MotorAnalysis
Design and Analysis of Induction Motors

Version 2.2
User Manual

Vladimir Kuptsov
2016
1. INTRODUCTION

Thank you for your interest in MotorAnalysis. MotorAnalysis is a FREE electric motor design software. The current version of MotorAnalysis includes squirrel cage induction motors. MotorAnalysis is based on automated finite element analysis (FEA) simulations and establishes a complete set of tools for design and analysis of induction motors. MotorAnalysis is a MALTAB-based application but stand-alone version working without MATLAB is now also available.

We believe that our work will help to save our environment supporting green technologies like electric vehicles or wind turbine energy generation as well as enhancing the efficiency and effectiveness of electric 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 and etc. 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:

\[ \nabla \times \left( \frac{1}{\mu_r} \cdot \nabla \times A \right) = J \]  
\[ J = \sigma \cdot \frac{U}{l} - \sigma \cdot \frac{\partial A}{\partial t}, \]

where \( A \) – magnetic vector potential (\( B = \nabla \times A \), \( B \) – magnetic flux density), \( \mu_r \) – relative magnetic permeability, \( J \) – current density, \( \sigma \) – conductivity, \( U \) – voltage applied to the FE area, \( l \) – length in z direction.

According to (2.2) the current density consists of two components, the first one is caused by external voltage and the second one is induced by varying magnetic field.

There are two methods used in MotorAnalysis to solve the problem defined by (2.1) and (2.2). First method is called a time-stepping finite element method (FEM) or transient analysis. The idea behind the time-stepping FEM is representation of the time derivative of the magnetic vector potential as follows:

\[ \frac{\partial A}{\partial t} = \frac{A_n - A_{n-1}}{\Delta t}, \]

where \( \Delta t \) – time step, \( A_n \) and \( A_{n-1} \) - magnetic vector potentials on n-th and (n-1)-th time steps.

Assuming zero initial magnetic vector potentials the iterative procedure can be used to find the magnetic vector potentials for each time step. This method allows taking into account the nonlinearity of iron and the rotation of the rotor and can be used for analysis of both transients and a steady state.

Another method used in MotorAnalysis is called a time-harmonic finite element method or AC analysis. It is based on the assumption of the sinusoidal time-dependence of the field which allows the computation time to be reduced radically comparing with the time-stepping FEM. According to the
time-harmonic FEM the magnetic vector potential, current density and voltage of (2.1) and (2.2) are considered as complex quantities and the time derivative of the magnetic vector potential becomes as follows:

$$\frac{\partial A}{\partial t} = j\omega A,$$

where \(\omega\) – angular supply frequency, \(j\) – complex unit.

To take into account the rotation of the rotor in the time-harmonic FEM, the rotor conductance is multiplied by the slip the same way as it is applied in the standard motor equivalent circuit. The nonlinearity of iron is taken into account by means of the equal average energy method. Note that the time-harmonic FEM provides only steady state motor parameters and cannot be used for analysis of transient regimes.

In both methods the 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:

$$K \cdot X = F,$$

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.

Figure 2.1. Generalized algorithm of solving a nonlinear problem with time-stepping FEM.
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, when the residual vector of the guess $\hat{X}$ will be defined as:

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

(2.4)

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:

$$ \max(|r|) < tol, $$

(2.5)

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

Generalized algorithm used in MotorAnalysis for a nonlinear case of the time stepping method is shown in Figure 2.1. An outer loop of the algorithm represents integration over time with step $\Delta t$. 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 while the time-stepping FE simulation is in progress 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. In MotorAnalysis the air gap always has odd number of mesh layers (3, 5, 7 or 9) and the sliding layer is always located at the centre of the air gap (see Figure 2.2).

![Figure 2.2. Three mesh layers in the air gap and the sliding layer at the center of the air gap.](image)
To reduce number of finite elements and as a consequence reduce computation time, periodic/antiperiodic boundary conditions can be applied. Using this type of boundary conditions is based on periodicity of the magnetic field of the motor.

