MotorAnalysis

Finite Element Analysis of Induction Motors

Version 1.1

User Manual

March, 2014

Vladimir Kuptsov
2v.kuptsov@gmail.com
http://motoranalysis.com

CONTENTS

1. INTRODUCTION

Thank you for your interest in MotorAnalysis.

MotorAnalysis is a Matlab application with graphical user interface for the finite element analysis of squirrel cage induction motors. MotorAnalysis is based on solving a nonlinear electromagnetic problem with coupled electrical circuit on two-dimensional and multi-slice domains.

MotorAnalysis is addressed to engineers and researchers engaged in design, optimization and analysis of induction machines.

2. BRIEF THEORETICAL BACKGROUND

The purpose of this section is to give the user a brief description of the problem examined. The section contains the formulation of the problem, short description of the nonlinear solver organization, rotation of the finite element mesh, torque and force calculation. This information is critical for proper application adjustment and getting desired results.

2.1. General description.

The magnetic field problems for an induction machine can be solved using a two-dimensional approximation, which is based on the assumption that the magnetic field does not depend on z-coordinate (*z*-axis being parallel to the axis of the rotor shaft). Thus, the magnetic field is solved in the plane of the machine's cross section (*x*-*y* plane). The current density and magnetic vector potential in two-dimensional problems only have the **z**-components and can be expressed as follows:

$$\nabla \times \left(\frac{1}{\mu_0 \mu_r} \cdot \nabla \times A \right) = J \tag{2.1}$$

$$J = \sigma \cdot \frac{U}{l} - \sigma \cdot \frac{\partial A}{\partial t} + \sigma \cdot v \times B\,, \tag{2.2}$$

where $\boldsymbol{A}$ – magnetic vector potential ($B = \nabla \times A$, $\boldsymbol{B}$ – magnetic flux density), μ_r – relative magnetic permeability, μ_0 – permeability of the free space, J – current density, σ – conductivity, U – voltage applied to the FE area, l – length in *z* direction, v – velocity of the conductor.

According to (2.2) the current density consists of three components, the first one is caused by external voltage, the second one is induced by varying the magnetic field, and the third one is caused by motional voltage $v \times B$.

Discretization of the problem over the plane of the machine's cross section using FEM results in the system of linear equations written in the matrix form as follows:

$$K \cdot X = F\,, \tag{2.3}$$

where K – stiffness matrix, X – vector of unknowns including values of magnetic vector potential in mesh nodes, rotor and stator currents and voltages , F – right side vector of the problem.

To solve the problem defined by (2.1) and (2.2) with a nonlinear B-H relationship the Gauss-Newton method is used. Assuming that we have a guess $\hat{X}$ of the solution and stiffness matrix $\hat{K}$ corresponding to the guess of the solution. Then the residual vector of the guess $\hat{X}$ will be defined as:

$$r = \hat{K} \cdot \hat{X} - F \tag{2.4}$$

The solving system (2.3) using Gauss-Newton iteration tends to be the minimization of the residual. The problem is said to be solved if the certain condition is satisfied. In MotorAnalysis this condition is defined by the maximum absolute value of the residual vector:

$$\max(|r|) < tol, \tag{2.5}$$

where *tol* –convergence tolerance setting the desired accuracy of the solution.

Generalized algorithm used in MotorAnalysis for a nonlinear case is shown in Figure 2.1.

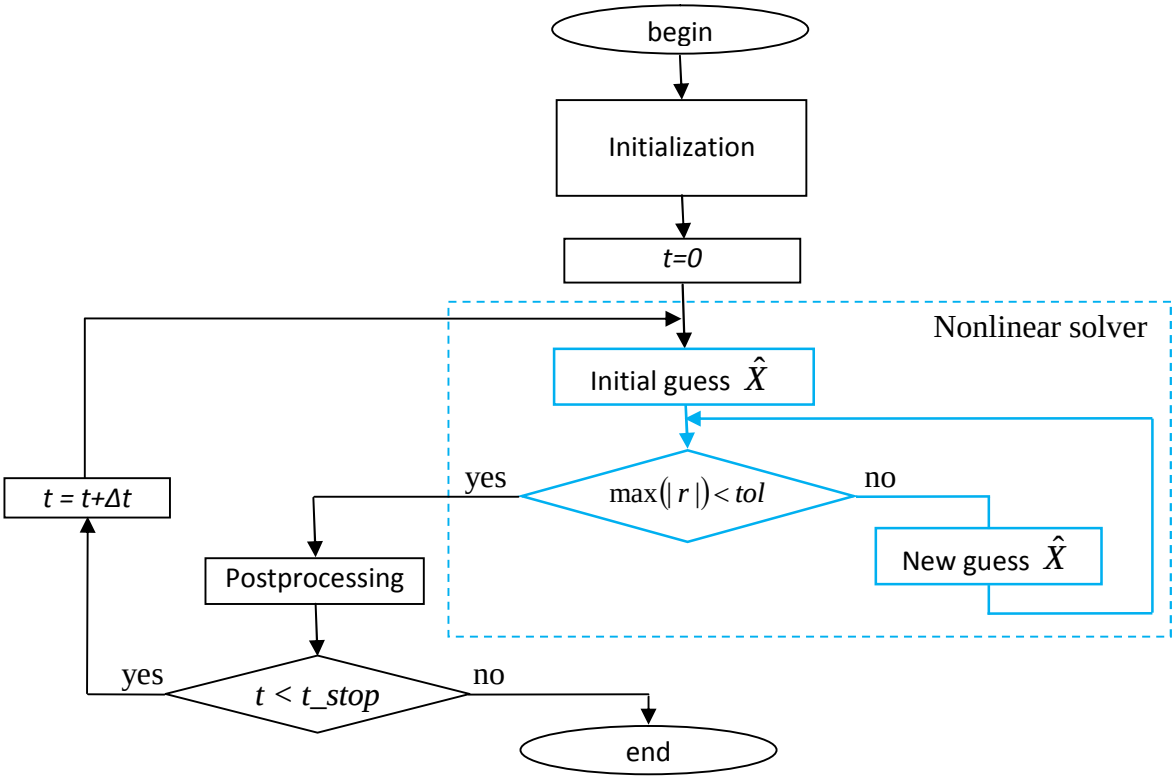


Figure 2.1. Generalized algorithm of solving a nonlinear problem.

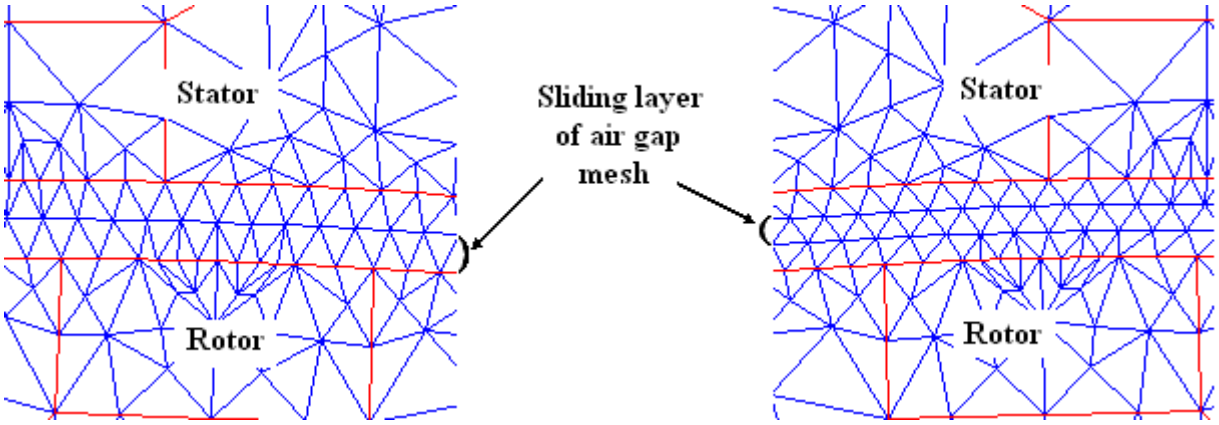
An outer loop of the algorithm represents integration over time with step Δt. The magnetic field problem is solved on the every integration step in the nonlinear solver loop (marked with blue color). Gauss-Newton iteration continues until the inequality (2.5) is satisfied.

2.2. Finite element mesh.

In MotorAnalysis the first-order triangular finite element mesh is used.

Every time when the rotor position is changed the mesh should be also changed. It is implemented by rotation of the rotor mesh connected to the fixed stator mesh via *sliding layer* located in the air gap. There are two strategies of sliding layer re-meshing applied in MotorAnalysis. First one produces a *regular mesh* in the sliding layer consisting of equal triangles, while the second one results in an *irregular mesh* consisting of triangles of arbitrary shape. The type of the mesh in the sliding layer

depends on the location of the sliding layer within the air gap mesh and total number of mesh layers in the air gap. The regular mesh usually gives a more accurate solution.



Two mesh layers in the air gap and the irregular mesh in the sliding layer (close to the rotor).

Three mesh layers in the air gap and the regular mesh in the sliding layer (at the center of the air gap)

Figure 2.2. Irregular and regular meshes in the sliding layer.

As it is seen from Figure 2.2 the regular mesh in the sliding layer is possible only when the mesh in the air gap contains more than three layers and the sliding layer does not adjoin the stator or rotor mesh. Most accurate simulation results are obtained when the sliding layer is located at the center of the air gap with an odd number (not less than three) of mesh layers. In this case the regular mesh in the sliding layer is guaranteed.

2.3. Calculation of electromagnetic torque and force.

There are two methods used in MotorAnalysis for calculation of the torque. These are the Maxwell stress tensor method and the virtual work method. For calculation of the radial force acting between the stator and rotor the virtual work method is used.

According to the Maxwell stress tensor method the electromagnetic torque can be expressed as the following:

$$T = -\frac{l \cdot \left(D_r + l_\delta\right)^2}{4\mu_0} \cdot \int_0^{2\pi} B_n B_t d\phi \quad , \tag{2.6}$$

where T – electromagnetic torque, l – lamination length in z direction, D_r – rotor diameter, l_δ – air gap length, μ_0 – permeability of the free space, B_n и B_t – normal and tangential components of the magnetic flux density in the air gap, as it is shown in Figure 2.3. This expression is used in MotorAnalysis for calculation of the torque with the Maxwell stress tensor method.

The integration contour used in (2.6) always lies inside the sliding layer of the air gap mesh, as it is shown in Figure 2.3.

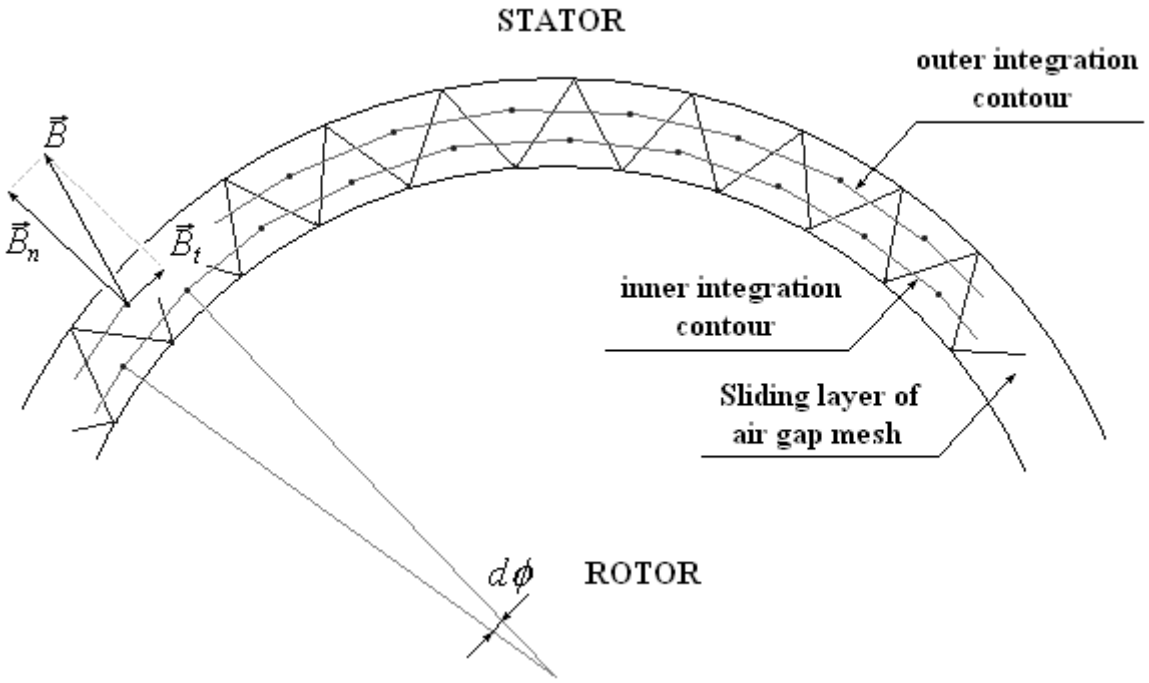


Figure 2.3. Calculation of the electromagnetic torque with the Maxwell stress tensor method.

Only the sliding layer of the air gap mesh is shown. Two integration contours are used - the inner contour passing through midpoints of the rotor-oriented triangles and the outer contour passing through midpoints of the stator-oriented triangles. The actual value of the torque is resulted as the average of two values calculated over the inner and outer integrated contours.

Calculation of the torque on the regular mesh in the sliding layer produces a more accurate result comparing with the irregular mesh. The most accurate torque value is obtained when the sliding layer is located at the center of the air gap. Calculation of the torque on the irregular mesh for the eccentric rotor is not recommended since the Maxwell stress tensor method in such cases gives highly incorrect results.

Matlab code for calculation of the electromagnetic torque with the Maxwell stress tensor method is presented in M- file MaxwellTorque.m.

According to the virtual work principle, the electromagnetic force is given by the derivation of the magnetic energy with respect to the virtual displacement with constant flux linkage:

$$F_u = -\left.\frac{\partial W}{\partial e}\right|_{\psi=const},$$

where ∂e - virtual displacement, ∂W - change of the magnetic energy between initial and final positions of the rotor.

Similarly, the expression for the torque calculation is defined as:

$$M = -\frac{\partial W}{\partial \phi}\bigg|_{\psi=const},$$

where $\partial \phi$ - virtual angular displacement.

When the virtual work principal is applied for the torque and force calculation, the accuracy significantly depends on the choice of the virtual displacement value. On the one hand the displacement should be small enough not to distort the finite element mesh. On the other hand, the round-off error arises when the displacement value is too small. The optimal value of the virtual displacement is not provided automatically and should be defined by a user in M-file MotorAnalysisSettings.m (refer to chapter 8 for more details). Some recommendations regarding this issue can be found in example 2 of chapter 5 of this manual.

It is considered that the virtual work method is more accurate comparing with the Maxwell stress tensor one since the first one implies the integration over the whole machine's cross section while using the Maxwell stress tensor method involves only finite elements of the air gap.

The calculated torque value is used to determine the current rotor position. The rotor rotation is defined as follows:

$$T = T_{load} + J\frac{d\omega}{dt}$$

$$\omega = \frac{d\varphi}{dt},$$

where T_{load} – load torque on the motor shaft, J – combined rotor and load moment of inertia, ω - rotor angular speed, φ – rotor angular position.

2.4. Multi-slice FEM.

As it was previously assumed the magnetic field of an induction motor does not depend on z-coordinate. In reality this assumption is not quite correct because of the end-winding leakage flux and rotor bars skewing. In 2D FEM the end-winding leakage flux can be taken into consideration with satisfactory accuracy using end-winding inductance. But the two-dimensional approximation is of no help when it comes to skewed geometries. Rotor bars skewing can be accurately simulated with 3D FEM. But these

simulations are usually long. As an alternative, multi-slice FEM is used in MotorAnalysis. Figure 2.4 illustrates the multi-slice approach.

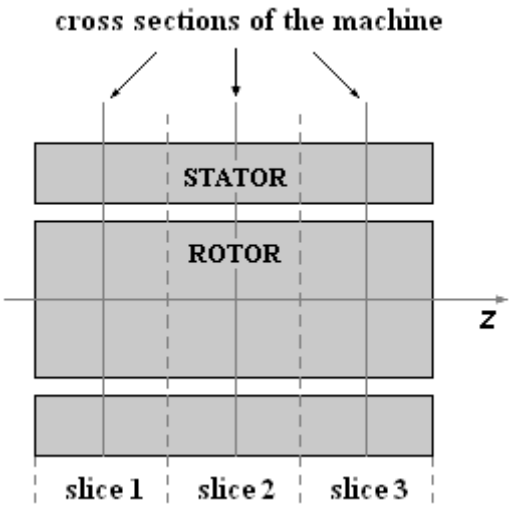


Figure 2.4. The multi-slice FEM principle.

The multi-slice FEM principle consists in dividing the machine along the z-coordinate into several slices. The cross section geometry changes from one slice to another according to applied rotor bar skew. Then the magnetic field problem is solved in every slice all at the same time with 2D FEM together with circuit equations of the stator and rotor windings. Electromagnetic torque is calculated for every slice and the actual torque is obtained as summation of all slice values.

Note that the calculation time is proportional to the number of slices. Set the number of slices to one to get the 2D FEM simulation. If there is no rotor bar skew, then using several slices will not be reasonable, since the result will be the same as with one slice used.

3. GETTING STARTED

3.1. First run of MotorAnalysis.

MotorAnalysis is a Matlab-application and to use it you should have Matlab installed on your computer. Matlab GUI objects are created with Matlab R2009b - the application may not work with earlier versions of Matlab. Matlab PDE Toolbox is required.

MotorAnalysis main window is shown in Figure 3.1.

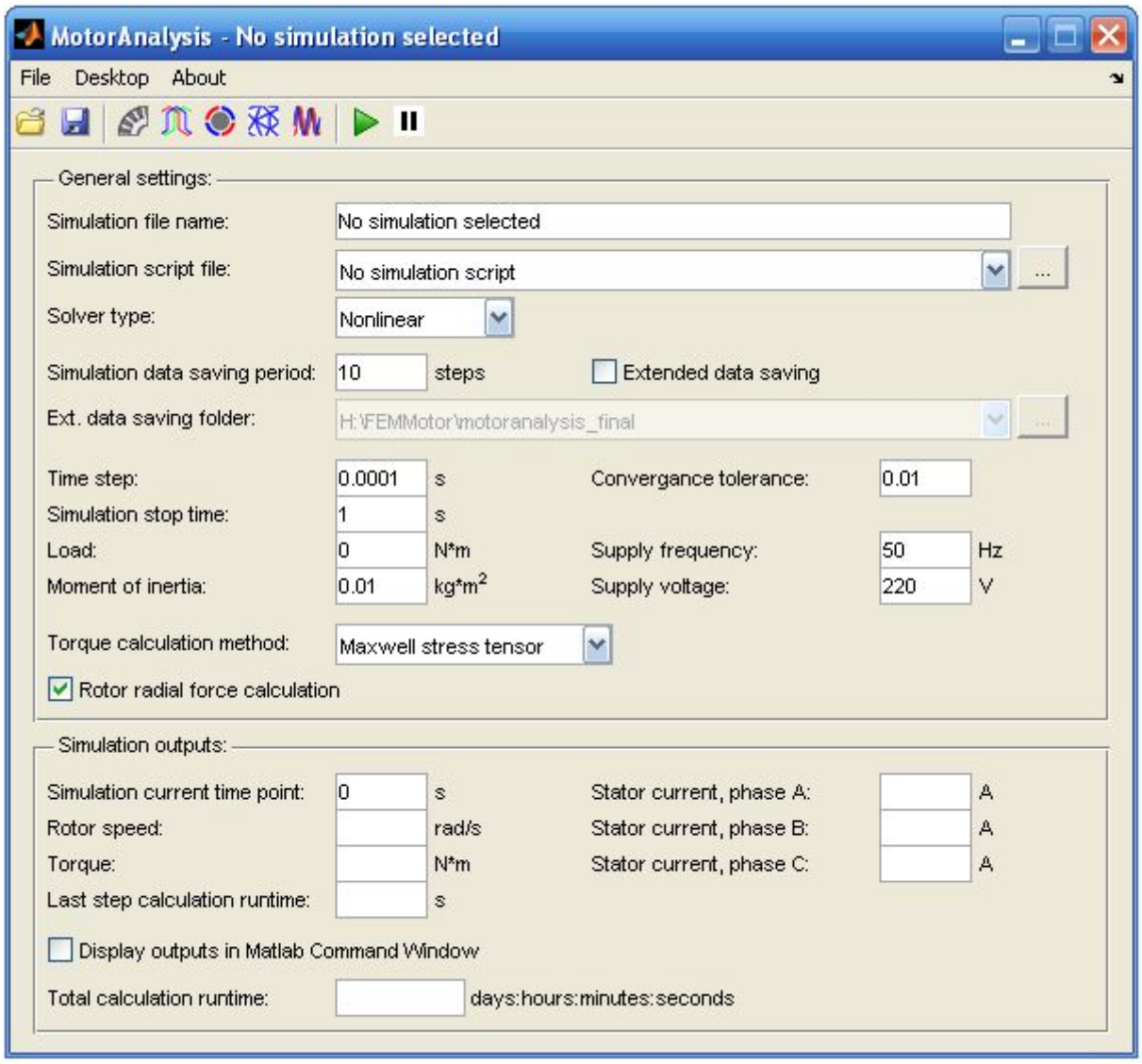


Figure 3.1. MotorAnalysis main window.

