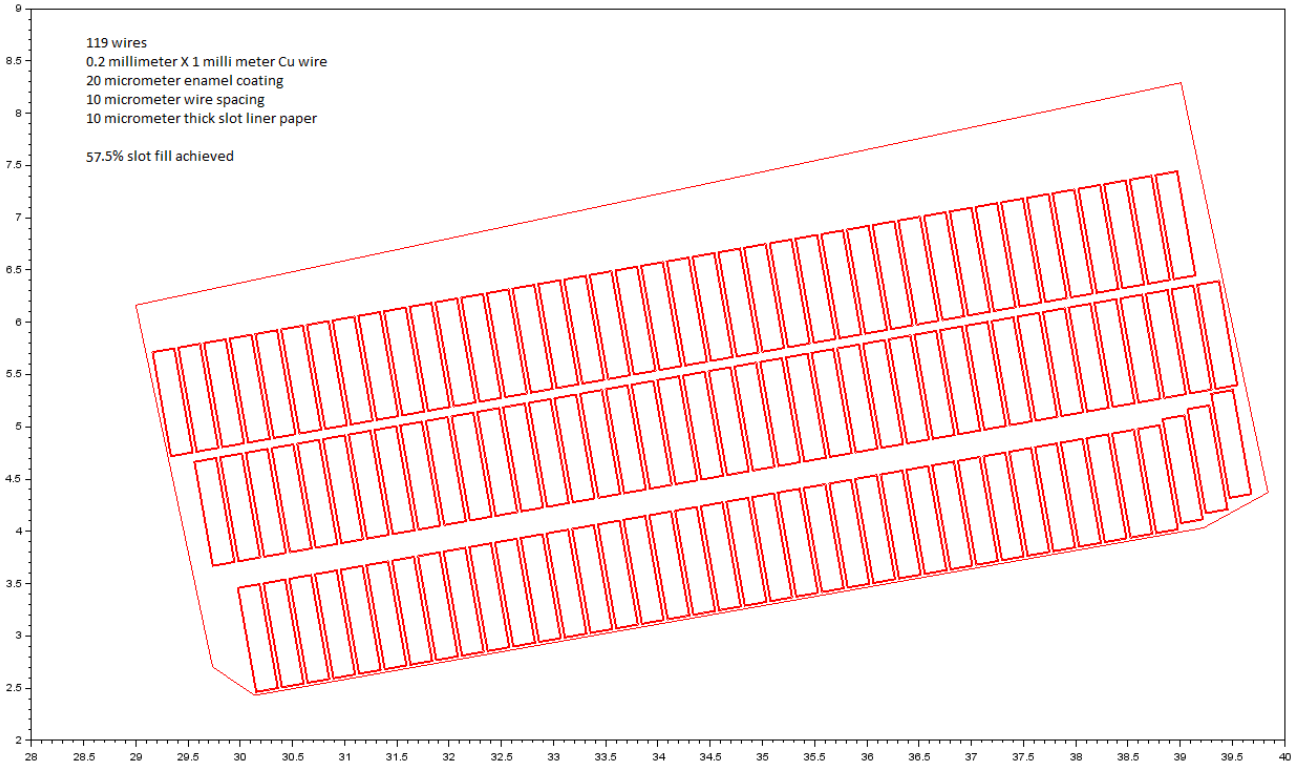
A different slot geometry



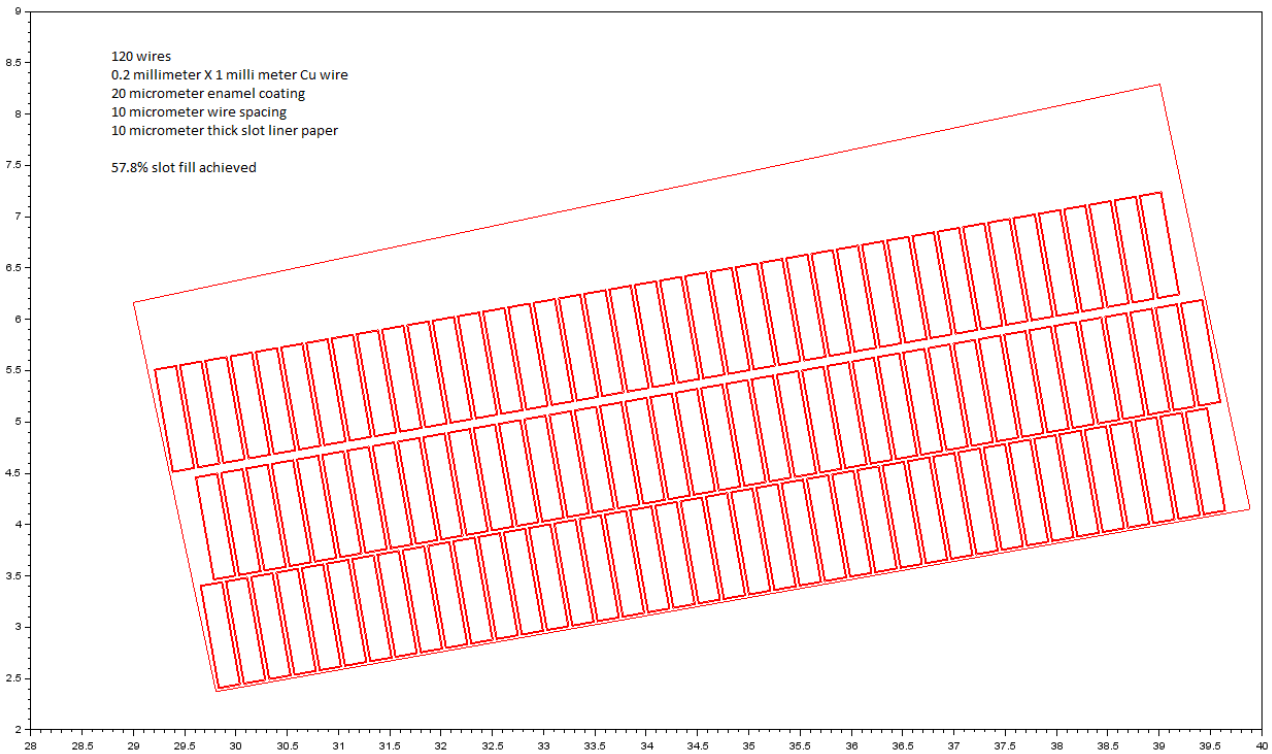
Same slot geometry a above, when the fillet radii are removed



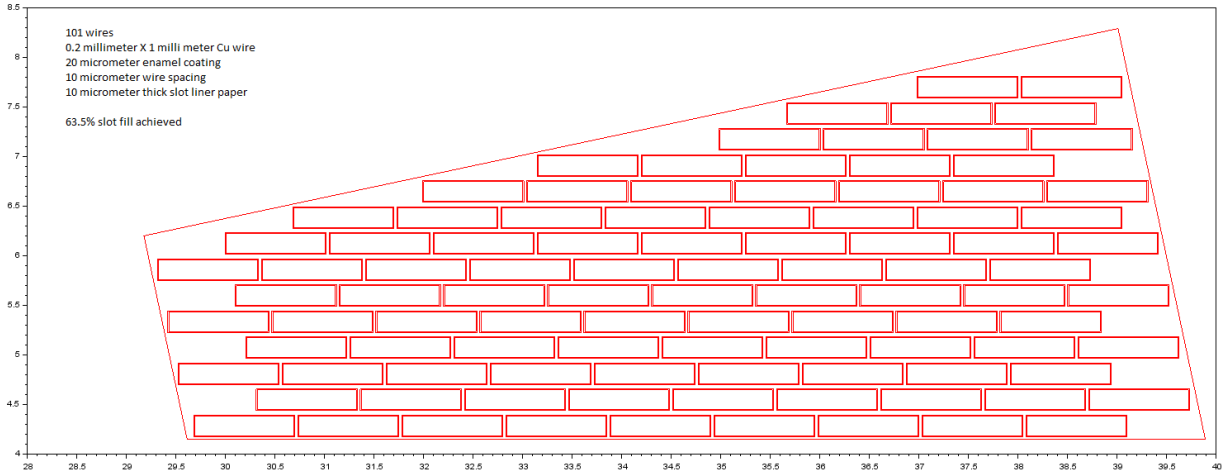
Same slot geometry as above, but the wire rotated and the fillet radii are included



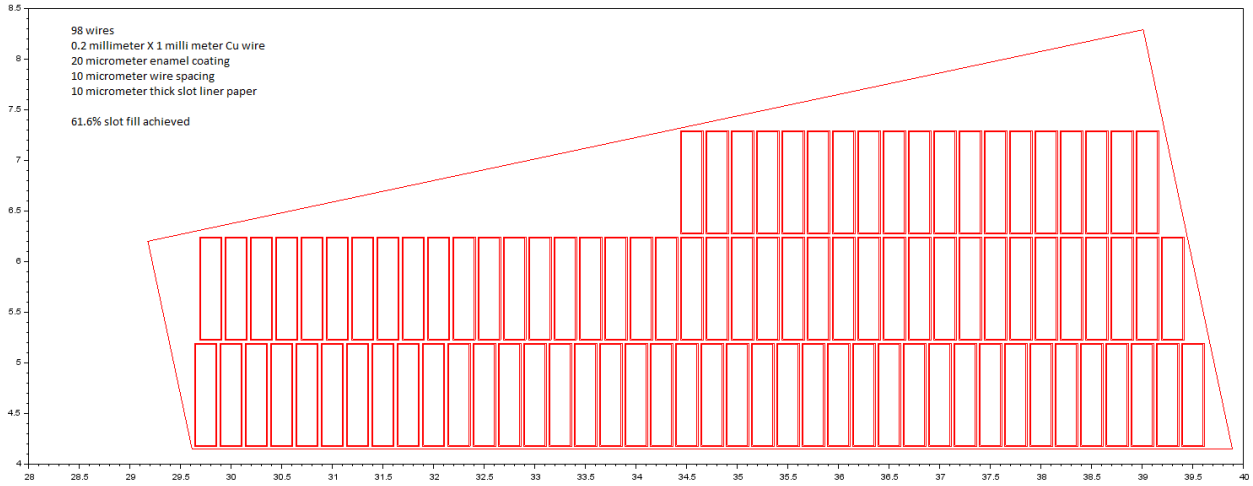
Same slot geometry as above, with the wire rotated and the fillet radii removed



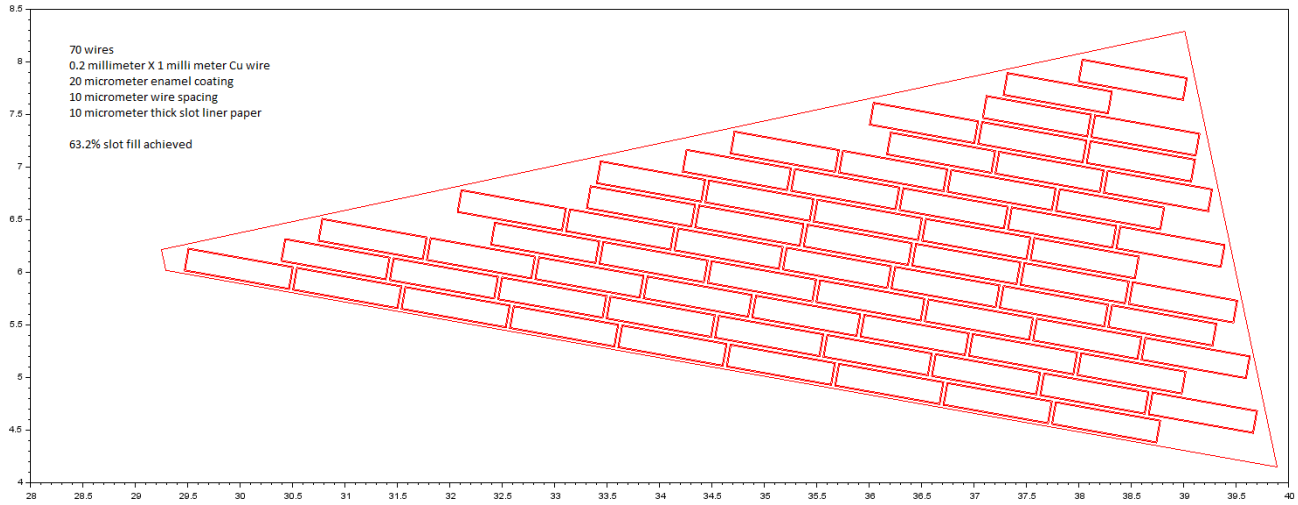
Another geometry with no fillet radii



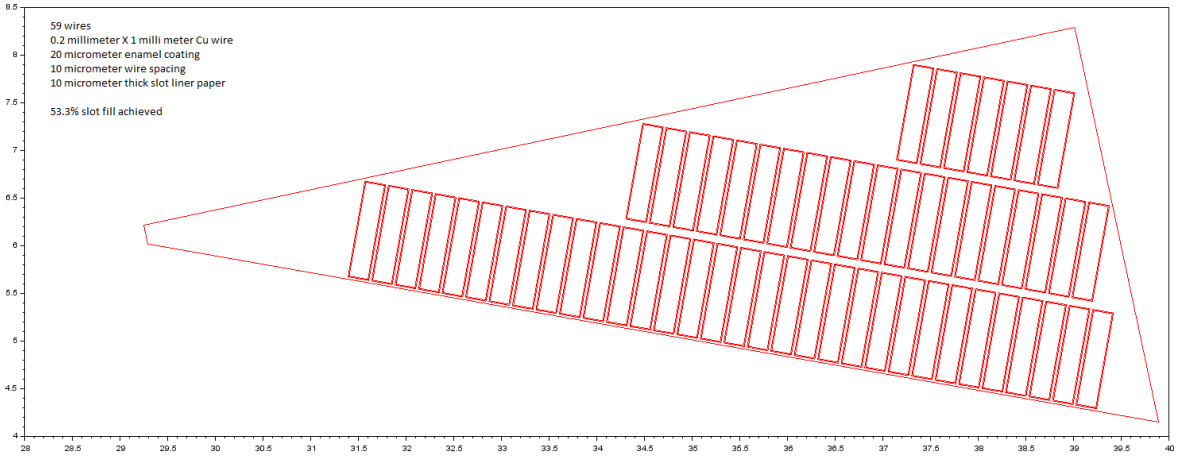
Same geometry as above with the wire rotated



Another geometry with no fillet radii



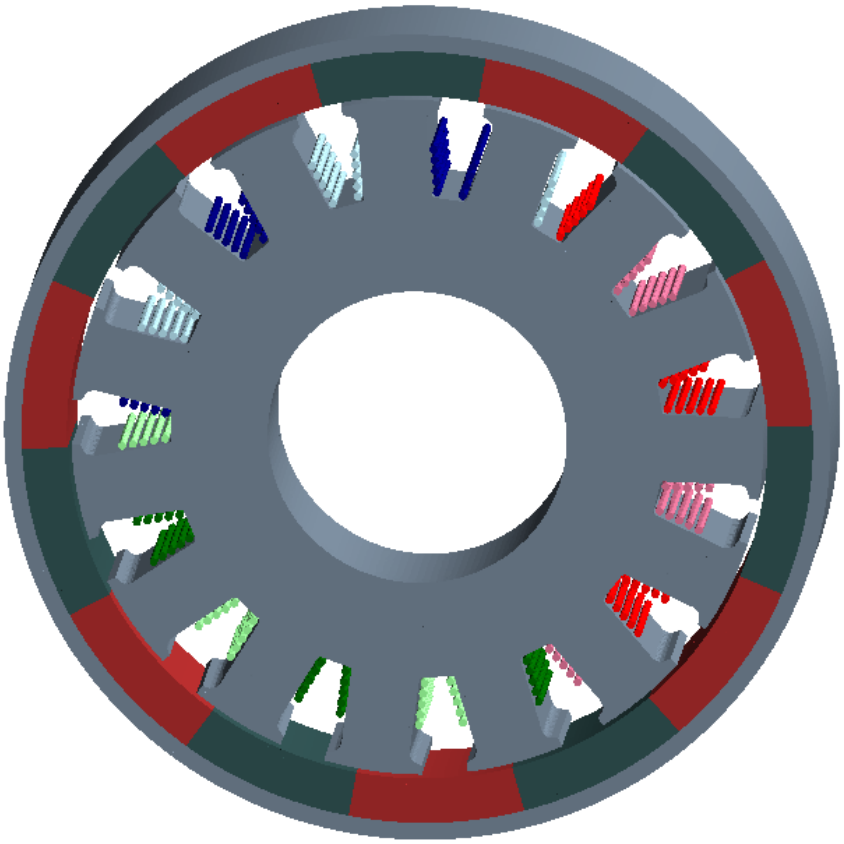
Same geometry as above, with the wire rotated



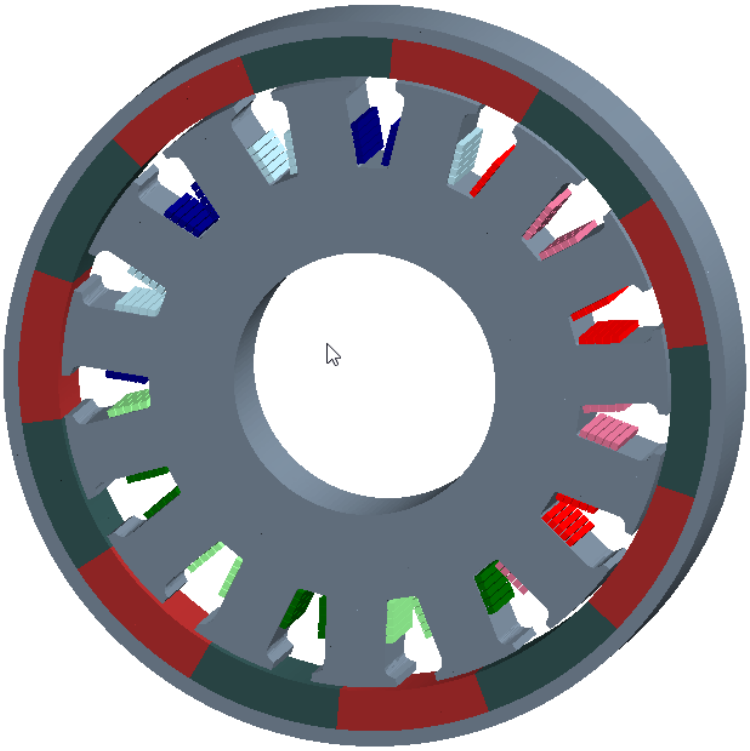
17.1.4 Some 3D examples

The rlib can create 3d models in Gmsh so far that can be meshed and imported to Elmer too. Below are some example images to demonstrate its 3D model building capabilities.