![Figure 2.3. Using boundary conditions.](image)

As it is seen from the example in Figure 2.3(a), the magnetic vector potentials repeat two times along the circle. Moreover, absolute values of the magnetic potential values repeat four times. Using this property of the field we can compute magnetic potentials for one quarter and then find the magnetic field for the whole cross-section as shown in Figure 2.3(b). In this case we use antiperiodic boundary conditions. If magnetic potentials are computed for the half, when periodic boundary conditions are applied. Number of periodicities of the magnetic field in MotorAnalysis is called periodicity factor. Periodicity factor basically equals to a number of pole pairs; in the presented example the periodicity factor is 2. Note that using periodic or antiperiodic boundary conditions are only possible if geometry of the cross section repeats together with the field. It means that to apply periodic boundary conditions the number of stator and rotor slots should be divided by number of pole pairs, while for antiperiodic boundary conditions the number of stator and rotor slots should be divided by number of poles.

### 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:

\[
T = -\frac{l \cdot (D_r + l_\delta)^2}{4\mu_0} \int_0^{2\pi} B_n B_t d\phi,
\]

where \(T\) – electromagnetic torque, \(l\) – lamination length in \(z\) direction, \(D_r\) – rotor diameter, \(l_\delta\) – air gap length, \(\mu_0\) – permeability of the free space, \(B_n\) and \(B_t\) – normal and tangential components of the magnetic flux density in the air gap, as it is shown in Figure 2.4. This expression is used in MotorAnalysis for calculation of the torque with the Maxwell stress tensor method.

The integration contour used in Eq. (2.6) always lies inside the sliding layer of the air gap mesh, as it is shown in Figure 2.4.

\[
T = -\frac{l \cdot (D_r + l_\delta)^2}{4\mu_0} \int_0^{2\pi} B_n B_t d\phi,
\]

Figure 2.4. 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.

MATLAB code for calculation of the electromagnetic torque with the Maxwell stress tensor method is presented in 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:

\[
F_u = -\frac{\partial W}{\partial e}_{\psi=\text{const}},
\]

where \(\partial e\) - virtual displacement, \(\partial W\) - change of the magnetic energy between initial and final positions of the rotor.

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

\[
M = -\frac{\partial W}{\partial \phi}_{\psi=\text{const}},
\]

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 file MotorAnalysisSettings.m.

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.

When the time-stepping FEM is applied the calculated torque value is used to determine the current rotor position. The rotor rotation is defined as follows:

\[
T = T_{\text{load}} + J \frac{d\omega}{dt},
\]

\[
\omega = \frac{d\phi}{dt},
\]

where \(T_{\text{load}}\) – load torque on the motor shaft, \(J\) – combined rotor and load moment of inertia, \(\omega\) - rotor angular speed, \(\phi\) – 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 2.5 illustrates the multi-slice approach.

![Cross sections of the machine](image)

Figure 2.5. 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.

In MotorAnalysis the multi-slice approach is only utilized for the time-stepping FEM (transient analysis). For time-harmonic (AC) analysis only 2D FEM is used.

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, when using several slices will not be reasonable, since the result will be the same as with one slice used.
2.4. Iron loss calculation

Iron core losses in MotorAnalysis are calculated during the post processing so it is assumed that the iron losses do not influence the magnetic field distribution. The iron loss estimation is based on the Steinmetz equation together with the eddy current term:

\[ P = K_h f^\alpha B_m^\beta + K_e (f \cdot B_m)^2 \]  

(2.7)

First term in 2.7 represents hysteresis losses, the second one – eddy current loss term. Coefficients \( K_h, \alpha, \beta \) and \( K_e \) are determined by fitting equation 2.7 to the measured iron loss data from the manufacturer at different frequencies.

When the time-harmonic FEM (AC analysis) is applied the iron losses are estimated at single frequency; for stator this is a supply frequency, for rotor – supply frequency multiplied by slip. For the time-stepping FEM (transient analysis) the average value of iron loss is calculated over the specified period of time and FFT is used to take the non-sinusoidal shape of the flux density waveform into consideration. It allows including into analysis the iron loss contribution from higher harmonics of the flux density such as slot harmonics and PWM produced harmonics.
3. GETTING STARTED

If you do not have MATLAB® refer to the next section to learn how to get started with standalone version of MotorAnalysis.
MotorAnalysis is a MATLAB-application and to use it you should have MATLAB® installed on your computer. MotorAnalysis may not work with versions of MATLAB® earlier than MATLAB R2009b. MATLAB PDE Toolbox is no longer required.
To start MotorAnalysis enter motoranalysis in MATLAB Command Window. Note that MATLAB Current Folder should be changed to the folder with the application files. MotorAnalysis main window shown in Figure 3.1 will appear.