Other windows (Geometry Editor, Iron Core Property Editor, Windings Property Editor, Mesh Editor and Plot Wizard) are available from the menu Desktop or using toolbar buttons.

3.2. MotorAnalysis general settings.

General settings of the simulation presented in MotorAnalysis main window include the following parameters:

Simulation file name – name of the opened simulation file.

Simulation script file – Matlab-function which is called on each simulation time step and allows the user to change all general simulation settings (except the time step), to calculate and store user defined variables, to control power supply sources, to implement user's motor control algorithms. Refer to chapter 5 for more details on using simulation script files.

Solver type – several solver types are available including **Nonlinear** solver, **Linear** solver using fixed values of the magnetic permeability and **Fast** solver which is a simplified linear solver (motion voltage component $\nu \times B$ of (2.2) is not taken into account).

Simulation data saving period – the number of time steps in which the simulation data are being rewritten on the hard drive. It allows you not to lose data when the simulation is incorrectly stopped.

Extended data saving – enables storing the magnetic potential, current density and permeability values for each time step. These values are not stored by default because of large amount of hard drive space required. Data for each time step are stored in a separate file so the number of files stored is equal to the number of calculated time steps.

Ext. data saving folder – the folder where files mentioned above are stored in, when **Extended data saving** checkbox is active.

Time step – simulation time step size in seconds. The time step should be defined before the simulation is started and cannot be changed during the simulation.

Simulation stop time – last point of the simulated time period.

Load – load torque on the motor shaft.

Moment of inertia – combined rotor and load moment of inertia.

Convergence tolerance – nonlinear simulation accuracy defined by (2.5).

Supply frequency – power supply frequency by default. The motor is fed by a three-phase sinusoidal symmetrical voltage unless otherwise specified in the simulation script file.

Supply voltage – power supply voltage by default.

Torque calculation method – two electromagnetic torque calculation methods are available: **Maxwell stress tensor** and **Virtual work**. Refer to section 2.3 for more details.

Rotor radial force calculation – enables calculation of the radial force acting between the stator and rotor.

3.3. Working with simulation files.

All motor parameters, material properties, simulation settings, finite element mesh and simulation results are stored in a *simulation file*. The simulation file is a Matlab data-file with .mat extension.

Standard file commands of opening and saving simulation files are available from the **File** menu or using toolbar buttons.

Motor parameters, material properties, finite element mesh, electrical circuit and simulation time step should be defined before the simulation is started and cannot be changed during the simulation.

Open As a New Simulation item of the **File** menu allows creating of a new simulation file from another simulation file. Simulation results and finite element mesh are not included in the new simulation file when using **Open As a New Simulation** command. Motor parameters, material properties, electrical circuit and simulation time step for the new simulation file can be changed before the simulation is started.

3.4. Starting and stopping the simulation.

To run the current simulation, click the **Run Simulation** button (green triangle) on the MotorAnalysis main window toolbar. The first run of the simulation begins at time zero. Every subsequent run of the simulation begins at the time where it was previously stopped. The simulation continues until an error occurs, until you stop the simulation or until simulation reaches the simulation stop time. The first run of the simulation starts with initialization which may take a few minutes. To stop the current simulation click the **Stop Simulation** button which is to the right of the **Run Simulation** button**.** The simulation will be stopped when the calculation of the current time step is completed. It is not recommended to interrupt the simulation using Ctrl+C shortcut otherwise the simulation file may be corrupted and all simulation data will be lost.

4. MOTOR PROTOTYPING AND MESH GENERATION

4.1. Geometry Editor.

Geometry Editor window allows you to set cross section dimensions of the motor as well as lamination length and rotor bar skew.

Geometry Editor window is shown in Figure 4.1:

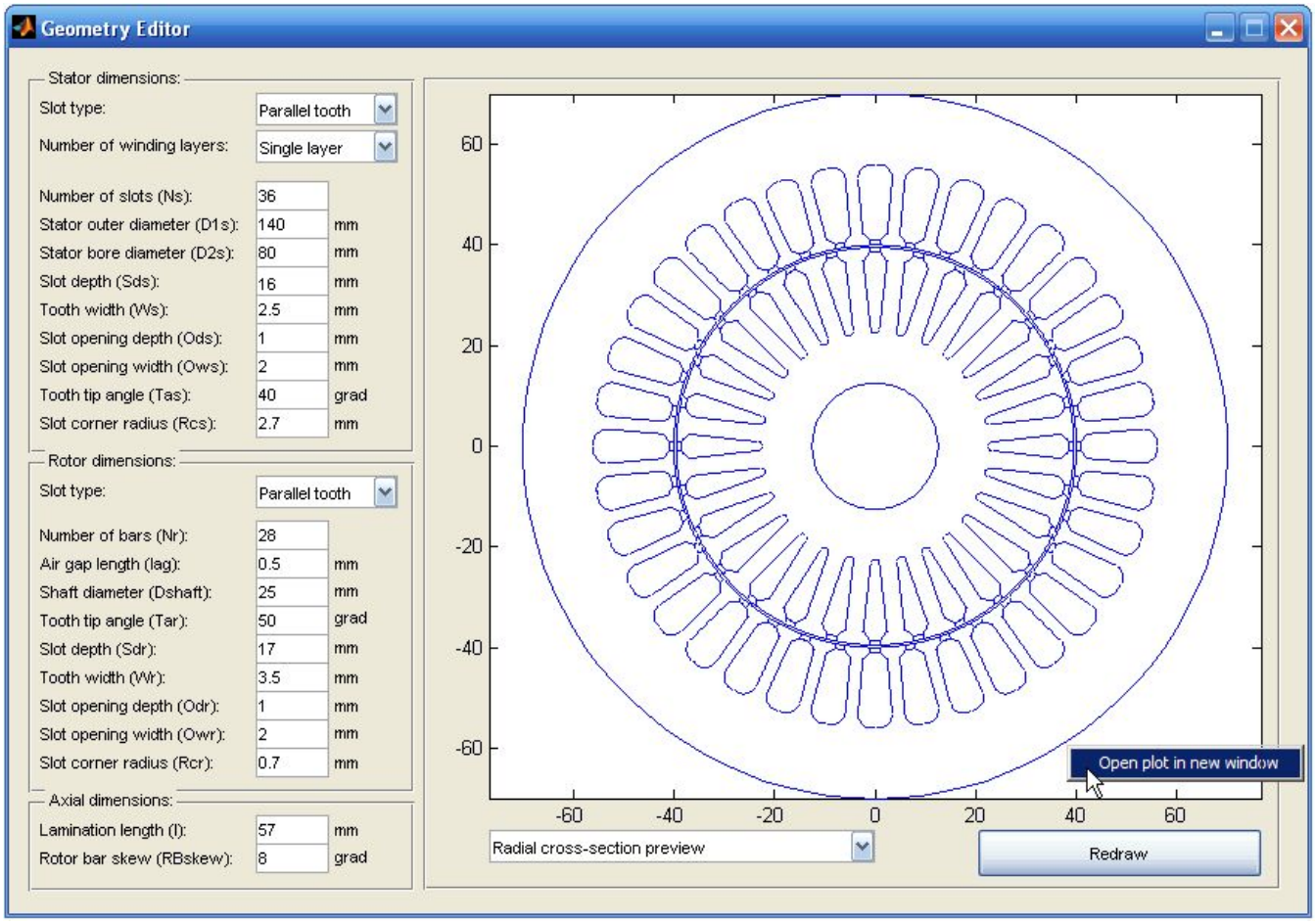


Figure 4.1. Geometry Editor window.

Slot type pop-up menus allow you to choose rotor and stator slot types. There are two stator slot types available: with **parallel tooth** and with **parallel slot** and three rotor slot types: with **parallel tooth**, with **parallel slot** and **round** slot. User defined slot types are not supported in this version and will be available in future versions of MotorAnalysis.

Number of winding layers pop-up menu allows you to choose either the **single layer** or the **double layer** stator winding. If the double layer stator winding is chosen, the stator slot will be divided horizontally into two equal parts.

Dimensions and geometry parameters of the motor presented in the Geometry Editor window are shown in Figures 4.2 - 4.6.

The cross section of the motor is displayed in the right part of the Geometry Editor window. To examine a stator or rotor slot on a large scale choose **Stator slot preview** or **Rotor slot preview**, respectively, from the pop-up menu below. When the user right-clicks the picture and selects **Open plot in new window** (as it is shown in Figure 4.1), the picture will be opened in a new window with Matlab figure tools.

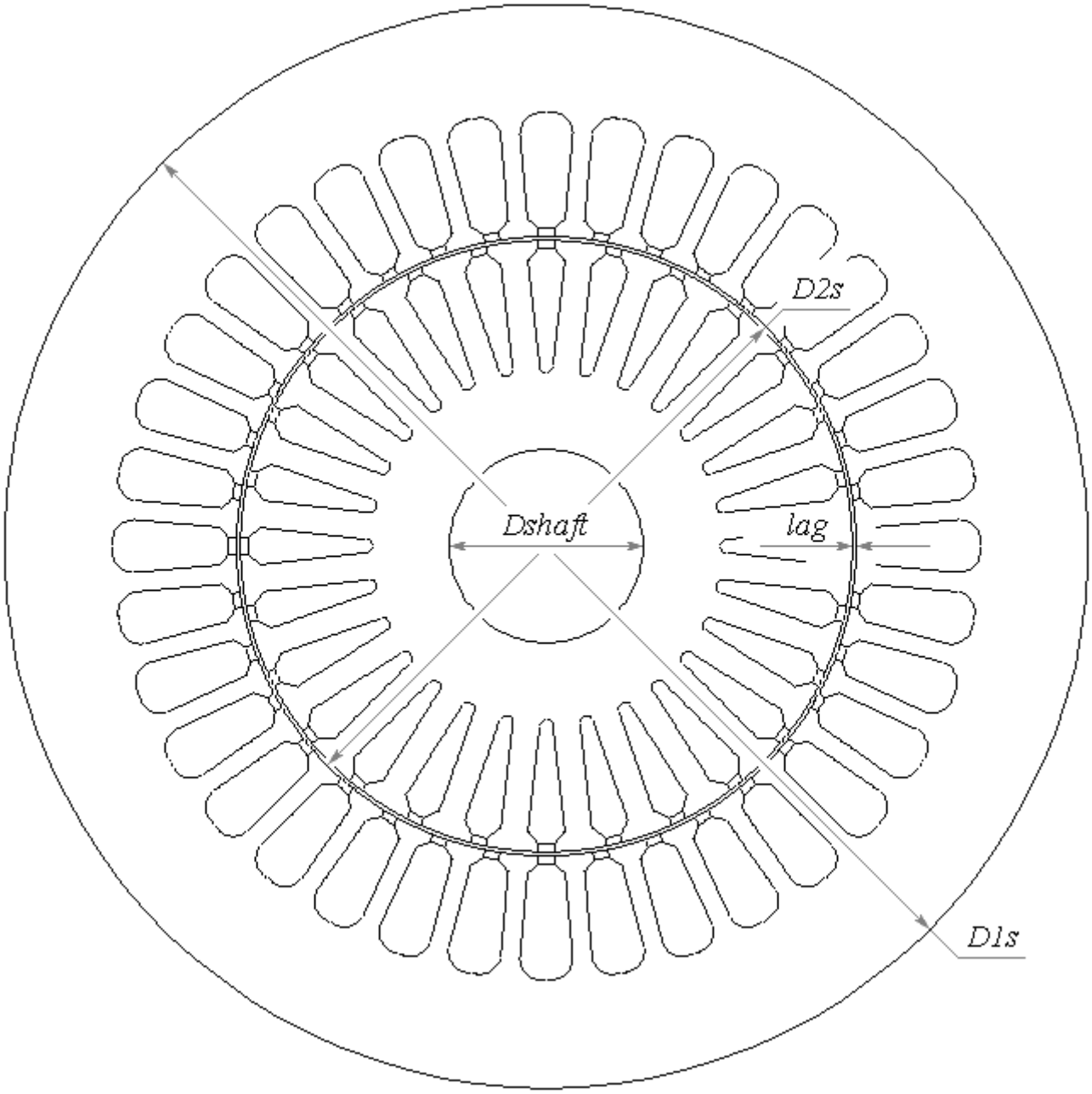


Figure 4.2. Cross section dimensions of the motor.

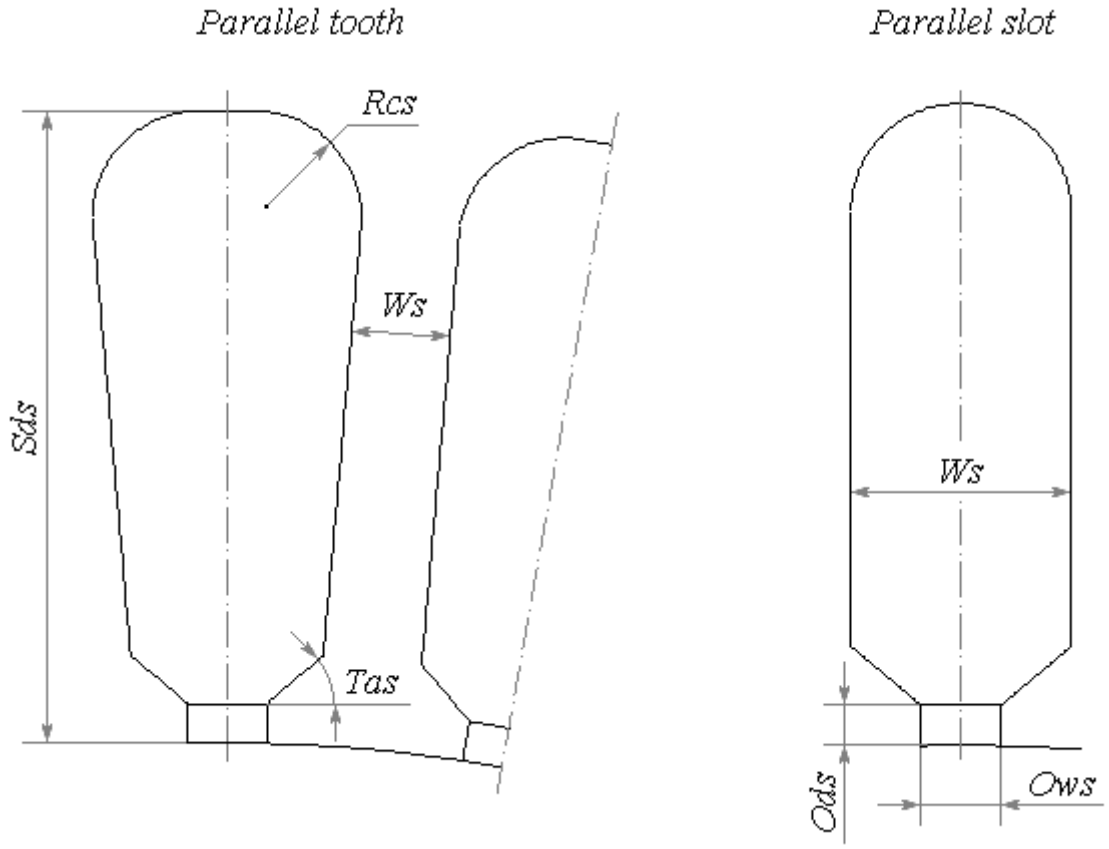


Figure 4.3. Dimensions of a stator slot.

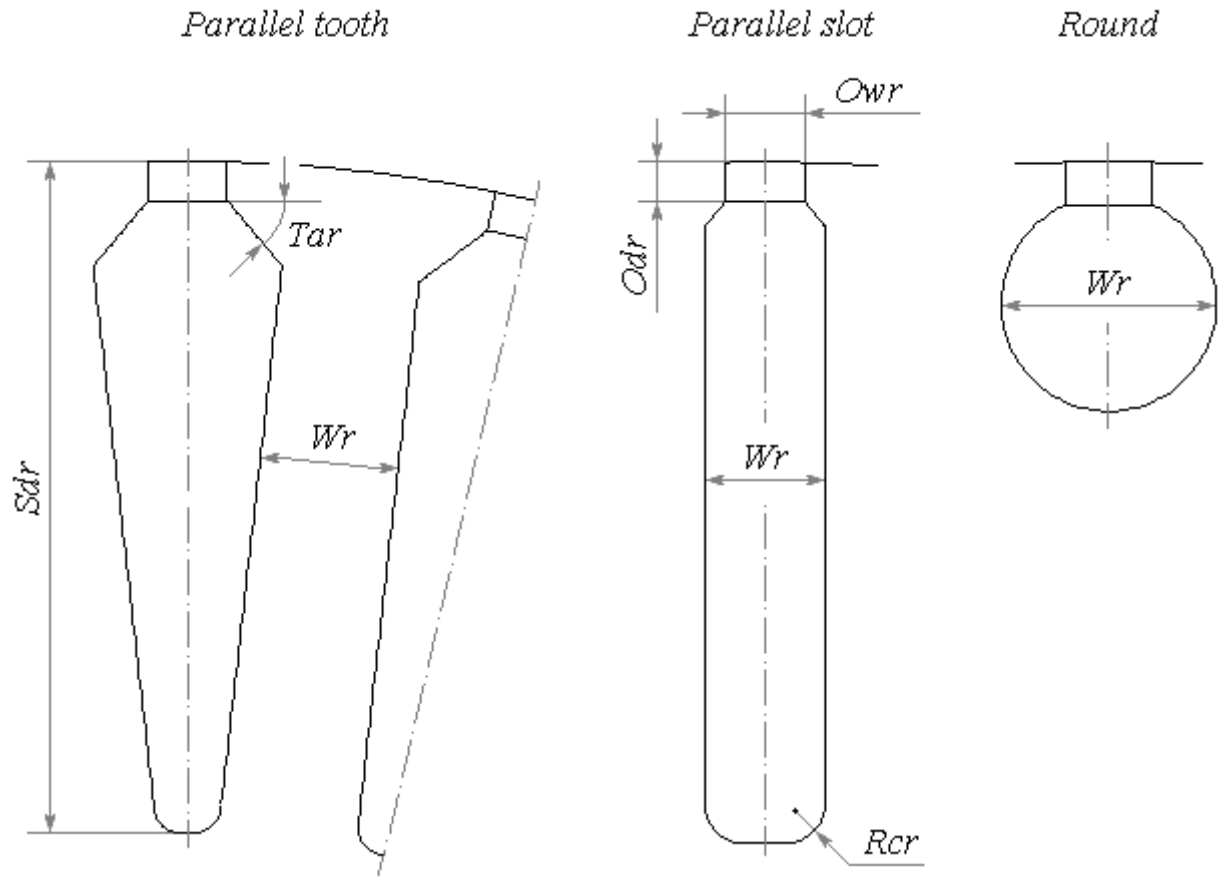


Figure 4.4. Dimensions of a rotor slot.