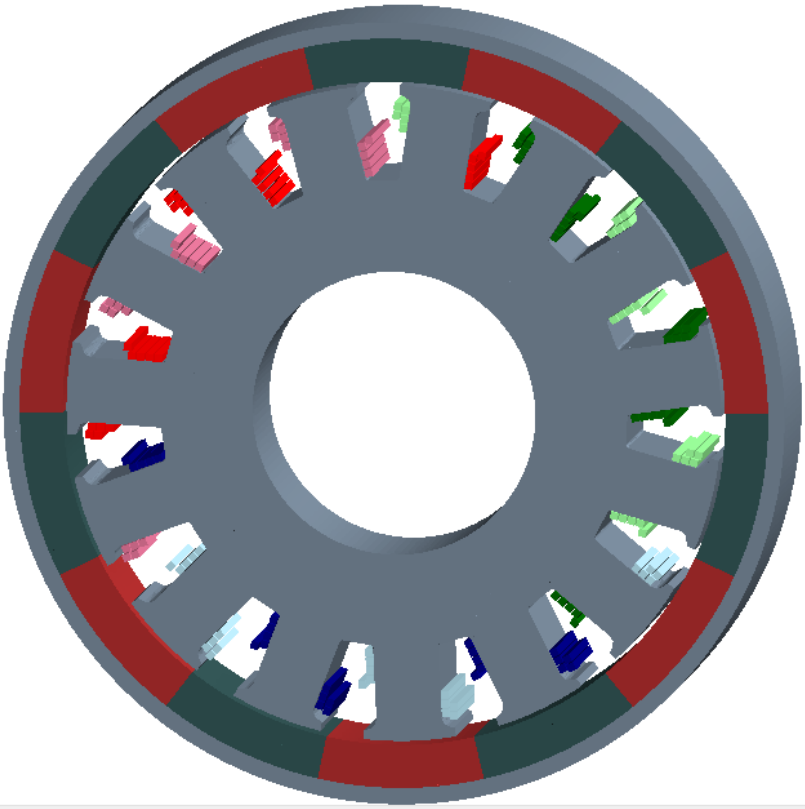
Circular wires in single throw winding configuration



Rectangular wires in single throw winding configuration



Rectangular wires in multi throw winding configuration



17.1.5 Software operational time comparison

When the individual wires are created in the geometry, the number of nodes increase dramatically. Different software are written to handle these nodes differently.

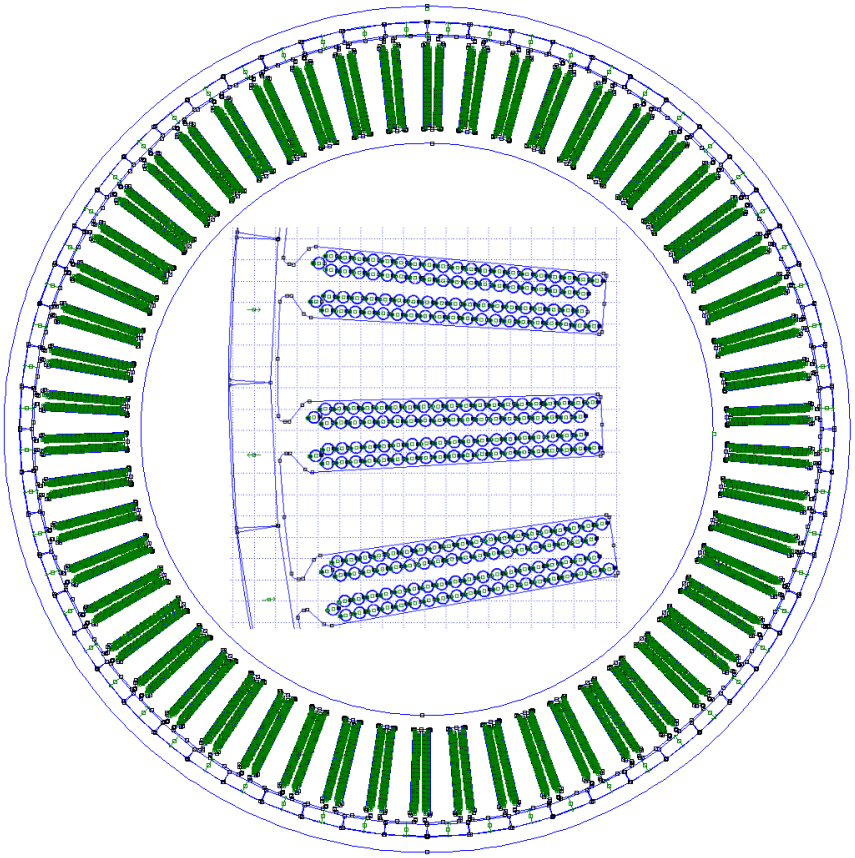
The following exercise of creating the motor geometry and mesh generation has been done with one of the motor designs with three software - FEMM, FLUX and GMSH.

A sample motor design from a Chinese manufacturer is taken for plotting these below geometries in different software. Individual wire placement is shown in the inside of each image.

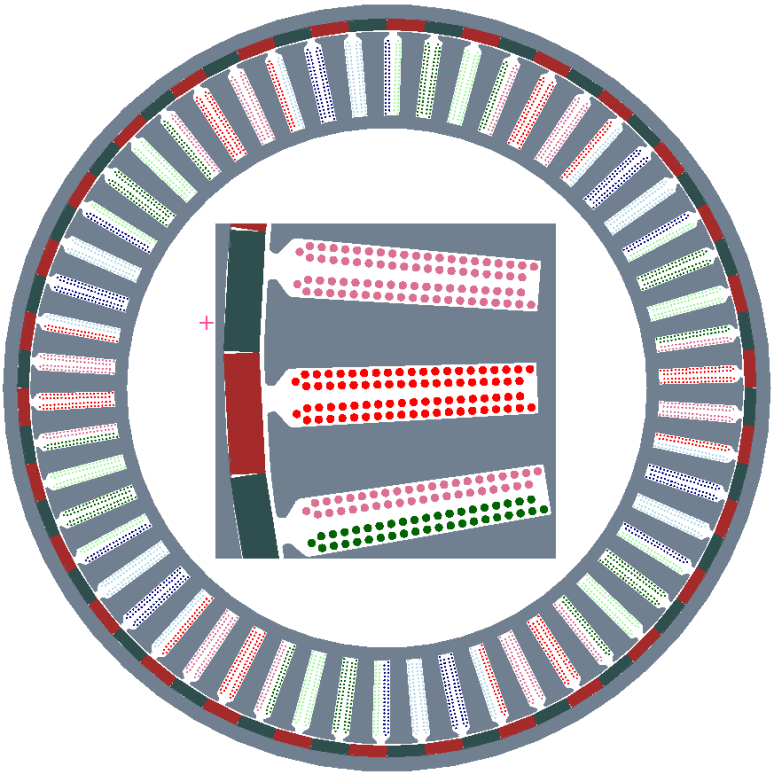
The results and comparisons are as below

TOOL	Time to Create Geometry	Time to Re-open Geometry	Time to Mesh domain	Nodes in the mesh
FEMM	6.5 minutes	3 seconds	1.5 minutes	2.6 million
FLUX	130.0 minutes	21 seconds	73.0 minutes	4.8 million
GMSH	38.0 minutes	17 seconds	2.5 minutes	3.3 million

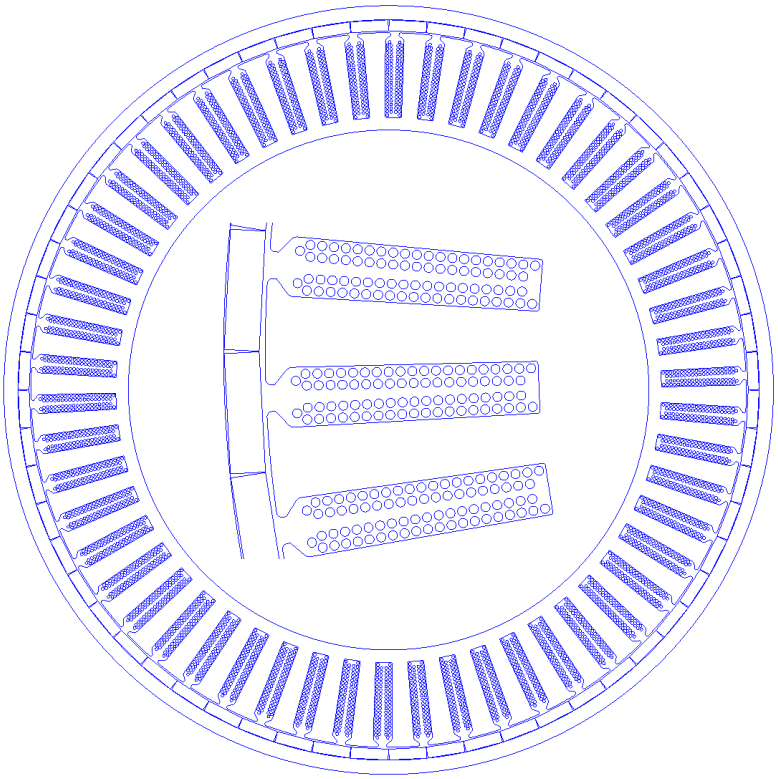
FEMM creates the following geometry



FLUX creates the following geometry



GMSH creates the following geometry



17.2 Magnet Overhang

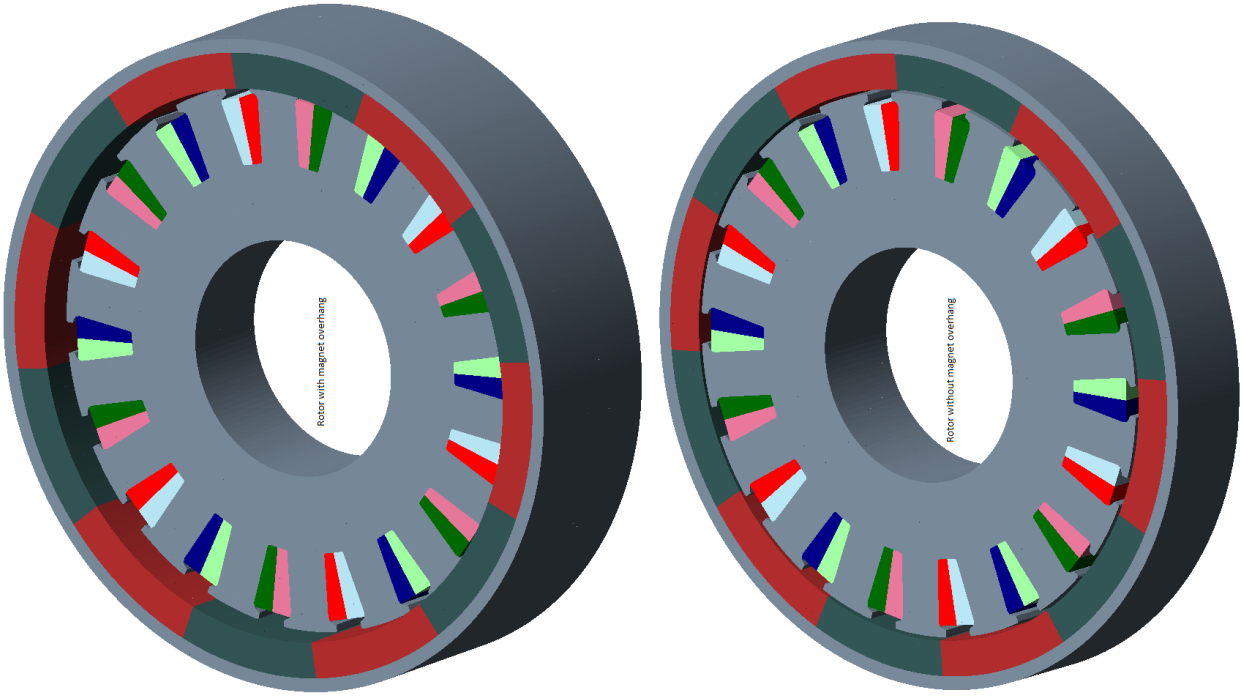
Magnet overhang is the excess magnet length beyond the stator stack.

The magnet overhang can be geometrically constructed for the surface mounted motor designs. It is believed to increase flux linkage by the coils and hence increases the torque production. Magnet overhang is also used by rlib to account for the rotor backiron weight, magnet weight and the total motor weight.

17.2.1 Code implementation

```
motor.rotor.magnetoverhang=5 // sets the magnet overhang on each side to 5mm. this also increases the backiron width to match that of the magnets.
```

17.2.2 Images of the Motors



17.3 Tooth stem angle

Tooth stem angle defines the taper of the stator teeth.

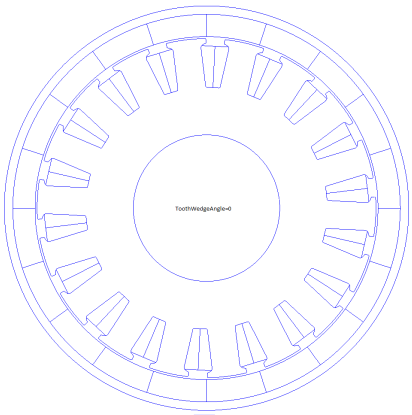
The rlib allows the stator tooth stem to be constructed with several angles as long as they are possible on the geometry. The parallel tooth and the parallel slot geometries are the subset of this implementation.