3.2. Using MotorAnalysis standalone version.
To use standalone version of MotorAnalysis you should first install the MATLAB Compiler Runtime (MCR) if it is not installed. To install MCR run file MCRInstaller.exe. It may be required to restart your computer after installation. To start MotorAnalysis run file motoranalysis.exe. MotorAnalysis main window shown in Figure 3.1 will appear.

Important notice: standalone version of MotorAnalysis has several limitations comparing with MATLAB version. These limitations are connected with that what the MCR application is not able to execute m-files which were not included while the application was built. Note that several m-files are provided together with the program. These m-files were wrapped into motoranalysis.exe while the application was built. If you change any of these m-files or add your own m-file it will not have any effect.

Limitations of MotorAnalysis standalone version:
1. You cannot use your own iron loss data files (see section 4.3) except those provided together with the program (NO20sura_ironloss.m, M1929Ga_ironloss.m, M1529Ga_ironloss_coef.m);
2. Transient analysis simulation script functions (see chapter 10) cannot be used except those provided together with the program (simscript_steadystate.m, simscript_pwm.m, simscript_SinglePhase.m);
3. Electrical circuit functions (see chapter 11) cannot be used except those provided together with the program (StarConnection.m, DeltaConnection.m, InverterCircuit.m, SinglePhaseConnection.m);
4. You cannot change MotorAnalysis additional settings stored in file MotorAnalysisSettings.m (see chapter 12).
If you use standalone version, keep in mind these limitations while reading this manual since some options described in the manual are available only for MATLAB version of MotorAnalysis.
Using toolbar buttons you can get access to all MotorAnalysis tools to create and analyze the motor prototypes such as Geometry Editor, Iron Core Property Editor, Windings Property Editor, Mesh Editor and Plot Wizard. These are also available from the menu Desktop. Next chapters provide the detailed information on using these tools.

There are four analysis types available in MotorAnalysis. These are Equivalent Circuit Analysis, AC Analysis, Transient Analysis and Advanced Equivalent Circuit (EC) Analysis. To choose the analysis type, use the corresponding button under toolbar.

Figure 3.1. MotorAnalysis main window.
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 and finite element mesh represented in Geometry Editor, Iron Core Property Editor, Windings Property Editor and Mesh Editor should be defined before any analysis is started and cannot be changed afterwards.

**Open As New (Empty) Simulation File** item of the File menu allows you to create a new simulation file based on another simulation file. All motor parameters, material properties and analysis setting will be copied in the new simulation file while the finite element mesh and analysis results will not be included. This is a convenient way to create several motor prototypes (each motor prototype corresponds to the separate simulation file) changing one or several motor parameters to find an optimal design.
4. CREATING MOTOR PROTOTYPES

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:

![Geometry Editor window](image)

Figure 4.1. Geometry Editor window.

Two Slot type pop-up menus allow you to choose the 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.

Two Bottom corner type menus allow you to choose the rotor and stator slot bottom shape making it whether of round shape or rectangular shape with round corners.

Dimensions and geometry parameters of the motor presented in the Geometry Editor window are shown in Figures 4.2 - 4.7.
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.
There is also an option of stator tooth shape optimization in MotorAnalysis. According to some publications the stator tooth shape optimization can lead to reduction of the torque-ripple and the noise radiation providing the same average torque *.

Changing stator tooth shape can be achieved by setting up tooth edge chamfers in Geometry Editor as it is shown in Figure 4.7.

4.2. Windings Property Editor.

Windings Property Editor window is shown in Figure 4.8:

![Windings Property Editor Window](image)

Figure 4.8. Windings Property Editor window.

**Stator winding**

**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 11 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.

**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.
There are two methods to specify stator winding layout: automatic and manual. When you choose Layout input method as Automatic you should also specify Winding type (Lap or Concentric) as well as Number of pole pairs and Coil span in slot pitches. When the layout input method is manual, values in these three fields do not matter. Winding layout is displayed as three tables corresponding to each phase. First column of each table is a coil number; second and third columns indicate the slot numbers of the coil and the fourth column is the number of the parallel path (number of the group of coils) corresponding to the coil. The same number of parallel path corresponds to coils of the same phase connected in series. Colored representation of the winding layout is displayed in the right part of the Windings Property Editor window, where the sign "+/-" specifies the forward and return direction of the conductors within a slot, the letter following the sign specifies a phase and the phase is followed by the number of parallel path.

End winding inductance field specifies the leakage inductance of the stator winding end-turns per phase. End winding inductance can be entered manually or calculated automatically based on the winding configuration and motor dimensions depending on the selected End winding inductance input method (Manual or Automatic). Note that when you choose the end winding inductance input method as automatic you should also specify End winding axial overhang.