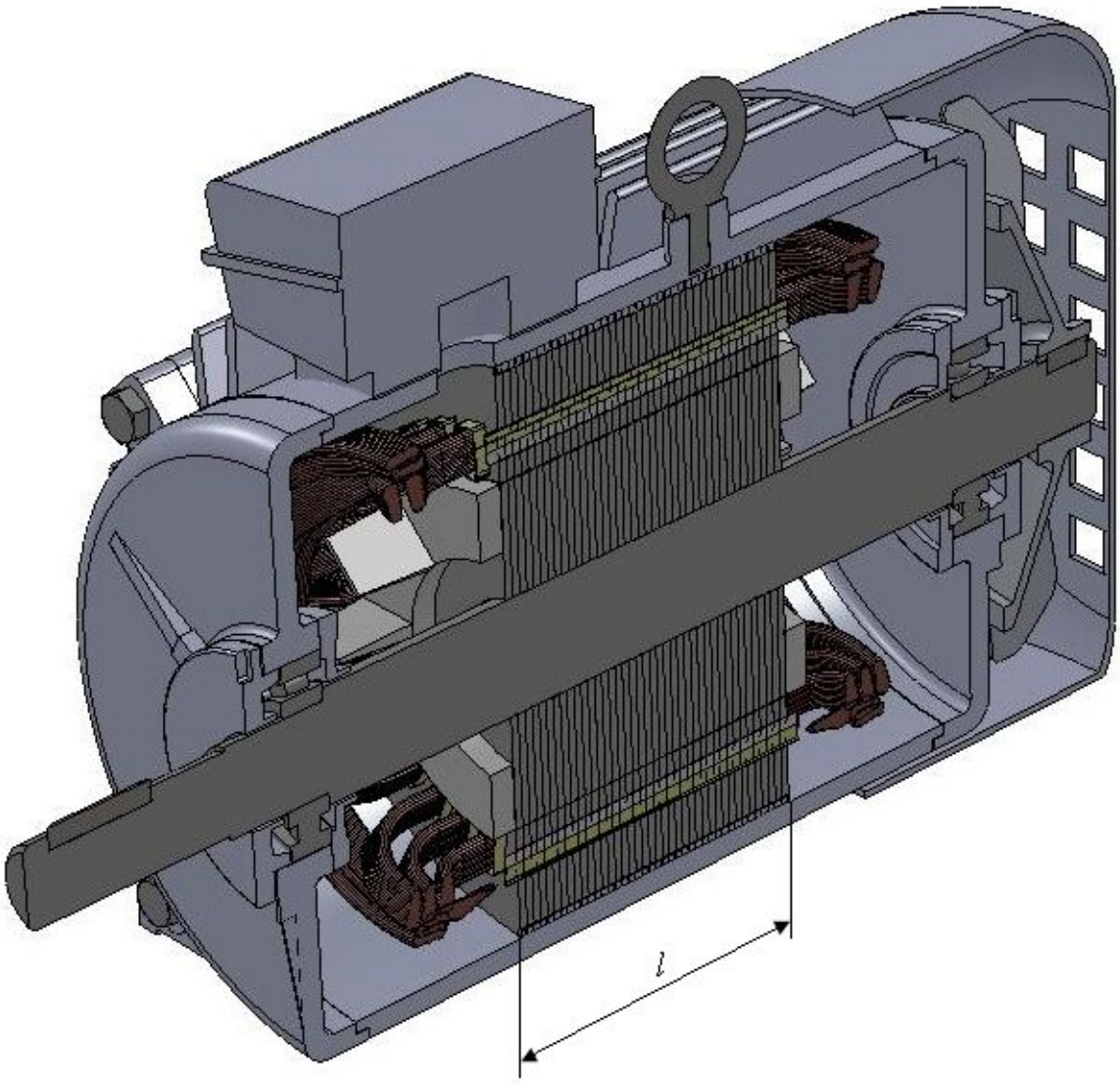


Figure 4.5. Lamination length.

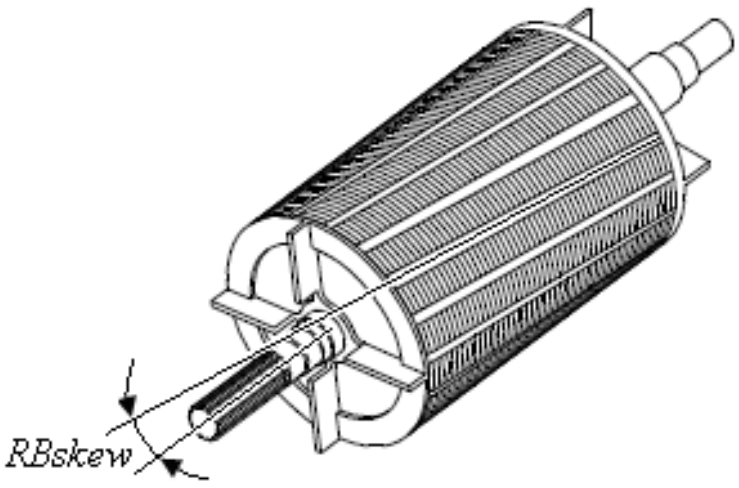


Figure 4.6. Rotor bar skew.

4.2. Windings Property Editor.

Windings Property Editor window is shown in Figure 4.7:

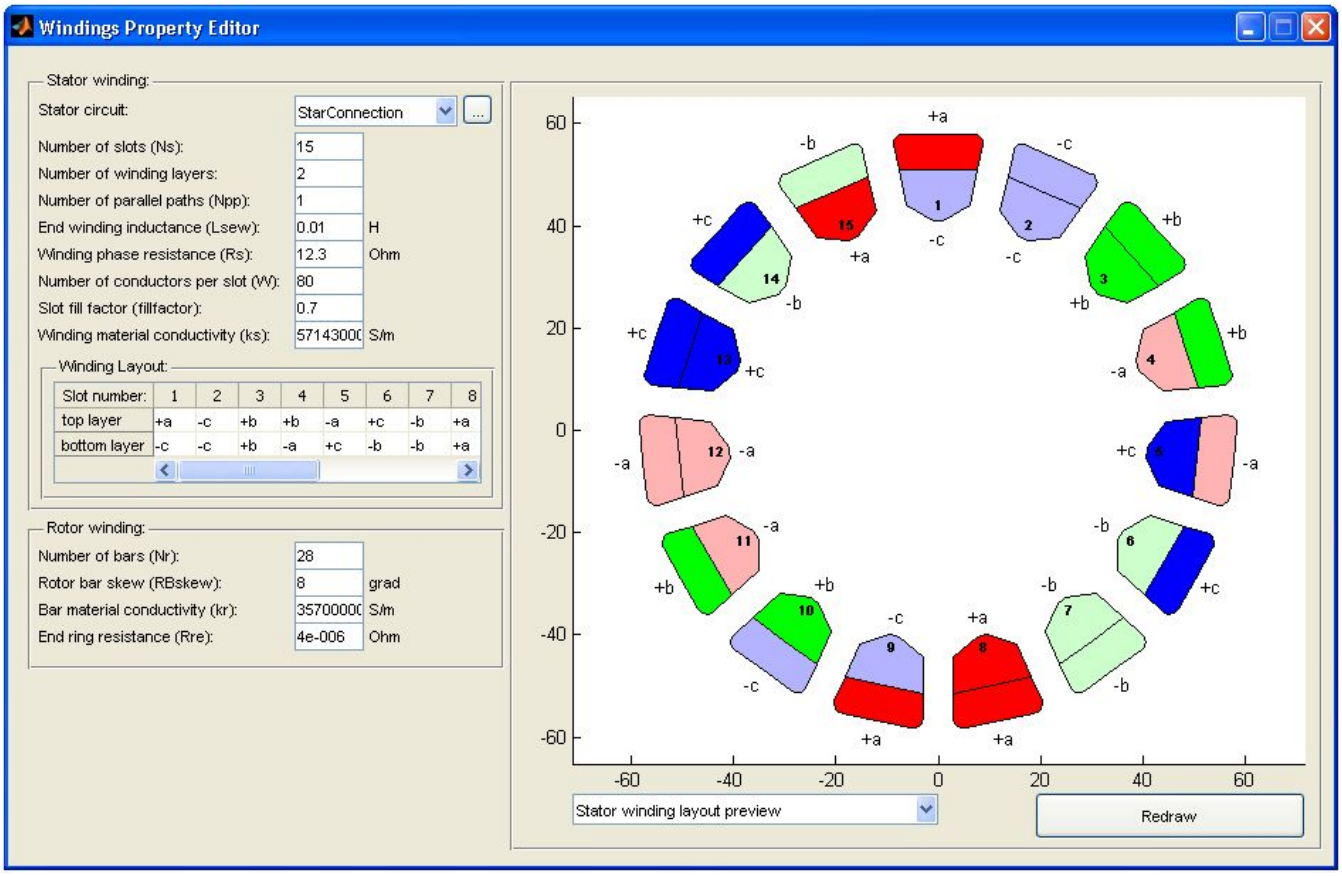


Figure 4.7. Windings Property Editor window.

Stator winding is specified by the following fields:

Stator circuit pop-up menu specifies the coupled electrical circuit which is solved simultaneously with magnetic field. There are two default circuits used in MotorAnalysis: circuit with the star connected stator winding and circuit with the delta connected stator winding corresponding to **StarConnection** and **DeltaConnection** selected, respectively. You can also specify your own electrical circuit as it is discussed in chapter 6 of this manual.

Number of slots and **Number of winding layers** fields display values of the corresponding fields of the Geometry Editor and can be edited only in Geometry Editor window.

Number of parallel paths field specifies the number of groups of coils per phase connected in parallel.

End winding inductance field specifies the leakage inductance of the stator winding end-turns per phase. End winding inductance will be calculated automatically depending on the winding configuration and motor dimensions in future versions of MotorAnalysis.

Winding phase resistance field specifies active resistance of the stator winding per phase. This value will be calculated automatically depending on the winding configuration and motor dimensions in future versions of MotorAnalysis.

Number of conductors per slot field specifies the total number of conductors placed in one stator slot. Note that this value should be a multiple of two for a double-layer winding.

Slot fill factor field specifies the ratio of the area of all conductors of a slot to the total slot area. Filling this field is optional and required only for calculation of the Joule loss distribution over a stator slots.

Winding material conductivity field is also optional and required for calculation of the Joule loss distribution over stator slots.

Winding layout table specifies stator winding configuration. Number of rows corresponds to the number of stator winding layers and the number of columns corresponds to the number of stator slots. Examples of **Winding layout** table values: +a, -c2, where the sign "+/-" specifies the forward and return direction of the conductors within a slot, respectively, the letter following the sign specifies a phase and the phase is followed by the number of the group of coils; so the same number of cell values corresponds to the coils connected in series. If there are no parallel paths, then the number of the group of coils (1 in this case) can be omitted. Example of the **Winding layout** table values corresponding to Figure 4.7:

Slot number:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
top layer	+a	-c	+b	+b	-a	+c	-b	+a	+a	-c	+b	-a	+c	+c	-b
bottom layer	-c	-c	+b	-a	+c	-b	-b	+a	-c	+b	-a	-a	+c	-b	+a

Colored representation of the **Winding layout** table is displayed in the right part of the **Windings Property Editor** window.

Rotor winding is specified by the following fields:

Number of bars and **Rotor bar skew** fields display values of corresponding fields of the **Geometry Editor** and can be edited only in **Geometry Editor** window.

Bar material conductivity field is used to calculate resistance of a rotor bar.

End ring resistance field specifies an active resistance of the arch of the rotor end ring measured between two adjacent rotor bars. End ring resistance will be calculated automatically in future versions of MotorAnalysis.

4.3. Iron Core Property Editor.

Iron Core Property Editor window is shown in Figure 4.8.

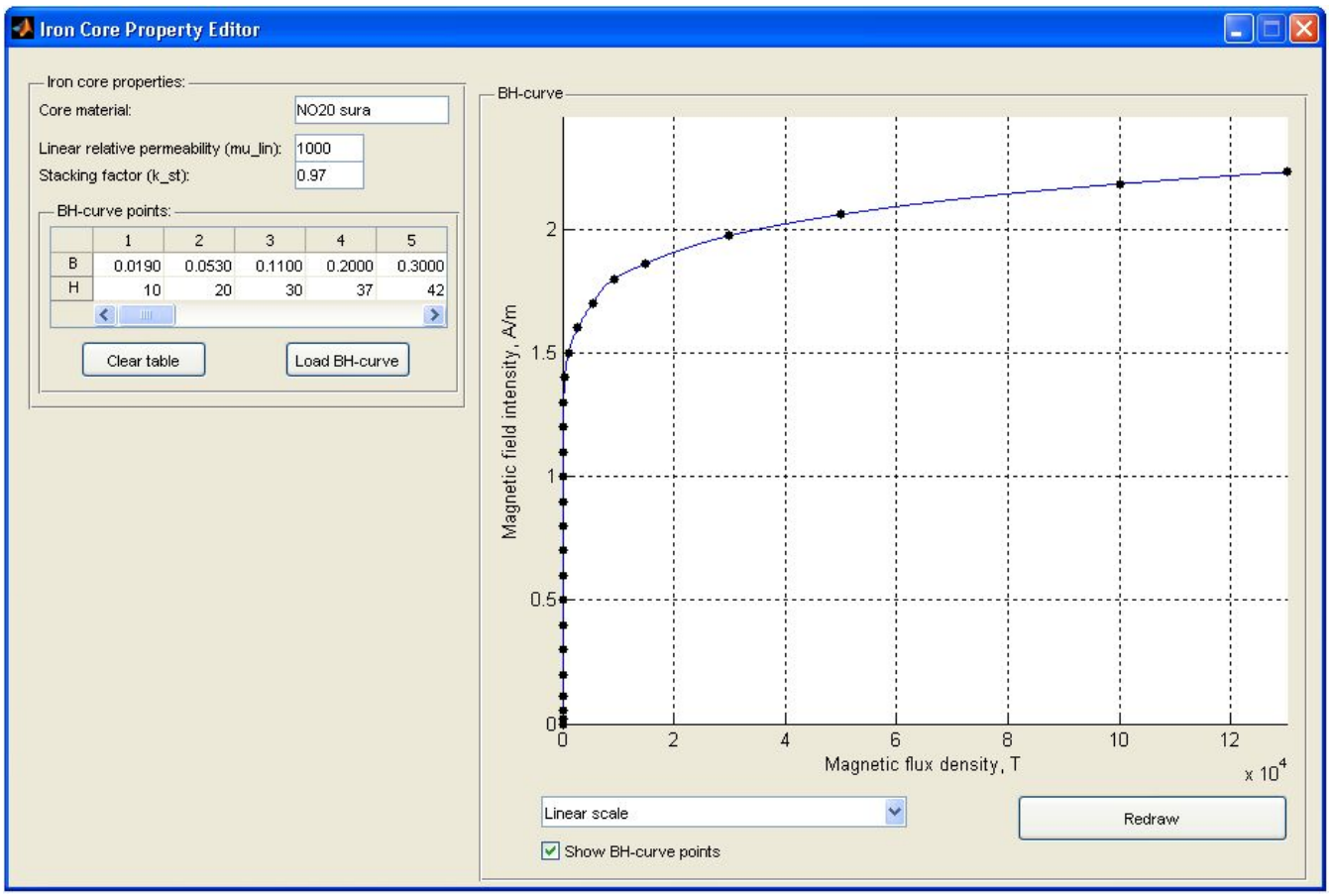


Figure 4.8. Iron Core Property Editor window.

Core material field is optional and can be left empty.

Linear relative permeability field specifies the relative permeability used by linear and fast solvers.

Stacking factor field specifies the ratio of the volume filled by electrical steel to the total volume of the iron core. The total volume of the iron core consists of the volume of lamination steel sheets and the volume of the coating between the sheets. Stacking factor reduces the flux carrying by the iron core which is taken into account by MotorAnalysis.

BH-curve points table defines the measured points of the BH-curve. **Load BH-curve** button allows you to load BH-curve from another simulation file; **Clear table** button cleans up all table cells.

BH-curve plot interpolated with cubic splines is presented in the right part of the **Iron Core Property Editor** window. The pop-up menu below allows you to choose linear or logarithmic scale for the BH-curve plot. When the user right-clicks the plot and selects **Open plot in new window**, the BH-curve plot will be opened in a new window with Matlab figure tools.

4.4. Mesh Editor.

Mesh Editor window is shown in Figure 4.8.

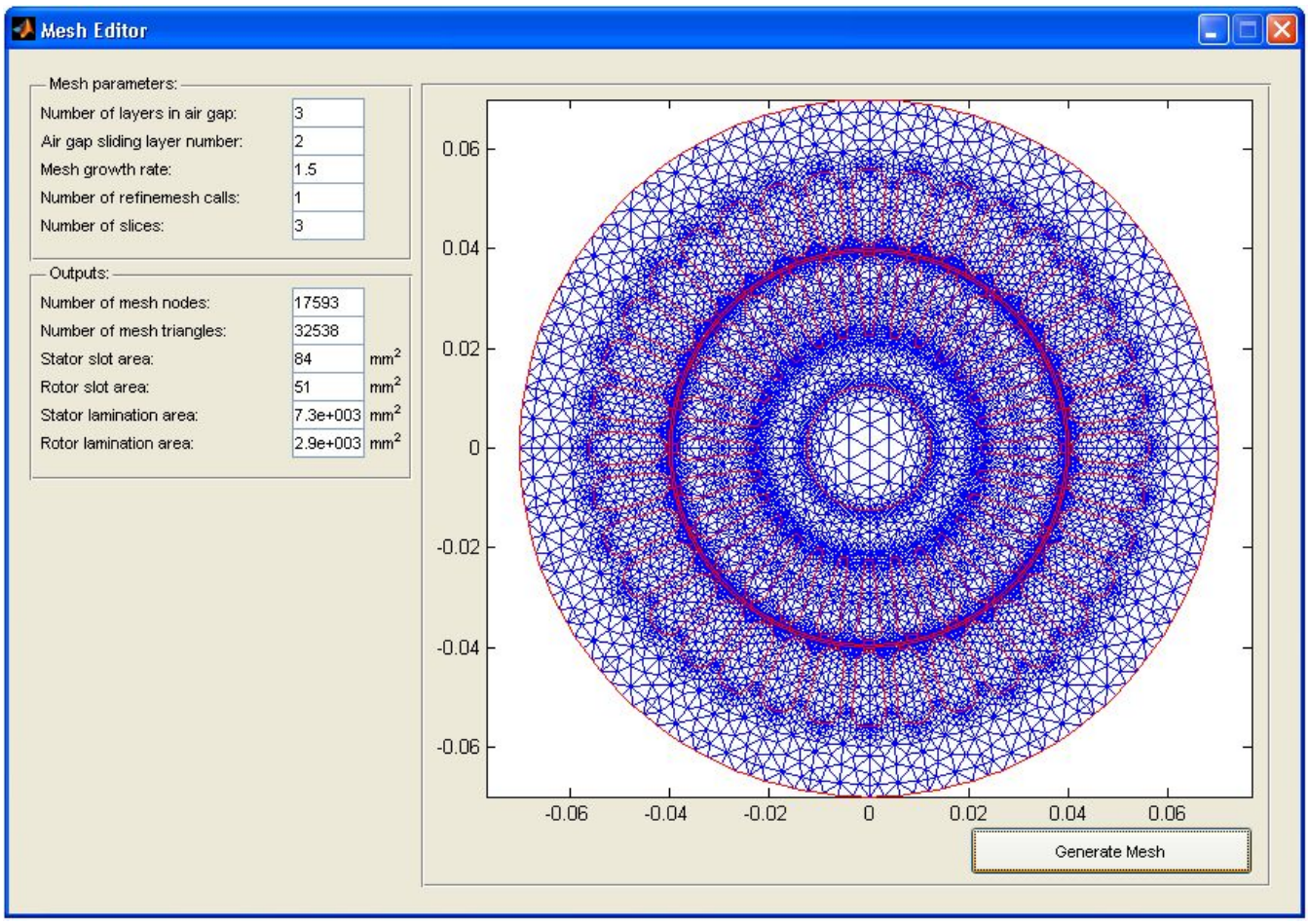


Figure 4.9. Iron Core Property Editor window.