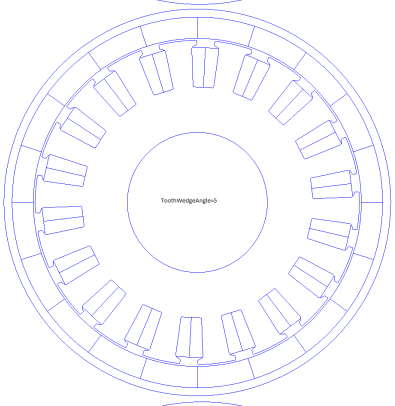
17.3.1 Code implementation

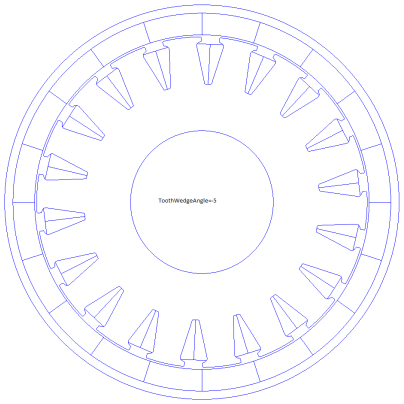
```
motor.stator.toothwedgeangle=-5// sets the angle
motor.stator.forcedparallelslot=1 // sets the angle so that the slot is parallel.
```

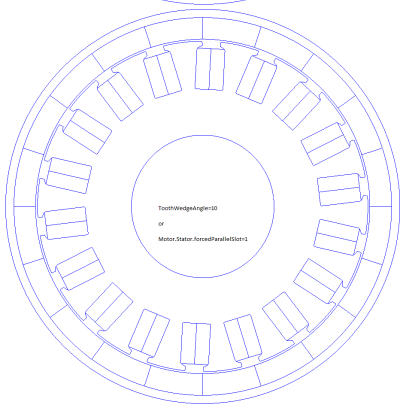
17.3.2 Motor models

Outer rotor motor model

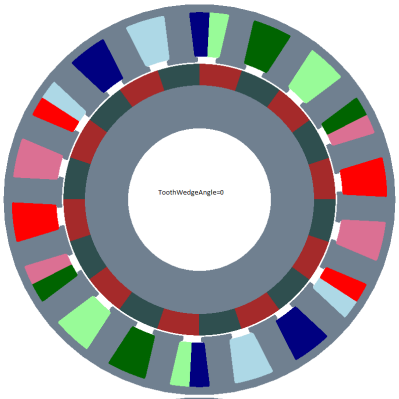


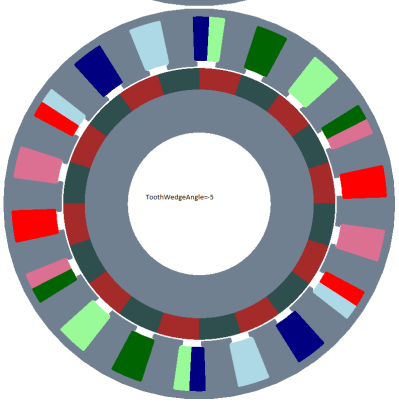


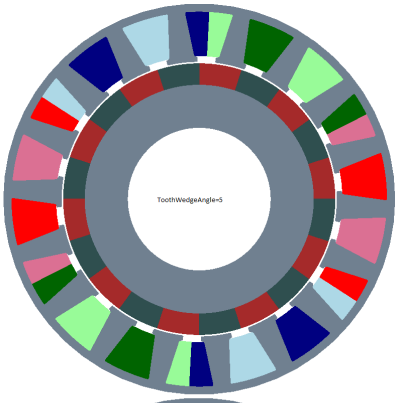


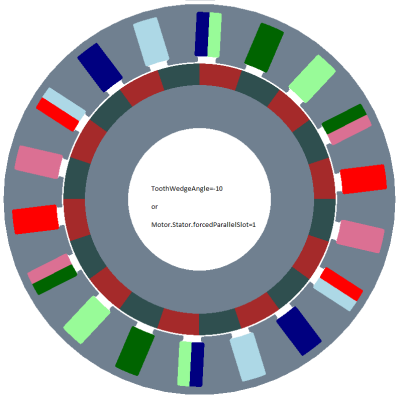


Inner rotor motor model









17.4 Eccentricity

The eccentricity of the stator-rotor assembly can be generated by rlib.

The Concentricity of the stator-rotor set may be disturbed due to various factors including those from assembly process, and the stacked up tolerance. The eccentricity may be simulated in the model. The static and dynamic eccentricity can be set with a simple setting.

Static eccentricity is the case where the rotor is offset from the stator's center, and the rotor is rotating about its own origin, and the airgap variation is static, is not rotating.

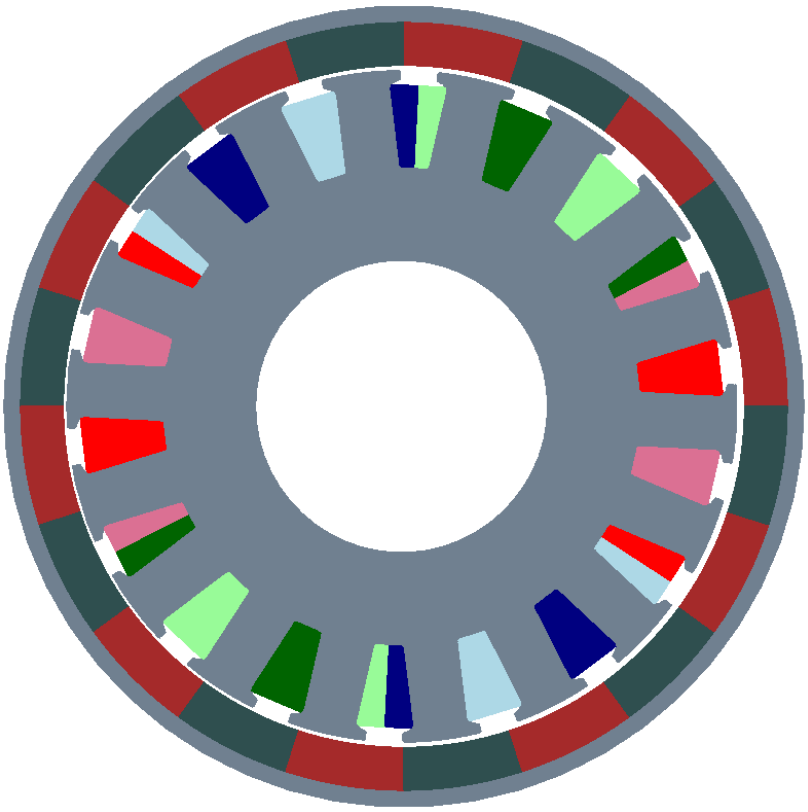
Dynamic eccentricity is the case where the rotor is offset from the stator's center, and the rotor is rotating about the center of the stator. In this case the airgap variation is not static. It rotates as the rotor rotates.

17.4.1 Code implementations

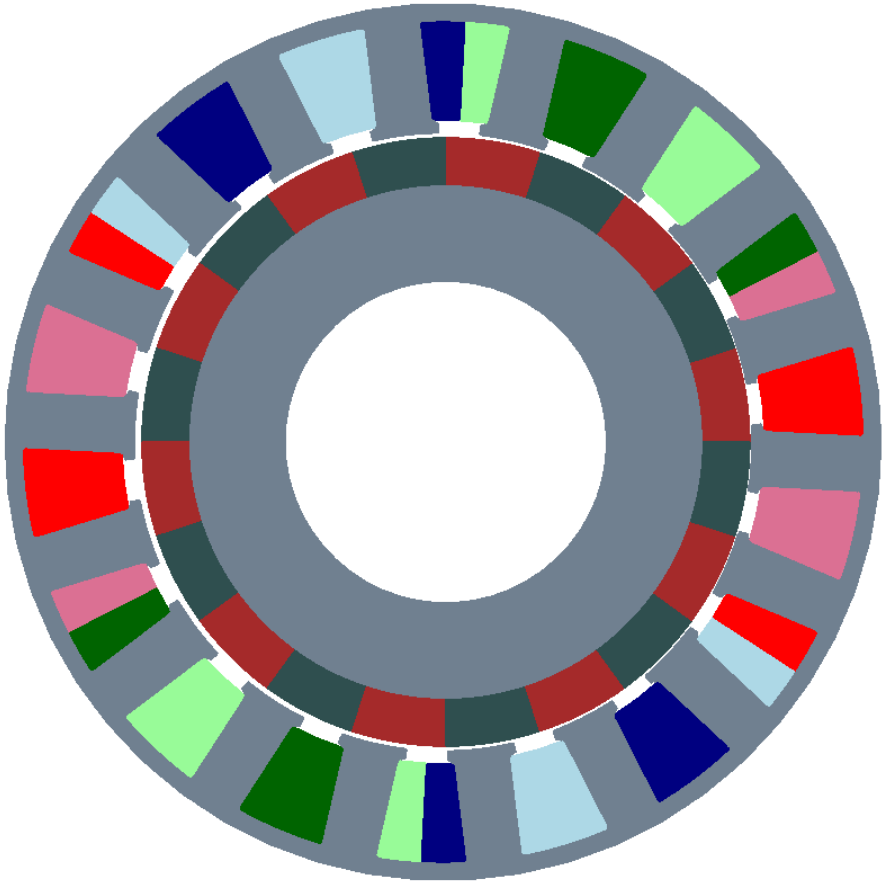
```
motor.rotor.offset.x=0.3

motor.rotor.offset.y=0.1

motor.rotor.offset.rotateaboutneworigin=1 // setting this to 1 results in static eccentricity. setting this to 0 results in dynamic eccentricity.
```

17.4.2 Motor models

17.4.2.1 Outer rotor motor



17.4.2.2 Inner rotor motor



17.5 Air pockets

The air pockets in the iron parts are the spaces where field lines do not go through.

These can be created through the raw coordinates or through the GUI. The air pockets created on the stator are replicated around the stator as many times as the number of teeth are. The air pockets created on the rotor are replicated as many times as the number of rotor poles are.

17.5.1 Code implementation & Workflow

```
// step 1
motor=r_buildvar('poles',20,'slots',18,'build','flux')
motor.buildmodelin='scilab' // set the build environment to scilab.


// step 2
motor=r_buildmotor(motor) // plot the motor model in scilab
// step 3
// click on the button that says 'pocket'
```

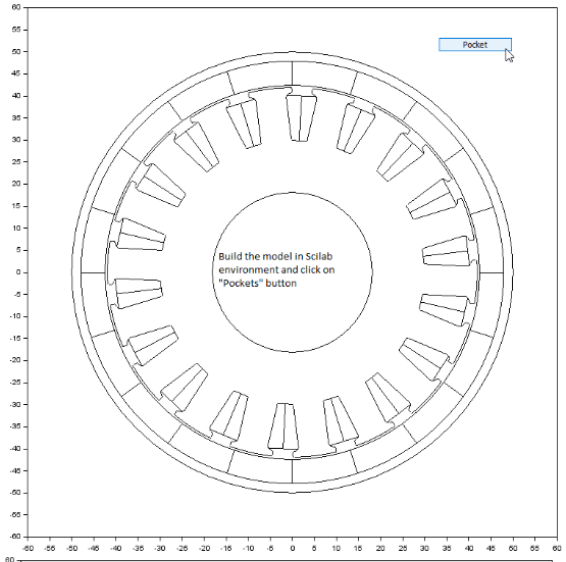
```
// step 4
// click on the motor's iron regions, either on the stator or on the rotor.
// notice the bottom of the graphic window that indicates the marked points.
// left click will keep adding points to the polygon.
// middle click adds spaces between polygons in case of multiple air pockets.
// to switch between air pockets in stator and in rotor, use middle click and go to the next air pocket.
// once done, use right click to come out of the pocket-addition context.
// step 5
motor=r_buildmotor(motor) // re-plot the motor model in scilab to confirm the entries
```

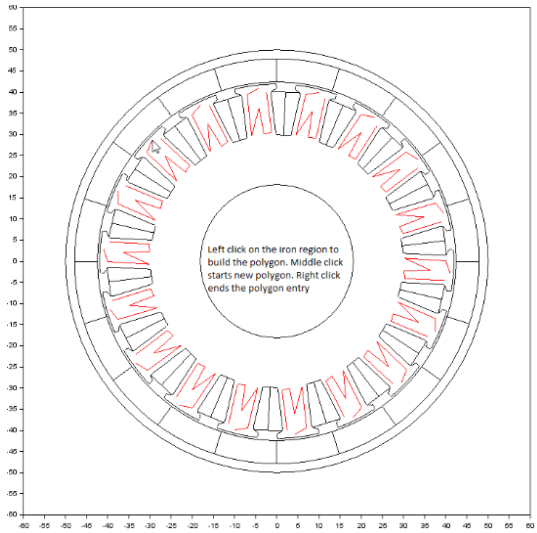
Alternatively, the following code may be used when we know of the motor's geometry and the location of the iron regions.

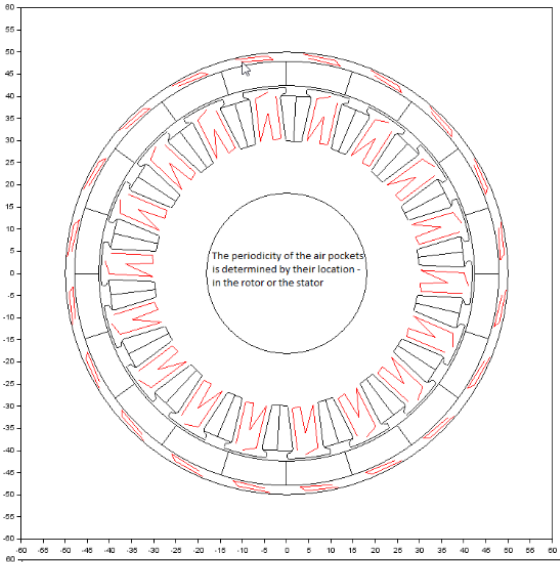
```
motor.stator.pocket.enable=1
motor.stator.pocket.polygon=[
   0.          0.
   9.3793785   38.032541
   5.3064478   39.159837
   8.9239024   46.723543
 ];
motor.rotor.pocket.enable=1
motor.rotor.pocket.polygon=[
   0.          0.
 - 5.3444676   26.443515
   2.0041754   20.753138
   1.0020877   27.447699
 ];
```

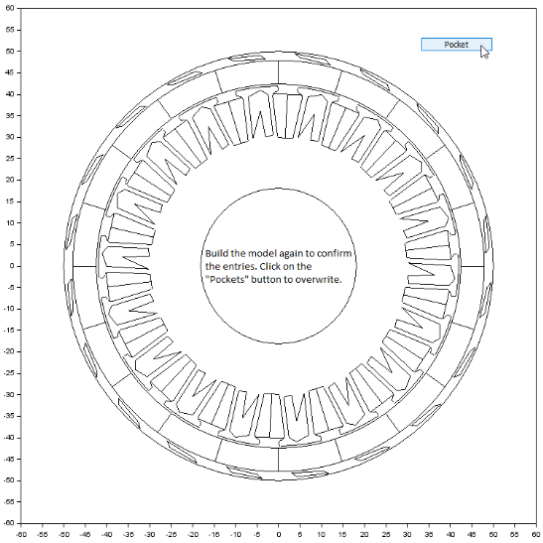
17.5.2 Models of the motors & steps to create pockets