Number of layers in air gap field specifies the total number of finite element layers placed in the air gap. For more details on choosing a number of layers in the air gap refer to section 2.2 of this manual.

Air gap sliding layer number field specifies the location of the sliding layer within the air gap with respect to the stator. For more details on choosing this parameter refer to section 2.2 of this manual.

Mesh growth rate field specifies the mesh growth rate corresponding to the property `Hgrad` used by function `initmesh` of the Matlab PDE Toolbox. Function `initmesh` is used in MotorAnalysis to create an initial mesh. If this field is left empty, its value will be specified automatically when the user clicks **Generate mesh** button.

Number of refinemesh calls field specifies the number of calls of function `refinemesh`. This function refines the initial mesh increasing a number of finite elements. If this field is left empty, its value will be specified automatically when the user clicks **Generate mesh** button.

Number of slices field specifies the number of slices used by multi-slice FEM, i.e. the number of the machine's cross section in which the magnetic field is simultaneously calculated. For more details on multi-slice FEM refer to section 2.4 of this manual.

To generate the mesh, click **Generate mesh** button. It is recommended to examine the mesh before the simulation is started. Right-click the mesh and select **Open plot in new window**, the mesh will be opened in a new window with Matlab figure tools for detailed examination. The accuracy of FEM significantly depends on the shape of finite elements especially within and close to the air gap area. The shape of finite elements should be as close as possible to the equilateral triangle shape. If the shape of finite elements is significantly distorted, try to regenerate the mesh changing mesh parameters.

If **Error while trying to generate mesh** message occurs, make sure that all mesh parameters are correct. Incorrectly defined motor geometry can also cause mesh generation errors – make sure that all motor dimensions are consistent. Sometimes incorrect motor dimensions can only be seen on the enlarged scale as it is shown in Figure 4.10. In this case a slot corner radius is inconsistent with a slot width.

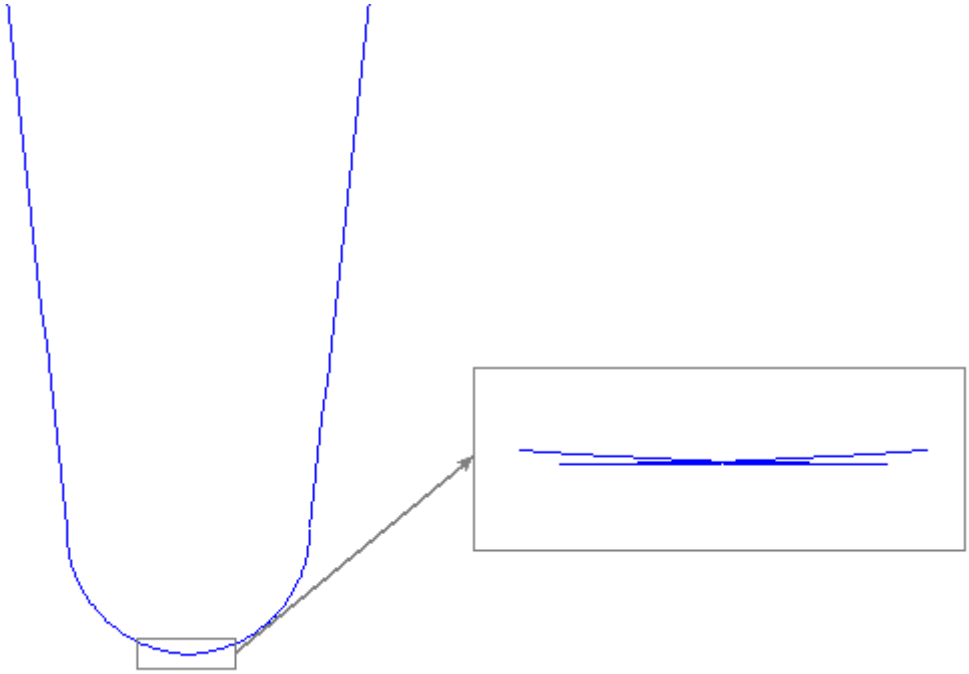


Figure 4.10. Example of incorrect motor dimensions.

Number of mesh nodes and **Number of mesh triangles** fields are calculated for one slice. To calculate total number of mesh nodes and total number of mesh triangles one should multiply values specified in **Number of mesh nodes** and **Number of mesh triangles** fields by the number of slices.

5. SIMULATION SCRIPT FUNCTIONS

A simulation script is a file with .m extension containing Matlab-function of a specific format. This function is called on each simulation time step and allows the user to change all simulation settings (except the time step), to calculate and store user defined variables, to control power supply sources, to implement motor control algorithms. For example, a simulation script function can perform motor equivalent circuit parameterization, PWM of supply voltage waveform when a motor is fed by a power converter and etc.

To understand this chapter, some familiarity with Matlab coding is required. Refer to Matlab help documentation for more details on Matlab coding principles.

5.1. General information.

The chosen simulation script file is displayed in the **Simulation script file** field of the MotorAnalysis main window. If the simulation script file is not chosen, **No simulation script** item will appear. To choose or change a simulation script file, click the button to the right of the **Simulation script file** field.

If the simulation script file is used all simulation settings displayed in the MotorAnalysis main window can be overlaid with those specified in the simulation script function. If supply voltage waveform is not specified in the simulation script function, the default voltage applied to the motor will be as follows:

$$\begin{aligned} u_a &= U \cdot \sqrt{2} \cdot \sin\left(2\pi \cdot f \cdot t\right) \\ u_b &= U \cdot \sqrt{2} \cdot \sin\left(2\pi \cdot f \cdot t + 2\pi/3\right) \\ u_c &= U \cdot \sqrt{2} \cdot \sin\left(2\pi \cdot f \cdot t + 4\pi/3\right) \end{aligned} \tag{5.1}$$

where U – value specified in the **Supply voltage** field, f - value specified in the **Supply frequency** field, t - value displayed in the **Simulation current time point** field of the MotorAnalysis main window.

5.2. Writing a simulation script function.

The definition of simulation script function `simscript` and a minimum amount of code appear in M-file simscript.m as follows:

```
function [ShaftPosition,PowerSupply,Settings,Enforce,Userdata] = ...
simscript (Outputs,Settings,Userdata,Circuit,Geometry,Mesh,...
            Windings,Core,A,cell_jr,cell_p,cell_t,cell_Nu,path)
ShaftPosition=[];
Enforce=[];
PowerSupply=[];
```

The first line of the function always starts with the keyword `function`. The names of the M-file and of the function should be the same. Refer to Matlab help documentation for more details on writing a Matlab-function.

The function's output arguments:

`ShaftPosition` – rotor shaft position; 1×2 array, containing rotor shaft coordinates`[x; y]` in meters. This variable allows you to apply static or dynamic eccentricity to the rotor. If array `ShaftPosition = []` or `ShaftPosition = [0; 0]`, there is no rotor eccentricity.

`PowerSupply` – structure containing current and voltage values which are supplied by all used current and voltage courses. If `PowerSupply = []` default voltage sources defined by (5.1) are used.

`Settings` – structure of simulation settings, the same as input argument `Settings`.

`Enforce` – structure reserved to be used in future versions of MotorAnalysis. Use `Enforce=[]` statement within a simulation script function to avoid errors.

`Userdata` – structure to store user data, the same as input argument `Userdata`. By default `Userdata` is empty.

The function's input arguments:

`Outputs` – structure containing simulation results.

`Settings` – structure of simulation settings.

`Userdata` – structure to store user data.

`Circuit` – structure containing stator electrical circuit parameters.

`Geometry` – structure containing the machine's dimensions data.

`Mesh` – structure containing mesh data.

`Windings` – structure containing the machine's windings data.

`Core` – structure containing the machine's iron core data.

`A` – array of magnetic vector potential node values at the current time step; `A[1:np]` – first slice values, `A[np+1:2*np]` – second slice values, `A[2*np+1:3*np]` – third slice values and etc., where `np` – number of mesh nodes in one slice.

`cell_jr` – cell-array containing current density midpoint values in the rotor bar finite elements for every slice.

`cell_p` – cell-array containing *x* and *y* coordinates of finite element mesh nodes for every slice, as it is described in Matlab PDE Toolbox help documentation (see help on `initmesh` function for more details).

`cell_t` – cell-array containing finite elements data for every slice, as it is described in Matlab PDE Toolbox help documentation (see help on `initmesh` function for more details).

`cell_Nu` – cell-array containing magnetic reluctivity midpoint values for every slice. Magnetic reluctivity is $\nu = \frac{1}{\mu_0 \mu_r}$, where μ_0 – permeability of the free space, μ_r - relative permeability.

`path` – directory with application files.

5.3. Main data structures.

`Outputs` structure fields:

Field	Size, type	Description
`Outputs.CurrentTime`	1×1, double	Simulation current time point.
`Outputs.Ua`	1×n, double	Instantaneous values of line to ground voltage; n – number of calculated time steps.
`Outputs.Ub`		
`Outputs.Uc`		
`Outputs.Isa`	Npp×n, double	Current flowing in each parallel path of each phase; number of array rows corresponds to the number of parallel paths. Npp – number of parallel paths.
`Outputs.Isb`		
`Outputs.Isc`		
`Outputs.Ir`	Nr×n, double	Rotor bar currents; number of array rows corresponds to the number of rotor bars. Nr – number of rotor bars.
`Outputs.time`	1×n, double	Time points.
`Outputs.Torque`	1×n, double	Electromagnetic torque; equals to one of the variables `Torque_maxwell` or `Torque_vwork` depending on the selected torque calculation method.
`Outputs.Load`	1×n, double	Load torque on the motor shaft.
`Outputs.Speed`	1×n, double	Rotor angular speed.
`Outputs.Rotang`	1×n, double	Rotor angular position.
`Outputs.Pinput`	1×n, double	Input apparent power delivered by all current and voltage sources.
`Outputs.Pcons`	1×n, double	Consumed apparent power.
`Outputs.Prb`	1×n, double	Resistive losses dissipated on the rotor bars.
`Outputs.Pre`	1×n, double	Resistive losses dissipated on the rotor end rings.
`Outputs.Ps`	1×n, double	Apparent power (real and reactive) consumed by the stator electrical circuit.
`Outputs.Pmf`	1×n, double	Magnetic energy time derivative.

`Outputs.Pmech`	1×n, double	Mechanical power on the shaft.
`Outputs.Torque_maxwell`	1×n, double	Electromagnetic torque calculated with the Maxwell stress tensor method.
`Outputs.Torque_vwork`	1×n, double	Electromagnetic torque calculated with the Virtual work method.
`Outputs.Fx`	1×n, double	Radial electromagnetic force acting between the stator and rotor along *x*-direction.
`Outputs.Fy`	1×n, double	Radial electromagnetic force acting between the stator and rotor along *y*-direction.

`Settings` structure fields:

Field	Value	Description
`Settings.script`	`'No simulation script'`	Simulation script file; corresponding to the **Simulation script file** field of the main window.
	Directory and/or file name	
`Settings.solvertype`	`'Nonlinear'`	Solver type; corresponding to the **Solver type** field of the main window.
	`'Linear'`	
	`'Fast'`	
`Settings.saveperiod`	Number > 0	Number of time steps in which the simulation data are being rewritten on the hard drive; corresponding to the **Simulation data saving period** field of the main window.
`Settings.extsave`	`0`	Enables the extended data saving mode; corresponding to the **Extended data saving** field of the main window.
	`1`	
`Settings.extsavefolder`	Directory	Folder to store data when the extended data saving mode is enabled; corresponding to the **Ext. data saving folder** field of the main window.
`Settings.timestep`	Number > 0	Simulation time step size; corresponding to the **Time step** field of the main window. This variable cannot be changed during the simulation.

`Settings.stoptime`	Number > 0	Simulation stop time; corresponding to the **Simulation stop time** field of the main window.
`Settings.motorload`	Number > = 0	Load torque on the motor shaft, corresponding to the **Load** field of the main window.
`Settings.momentinertia`	Number > = 0	Combined rotor and load moment of inertia; corresponding to the **Moment of inertia** field of the main window.
`Settings.tol`	Number > 0	Convergence tolerance; corresponding to the **Convergence tolerance** field of the main window.
`Settings.f1`	Number > 0	Default supply frequency; corresponding to the **Supply frequency** field of the main window.
`Settings.U`	Number	Default supply voltage; corresponding to the **Supply voltage** field of the main window.
`Settings.torquemethod`	`'Maxwell stress tensor'`	Electromagnetic torque calculation method; corresponding to the **Torque calculation method** field of the main window.
	`'Virtual work'`	
`Settings.force`	`0`	Enables calculation of the radial force acting between the stator and rotor; corresponding to the **Rotor radial force calculation** field of the main window.
	`1`	
`Settings.report`	`0`	Enables output of simulation data in Matlab Command window; corresponding to the **Display outputs in Matlab Command Window** field of the main window.
	`1`	

`Geometry` structure fields:

Field	Value	Description
`Geometry.D1s`	Number > = 0	Machine's cross section dimensions according to section 4.1 of this manual.
`Geometry.D2s`		
`Geometry.lag`		
`Geometry.Dshaft`		
`Geometry.Ods`		
`Geometry.Ows`		
`Geometry.Tas`		
`Geometry.Sds`		
`Geometry.Rcs`		
`Geometry.Ws`		
`Geometry.Odr`		
`Geometry.Owr`		
`Geometry.Tar`		
`Geometry.Sdr`		
`Geometry.Rcr`		
`Geometry.Wr`		
`Geometry.Ns`	Number > 0	Number of stator slots; corresponding to the **Number of slots** field of the **Geometry Editor** window.
`Geometry.Nr`	Number > 0	Number of rotor bars; corresponding to the **Number of bars** field of the **Geometry Editor** window.
`Geometry.statorslottype`	`'Parallel tooth'`	Stator slot type; corresponding to the **Slot type** field of the **Stator dimensions** panel of the **Geometry Editor** window.
	`'Parallel slot'`	
`Geometry.slotlayertype`	`'Single layer'`	Number of stator winding layers; corresponding to the **Number of winding layers** field of the **Geometry Editor** window.
	`'Double layer'`	
`Geometry.rotorslottype`	`'Parallel tooth'`	Rotor slot type; corresponding to the **Slot type** field of the **Rotor dimensions** panel of the **Geometry**
	`'Parallel slot'`	
	`'Round'`	

		Editor window.
`Geometry.l`	Number > 0	Machine's axial dimensions according to section 4.1 of this manual.
`Geometry.RBskew`	Number > 0	

`Windings` structure fields:

Field	Value	Description
`Windings.Npp`	Number > 0	Number of parallel paths of the stator winding per phase; corresponding to the **Number of parallel paths** field of the **Windings Property Editor** window.
`Windings.Lsew`	Number > = 0	Leakage inductance of the stator winding end-turns per phase; corresponding to the **End winding inductance** field of the **Windings Property Editor** window.
`Windings.Rs`	Number > = 0	Active resistance of the stator winding per phase; corresponding to the **Winding phase resistance** field of the **Windings Property Editor** window.
`Windings.layout`	Nsl×Ns cell array	Stator winding layout; corresponding to the **Winding layout** table of the **Windings Property Editor** window. Nsl – number of stator winding layers, Ns – number of stator slots.
`Windings.kr`	Number > 0	Rotor bar material conductivity; corresponding to the **Bar material conductivity** field of the **Windings Property Editor** window.
`Windings.Rre`	Number > 0	Active resistance of the arch of the rotor end ring measured between two adjacent rotor bars; corresponding to the **End ring resistance** field of the **Windings Property Editor** window.
`Windings.W`	Number > 0	Total number of conductors placed in one stator slot; corresponding to the **Number of conductors per slot** field of the **Windings Property Editor** window.

`Windings.fillfactor`	Number > 0	Stator slot fill factor; corresponding to the **Slot fill factor** field of the **Windings Property Editor** window.
`Windings.ks`	Number > 0	Stator winding material conductivity; corresponding to the **Winding material conductivity** field of the **Windings Property Editor** window.
`Windings.statorcircuit`	Function name	Name of the function specifying the stator electrical circuit; corresponding to the **Stator circuit** field of the **Windings Property Editor** window.

`Mesh` structure fields:

Field	Value	Description
`Mesh.nAirGapLayers`	0 < number < 10	Number of mesh layers in the air gap; corresponding to the **Number of layers in air gap** field of the **Mesh Editor** window.
`Mesh.AirGapSlidingLayer`	Number > 0	Sliding layer number within the air gap mesh; corresponding to the **Air gap sliding layer number** field of the **Mesh Editor** window.
`Mesh.Hgrad`	1 < number < 2	Mesh growth rate; corresponding to the **Mesh growth rate** field of the **Mesh Editor** window.
`Mesh.nrfmeshcalls`	Number > 0	Number of `refinemesh` function calls; corresponding to the **Number of refinemesh calls** field of the **Mesh Editor** window.
`Mesh.nSlices`	Number > 0	Number of slices; corresponding to the **Number of slices** field of the **Mesh Editor** window.
`Mesh.p`	2×np double	Initial mesh data; these correspond to `p`, `t` and `e` arrays used in Matlab PDE Toolbox as mesh data.
`Mesh.t`	4×nt double	
`Mesh.e`	7×ne double	

`Mesh.ipSlidingLayerStator`	array of doubles	Arrays of node indexes placed on the borders of the air gap mesh layers between the sliding layer and stator and between the sliding layer and rotor, respectively.
`Mesh.ipSlidingLayerRotor`	array of doubles	

`Core` structure fields:

Field	Value	Description
`Core.corematerial`	String	Iron core material; corresponding to the **Core material** field of the **Core Property Editor** window.
`Core.mu_lin`	Number > 0	Relative permeability used by linear and fast solvers; corresponding to the **Linear relative permeability** field of the **Core Property Editor** window.
`Core.k_st`	0 < number < 1	Iron core stacking factor; corresponding to the **Stacking factor** field of the **Core Property Editor** window.
`Core.BHcurve`	2×nBH cell array	Measured points of the BH-curve; corresponding to the **BH-curve points** table of the **Core Property Editor** window. First row corresponds to the magnetic flux density values, second row – to the magnetic field intensity values. nBH – number of the BH-curve points.

5.4. Simulation script examples.

Example 1

This example involves a simulation of an induction motor with a 40% static eccentricity. The purpose is to estimate the radial electromagnetic force acting between the stator and rotor and to analyze overall machine's performance. It is known the rated load is 9 N*m. It is also known that the motor reaches its synchronous speed at no load in about 0.5 second.

Implementation of the simulation script function for this example is given in file simscript_eccentricrotor.m, simulation file is example_eccentricrotor.mat.

The following piece of code controls voltage sources and specifies the moment of inertia which is constant during the entire simulation:

```
CurrentTime = Outputs.CurrentTime;
PowerSupply=[];
Enforce=[];
U = 220;
f1 = 50;
ua=U*sqrt(2)*sin(2*pi*f1*CurrentTime);
ub=U*sqrt(2)*sin(2*pi*f1*CurrentTime+2*pi/3);
uc=U*sqrt(2)*sin(2*pi*f1*CurrentTime+4*pi/3);
PowerSupply.ua=ua;
PowerSupply.ub=ub;
PowerSupply.uc=uc;

Settings.momentinertia=0.03;
```

Voltage values are stored in corresponding fields of the `PowerSupply` structure. As it was previously said the simulation script function overlays default voltage values specified by fields **Supply voltage** and **Supply frequency** of the main window. Default voltage values will be used if a simulation script function retrieves empty `PowerSupply` structure or if a simulation script function is not in use (**No simulation script** item is selected in the **Simulation script file** field). You must not use other names for fields of the `PowerSupply` structure different form `ua`, `ub`, `uc`, if one of the default stator circuits is specified (**StarConnection** or **DeltaConnection** item in the **Stator circuit** field of the **Windings Property Editor** window). Refer to section 6.2 for more details on controlling power sources.

The following piece of code changes simulation settings and controls the motor performance:

```
if CurrentTime<1
    ShaftPosition=[0; 0];                    % 0% eccentricity
    Settings.solvertype='Fast';
    Settings.extsave=0;
    Settings.force=0;
    if CurrentTime>0.5
        Settings.motorload=9;
    else
        Settings.motorload=0;
    end
else
    Settings.solvertype='Nonlinear';
    Settings.tol=10^-4;
```

```
    Settings.extsave=1;
    Settings.force=1;
    if CurrentTime>1.3
        lag=Geometry.lag/1000;              % air gap length
        ShaftPosition=[0.2*lag; 0.35*lag];  % 40% static eccentricity
    else
        ShaftPosition=[0; 0];               % 0% eccentricity
    end
end
```

Since the startup transient is out of interest first second of the motor operation is simulated using 'Fast' solver to determine the initial conditions for nonlinear simulation under rated load. When the simulation reaches the one second time point, solver type is set to 'Nonlinear', the extended data saving mode and force calculation are enabled. First 0.3 seconds the simulation continues without rotor eccentricity, thereafter the 40% eccentricity is applied. To shift the rotor shaft position, two elements of the `ShaftPosition` array are used specifying *x* and *y* coordinates of the rotor shaft in meters, respectively. Note that all machine's dimensions in the `Geometry` structure are given in millimeters. The following piece of code executes calculation and storing of the user's data:

```
if ~isfield(Userdata,'Fm')
    Userdata.Fm=[];
    Userdata.Fang=[];
else
    Fx=Outputs.Fx;
    Fy=Outputs.Fy;
    Fx=Fx(end);
    Fy=Fy(end);
    Fm=sqrt(Fx^2+Fy^2);             % force magnitude
    Fang=atan2(Fx,Fy)*180/pi;       % force angle

    Userdata.Fm=[Userdata.Fm Fm];
    Userdata.Fang=[Userdata.Fang Fang];
end
```

In this example the force magnitude and force angle are additionally calculated and stored in the `Userdata` structure. With the first call of the simulation script function, `Userdata` structure fields are initialized with the empty arrays. With the following function calls the force magnitude and force

angle variables are calculated and stored in the corresponding fields of the `Userdata` structure. Note that when a simulation script function is called for the first time, all fields of the `Outputs` structure contain empty arrays.

Example 2

When the virtual work principle is applied for the torque and force calculation, the accuracy significantly depends on the choice of the virtual displacement value. On the one hand, the displacement should be small enough not to distort the finite element mesh. On the other hand the round-off error arises when the displacement value is too small. The optimal value of the virtual displacement is not provided automatically and should be defined by the user in file MotorAnalysisSettings.m (refer to chapter 8 for more details).

This example suggests the algorithm for estimation of the accuracy of the virtual work force calculation versus the chosen virtual displacement value. The algorithm is based on the assumption that the value of the calculated force does not depend on the virtual displacement direction. So, comparing the force values calculated for two opposite displacements of the same value, it is possible to estimate the force calculation accuracy depending on the choice of the virtual displacement value.

Implementation of the simulation script function for this example is given in file simscript_virtualforce.m, simulation file is example_ virtualforce.mat. Motor with a 40% static eccentricity is used operating under 2 N*m load. Since taking into account the saturation of the iron core is not essential in this case, the solver of the ‘Linear’ type is used.

Only a part of the source code is shown. The full code of the function can be found in file simscript_virtualforce.m.

Since the function retrieves empty `PowerSupply` structure the default voltage sources defined by (5.1) are used:

```
PowerSupply=[];
```

The following code calculates force with the virtual work method for several virtual displacement values and stores the results in the `Userdata` structure:

```
d=[10^-7 10^-8 10^-9 10^-10 10^-11 10^-12];   % virtual displacement
e=Mesh.e;
t=Mesh.t;
nSlices=Mesh.nSlices;
nAirGapLayers=Mesh.nAirGapLayers;
AirGapSlidingLayer=Mesh.AirGapSlidingLayer;
ipSlidingLayerRotor=Mesh.ipSlidingLayerRotor;
ipSlidingLayerStator=Mesh.ipSlidingLayerStator;
```

```
Ns=Geometry.Ns;
Nr=Geometry.Nr;
l=Geometry.l/1000;
% indexes of the mesh triangles of the rotor
it_rotor=find((t(4,:)>4+Ns & t(4,:)<=4+Ns+Nr) | t(4,:)==3  |...
t(4,:)==4 | (t(4,:)>4+2*Ns+Nr & t(4,:)<=4+2*Ns+2*Nr));
% indexes of the mesh points which are virtually displaced
ip_displaced=[t(1,it_rotor)    t(2,it_rotor)    t(3,it_rotor)    ...
reshape(ipSlidingLayerRotor,1, ...
size(ipSlidingLayerRotor,1)*size(ipSlidingLayerRotor,2))];
ip_displaced=sort(ip_displaced);
ip_displaced=ip_displaced([find(diff(ip_displaced)) ...
length(ip_displaced)]);
if ~isfield(Userdata,'Fx1')
    Userdata.Fx1=[];   Userdata.Fy1=[];
    Userdata.Fx1_=[];  Userdata.Fy1_=[];
    Userdata.Fx2=[];   Userdata.Fy2=[];
    Userdata.Fx2_=[];  Userdata.Fy2_=[];
    Userdata.Fx3=[];   Userdata.Fy3=[];
    Userdata.Fx3_=[];  Userdata.Fy3_=[];
    Userdata.Fx4=[];   Userdata.Fy4=[];
    Userdata.Fx4_=[];  Userdata.Fy4_=[];
    Userdata.Fx5=[];   Userdata.Fy5=[];
    Userdata.Fx5_=[];  Userdata.Fy5_=[];
    Userdata.Fx6=[];   Userdata.Fy6=[];
    Userdata.Fx6_=[];  Userdata.Fy6_=[];
else
    for i=1:length(d)
        [Fx_slice Fy_slice]=VirtualWorkForce ...
        (d(i),cell_p,cell_t,e,cell_Nu,A, ...
        ip_displaced,ipSlidingLayerRotor,ipSlidingLayerStator,l, ...
        nSlices,nAirGapLayers,AirGapSlidingLayer);
        % force components
        Fx=sum(Fx_slice);
        Fy=sum(Fy_slice);
        ExeStr=['Userdata.Fx' num2str(i) ...
```

```
        '=[Userdata.Fx' num2str(i) ' Fx];'];
        eval(ExeStr);
        ExeStr=['Userdata.Fy' num2str(i) ...
        '=[Userdata.Fy' num2str(i) ' Fy];'];
        eval(ExeStr);
        [Fx_slice_ Fy_slice_]=VirtualWorkForce ...
        (-d(i),cell_p,cell_t,e,cell_Nu,A, ...
        ip_displaced,ipSlidingLayerRotor,ipSlidingLayerStator,l, ...
        nSlices,nAirGapLayers,AirGapSlidingLayer);
        % force components
        Fx_=sum(Fx_slice_);
        Fy_=sum(Fy_slice_);
        ExeStr=['Userdata.Fx' num2str(i) ...
        '_=[Userdata.Fx' num2str(i) '_ Fx_];'];
        eval(ExeStr);
        ExeStr=['Userdata.Fy' num2str(i) ...
        '_=[Userdata.Fy' num2str(i) '_ Fy_];'];
        eval(ExeStr);
    end
end
```

On the first line of the code the array of virtual displacements is defined. Force values are calculated using `VirtualWorkForce` function retrieving values for every slice and the actual force is obtained as a summation of all slice values. Forces are calculated for positive and negative displacement values. Variables related to negative displacements are marked with the underline character at the end of the variable name. Calculated values are stored in corresponding fields of the `Userdata` structure.

6. USING ELECTRICAL CIRCUITS

MotorAnalysis allows you to connect a stator winding to an electrical circuit consisting of the following components:

- resistors;
- capacitors;
- inductances;
- voltage sources;
- current sources.

There is no graphical user interface for editing electrical circuits in this version of MotorAnalysis. Special functions are used instead.

The electrical circuit is specified through the function which retrieves the electrical circuit object. The name of the function appears in the **Stator circuit** pop-up menu of the **Windings Property Editor** window. Clicking the button to the right allows you to specify your own electrical circuit choosing corresponding M-file from the list of files. Functions constructing default electrical circuits of the star and delta connection are implemented in files StarConnection.m and DeltaConnection.m, respectively. You can use these files as examples to write a function for your own electrical circuit.

To understand this chapter some familiarity with Matlab coding is required. Refer to Matlab help documentation for more details on Matlab coding principles.

6.1. Writing an electrical circuit function.

The definition of the electrical circuit function `StarConnection` appears in M-file StarConnection.m as follows:

```
function Schematic = ...
StarConnection(Rs,Lsew,W,Npp,layout,p,t,Nr,Ns,l,delta_t,nSlices)
```

The function retrieves one argument `Schematic` which contains the electrical circuit description.

The input arguments of the function:

`Rs` – active resistance of the stator winding per phase; this variable is taken from the **Winding phase resistance** field of the **Windings Property Editor** window.

`Lsew` – leakage inductance of the stator winding end-turns per phase; this variable is taken from the **End winding inductance** field of the **Windings Property Editor** window.

`W` – total number of conductors placed in one stator slot; this variable is taken from the **Number of conductors per slot** field of the **Windings Property Editor** window.

`Npp` – number of parallel paths of the stator winding per phase; this variable is taken from the **Number of parallel paths** field of the **Windings Property Editor** window.

`layout` – stator winding layout; this variable is taken from the **Winding layout** table of the **Windings Property Editor** window.

`p` – variable taken from `Mesh.p`.

`t` – variable taken from `Mesh.t`.

`Nr` – number of rotor bars; this variable is taken from the **Number of bars** field of the **Geometry Editor** window.

`Ns` – number of stator slots; this variable is taken from the **Number of slots** field of the **Geometry Editor** window.

`l` – lamination length; this variable is taken from the **Lamination length** field of the **Geometry Editor** window.

`delta_t` – simulation time step size; this variable is taken from the **Time step** field of the main window.

`nSlices` – number of slices used by multi-slice FEM; this variable is taken from the **Number of slices** field of the **Mesh Editor** window.

Circuit construction procedure always begins with the following function:

```
Schematic = CircuitCreateSchematic();
```

This function does not have input arguments and retrieves the empty circuit object `Schematic`.

The procedure of circuit construction consists of building circuit branches and adding branches to the circuit object.

6.1.1. Electrical circuit branches.

An electrical circuit branch building begins with the following function:

```
Branch = CircuitCreateBranch(node1,node2);
```

All circuit nodes should be previously numbered beginning with zero node. The function receives start and end node numbers of the circuit branch, respectively, and retrieves an empty branch object.

When all circuit components are added to the branch object (see section 6.1.2), it should be added to the circuit object using the following function:

```
Schematic = CircuitAddBranch(Schematic,Branch);
```

The function receives the circuit and branch objects and retrieves the circuit objects with the new branch added to the circuit.

6.1.2. Adding circuit components.

6.1.2.1. To add a resistor to a branch, the following function is used:

```
Branch = CircuitAddR(Branch,name,value);
```

`Branch` – variable corresponding to a branch which a resistor is added to, `name` – unique circuit component name, `value` – resistance value in Ohm.

Example of adding resistor R1 of 1 kOhm to the branch defined by variable `Branch`:

```
Branch = CircuitAddR(Branch, 'R1', 1000);
```

6.1.2.2. To add an inductance to a branch, the following function is used:

```
Branch = CircuitAddL(Branch,name,value);
```

`Branch` – variable corresponding to a branch which inductance is added to, `name` – unique circuit component name, `value` – inductance value in Henries.

Example of adding inductance L1 of 100 mH to the branch defined by variable `Branch`:

```
Branch = CircuitAddL(Branch, 'L1', 0.1);
```

6.1.2.3. To add a capacitor to a branch, the following function is used:

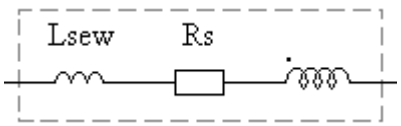
```
Branch = CircuitAddC(Branch,name,value);
```

`Branch` – variable corresponding to a branch which a capacitor is added to, `name` – unique circuit component name, `value` – capacitance value in Farads.

Example of adding capacitor C1 of 100 μF to the branch defined by variable `Branch`:

```
Branch = CircuitAddC(Branch, 'C1', 100*10^-6);
```

6.1.2.4. Each stator winding coil or a group of coils can be treated as an independent circuit component consisting of an active resistance, stator end winding inductance and conductors within stator slots designated as a spiral with a dot at the input terminal:



The coil or the group of coils is herein referred to as a coil object or a coil component. Each coil object combines all stator slots or stator slot layers associated with the same phase and parallel path, i.e. having the same letter and number in the stator layout table.

To add a coil object to a branch, the following function is used:

```
Branch = CircuitAddCoil(Branch,name,phase,R,Lsew,Wcoil,npath,...
         coilorientation,layout,p,t,Nr,Ns,l,delta_t,nSlices);
```

`Branch` – variable corresponding to a branch which a coil object is added to, `name` – unique circuit component name, `phase` – phase accepts one of the following values: `'a'`, `'b'`, `'c'`; `R` and `Lsew` – active resistance and end winding inductance of a coil component, respectively, `npath` – parallel path number (=1 if there are no parallel paths), `coilorientation=1`, if a coil component is oriented concordantly with the branch which a coil component is added to, otherwise `coilorientation=-1`. The branch orientation is defined by `node1` and `node2` arguments of `CircuitAddBranch` function and the coil component orientation is defined by a dot at the input terminal. Input arguments `layout`, `p`, `t`, `Nr`, `Ns`, `l`, `delta_t`, `nSlices` are defined form the circuit function definition.

Note that in order to prevent incorrect results **each branch should include only one coil component**, otherwise coil components are to be divided by additional nodes as it is shown in example 3 of this chapter.

Example of adding a coil component to a branch defined by variable `Branch`:

```
Branch = CircuitAddCoil(Branch,'coil_c2','c',...
                Rs,Lsew,Wcoil,2,1,layout,p,t,Nr,Ns,l,delta_t,nSlices);
```

In this case a coil component corresponds to the second group of coils of phase “c” named as coil_c2, oriented concordantly with the branch, with the active resistance `Rs`, end winding inductance `Lsew` and number of conductors placed in stator slot `Wcoil`.

6.1.2.5. To add a voltage source to a branch the following function is used:

```
Branch = CircuitAddE(Branch, name, data);
```

`Branch` – variable corresponding to a branch which a voltage source is added to, `name` – unique circuit component name, `data` – `PowerSupply` structure field used by a simulation script function to control a voltage source (see section 6.2).

6.1.2.6. To add a current source to a branch, the following function is used:

```
Branch = CircuitAddJ(Branch, name, data);
```

`Branch` – variable corresponding to a branch which a current source is added to, `name` – unique circuit component name, `data` – `PowerSupply` structure field used by a simulation script function to control a current source (see section 6.2).

6.2. Controlling the voltage and current values of the power sources.

The voltage and current values supplied by the voltage and current sources can be specified at each time step by a simulation script function. For this purpose the `PowerSupply` structure is used. Each voltage or current source is associated with its field of the `PowerSupply` structure through the input argument `data`, when the `CircuitAddE` or `CircuitAddJ` function is called. A value assigned to a field of the `PowerSupply` structure at the specified time step will appear on the voltage or current source output associated with this field.

The following example provides an explanation for the power source control principle. Assume that the voltage source named U1 was added to the stator circuit calling the function

```
Branch = CircuitAddE(Branch,'U1','PowerSupply.u1');
```

In this case the voltage source U1 is associated with field `u1` of the `PowerSupply` structure. The following piece of code can be used to specify the voltage value supplied by voltage course U1 at the given simulation time step:

```
CurrentTime = Outputs.CurrentTime;
PowerSupply.u1 = 311*sin(2*pi*50*CurrentTime);
```

Note that the power sources can have any desired voltage or current waveforms.

If a simulation script function retrieves empty `PowerSupply` structure or if a simulation script function is not in use (**No simulation script** item is selected in the **Simulation script file** field), default voltage waveform defined by (5.1) will be used. `PowerSupply` structure fields `ua`, `ub`, `uc` are reserved to control voltage sources of default stator circuits (star and delta connection) then user defined voltage waveform is required. Below the piece of code of a simulation script function for adding higher order harmonics to the supply voltage:

```
CurrentTime = Outputs.CurrentTime;
PowerSupply.ua=U*sqrt(2)*sin(2*pi*f1*CurrentTime)+ ...
0.02*U*sqrt(2)*sin(2*pi*5*f1*CurrentTime)+ ...
0.01*U*sqrt(2)*sin(2*pi*7*f1*CurrentTime);
PowerSupply.ub=U*sqrt(2)*sin(2*pi*f1*CurrentTime+2*pi/3)+
0.02*U*sqrt(2)*sin(2*pi*5*f1*CurrentTime+2*pi/3)+ ...
0.01*U*sqrt(2)*sin(2*pi*7*f1*CurrentTime+2*pi/3);
```

```
PowerSupply.uc=U*sqrt(2)*sin(2*pi*f1*CurrentTime+4*pi/3)+ ...
0.02*U*sqrt(2)*sin(2*pi*5*f1*CurrentTime+4*pi/3)+ ...
0.01*U*sqrt(2)*sin(2*pi*7*f1*CurrentTime+4*pi/3);
```

If you use your own stator circuit you can also use names `ua`, `ub`, `uc` for the `PowerSupply` structure fields associating them with voltage sources of your circuit. In this case, similarly to using a default circuit, if a simulation script function retrieves empty `PowerSupply` structure or if a simulation script function is not in use, default voltage waveform defined by (5.1) will be used. Otherwise if other field names of the `PowerSupply` structure are assigned, the voltage or current waveforms must be specified through a simulation script function.

6.3. Examples of using stator electrical circuits.

Example 1

This example provides an explonation for the construction procedure of the electrical circuit with a star connected stator winding which is one of the default stator circuits used in MotorAnalysis. Corresponding electrical circuits for stator winding without parallel paths and for one with two parallel paths are shown in Figure 6.1. Implementation of the function discussed in this example is given in M-file StarConnection.m.

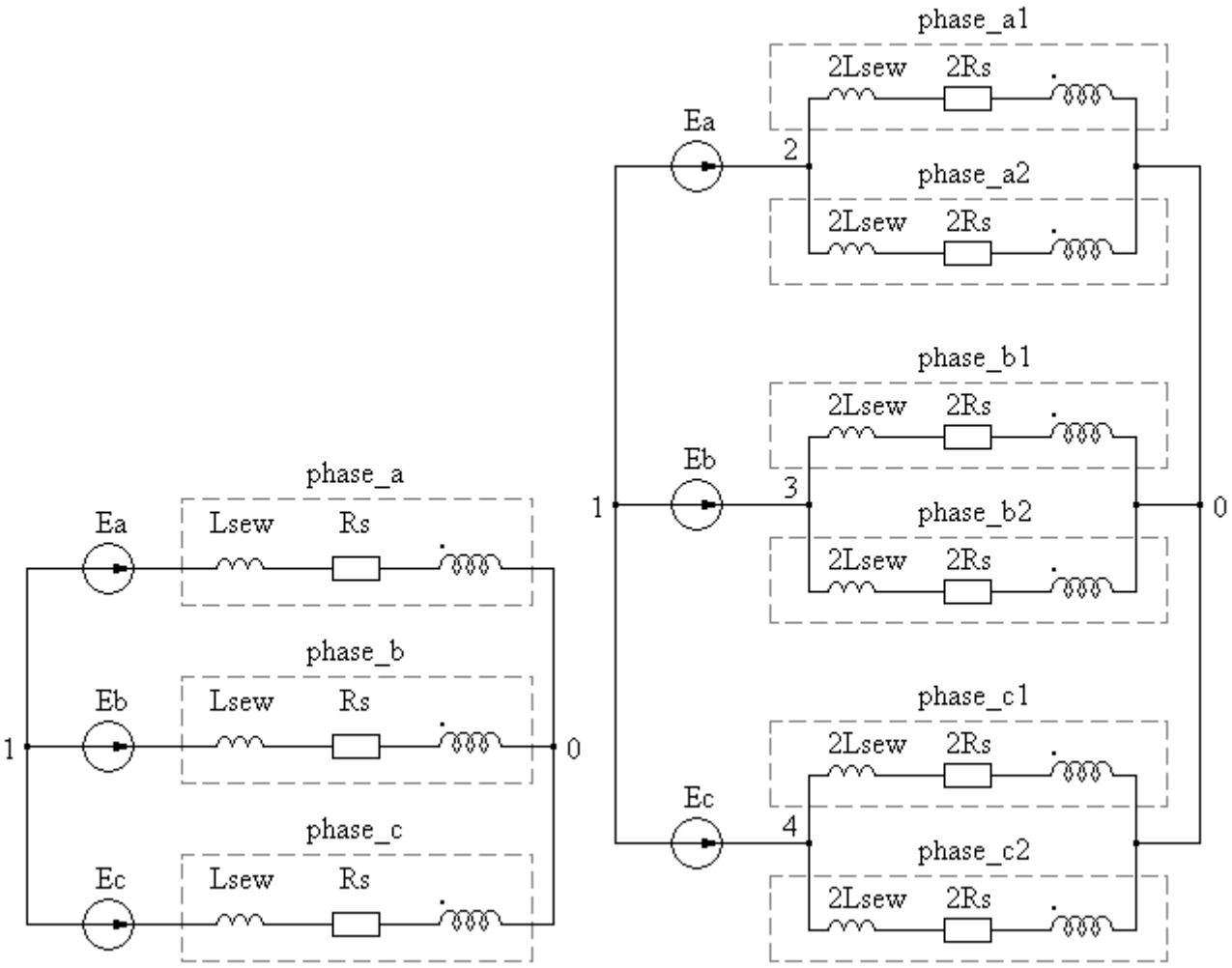
The definition of the function appears in M-file StarConnection.m as follows:

```
function Schematic =  ...
StarConnection(Rs,Lsew,W,Npp,layout,p,t,Nr,Ns,l,delta_t,nSlices)
```

The function has input and output arguments as it described in section 6.1.