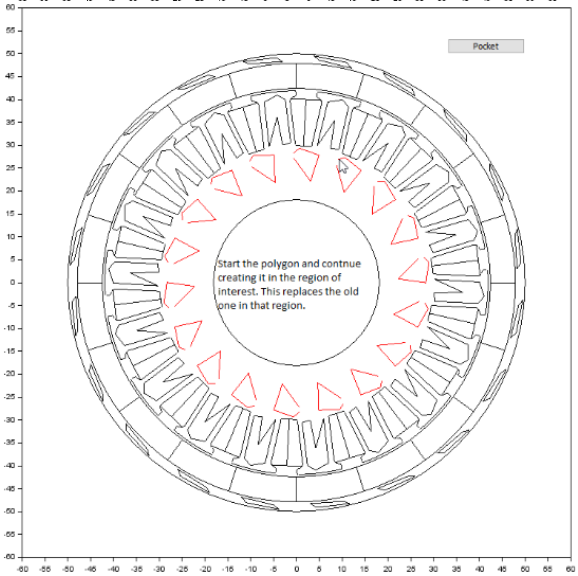
17.5.2.1 OR BLDC

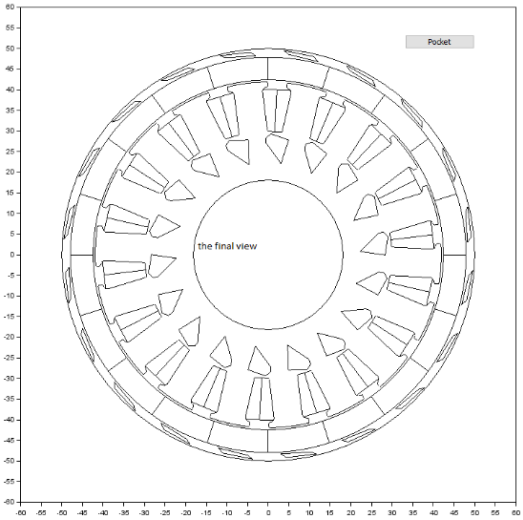


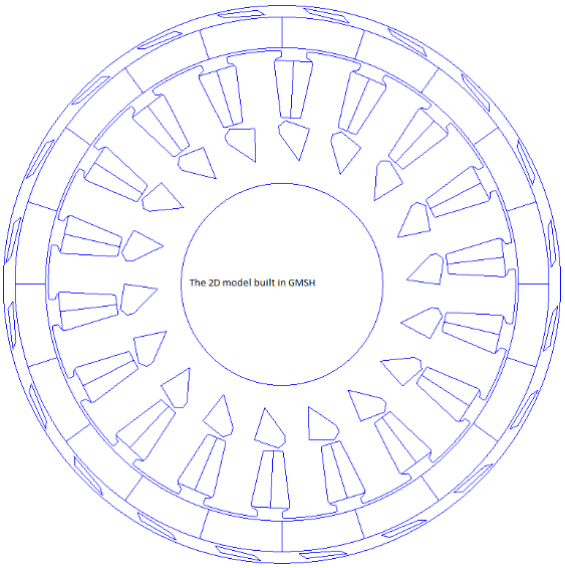


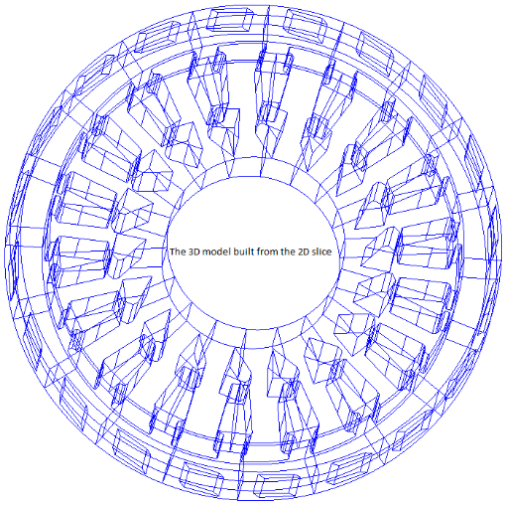


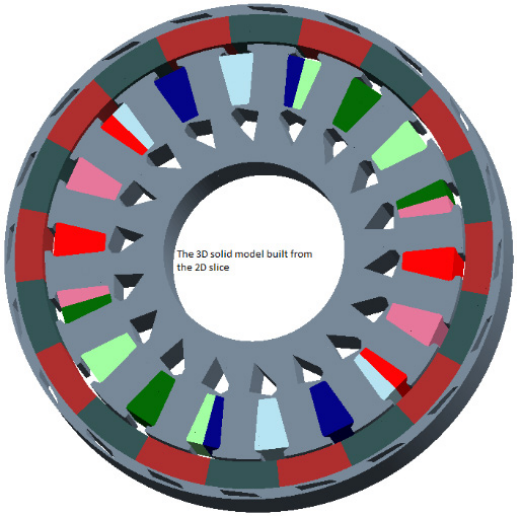




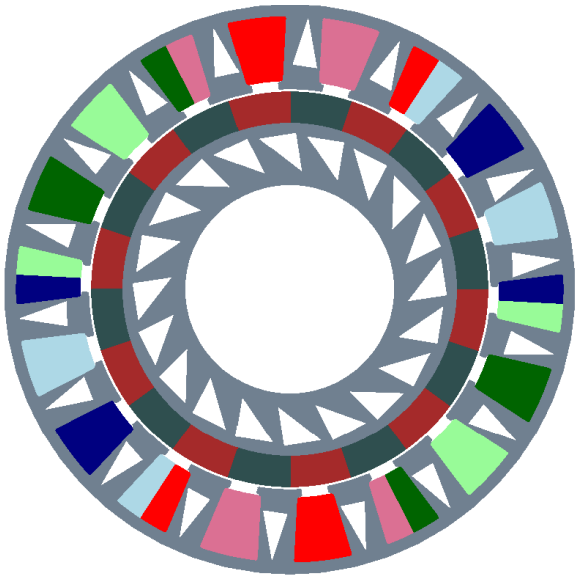


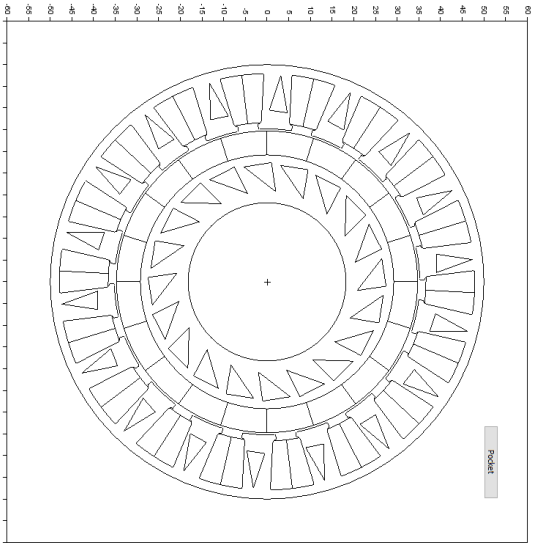


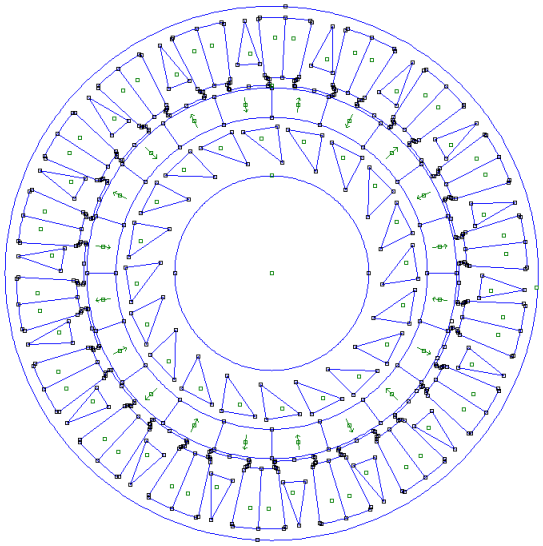




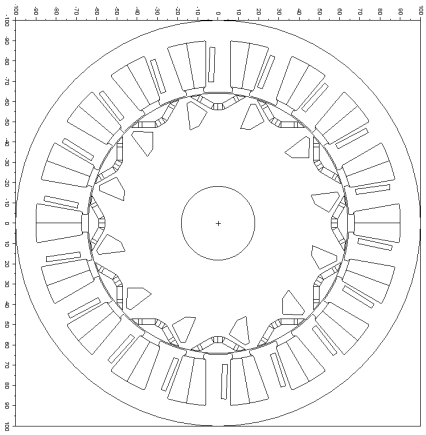
17.5.2.2 IR BLDC

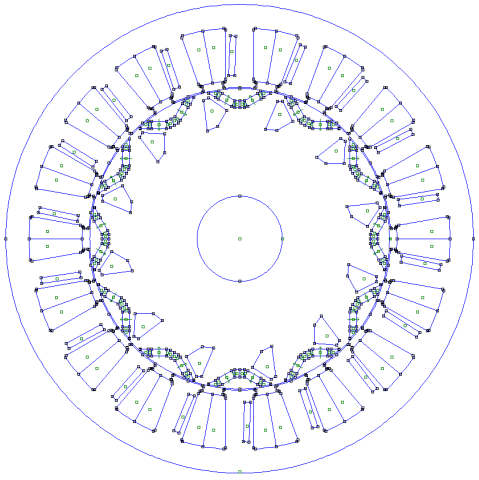






17.5.2.3 IR IPM





17.6 Rectangular magnets

Rectangular magnet geometry can be generated with rlib.

The Surface mounted PM BLDC motor comes with either arc or rectangular magnets. Both the inner rotor and outer rotor motors are configured to have rectangular magnets. The shortest (not the average) distance between the stator and the rotor is considered the airgap in the motor.

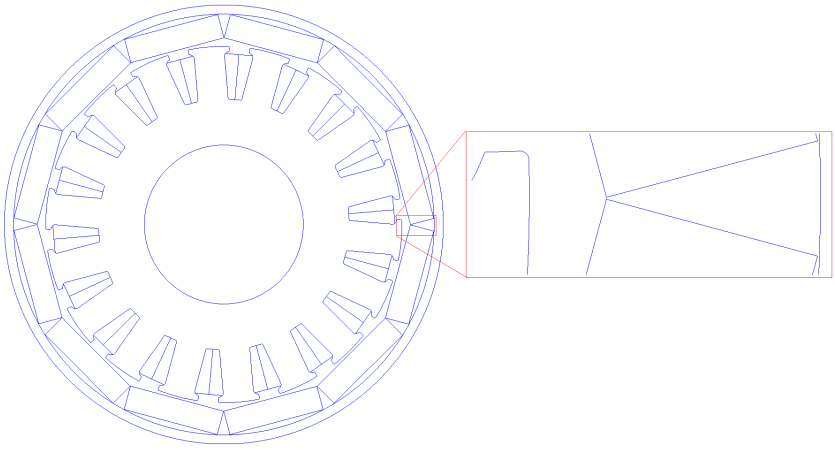
17.6.1 Code implementations

The following code creates and configures the rectangular magnets, irrespective of IR or OR BLDC.

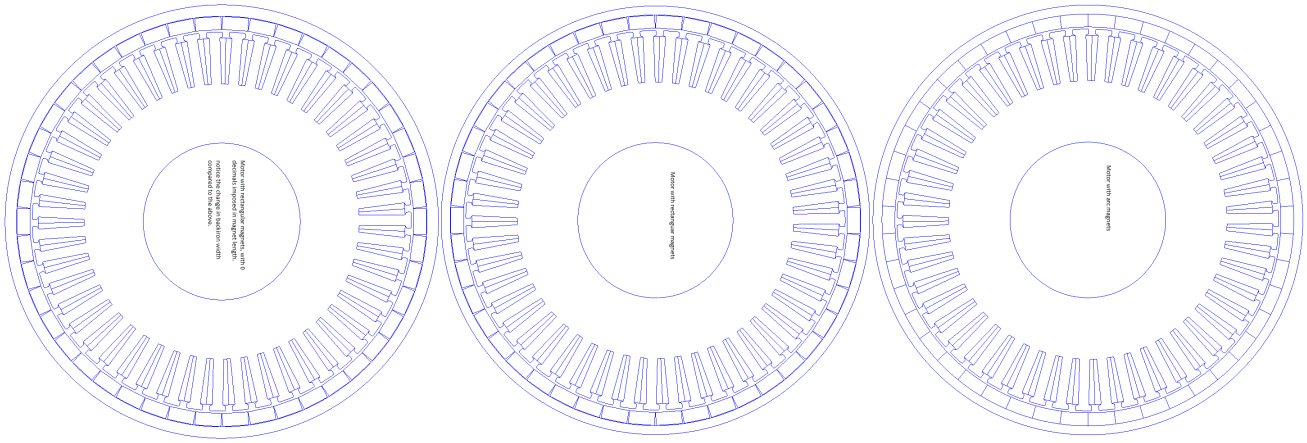
```
motor.rotor.rectangularmagnet=1 // sets the magnets to rectangular shape
motor.rotor.polearc=180 // has to be 180.
motor.rotor.halbach_type=0 // has to be of no halbach type
motor.rotor.halbach_fraction is not applicable
motor.rotor.rectangularmagspacing=0.05 // shortest space between corners of the magnets
motor.rotor.rectangularmaggluethick=0.05 // shortest space between magnet and backiron
motor.rotor.magnet_thickness=5.5 // thickness of the magnet in radial direction
motor.rotor.rectmagwidresoldecimals=-1 // this is a critical parameter in determining the magnet length along tangential direction
// when set to -1, the magnet length is not modified.
// when set to 0 or 1 or 2 or 3, or etc, this represents the number of decimals in the magnet length. if this factor is set to 1, the magnet length can be 10.0 or 10.1; nothing in between.
```

17.6.2 Images of the models built

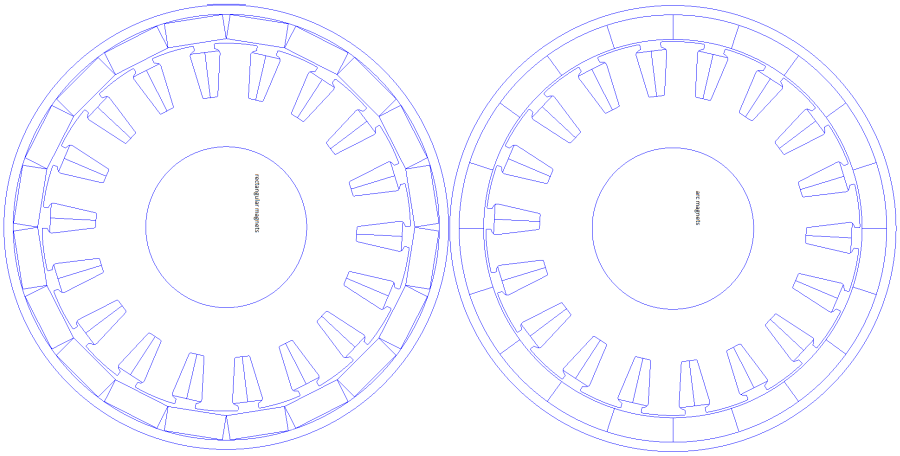
The image below highlights the space between magnets and the space between magnet and the backiron.



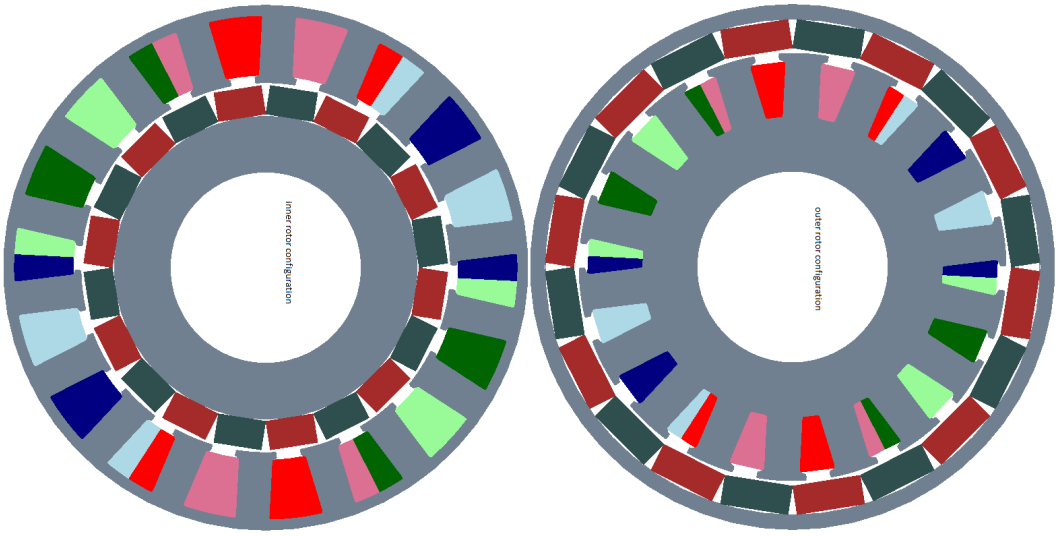
The image below highlights the contrast between arc, rectangular magnet (unregulated length) and rectangular magnet with its length regulated. The series of images below show the high pole count scenario.



The image below again highlights the contrast between arc and rectangular magnets.



The image below shows the configurations with inner rotor and outer rotor configurations



17.7 Consequent poles

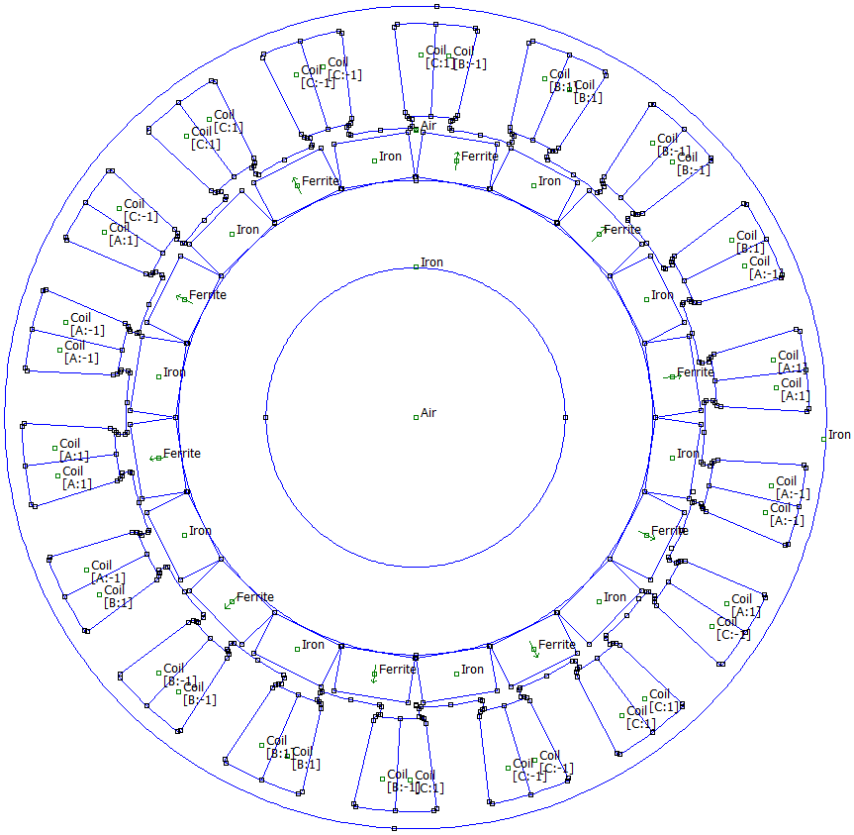
Consequent pole motors typically have magnets followed by an iron part instead of another magnet. These configurations have been implemented for IR and OR BLDC configurations.

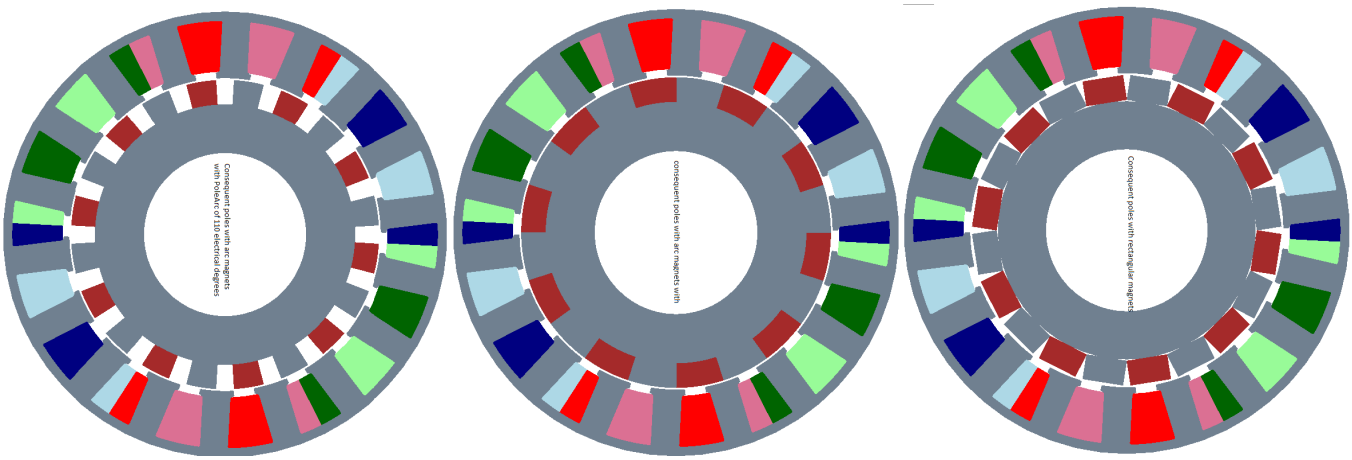
17.7.1 Code implementation

```
motor.rotor.consequentpoles=1 // this sets the consequent poles.
```

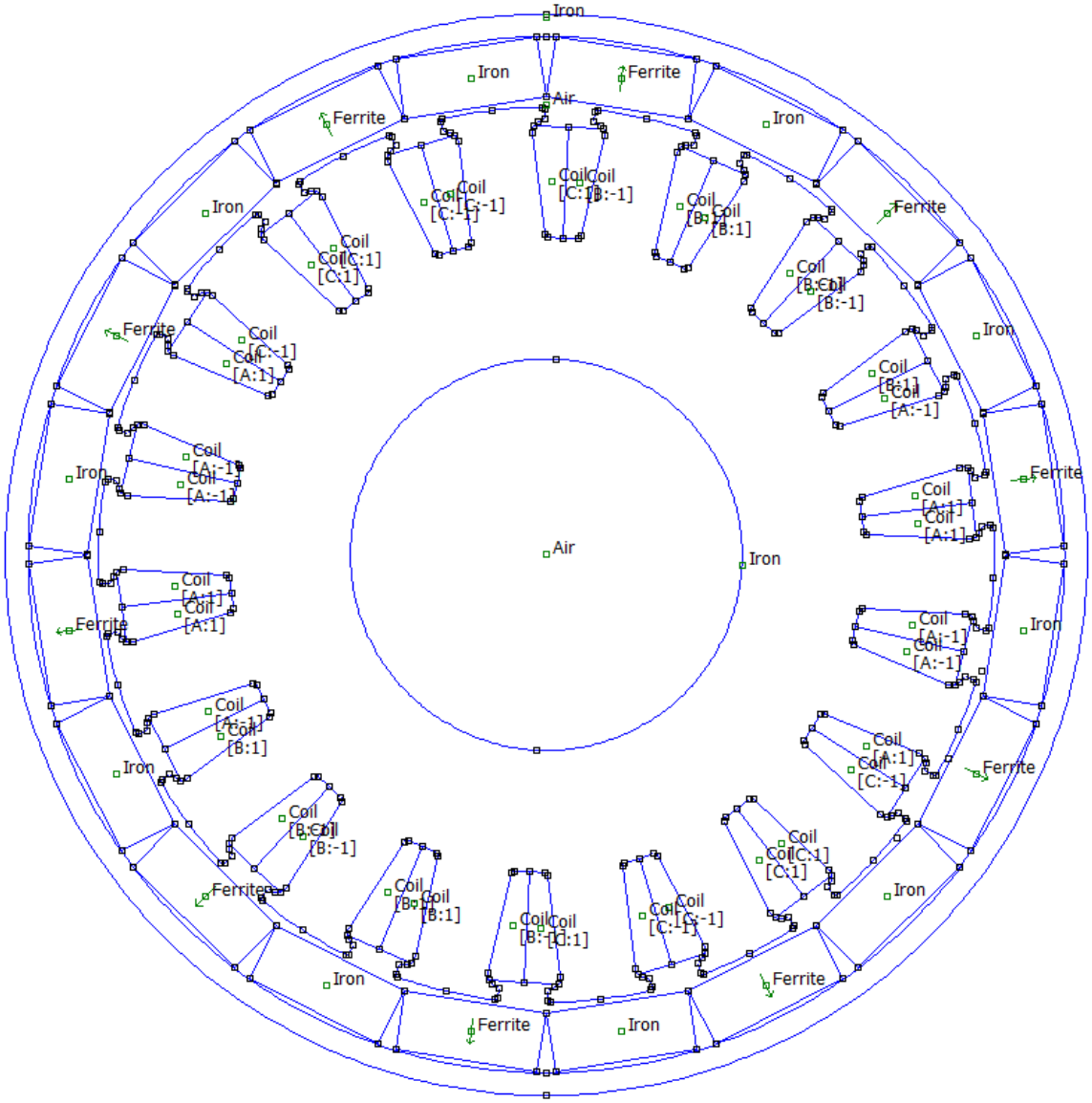
17.7.2 Example configurations

Inner rotor BLDC variations with consequent poles.





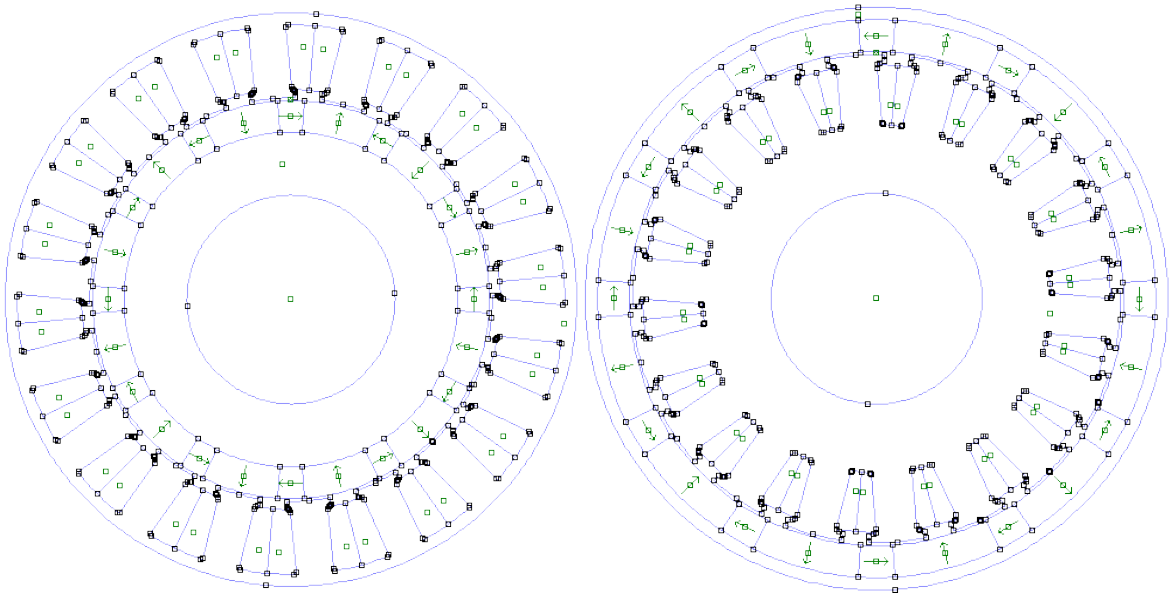
Outer rotor BLDC variations with consequent poles.



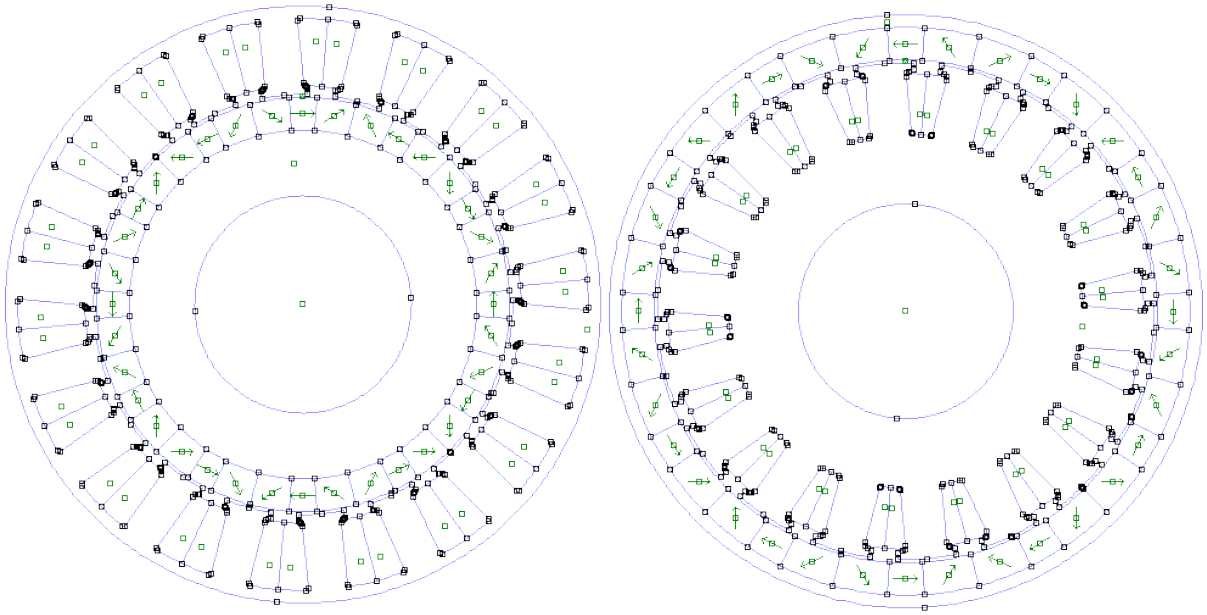
17.8 Halbach arrays

This tool rlib implements two types of halbach configurations.

First configuration is with magnets in two orientations (radially in/out and tangential). The following are the images for IR and OR type BLDCs with type 1 halbach implemented.



Second configuration is with magnets in three orientations (two magnets symmetrically inclined to radial direction and tangential). The following are the images for IR and OR type BLDCs with type 2 halbach implemented.



17.9 Motor losses and efficiency maps

An example motor is considered and solved using FEA for different speed and torque points to predict the losses using Bertotti loss model and LS loss model. The bertotti losses are used along with the magnet losses and the copper losses to predict the input and hence the efficiency of the motor.

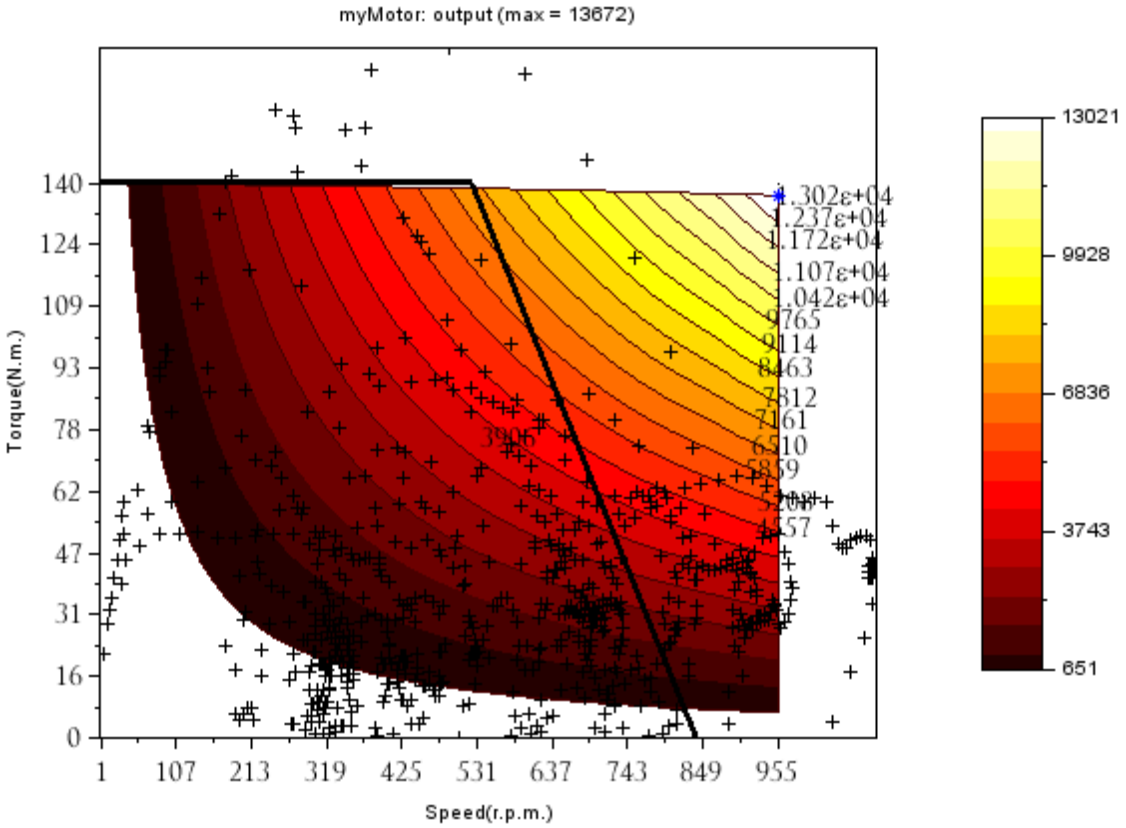
LS Loss model was just plotted to be able to compare with the bertotti losses.

Bertotti losses can be split into hysteresis losses, eddy losses and excess losses.