The following piece of code executes the calculation of the internal variables and data validation:

```
if size(layout,1)==2
    Wcoil=W/2;          % double-layer winding
    if round(Wcoil)~=W/2
        error('Number of conductors per slot must be a multiple of
               two for a double-layer winding')
    end
else
    Wcoil=W;            % single-layer winding
end
Rs = Npp*Rs;            % parallel path phase resistance
Lsew = Npp*Lsew;        % parallel path end winding inductance
```



a - “star” with no parallel paths b - “star” with two parallel paths per phase

Figure 6.1. Electrical circuit with star connected stator winding for different number of parallel paths.

```
[ok_layout ok_Npp] = ValidateLayout(layout,Npp);
if ~ok_layout
    error('ValidateLayout: desequencing of parallel path numbering');
end
if ~ok_Npp
    error('ValidateLayout: number of parallel paths is not consistent
           with winding layout table');
end
```

`Wcoil` – is the number of conductors within a stator slot per one coil or a group of coils (i.e. coil component). For a single-layer winding this variable equals to the entire number of conductors within a stator slot. For a double-layer winding this variable equals to the number of conductors within a stator slot divided by two.

If a stator winding has several parallel paths, the active resistance and end winding inductance for each parallel path are defined as follows:

$$Rs\_par = Npp*Rs$$
$$Lsew\_par = Npp*Lsew, \quad (6.1)$$

where *Rs* and *Lsew* – values specified in **Winding phase resistance** and **End winding inductance** fields, respectively, *Npp* – number of parallel paths per phase, specified in **Number of parallel paths** field. According to (6.1) the active resistance and end winding inductance of each parallel path of the circuit shown in Figure 6.1(b) are doubled.

`ValidateLayout` function checks whether the stator layout table is consistent with the number of parallel paths and whether the parallel paths in the stator layout table are numbered in consecutive order.

The electrical circuit construction procedure is implemented by the following piece of code:

```
Schematic = CircuitCreateSchematic();
if Npp==1
    Branch = CircuitCreateBranch(1,0);
    Branch = CircuitAddE(Branch,'Ea','PowerSupply.ua');
    Branch = CircuitAddCoil(Branch,'coil_a','a',Rs,Lsew,Wcoil, ...
             1,1,layout,p,t,Nr,Ns,l,delta_t,nSlices);
    Schematic = CircuitAddBranch(Schematic,Branch);

    Branch = CircuitCreateBranch(1,0);
    Branch = CircuitAddE(Branch,'Eb','PowerSupply.ub');
    Branch = CircuitAddCoil(Branch,'coil_b','b',Rs,Lsew,Wcoil, ...
             1,1,layout,p,t,Nr,Ns,l,delta_t,nSlices);
    Schematic = CircuitAddBranch(Schematic,Branch);

    Branch = CircuitCreateBranch(1,0);
    Branch = CircuitAddE(Branch,'Ec','PowerSupply.uc');
    Branch = CircuitAddCoil(Branch,'coil_c','c',Rs,Lsew,Wcoil, ...
             1,1,layout,p,t,Nr,Ns,l,delta_t,nSlices);
    Schematic = CircuitAddBranch(Schematic,Branch);
elseif Npp>1
    Branch = CircuitCreateBranch(1,2);
    Branch = CircuitAddE(Branch,'Ea','PowerSupply.ua');
```

```
    Schematic = CircuitAddBranch(Schematic,Branch);

    Branch = CircuitCreateBranch(1,3);
    Branch = CircuitAddE(Branch,'Eb','PowerSupply.ub');
    Schematic = CircuitAddBranch(Schematic,Branch);

    Branch = CircuitCreateBranch(1,4);
    Branch = CircuitAddE(Branch,'Ec','PowerSupply.uc');
    Schematic = CircuitAddBranch(Schematic,Branch);
    for i=1:Npp
        Branch = CircuitCreateBranch(2,0);
        Branch = CircuitAddCoil(Branch,['coil_a' num2str(i)],'a', ...
               Rs,Lsew,Wcoil,i,1,layout,p,t,Nr,Ns,l,delta_t,nSlices);
        Schematic = CircuitAddBranch(Schematic,Branch);

        Branch = CircuitCreateBranch(3,0);
        Branch = CircuitAddCoil(Branch,['coil_b' num2str(i)],'b', ...
               Rs,Lsew,Wcoil,i,1,layout,p,t,Nr,Ns,l,delta_t,nSlices);
        Schematic = CircuitAddBranch(Schematic,Branch);

        Branch = CircuitCreateBranch(4,0);
        Branch = CircuitAddCoil(Branch,['coil_c' num2str(i)],'c', ...
               Rs,Lsew,Wcoil,i,1,layout,p,t,Nr,Ns,l,delta_t,nSlices);
        Schematic = CircuitAddBranch(Schematic,Branch);
    end
end
```

Default voltage sources are used associated with `ua`, `ub`, `uc` fields of the `PowerSupply` structure. Using a simulation script function for this circuit is not required (though still possible). If a stator winding has parallel paths, each parallel path (branch) is constructed within the `for` loop.

Example 2

In this example another default stator electrical circuit with a delta connected stator winding is discussed.

Corresponding electrical circuits for stator winding with two parallel paths are shown in Figure 6.2.

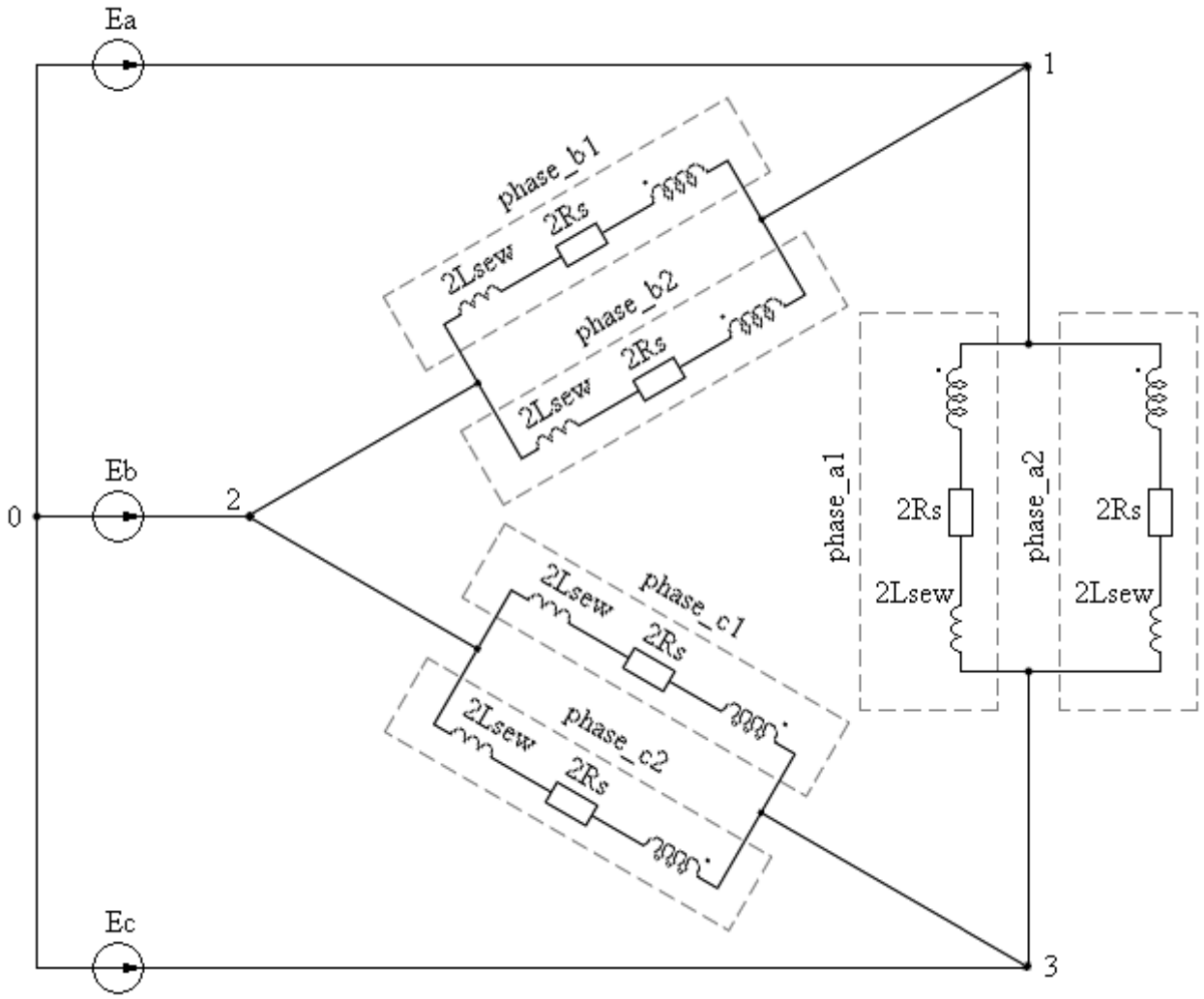


Figure 6.2. Electrical circuit with the delta connected stator winding for two parallel paths.

Sourse code of function `DeltaConnection` corresponding to the circuit with the delta connected stator winding is given in M-file DeltaConnection.m and is not provided in this example.

Example 3

This example deals with a three-phase induction motor connected to a single-phase supply. The electrical circuit for single phase connection of a stator winding used in the example is shown in Figure 6.3.

Some important issues concerning the circuit construction when coils of different phases are placed at one circuit branch are also discussed in this example.

Implementation of the function of this example is given in M-file SinglePhaseConnection.m, the simulation script function related to the example can be found in M-file simscript_SinglePhase.m, simulation file - example_SinglePhase.mat.

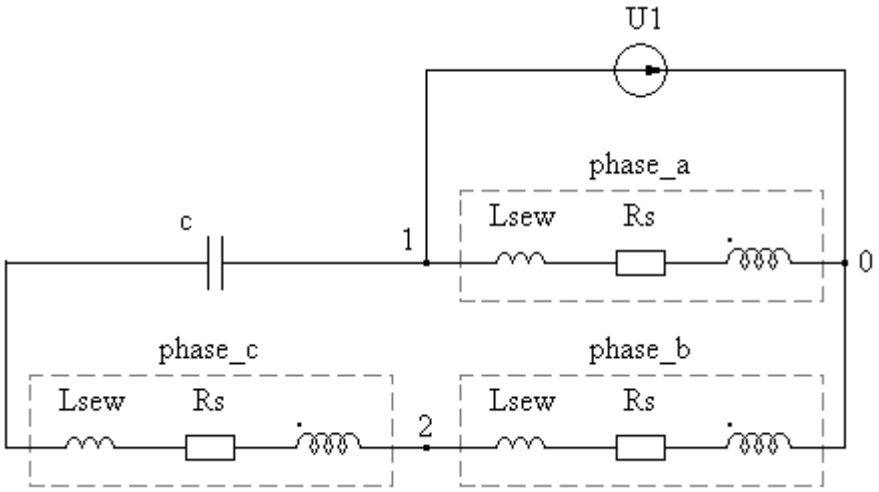


Figure 6.3. Electrical circuit for single phase connection of a stator winding.

`SinglePhaseConnection` function source code:

```
function Schematic = SinglePhaseConnection(Rs,Lsew,W,Npp,layout,...
                     p,t,Nr,Ns,l,delta_t,nSlices)
if Npp>1
    error('SinglePhaseConnection function does not support parallel
coil connection. You can provide your own code to fulfill this
requirement');
end
% W - number of conductors per slot
% Wcoil - number of conductors per coil per slot
if size(layout,1)==2
    Wcoil=W/2;        % double-layer winding
    if round(Wcoil)~=W/2
        error('Number of conductors per slot must be a multiple of
               two for a double-layer winding')
    end
else
    Wcoil=W;          % single-layer winding
end
Schematic = CircuitCreateSchematic();
Branch = CircuitCreateBranch(0,1);
Branch = CircuitAddE(Branch,'U1','PowerSupply.u1');
```

```
Schematic = CircuitAddBranch(Schematic,Branch);

Branch = CircuitCreateBranch(1,0);
Branch = CircuitAddCoil(Branch,'coil_a','a',Rs,Lsew,Wcoil,1,1,...
                        layout,p,t,Nr,Ns,l,delta_t,nSlices);
Schematic = CircuitAddBranch(Schematic,Branch);

Branch = CircuitCreateBranch(0,2);
Branch = CircuitAddCoil(Branch,'coil_b','b',Rs,Lsew,Wcoil,1,1,...
                        layout,p,t,Nr,Ns,l,delta_t,nSlices);
Schematic = CircuitAddBranch(Schematic,Branch);

Branch = CircuitCreateBranch(1,2);
Branch = CircuitAddC(Branch,'c',30*10^-6);
Branch = CircuitAddCoil(Branch,'coil_c','c',Rs,Lsew,Wcoil,1,1,...
                        layout,p,t,Nr,Ns,l,delta_t,nSlices);
Schematic = CircuitAddBranch(Schematic,Branch);
```

In the circuit two coils of phase B and phase C are connected in series and appear to be placed at one circuit branch. According to section 6.1.2.4, in such cases the branch is required to be divided by an additional node into two branches, so each branch includes only one coil. As it is shown in Figure 6.3, node 2 was added to the circuit to fulfill this requirement.

Implementation of the function for single phase connection of a stator winding given in this example does not support parallel paths; so, if the number of parallel paths is more than one, an error occurs. You can provide your own code, if parallel paths are needed.

To control voltage source U1, the simulation script function implemented in file simscript_SinglePhase.m is used. The voltage value is controlled through the `u1` field of the `PowerSupply` structure. The source code of the simulation script function:

```
function [ShaftPosition,PowerSupply,Settings,Enforce,Userdata] = ...
simscript_SinglePhase(Outputs,Settings,Userdata,Circuit,...
Geometry,Mesh,Windings,Core,A,cell_jr,cell_p,cell_t,cell_Nu,path)

ShaftPosition=[];
Enforce=[];
PowerSupply=[];
CurrentTime = Outputs.CurrentTime;
```

```
U = Settings.U;
f1 = Settings.f1;
u1=U*sqrt(2)*sin(2*pi*f1*CurrentTime);
PowerSupply.u1=u1;
% save capacitor voltage
Branch=Circuit.Schematic{4,1};
c=Branch.Components{1,1};
Uc=c.U;
if ~isfield(Userdata,'Uc')
    Userdata.Uc=[];
else
    Userdata.Uc=[Userdata.Uc Uc];
end
```

Besides voltage source control, the function also stores the capacitor voltage values at each time step.

Example 4

In this example the electrical circuit with a star connected stator winding fed by current sources is discussed. The electrical circuit is shown in Figure 6.4.

Implementation of the function of this example is given in M-file CurrentSourceStarConnection.m, the simulation script function related to the example can be found in M-file simscript_JStar.m, simulation file - example_JStar.mat.

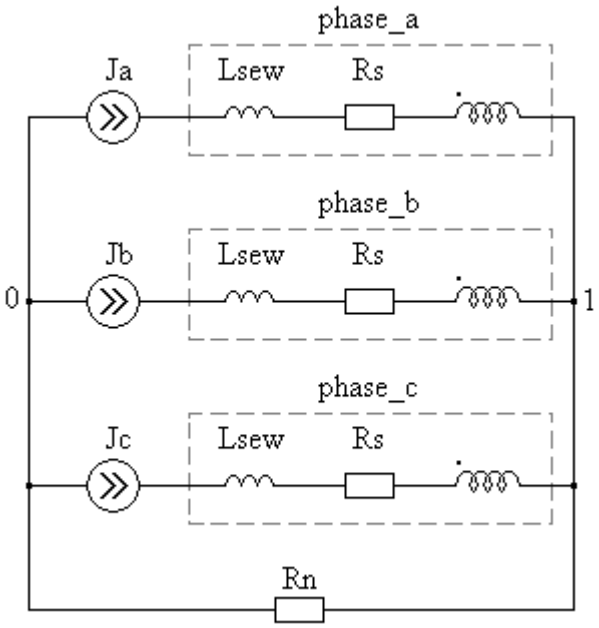


Figure 6.4. Electrical circuit with the star connected stator winding fed by current sources.

`CurrentSourceStarConnection` function source code:

```
function Circuit = CurrentSourceStarConnection(Rs,Lsew,W,Npp,...
                   layout,p,t,Nr,Ns,l,delta_t,nSlices)

if Npp>1
    error('CurrentSourceStarConnection function does not support
           parallel coil connection. You can provide your own code to
           fulfill this requirement');
end
% W - number of conductors per slot
% Wcoil - number of conductors per coil per slot
if size(layout,1)==2
    Wcoil=W/2;        % double-layer winding
    if round(Wcoil)~=W/2
        error('Number of conductors per slot must be a multiple of
               two for a double-layer winding')
    end
else
    Wcoil=W;          % single-layer winding
end

Schematic = CircuitCreateSchematic();

Branch = CircuitCreateBranch(0,1);
Branch = CircuitAddJ(Branch,'Ja','PowerSupply.ia');
Branch = CircuitAddCoil(Branch,'coil_a','a',Rs,Lsew,Wcoil,1,1,...
                        layout,p,t,Nr,Ns,l,delta_t,nSlices);
Schematic = CircuitAddBranch(Schematic,Branch);

Branch = CircuitCreateBranch(0,1);
Branch = CircuitAddJ(Branch,'Jb','PowerSupply.ib');
Branch = CircuitAddCoil(Branch,'coil_b','b',Rs,Lsew,Wcoil,1,1,...
                        layout,p,t,Nr,Ns,l,delta_t,nSlices);
Schematic = CircuitAddBranch(Schematic,Branch);
```

```
Branch = CircuitCreateBranch(0,1);
Branch = CircuitAddJ(Branch,'Jc','PowerSupply.ic');
Branch = CircuitAddCoil(Branch,'coil_c','c',Rs,Lsew,Wcoil,1,1,...
                        layout,p,t,Nr,Ns,l,delta_t,nSlices);
Schematic = CircuitAddBranch(Schematic,Branch);

Branch = CircuitCreateBranch(0,1);
Branch = CircuitAddR(Branch,'Rn',1000000);
Schematic = CircuitAddBranch(Schematic,Branch);
```

As it is seen in Figure 6.4 one more branch with resistor Rn of 1MOhm was added to the circuit. Branches with current sources have infinite resistance that destabilizes the FEM solver performance, so the branch with finite resistance connected in parallel is required. Since the resistor value is large enough, the influence of this additional branch on the circuit is minor.