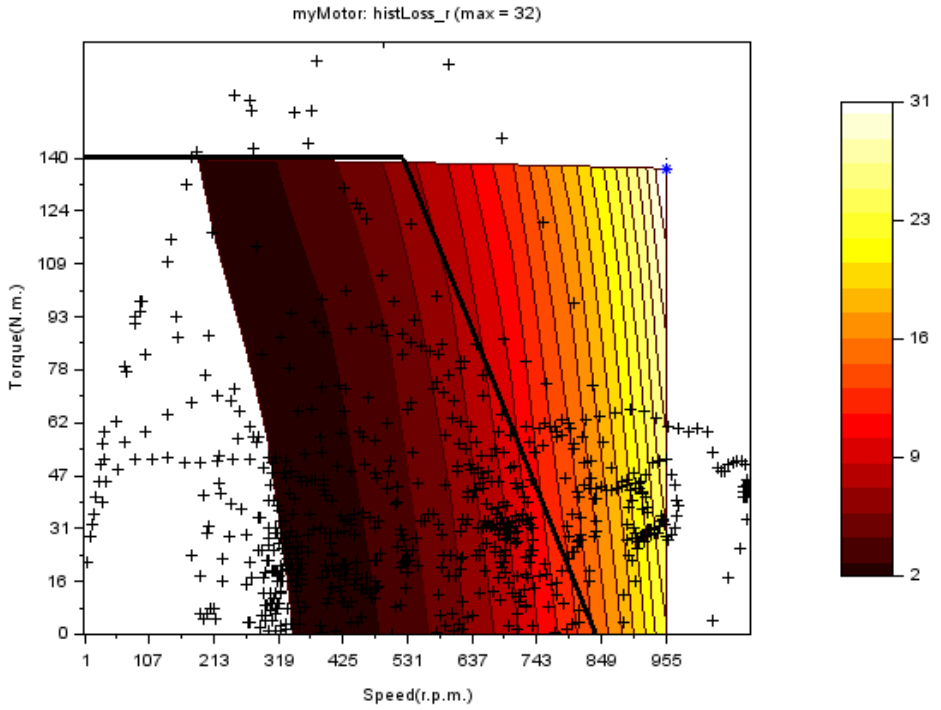
In each of the following images:

- the dark-thick line represents the speed-torque characteristic curve.
- the black + dots represent the operating points of the motor when used as a traction motor on a certain vehicle with a certain drive cycle.
- the blue * dot represent the peak of the corresponding map.

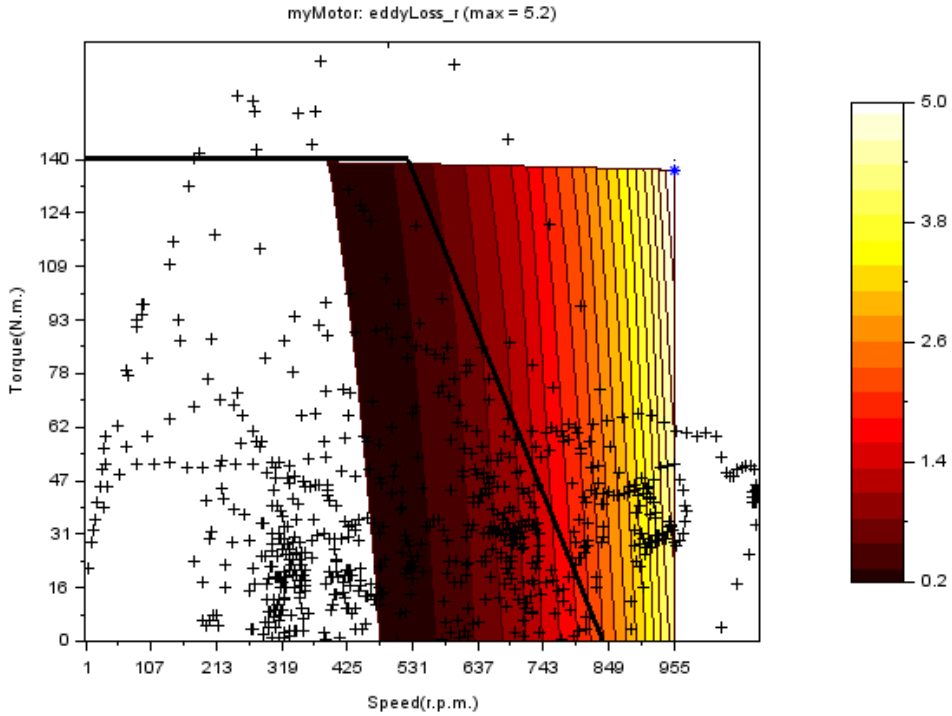
The following output map can be generated from the knowledge of speed and torque values.



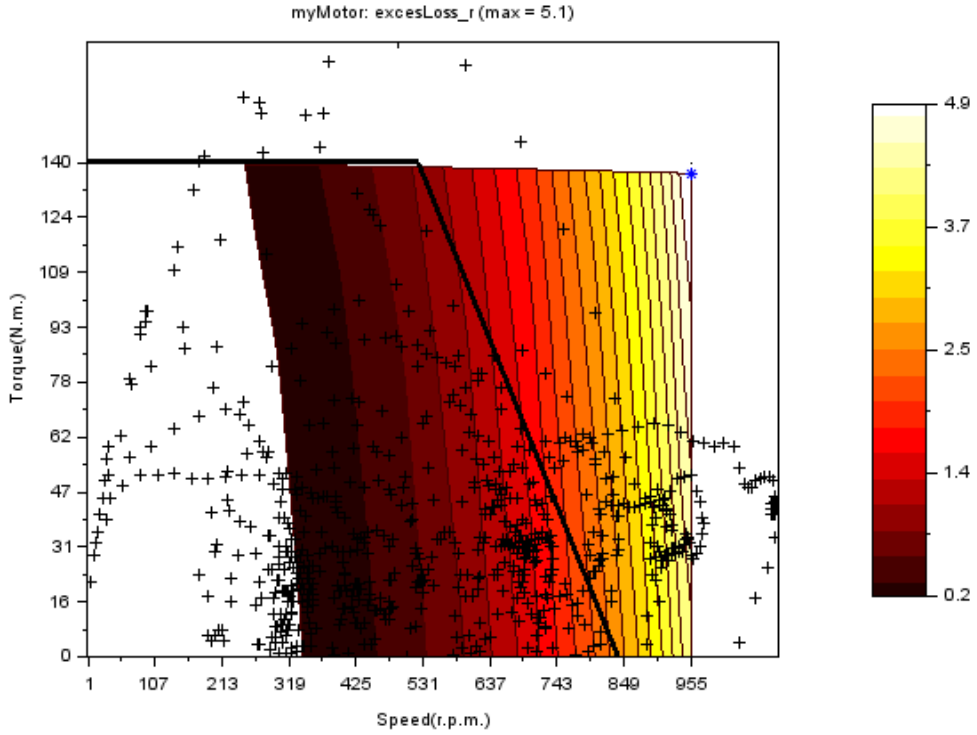
The first component of bertotti losses in the rotor - hysteresis losses



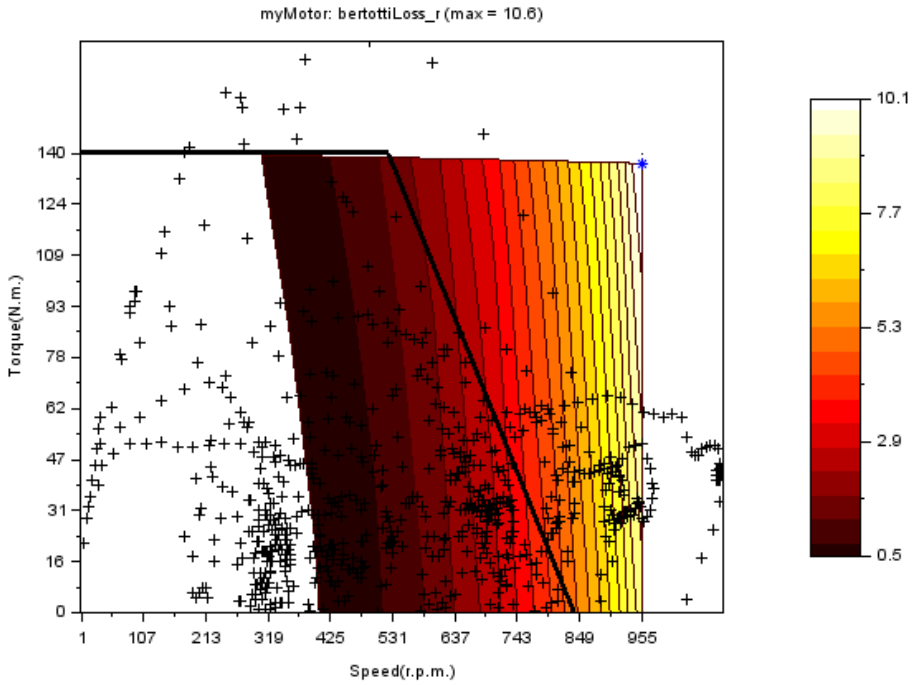
The second component of bertotti losses in the rotor - eddy losses



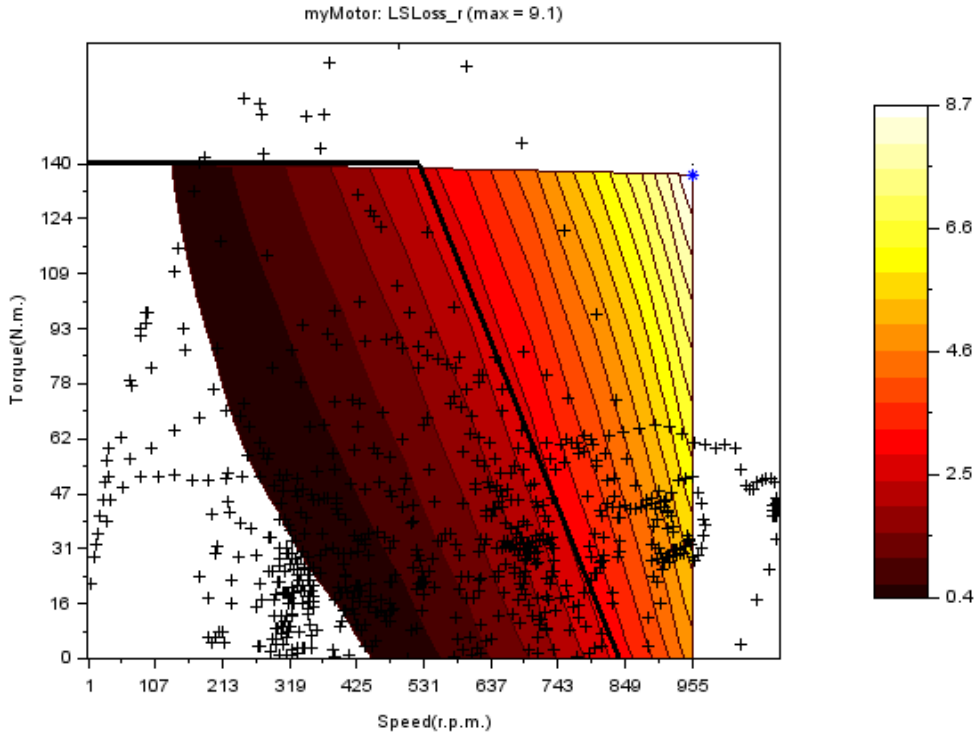
The third component of bertotti losses in the rotor - excess losses



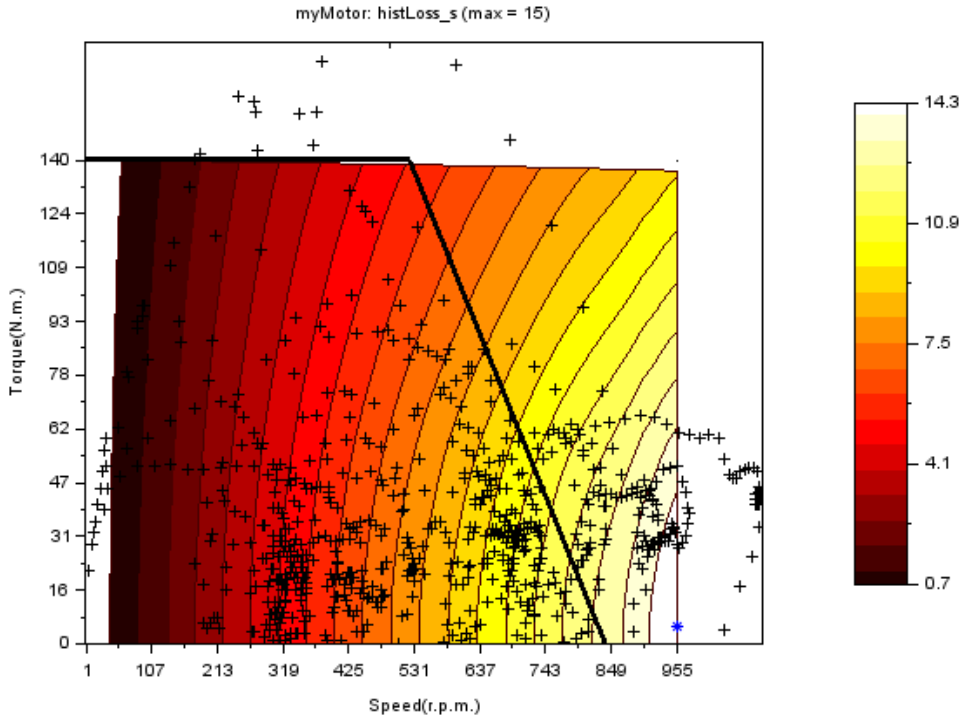
Total bertotti losses in the rotor



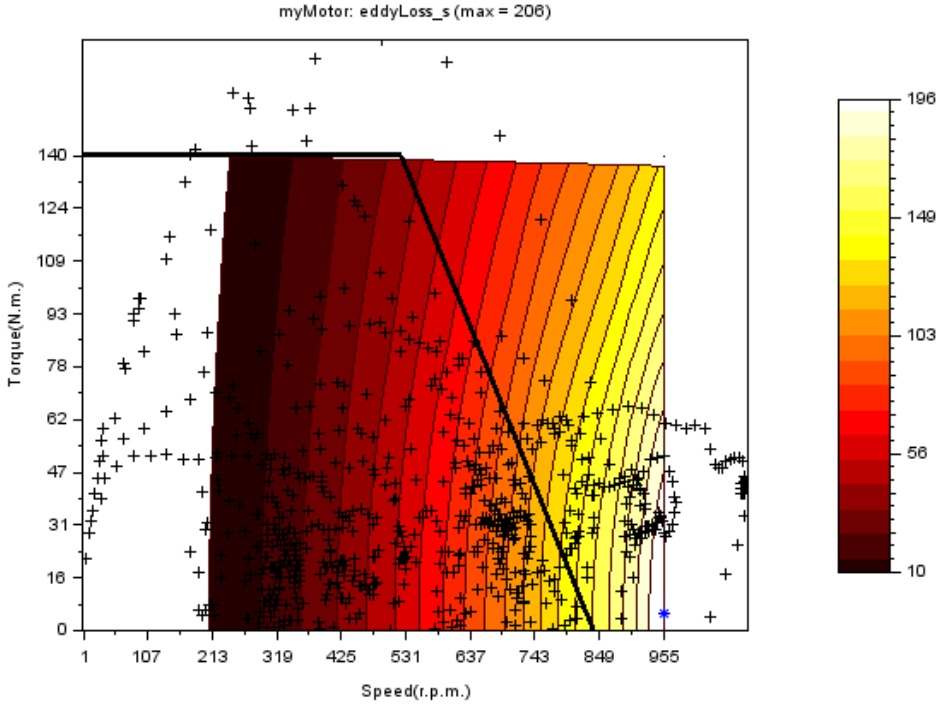
LS losses in the rotor - an equivalent of Bertotti losses, but calculated differently