To control current sources, the simulation script function implemented in M-file simscript_JStar.m is used. The current values are controlled through the `ia`, `ib`, `ic` fields of the `PowerSupply` structure. The source code of the simulation script function:

```
function [ShaftPosition,PowerSupply,Settings,Enforce,Userdata] = ...
simscript_JStar(Outputs,Settings,Userdata,Circuit,Geometry,Mesh,...
                Windings,Core,A,cell_jr,cell_p,cell_t,cell_Nu,path)

ShaftPosition=[];
Enforce=[];
PowerSupply=[];
CurrentTime = Outputs.CurrentTime;
f1 = Settings.f1;
ia=5*sin(2*pi*f1*CurrentTime);
ib=5*sin(2*pi*f1*CurrentTime+2*pi/3);
ic=5*sin(2*pi*f1*CurrentTime+4*pi/3);
PowerSupply.ia=ia;
PowerSupply.ib=ib;
PowerSupply.ic=ic;
```

7. VIEWING RESULTS

A tool for viewing simulation results in MotorAnalysis is called **Plot Wizard**. Figure 7.1 shows the **Plot Wizard** window.

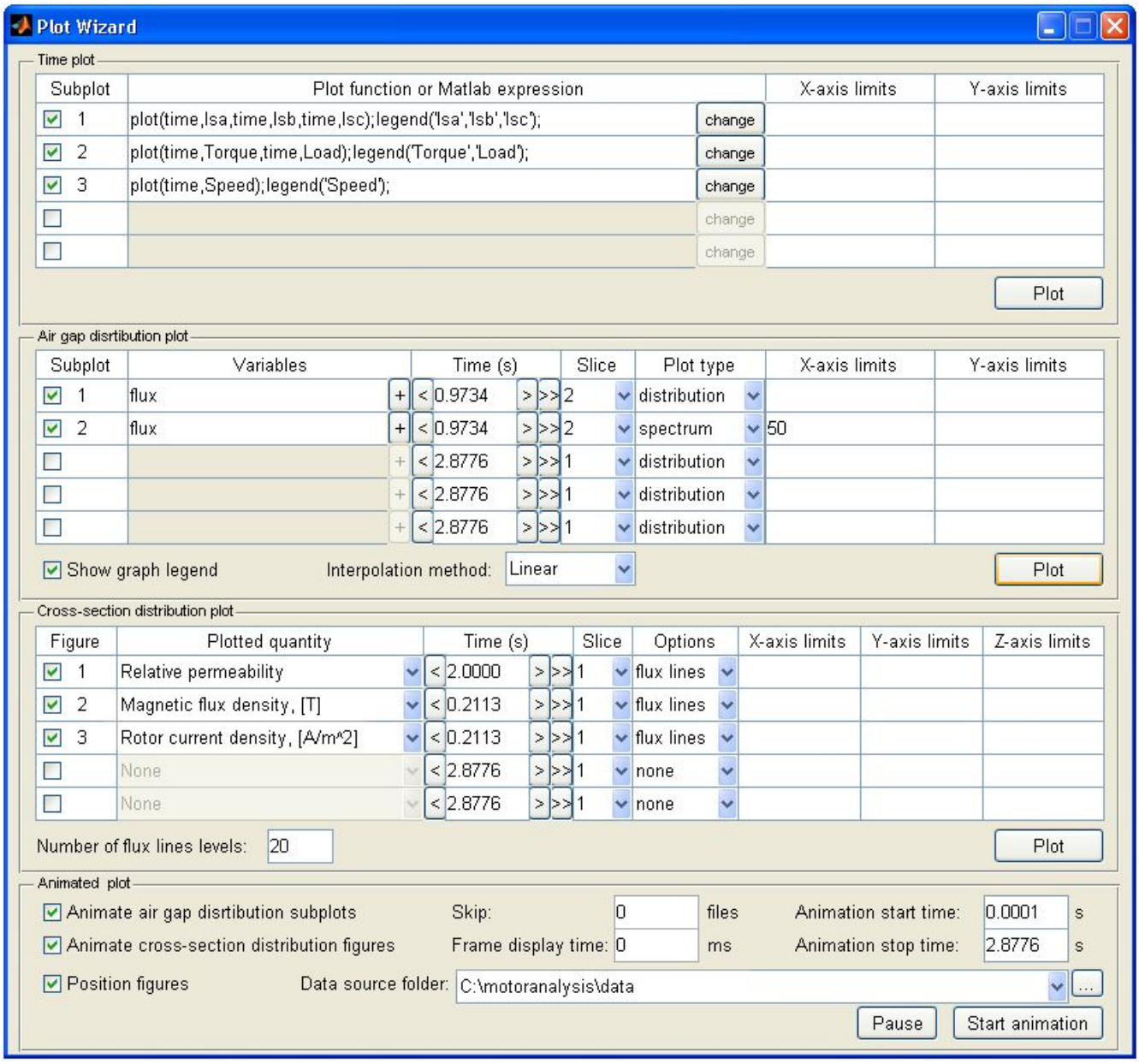


Figure 7.1. Plot Wizard window.

A few plot types are available in MotorAnalysis:

- time plot;
- air gap distribution plot;
- frequency spectrum of the air gap distribution;
- cross-section distribution plot;
- animation.

7.1. Time plots.

Time plots are available from the **Time plot** panel of the **Plot Wizard** window. It is organized as a standard Matlab plotting construction consisting of a set of `subplot` and `plot` functions. **Subplot** checkboxes allow controlling the number of axes or rectangular panes displayed within a current figure window. The corresponding axes are activated or deactivated by a mouse click within a cell of the **Subplot** column. When the subplot is activated, the **Change** button allows you to choose variables to be plotted into the selected axes. Variables can be chosen from the dialog shown in Figure 7.2. Use Ctrl button to select several variables.

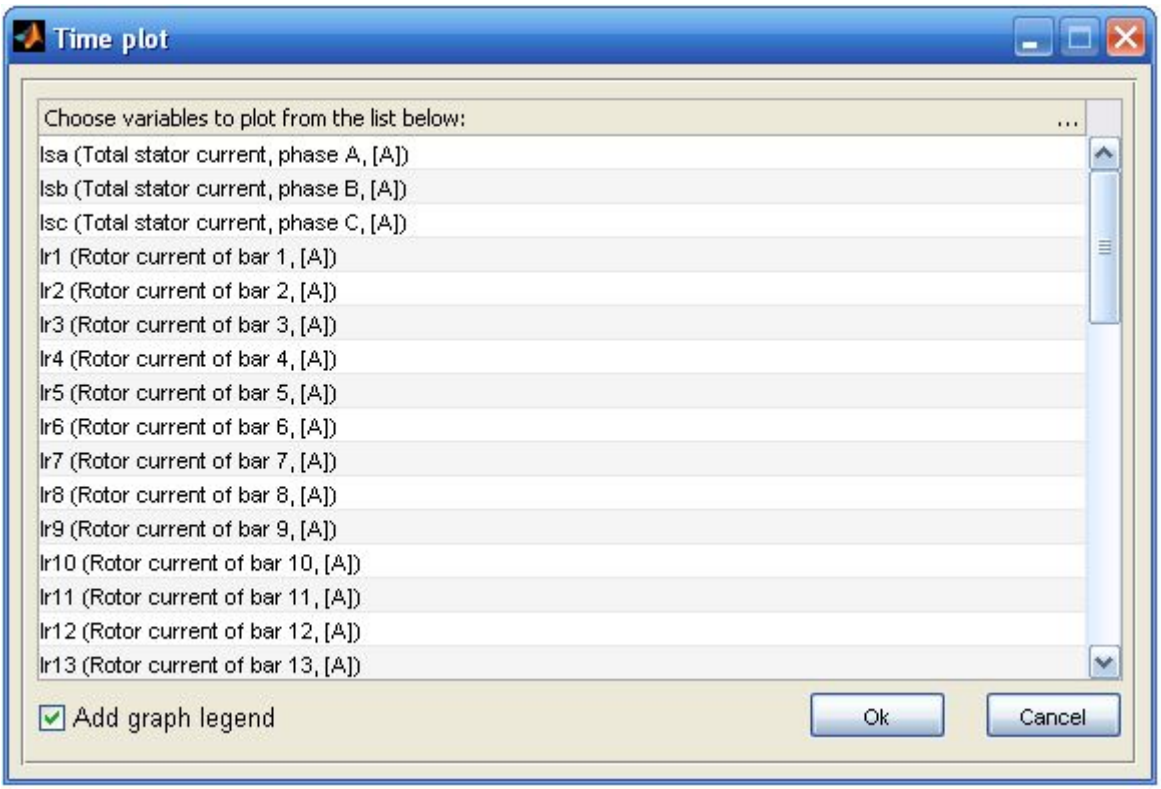


Figure 7.2. Choosing variables to be plotted.

The following variables are listed in the dialog:

Variable	Comments
Isa (Total stator current, phase A, [A])	Current summarized over all parallel paths of a given phase.
Isb (Total stator current, phase B, [A])	
Isc (Total stator current, phase C, [A])	
Isa1 (Stator current of parallel path 1, phase A, [A])	Available for each parallel path of each phase.
. . . . .	
Ir1 (Rotor current of bar 1, [A])	Available for each rotor bar.
. . . . .	

Ua (Supply voltage, phase A, [V])	Can be unavailable if default voltage sources are not in use.
Ub (Supply voltage, phase B, [V])	
Uc (Supply voltage, phase C, [V])	
Pinput (Input apparent power, [VA])	For more details on these variables refer to Appendix A
Pcons (Consumed apparent power, [VA])	
Prb (Resistive losses dissipated on the rotor bars, [W])	
Pre (Resistive losses dissipated on the rotor end rings, [W])	
Ps (Apparent power (real and reactive) consumed by a stator circuit, [VA])	
Pmech (Mechanical power on the rotor shaft, [W])	
Pmf (Time derivative of the magnetic field energy, [VAR])	
Torque (Electromagnetic torque, [N*m])	Equals to one of the variables `Torque_maxwell` or `Torque_vwork` depending on the selected torque calculation method.
Load (Load torque on the motor shaft, [N*m])	
Speed (Rotor angular speed, [rad/s])	
Rotang (Rotor angular position, [rad])	
Torque_maxwell (Electromagnetic torque calculated using Maxwell's stress tensor method, [N*m])	For more details on these variables refer to section 2.3.
Torque_vwork (Electromagnetic torque calculated using Virtual work method, [N*m])	
Fx (Radial electromagnetic force acting between the stator and rotor along x-direction, [N])	
Fy (Radial electromagnetic force acting between the stator and rotor along y-direction, [N])	
User defined variable	All variables stored in the `Userdata` structure are available at the end of this list.

By clicking the **OK** button of the dialog (Figure 7.2), the Matlab-expression is constructed to plot the selected variables appearing in the corresponding line as it is shown in Figure 7.3 for plotting stator currents (first line), torque and load (second line) and rotor speed (third line). The fourth line is not active and, therefore, will not be plotted.

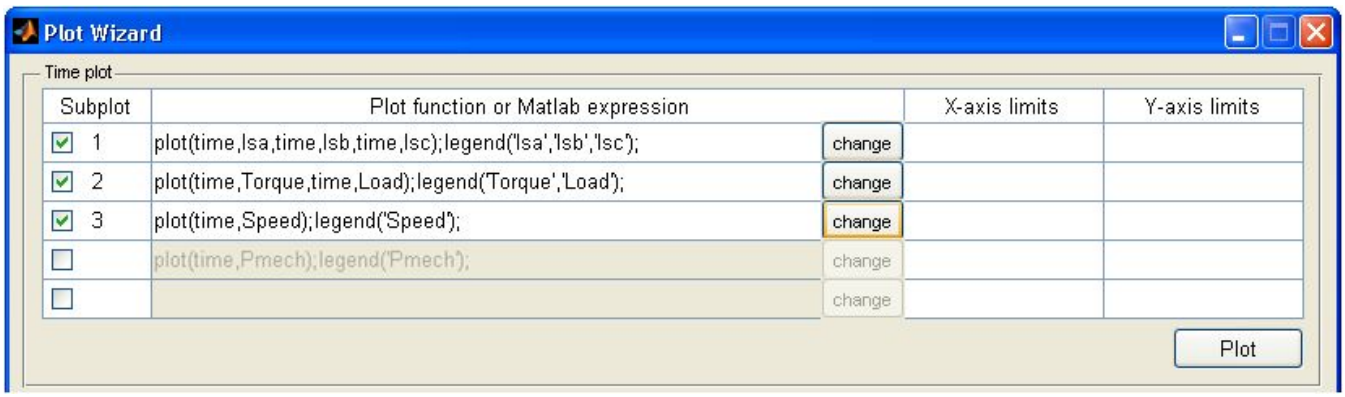


Figure 7.3. Example of using **Plot Wizard** for creating time plots.

When the **Plot** button is clicked, the Matlab figure window with plots of the selected variables will appear (see Figure 7.4).

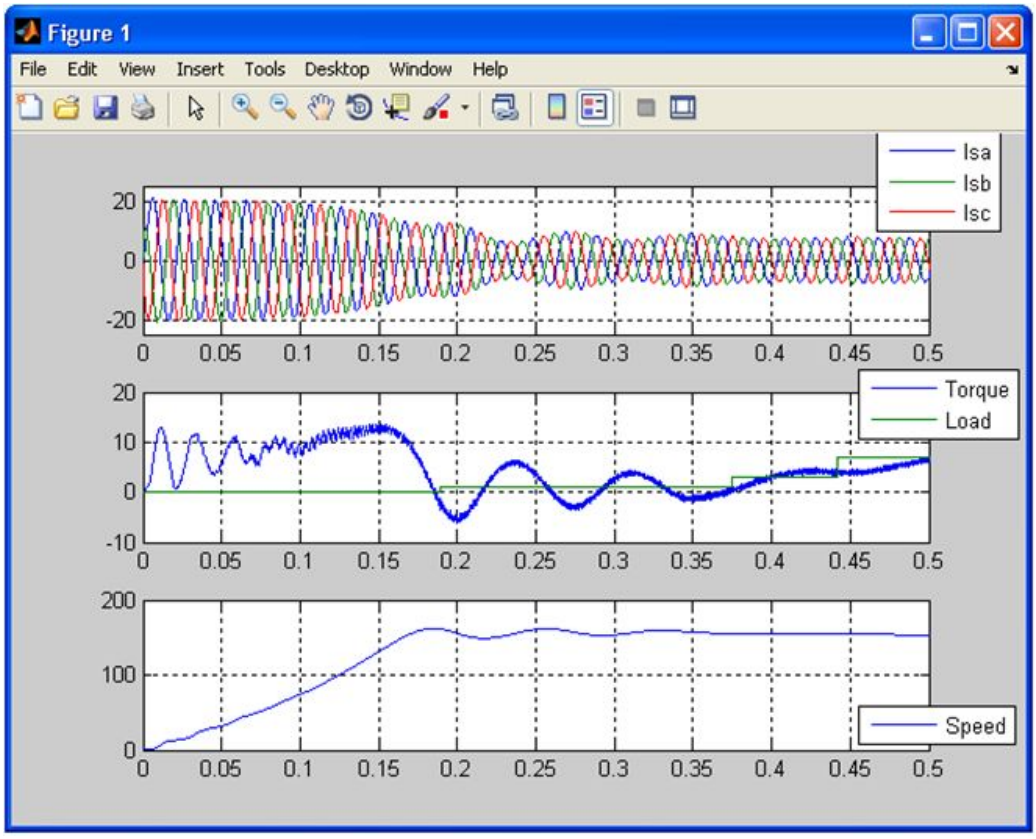


Figure 7.4. Matlab figure window with time plots of the selected variables.

As it is seen, a plotting expression consists of the `plot` function with selected variables used as input arguments. If **Add graph legend** checkbox of the dialog is active, the `legend` function is added to the plotting expression so the graph legend of the corresponding axes will be shown. All plotting expressions are editable, so you can use any Matlab plotting options and functions to change the way plots appear on the screen. You can also change `time` variable to any other variable you would like to use as an input argument of the `plot` function. Moreover, all variables stored in the `Geometry`, `Windings`, `Core`, `Mesh`, `Settings` and `Userdata` structures (see section 5.3) can be used within a plotting expression. For example, to obtain a plot of the resistive losses dissipated on the stator winding, one can use the following plotting expression:

```
plot(time,(Isa.^2+Isb.^2+Isc.^2)*Rs);
```

`Rs` – `Windings` structure field corresponding to the active resistance of the stator winding per phase.

X-axis limits and **Y-axis limits** fields of the **Time plot** panel allow you to set the x-axis and y-axis limits, respectively, to the specified values. Two limit values within a cell are divided by a space, comma [,] or semicolon [;]. If a cell is empty, the limits of the corresponding axis will be chosen automatically.

7.2. Air gap distribution plots.

Air gap distribution plots allow viewing the distribution of the particular quantity over the machine's air gap as well as its frequency spectrum. It is available from the **Air gap distribution plot** panel. This type of plots is only allowed when the regular mesh in the sliding layer of the air gap is used (see section 2.2). **Subplot** checkboxes allow controlling the number of axes or rectangular panes displayed within a current figure window. The corresponding axes are activated or deactivated by a mouse click within a cell of the **Subplot** column. When the subplot is activated, the **"+"** button allows you to choose variables to be plotted from the dialog shown in Figure 7.5. Use Ctrl button to select several variables.

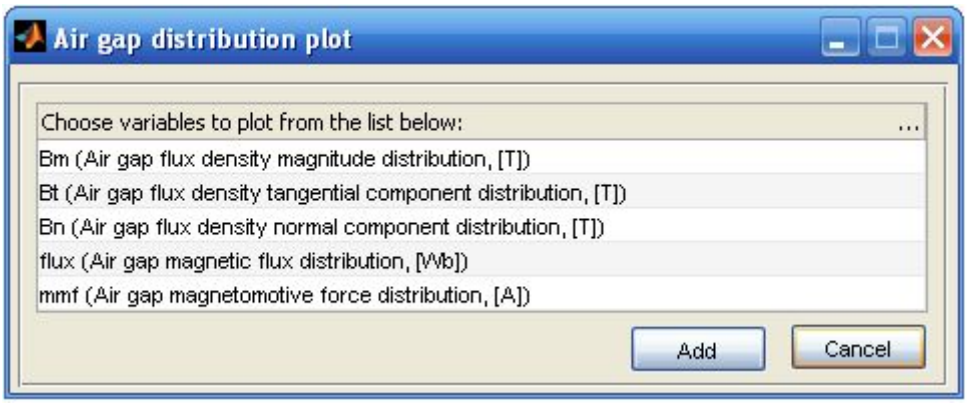


Figure 7.5. Choosing variables to be plotted.

Variable	Comments
Bm (Air gap flux density magnitude distribution, [T])	Defined as $B_m = \sqrt{B_t^2 + B_n^2}$ where B_n and B_t – normal and tangential components of the magnetic flux density in the sliding layer of the air gap, as it is shown in Figure 2.3.
Bt (Air gap flux density tangential component distribution, [T])	B_n and B_t – normal and tangential components of the magnetic flux density in the sliding layer of the air gap, as it is shown in Figure 2.3.
Bn (Air gap flux density normal component distribution, [T])	
flux (Air gap magnetic flux distribution, [Wb])	Integration of the magnetic flux distribution over an air gap always gives zero: $\Phi_{airgap} = l \oint_{airgap} B_n \cdot d\ell = 0$, where l – lamination length in z direction. Air gap magnetic flux distribution is calculated over inner and outer contours as it is shown in Figure 2.3 and the actual values are resulted as the average.
mmf (Air gap magnetomotive force distribution, [A])	Integration of the magnetomotive force distribution over an air gap always gives zero: $F_{airgap} = \frac{1}{\mu_0} \oint_{airgap} B_t \cdot d\ell = 0$ Air gap magnetomotive force distribution is calculated over inner and outer contours as it is shown in Figure 2.3 and the actual values are resulted as the average.