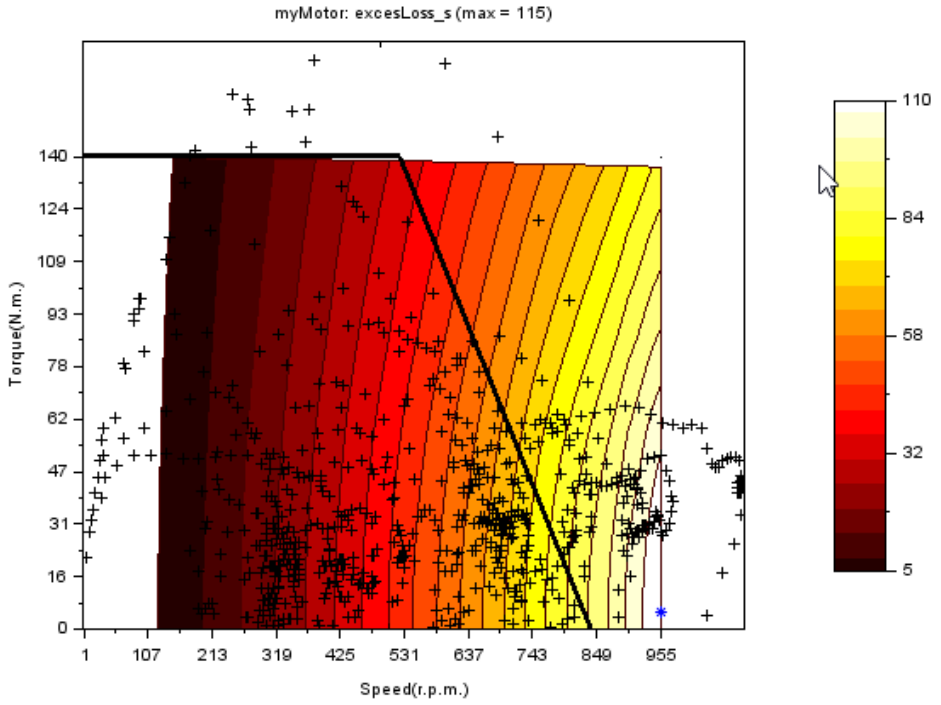
The first component of bertotti losses in the stator - hysteresis losses



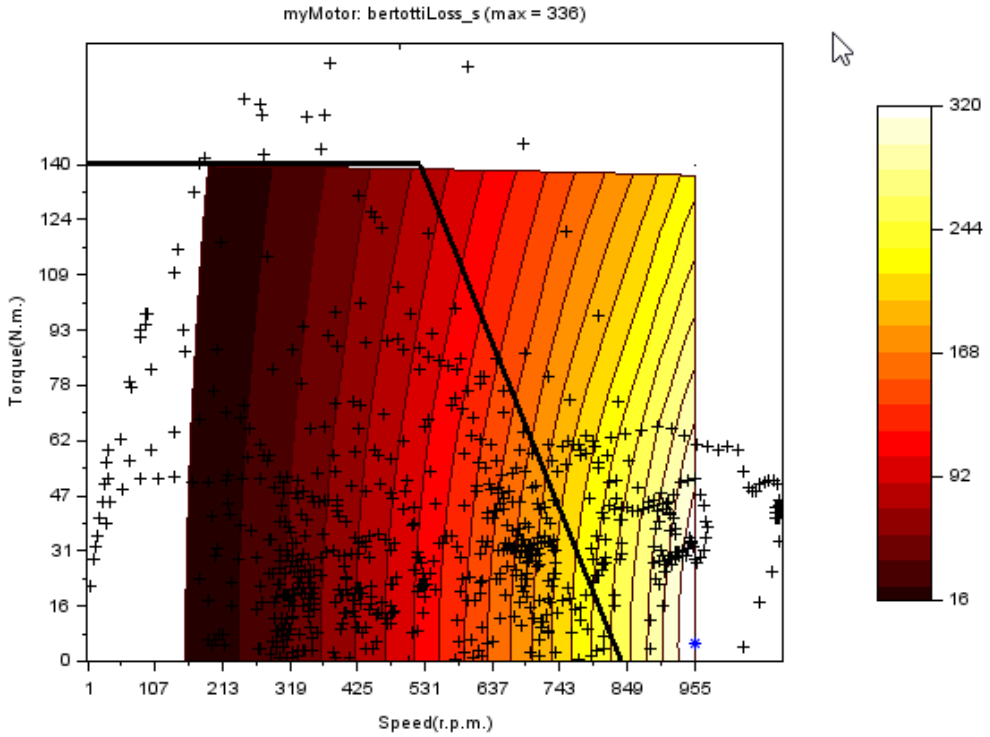
The second component of bertotti losses in the stator - eddy losses



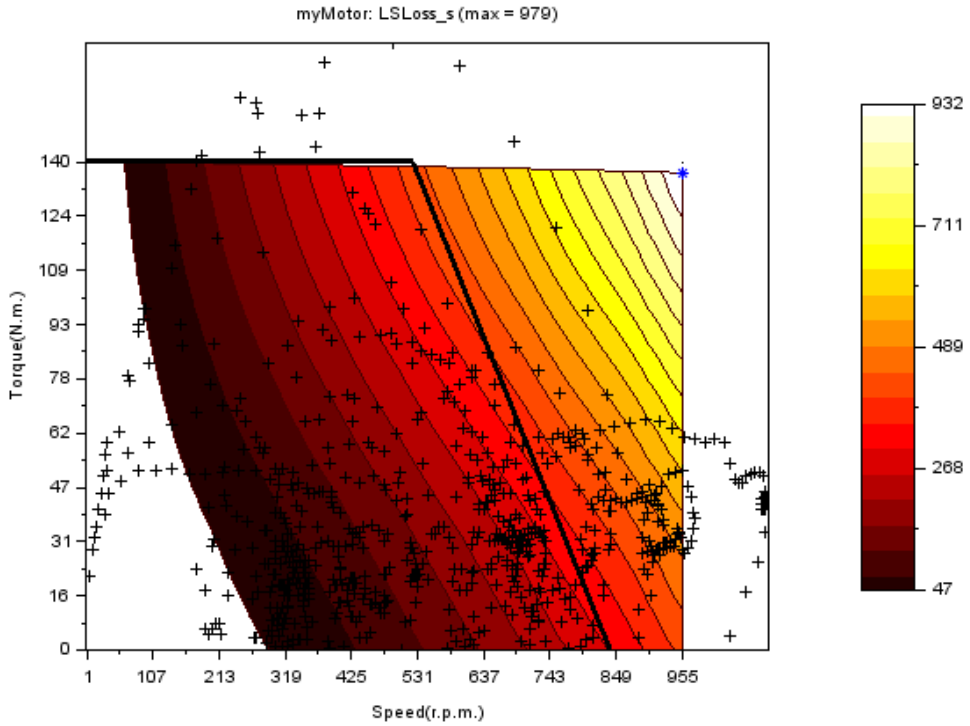
The third component of bertotti losses in the stator - excess losses



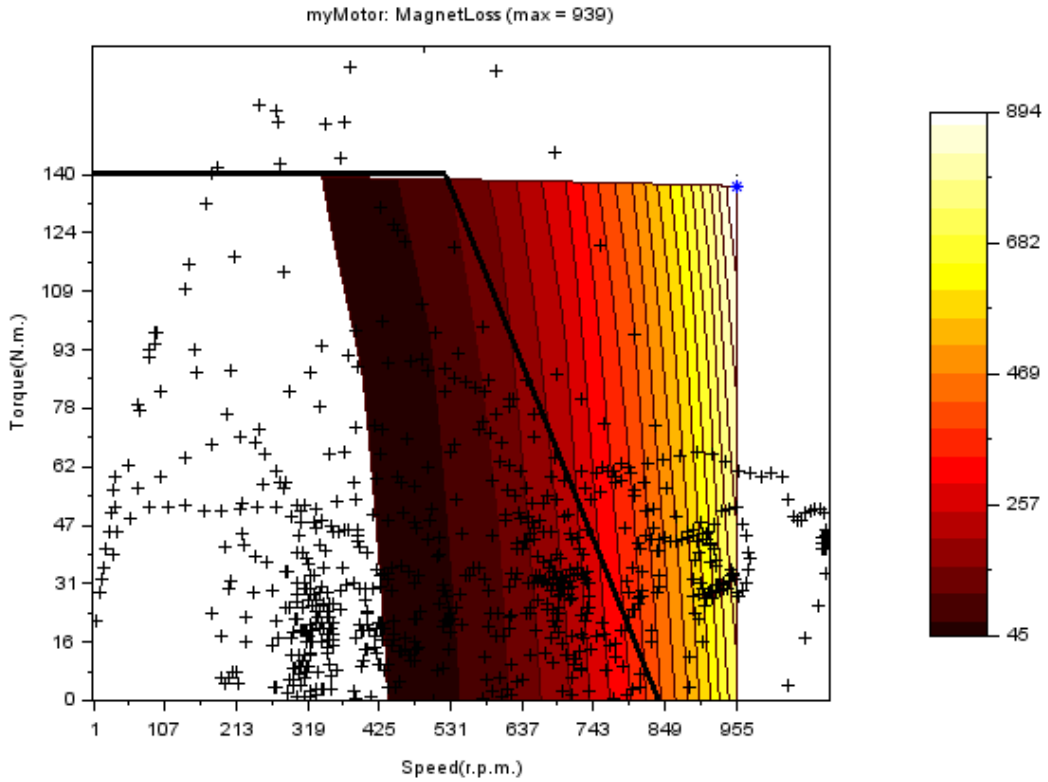
Total bertotti losses in the stator



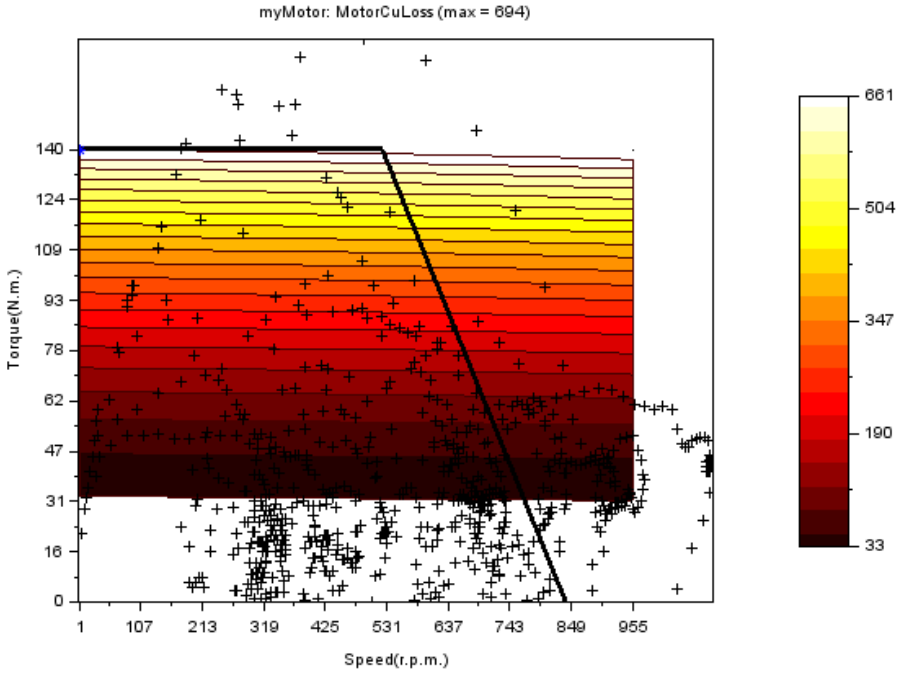
LS losses in the stator - an equivalent of Bertotti losses, but calculated differently



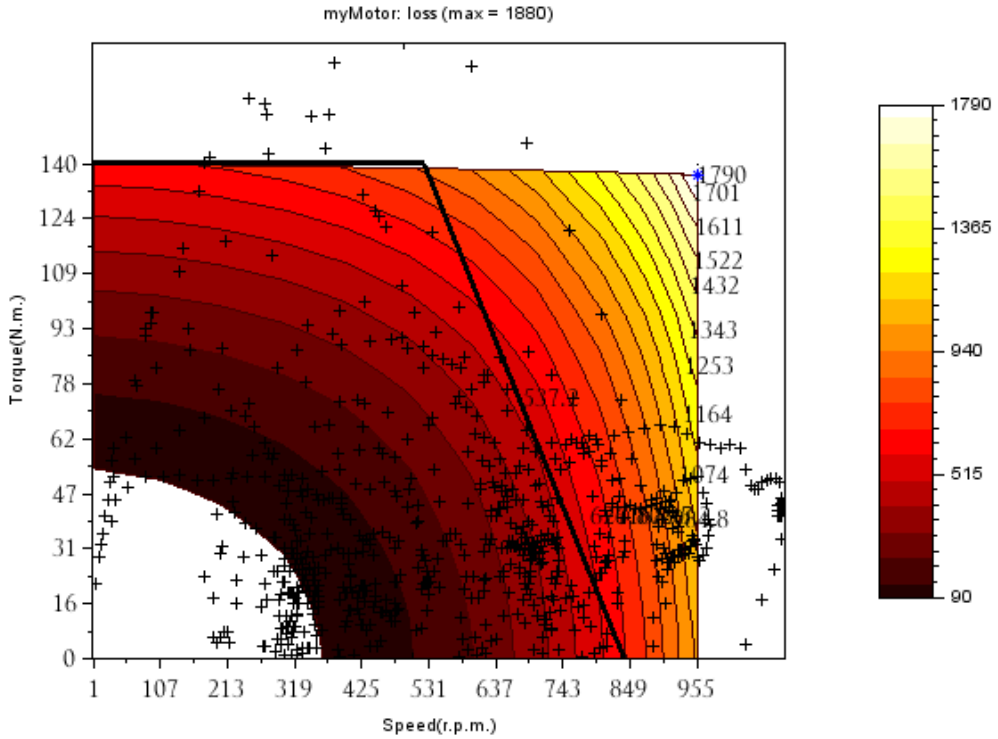
Losses in the magnet



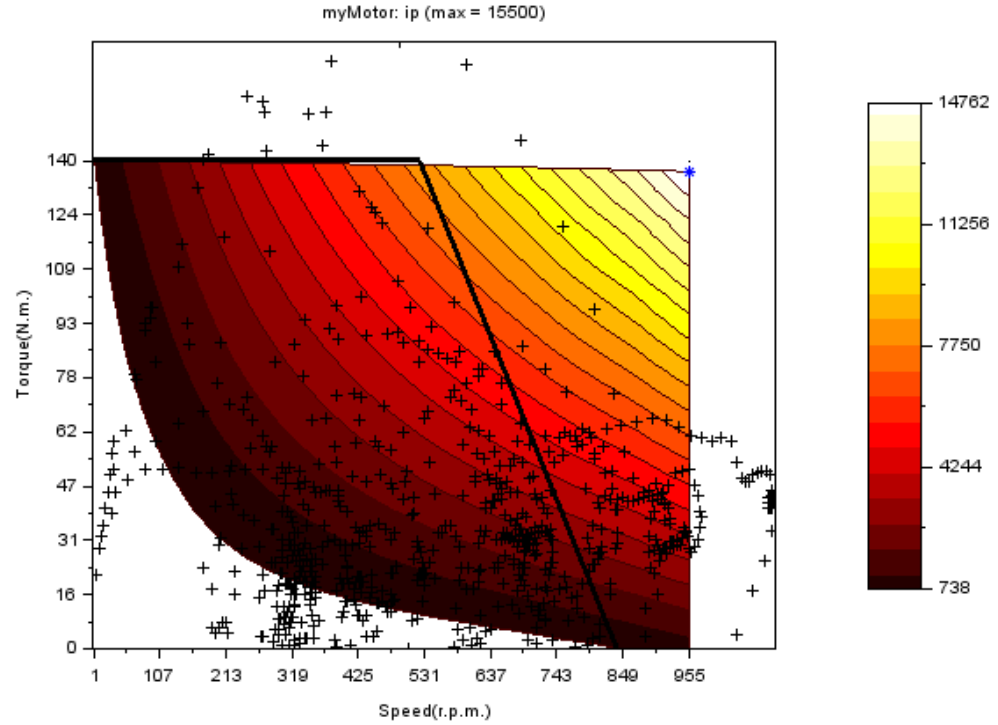
Copper losses



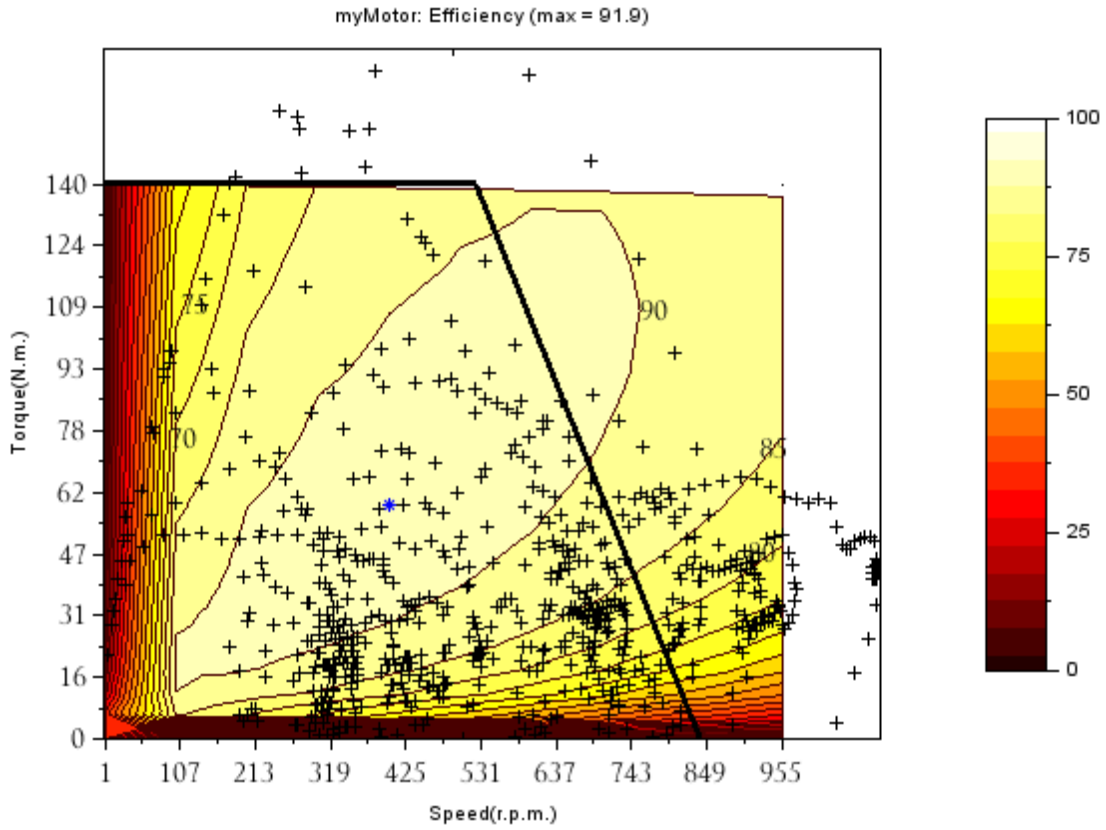
Total losses



Output + Losses = input



Predicted efficiency



Now, the above efficiency map assumes a default material, and certain ideal processes. In reality, the losses may be more than predicted. It is recommended to use a factor of safety for the losses estimated while calculating the efficiencies.

18 Example Scripts

The following is a common code to load the libraries. You need to pick the corresponding geometry file.

```
exec('D:\femm42\scifemm\scifemm.sci',-1)
//exec('E:\work\rlibtrials\motorrlibdllexe\rlib_sci\rlib_iripm.sci', -1)
//exec('E:\work\rlibtrials\motorrlibdllexe\rlib_sci\rlib_iripm.sci', -1)
exec('E:\work\rlibtrials\motorrlibdllexe\rlib_sci\rlib_orbldc.sci', -1)
//exec('E:\work\rlibtrials\motorrlibdllexe\rlib_sci\rlib_orim.sci', -1)
exec('E:\work\rlibtrials\motorrlibdllexe\rlib_sci\rlib_common.sci', -1)
exec('E:\work\rlibtrials\motorrlibdllexe\rlib_sci\rlib_odr.sci', -1)
```

18.1 Set Magnet Type

```
motor=r_buildvar('poles',12,'slots',18,'dia',115);
//  all the geometrical inputs are in millimeters.
this function requires one input (motor structure) to generate optimization structure.
this may accept upto 8 sequantial inputs (poles, slots, outer radius, stack length, airgap, winding, vdc, idcmax) to return a motor structure.
2 inputs will set the motor orad to 50mm, stack length to 15mm, airgap to 0.5mm, winding type to star type, voltage to 48v, idcmax to 80a.
provide the vdc as rated v(typically 12v and 48v) and then change the soc when required.
if 48v mode is selected, the cells are li-ion type of 3.6v nominal voltage. set the battery cells in series to set the operating voltage.
if 12v mode is selected, the cells are lead acid type of 12v nominal voltage. set the battery cells in series to set the operating votlage.
any voltage less than 18v is taken as 12v band, and more is taken as 48v band.


use the new calling method: motor=r_buildvar('poles',12,'slots',18,'dia',150)
special calling refrences: poles, slots, outerrad, outerdia, stacklen, airgap, winding, vdcrated, idcmax, and coilthrow
if any other field needs to be set, it can be set with a pair of arguments such as 'stator.shaft_rad',12.0
motor.solve.tooutput has the format
```

[b/h]: name,starting radius, ending radius, number of points between radii(including), number of points on each circle, b/h.

[fill factor, slot area]: name, x coordinate, y coordinate,,,fill/area.

[circuit properties, volume, i2r]: circprop/volume/i2r,,,,,circprop/volume/i2r.

[torque]: torq,startangle,endangle,incrementalangle,,torque.

when the start/end/increments are given as empty, instantanious fea is done for the single rotor position.

when the motor.run.dangle_md is given to be other than 0, rotor is rotated to that position and solved once.

to solve for no load speed, backenmf constant, keep the torque fields active, and provide a zero motor.run.dangle_md. once solved, the rotor is placed at initial position.

1 input corresponds to creating optimization variable. the input can be any motor structure or any other fromat(in this case, a dummy motor model is created)

opti=r_buildvar(motor);

opti.thresholds has the format [extractfrom, parametrname, toconsidervalue, thresholdvalue, passiflessthanthreshold, additional parameter under sub-heading, val of addnl parameter, handlearray]

opti.limits has the format [parametername, changecriteria, [changeparam, paramval1], to change val/index]

motor.stackdepth=15;

motor.airgap=0.5

motor.rotor.magnettype='Ferrite';

18.2 Set Battery parameters

motor=r_buildvar('poles',12,'slots',18,'dia',115);

motor.controller.vdcrated=48;

motor.controller.imax=80;

18.3 Set Meshing parameters

```
motor=r_buildvar('poles',12,'slots',18,'dia',115);
motor.solve.frugalmeshing=1;//
motor.solve.meshminangle=15;//
motor.solve.maxseginarc=1;
motor.solve.magnetmeshsize=0.8;
motor.solve.spacemeshsize=2;
motor.solve.ironmeshsize=0.6;
motor.solve.wiremeshsize=1;
motor.solve.airmeshsize=0.8;
```

18.4 Set Solving parameters

```
motor=r_buildvar('poles',12,'slots',18,'dia',115);
motor.solve.ptsin60degree=10;
motor.solve.valuesbelowpeak=2;
motor.solve.tooutput=[.....];// this is an important array of string
motor.solve.mini=0;
motor.solve.instantaneous=0;
motor.solve.femm.symboundarygappct=24;
motor.solve.customchar=0;
motor.run.sw_currentfr=[0 80 -80];
motor.run.wradps=100;
motor=r_solve(motor);//  this function requires a motor structure as an input and optional phase currents [i1,i2,i3,...] to solve using fea. if the phase currents are not provided, the geometry is used to calculate resistance, and the controller settings are used to calculate the currents in each phase. this functions requires the femm model to have been created.  this function returns a motor structure with output values updated based on the solution. then the motor.run.dangle_md is set to non zero values (or when motor.solve.tooutput's torque line has no starting and ending and incrementa angles), the perturbation is not done to calculate the kb and nlrpm. only the torque after rotating by the angle motor.run.dangle_md is calculated, along with few other parameters. output is populated based on motor.solve.tooutput parameters. the syntax is b/h: name,starting radius, ending radius, number of points between radii(including), number of points on each circle, b/h. fill factor, slot area: name, x coordinate, y
```

coordinate,,,fill/area. circuit properties, volume, i2r: circprop/volume/i2r,,,,,circprop/volume/i2r. torque: torq,startangle,endangle,incrementalangle,,torque. when the start/end/increments are given as empty, instantanious fea is done for the single rotor position. when the motor.run.dangle_md is given to be other than 0, rotor is rotated to that position and solved once. to solve for no load speed, backenmf constant, keep the torque fields active, and provide a zero motor.run.dangle_md. once solved, the rotor is placed at initial position.

18.5 Set characterization parameters

motor=r_buildvar('poles',12,'slots',18,'dia',115);

motor.solve.customchar=1;

r1_findmotorchar(...);//r1_findmotorchar(motor,c/m/m_sw/g/ind/ind_a/ind_a_vs_i/ind_sw/ind_sw_vs_i/m_vs_i,[imax,imin,steps],erev)

this function plots the motor's torque or back emf waveforms or inductance for one full electrical cycle. use second argument 'c' or 'm' or 'g' or 'm_sw' or 'ind' or 'ind_a' or 'ind_a_vs_i' or 'm_vs_i' for only cogging mode or motoring mode or generation mode plots, motoring mode with switching, phase inductances, phase a inductance, and inductance of phase a with current. this function consumes considerable time. m mode considers same switch combinations all around, while m_sw considers switches for the corresponidng rotor position. ind or ind_a accept an optional third argument of current at which inductance needs to be calculated. exclusion will result in calculation at idcmax. ind_a_vs_i accepts 3 inputs that are max current, min current and steps. exclusion of any of these values will cause : max to be set at idcmax, min to be set at -max, and 11 steps. ind_a_vs_i will take considetable time to finish. ind_sw will calculate the cumulative inductance of as seen by the controller for one switching instance for a given dc bus current limit, ind_sw_vs_i will range the currents and plot.

if the pc supports and when you want to run parallel agents for faster characterization for ind, ind_a_vs_i and ind_sw_vs_i; use motor.par=1 and motor.paths.femm='yourfemmexecutablepath\femm.exe'. select motor.agents=#of concurrent parallel sessions to be run. these are not native fields, and so you have to create. to disable parallelization, set par=0. it is disabled by default when the motor structure is created.

ind executes all three inductances in one go, ind_a_vs_i and ind_sw_vs_i execute all current variations in one go. position increments are taken as anoher parallel run.

m, g and c run for multiple points across the electrical angle. m_sw implementation is buggy.

18.6 Loss map from Flux

motor=r_buildvar('poles',12,'slots',18,'dia',115);

motor.buildmodelin='flux';

motor.solve.flux.magstic0_stdyst1_trnsnt2=2;

motor.solve.flux.losspoints1grid2=1;

// set the bertotti loss model parameters in motor.solve.betrotti

motor=r_buildmotor(motor); // this builds flux model and auto solves for loss map generating a csv file.

r_2dfcontour('1.csv');// this function plots the 2d contour maps. the first entry is the array (effmap) of n x 3 (or 4 or 5 etc) entries. first column is rpm, second column is torque. third column onwards is taken as the efficiency (or the loss)

the optional entries are 'lines' or 'line' = 1, 'levels' or 'level' = 32, 'dot' or 'dots' = 1,'cmap' or 'colormap' or color' = hotcolormap(levels), 'title' = ' ' , 'tags' or 'tag' = ['speed(r.p.m.)' 'torque(n.m.)' 'efficiency']. additional tags are assumed to be loss if not specified for more than 3 columns in effmap entry.

to plot only the efficiency from the file/array, use 'only' followed by 'eff'

alternatively, to plot only the efficiency from the file/array, use 'onlyeff' followed by '1'

to overlay another line plot, use the optional argument 'lineplot' or 'draw' followed by a 2 column matrix.

to overlay another scatter plot, use the optional argument 'scatter' or 'scatterdata' followed by a 2 column matrix.

to set the limits on x or y axes bounds, use the optional argument 'xlim' or 'xlimits' or 'ylim' or 'ylimits' followed by a 2 element vector.

to set the torque values due to cogging to zero at zero currents use 'nocogging' followed by 1

to display drivecycle data, use 'drivecycle' followed by a drive cycle data, such as r1_getwmtcdata(2). a vehicle may also be speficied which has more info such as 'vehicle' followed by a vehicle varaible from r1_getmyvehicle.

one of the ways to use this function is the following:

vehicle=r1_drivecycledensity(r1_getwmtcdata())

r_2dfcontour('motor2d_20190913090100_demag\20190913090100_demag_losspoints.csv','nocogging',1,'useall',1,'draw',[0 63; 470 63; 580 0],'vehicle',vehicle,'drivecycle',r1_getwmtcdata(2))

18.7 Find phase resistance after building the model

motor=r_buildvar('poles',12,'slots',18,'dia',115);

motor=r_buildmotor(motor);

r1_findrphase(motor);// this function requires a motor structure to find the phase resistance. this requires the motor to have been built with motor=r_buildmotor(motor) call so that the geometrical construction parameters are updated in the motor structure.

18.8 Find Winding pattern

motor=r_buildvar('poles',12,'slots',18,'dia',115);

output=r1_gettoothwinding(12,18,{'delta'},'presentwindingpattern')

this function requires two integers as pole, slot values to return an output that stores winding pattern for single throw .

whenthe present winding pattern is provided, this function returns the same, in addition to the stator and rotor angles.

capital letter indicates that the magnetic field lines are horizontal, and going to the right.

output=r1_getwinding(motor)

this function requires a motor structure as input.

toothwinding nomenclature for both the inner rotor and outer rotor motors:

a,b,c will give a magnetic field radially out of the motor.

a,b,c will give a magnetic field radially into the motor

slotwinding nomenclature for both the inner rotor and outer rotor motors:

a,b,c represent a coil coming out of the screen. this is denoted by positive turn number in femm

a,b,c represent a coil going into the screen. this is denoted by a negative turn number in femm

the windingpattern is represented clockwise from the horizontal, in the direction of the phases a,b, and c.

18.9 Solving for model in Gmsh

motor=r_buildvar('poles',12,'slots',18,'dia',115);

motor.solve.gmsh.showallgeo=1 sets all the lines to be shown. Setting this to 0 gives a clean view of the motor model.

Motor.solve.gmsh.

Useparsolvers utilizes the parallel solves features in getdp, with the threads limited by the numberthreads field.

The loss model is a custom model, and at the moment the standard bertotti loss model.

We can enable the type of solver code to be generated by the choise in

Motor.solve.gmsh.analyses and selecting the corresponding sub fields.

18.10Building with symmetry

motor=r_buildvar('poles',12,'slots',18,'dia',115);

motor.buildwithsymmetry=1 results in a symmetric model built in all three tools.

The FEMM script creates a symmetric model with periodic air gap boundary condition that keeps the stator and rotor in place while the rotor's boundary conditions are rotated. Earlier implementation of this tool used sliding mesh control.

The corresponding airgaps and the split into the regions are all handled by the tool.

The flux and gmsh scripts create a symmetric model where the rotor part rotates.

18.11FEMM parallel characterization of the motor model

motor=r_buildvar('poles',12,'slots',18,'dia',115);

motor.par=1 enables parallel computation in FEMM

motor.agents=10 indicates the parallel instances of femm

The FEMM sessions are parallelly created to ensure faster characterization of the motor. This is useful for creating back-emf profile, torque-rotational angle profile, and many more.

18.12Set Optimization parameters

motor=r_buildvar('poles',12,'slots',18,'dia',115);

motor.solve.optimode=1;

r_optimize(motor); // this runs the optimization and creates odr files.

R_odr2csv('r_common_odr.bin');// r_odr2csv('y:\motor\scripts\rlib\r_common_odr.bin',50,'par');

r_odr2csv('y:\motor\scripts\rlib\r_common_odr.bin','par');

this function requires the path of compiled function file(*.bin) which has this function. most likely it will be r_common_odr.bin

each background session is expected to occupy 200mb to 250mb. carefully choose the parallel sessions to leave enough space for other programs on the pc

if you have 12gb ram installed, 7gb still utilized from task manager, then use 5gb (about 70 percent) for scilab, i.e. you can run 20 scilab sessions, max. if you have n odr files, then call odr2csv (lib , n/20 , par).

if the n/20 becomes more than 500, consider that each scilab takes more memory. it becomes iterative. instead, use the feature without par, plain old way.

18.13FEMM parallel optimization

```
motor=r_buildvar('poles',12,'slots',18,'dia',115);
r_optimize(motor)
```

Optimization instance runs in the scilab window. Many optimization instances can be run in the same computer without interfering with the other instance, running out of the same folder. The temporary file names and the resulting file names are handled to avoid conflict.

The seed values for optimization jumps are different for different sessions, powered by the rand() in scilab. Two sessions started at the exact moment will result in entirely different path travel.

18.14FEMM routine for optimization

```
motor=r_buildvar('poles',12,'slots',18,'dia',115);
r_optimize(motor)
```

Optimization implemented in scilab is a combination of random walk and simulated annealing. This combination helps achieve global max as opposed to local max of the standard approaches.