By clicking the **Add** button of the dialog (Figure 7.5), the selected variables are added to the corresponding line separated by commas as it is shown in Figure 7.6. Each line of the **Variables** column is editable, so you can you use Matlab arithmetic operations to obtain desired plots. For example, the second and third lines in Figure 7.6 produce plots of the tangential air gap force density distribution and the radial air gap force density distribution, respectively, where `mu0` – variable corresponding to the permeability of free space. According to (2.6) the tangential air gap force produces the electromagnetic torque so the plot of the electromagnetic torque distribution over an air gap will be obtained.

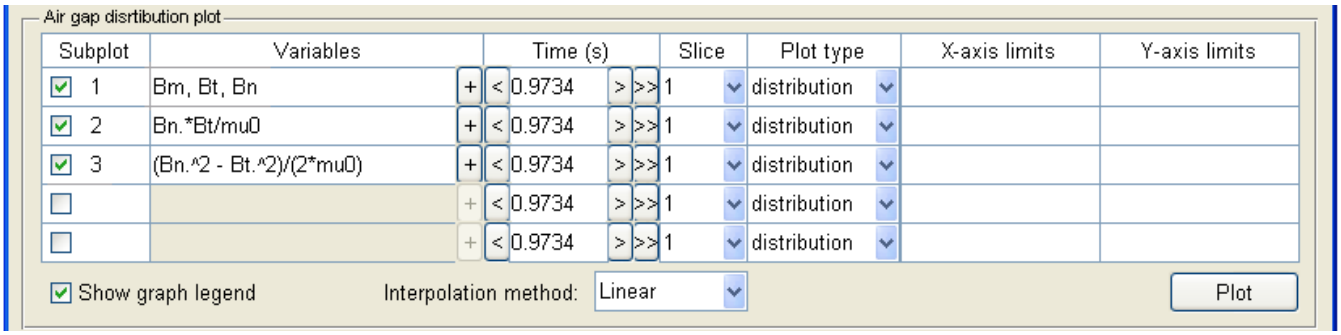


Figure 7.6. Example of using **Plot Wizard** for creating air gap plots.

Time fields of the **Air gap distribution plot** panel allow specifying the time point which the selected variables are plotted for. **X-axis limits** and **Y-axis limits** fields allow you to set the x-axis and y-axis limits, respectively, to the specified values in the same way as for **Time plot** panel. **Slice** fields allow choosing the machine's cross-section which the selected variables are plotted for, if several slices are used.

Besides the air gap distribution, it is also possible to calculate the air gap harmonic components for the selected quantities. To obtain the frequency spectrum, select the **spectrum** item in the corresponding **Plot type** pop-up menu. An example of plotting the air gap distribution of the magnetic flux and its spectrum at time point of 0.9734 seconds in the second slice's cross-section is shown in Figure 7.7. Only first 50 harmonics are shown, since the value specified in the corresponding **X-axis limits** field is 50 (if a minimum limit equals to 0, it can be omitted).

Show graph legend field allows you to choose whether the graph legend will be displayed on the screen, **Interpolation method** field chooses how the missing values are interpolated.

Plotting of the air gap distribution is only possible, if a file containing data for desired time point was previously stored. These data-files are stored when the **Extended data saving** mode is enabled in the MotorAnalysis main window. So, if you are interested in viewing the air gap distribution plots make sure that the corresponding data-files are being stored. The folder used as a source of data-files is specified in the **Data source folder** field located in the bottom part of the **Plot Wizard** window (**Animated plot** panel). By default the source folder is the same as specified in the **Ext. data saving folder** field of the MotorAnalysis main window. You can change it by clicking the button to the right, if needed.

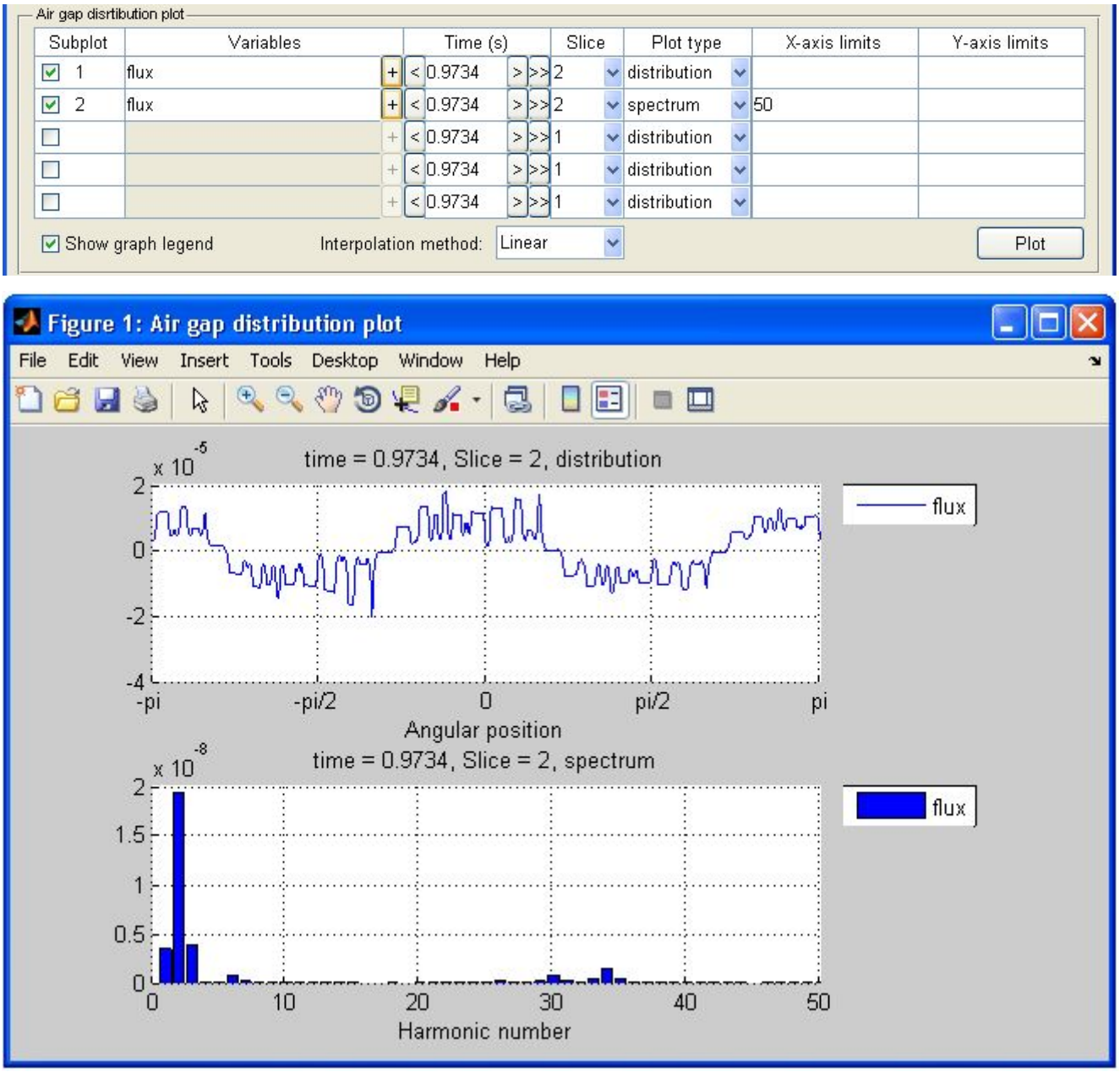


Figure 7.7. Plotting of the air gap distribution of the magnetic flux and its spectrum.

7.3. Cross-section distribution plots.

Cross-section distribution plots are available from the **Cross-section distribution plot** panel of the **Plot Wizard** window. As opposed to two previous plot types, the cross-section distribution for each quantity is plotted in a separated window which contains only one axes. **Figure** checkboxes allow controlling the number of windows appearing when the **Plot** button is clicked. The corresponding figure is activated or deactivated by a mouse click within a cell of **Figure** column. When the figure is activated, the corresponding **Plotted quantity** pop-up menu allows you to choose the quantity to be plotted. If **None** item is selected, the machine's cross-section geometry will be plotted.

The following items are available from the **Plotted quantity** pop-up menu:

- Magnetic vector potential, [T*m];
- Magnetic flux density, [T];
- Magnetic field intensity, [A/m];
- Relative permeability;
- Current density, [A/m^2];
- Stator current density, [A/m^2];
- Rotor current density, [A/m^2];
- Squared flux density, [T];
- Magnetic field energy, [J];
- Magnetic field energy density, [J/m^3];
- Joule loss, [W];
- Joule loss density, [W/m^3];
- Finite element mesh.

Time and **Slice** fields allow specifying the time point and the machine's cross-section, respectively. **Options** field allows showing the magnetic flux lines (if a **flux lines** item is selected) or magnetic flux arrows (if a **flux arrows** item is selected) on the corresponding plot. Flux arrows are plotted such that the direction of the arrow indicates the direction of the flux and the size of the arrow indicates the magnitude of the flux density. **Number of flux lines levels** field allows altering the flux lines density.

X-axis limits and **Y-axis limits** fields allow you to set the x-axis and y-axis limits, respectively. If x-axis and y-axis limits are not specified, their values will be determined by the size of the machine's cross-section. **Z-axis limits** field sets the color scale limits to specified minimum and maximum values divided by a space, comma [,] or semicolon [;].Values between z-axis limits linearly map to the used color scale (colormap). Data values less or greater than specified z-axis limits map to the minimum limit or to the maximum limit, respectively.

An example of plotting the cross-section distribution of the relative permeability at time point of 2 seconds in the first and second slice's cross-sections is shown in Figure 7.8. When the **Plot** button is clicked two windows appear. As it is seen, the **Flux lines** option is chosen for the first figure, so the flux lines are additionally shown in the first window. Since the third figure is not active, the relative permeability for the third slice is not plotted.

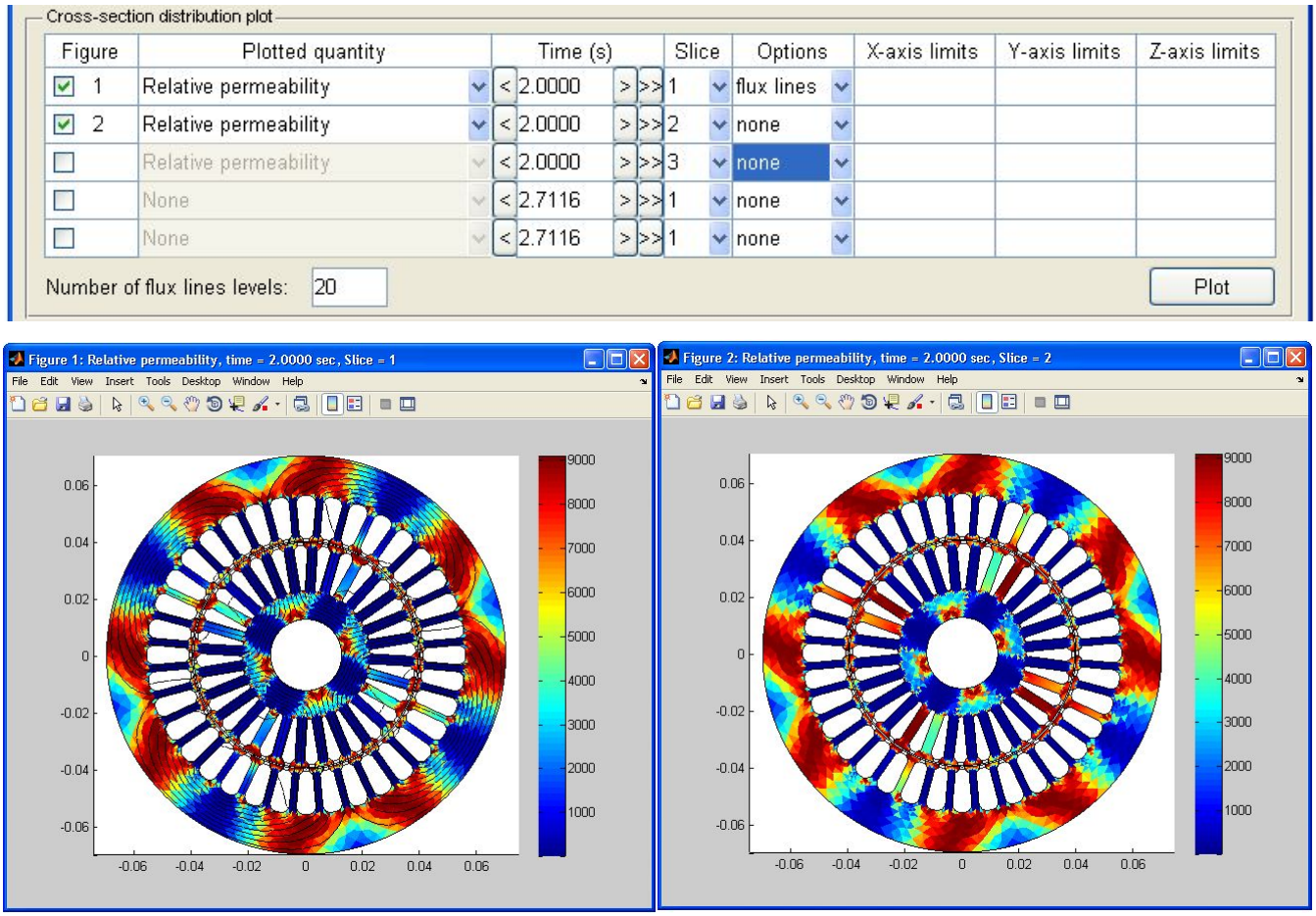


Figure 7.8. Example of using **Plot Wizard** for creating cross-section distribution plots.

Plotting of the cross-section distribution is only possible, if a file containing data for desired time point was previously stored. These data-files are stored when the **Extended data saving** mode is enabled in the MotorAnalysis main window. So, if you are interested in viewing the cross-section distribution plots, make sure that the corresponding data-files are being stored. The folder used as a source of data-files is specified in the **Data source folder** field located at the bottom part of the **Plot Wizard** window (**Animated plot** panel). By default the source folder is the same as specified in the **Ext. data saving folder** field of the MotorAnalysis main window. You can change it by clicking the button to the right, if needed.

7.4. Animation.

Animated plot panel is used to create animated sequences of the air gap distribution plots and/or the cross-section distribution plots. **Animate air gap distribution subplots** and **Animate cross-section distribution figures** checkboxes allow you to choose plot types to be animated. If the **Animate air gap distribution subplots** checkbox is active, all selected subplots of the **Air gap distribution plot** panel will be animated. Similarly, if the **Animate cross-section distribution figures** checkbox is active, all selected figures of the **Cross-section distribution plot** panel will be animated. If **Position figures**

control is selected, the size and location on the screen for all figure windows will be specified automatically so that the windows fit the screen size best.

Skip and **Frame display time** fields allow you to specify the way how frames change each other while the animation is in progress. **Skip** field specifies the number of data-files or frames skipped. So, if **Skip** field is set to 0, data for each calculated time step will be shown on the screen, if **Skip** field is set to 1, data for each second time step will be shown, if **Skip** field is set to 2 – for each third time step and so on. **Frame display time** field specifies the time each frame is displayed on the screen. Note that the least time a frame is displayed on the screen is limited by the animation function runtime. So, if the **Frame display time** field is set to 0, the actual time a frame is displayed on the screen will be the least possible in this case.

Animation start time and **Animation stop time** fields set the time period for the animation. **Data source folder** field specifies the folder with data-files to be animated. The same folder is used for animations and for air gap distribution and cross-section distribution plots. Using animations is only possible, if the files containing data for the time period to be animated were previously stored.

To start the animation, click the **Start animation** button. The current time point, animated quantity and other information is displayed at the top of each window involved in the animation. The animation continues until the time specified in the **Animation stop time** field is reached or until all windows involved in the animation are closed. You can also pause the animation using **Pause** button. While the animation is in progress or paused, you can use all Matlab figure tools changing, for example, the scale and position of the plots.

8. ADDITIONAL SETTINGS

MotorAnalysis additional settings stored in M-file MotorAnalysisSettings.m include the following:

Nonlinear solver settings:

`substepmax` – maximum number of Newton algorithm iterations. If the number of Newton algorithm iterations exceeds `substepmax` value, it is said that the solution does not converge. The subsequent action will depend on the `NonconvHnd` variable value.

`relaxmin` – minimum relaxation factor. If a solution does not converge at the current Newton algorithm iteration, the nonlinear solver damps down the search step with relaxation factor $0 < \alpha < 1$. The relaxation factor iteratively decreases until the convergence occurs. The smaller relaxation factor, the worse convergence. If the relaxation factor becomes less than `relaxmin`, it is said that the solution does not converge. The subsequent action will depend on the `NonconvHnd` variable value.

`convreport` – enables (if `'on'`) or disables (if `'off'`) displaying the convergence information (i.e. Newton algorithm iteration number, the maximum and average residual and the relaxation factor) in the Matlab Command window.

`NonconvHnd` – determines the action the nonlinear solver takes when the solution does not converge. It can be set to the following values: 1 – continue simulation with the accuracy reached regardless the value specified in the **Convergence tolerance** field of the main window; 2 – produce the dialog allowing the user to decide either to continue or interrupt the simulation; 3 - interrupt the simulation and report an error.

Virtual work method settings:

`vd` – default virtual displacement value for force calculation using the Virtual work method.

`vad` – default virtual angular displacement value for torque calculation using the Virtual work method.

If `vad=[]`, the torque calculation using Virtual work method is disabled and `Outputs.Torque_vwork` variable is filled with zero values.

APPENDIX

Appendix A. Power balance and accuracy of the results.

If the power balance is satisfied, it means that the input apparent power delivered by all current and voltage sources equals to the consumed apparent power:

$$P_{input} = P_{cons}$$

The input power is calculated from voltages and currents of all power sources:

$$P_{input} = \sum_{n=1}^{m} i_n u_n$$

The consumed power is defined as follows:

$$P_{cons} = P_{rb} + P_{re} + P_s + P_{mech} + \frac{\Delta W_{mf}}{\Delta t}$$

where P_{rb} and P_{re} – resistive losses dissipated on the rotor bars and rotor end rings, respectively, P_s – apparent power (real and reactive) of the stator electrical circuit, P_{mech} – mechanical power on the shaft, $\Delta W_{mf}/\Delta t$ – magnetic energy time derivative.

All these quantities are available from the **Plot Wizard** tool. Difference between the input power and consumed power allows estimating the reliability of the simulation results.

Appendix B. Out of memory errors handling and choice of Matlab version.

MotorAnalysis has been currently tested with Matlab 2006a, Matlab 2009b and Matlab 2013a. Graphical user interface of MotorAnalysis does not work with Matlab 2006a. Though using MotorAnalysis without graphical interface is still possible, this issue is out of the scope of this manual. Matlab 2009b demonstrated better performance comparing with Matlab 2013a in terms of number of finite elements and speed. Matlab 2009b worked stably with 10 slices and the mesh in each slice consisting of ~17,000 nodes and ~ 34,000 elements, so the total amount of mesh nodes and finite elements were ~170,000 and 340,000, respectively. The same mesh with Matlab 2013a caused the “out of memory” error.

“Out of memory” errors can occur when using the mesh of too large size. According to Matlab help documentation, to avoid “out of memory” errors when using MotorAnalysis, you can increase the size

of the swap file or add more memory to the system. Sometimes, restarting Matlab when “out of memory” error occurs also helps.

Appendix C. Determining the initial conditions.

The ‘Fast’ solver is usually used to determine the initial conditions, i.e. rotor speed, stator and rotor currents and voltages, avoiding the lengthy simulations, e.g., the simulation of the startup transient. Since the number of slices and the time step cannot be changed, it still may take quite long time to determine the initial conditions, especially when several slices are used. `SetInitCnd` function is designed to facilitate determining the initial conditions in such cases.

```
SetInitCnd(sim_old,sim_new,nSlices,timestep)
```

The function creates the new simulation file with the name specified in `sim_new` using the initial conditions of the `sim_old` simulation file. The number of slices and the time step of the new simulation file are set to `nSlices` and `timestep`, respectively. You can use one slice and the larger time step for `sim_old` to determine the initial conditions and then set the number of slices and the time step to the desired values creating the new simulation file `sim_new`.