Winding phase resistance field specifies active resistance of the stator winding per phase. Winding phase resistance can be entered manually or calculated automatically based on the winding configuration and motor dimensions depending on the selected Winding phase resistance input method (Manual or Automatic). Note that when you choose the winding phase resistance input method as automatic you should also specify Slot fill factor and Winding material conductivity.

Slot fill factor field specifies the ratio of the area of all conductors of a slot to the total slot area. If the winding phase resistance input method is manual then filling the Slot fill factor and Winding material conductivity fields is optional and required only for calculation of the Joule loss distribution over stator slots. Similarly, if the end winding inductance input method is manual then filling the End winding axial overhang field is optional.

Rotor winding

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

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

End ring inductance field specifies the leakage inductance of the arch of the rotor end ring measured between two adjacent rotor bars. End ring resistance field specifies an active resistance of the arch of the rotor end ring measured between two adjacent rotor bars. End ring inductance and resistance can be entered manually or calculated automatically depending on the selected End ring parameters input method (Manual or Automatic). Note that when you choose the end ring parameters input method as
automatic you should also specify end ring dimensions (**End ring outer diameter**, **End ring inner diameter** and **End ring thickness** fields).

**Temperature effects on resistance**

Stator winding resistance, rotor end ring resistance as well as stator and rotor winding material conductivities are given for 20°C and, by default, temperature of both stator and rotor windings is equal to ambient temperature 20°C. You can take into account how resistance depends on the temperature by specifying the temperature coefficient of resistance and winding temperature for both stator and rotor (**Stator winding temperature**, **Rotor winding temperature**, **Stator winding temperature coefficient of resistance** and **Rotor winding temperature coefficient of resistance** fields).

The generic formula for temperature effects on resistance is as follows:

\[ R = R_{20°C} \times [1 + \alpha \times (T - 20°C)] , \]

where

R – resistance at temperature T;
R_{20°C} – resistance at 20°C;
\alpha – temperature coefficient of resistance;
T – winding temperature.

By default, the temperature coefficient of resistance is zero i.e. resistance does not depend on the temperature.
4.3. Iron Core Property Editor.

Iron Core Property Editor Window is shown in Figure 4.9.

![Iron Core Property Editor Window](image)

- **Core material** field is optional and can be left empty.
- **Linear relative permeability** field specifies the relative permeability used by linear solver.
- **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 through the stacking factor.
- **BH-curve points** table defines the measured points of the BH-curve. **Load BH-curve** button allows you to load the BH-curve from another simulation file; **Clear table** button cleans up all table cells.

BH-curve plot interpolated with cubic spline is presented in the right part of the Iron Core Property Editor Window. 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.
Iron loss panel allows you to include into analysis eddy current and hysteresis losses in the iron core. Iron losses are specified by Iron loss data file. Use the Change button to select iron loss data file. If iron loss data file is not specified (you can delete it using Delete button) it is assumed that the iron losses are zero. Iron loss data file is an m-file which contains the measured iron loss data at different frequencies and magnetic flux density values. Reference to the file NO20sura_ironloss.m as an example to define iron losses for your own iron core material. MotorAnalysis uses an iron loss data file to determine coefficients of equation 2.7. You can also directly define iron loss coefficients in the iron loss data file (see file M1529Ga_ironloss_coef.m). Thus there are two ways to define iron losses: through measured iron loss data or using iron loss coefficients.
4.4. Mesh Editor.

Mesh Editor window is shown in Figure 4.10.

**Number of layers in air gap** field specifies the total number of finite element layers placed in the air gap. You can specify only odd number of finite element layers from 1 to 9 (refer to section 2.2 of this manual for more details).

**Mesh growth rate** field determines the mesh growth rate away from a small part of the geometry. The mesh growth rate should be strictly between 1 and 2 and cannot be equal to either of its bounds, 1 and 2. The default value is 1.4, i.e., a growth rate of 40%.

**Air gap mesh quality** field specifies the quality of the mesh in the air gap region (Low, Medium (by default) or High). The mesh of the highest quality consists of equilateral triangles which is practically not possible for complex geometry. The lower air gap mesh quality usually leads to the less number of mesh triangles and faster computational speed sacrificing the accuracy.

Use the **Mesh growth rate** and the **Air gap mesh quality** fields to control the number of mesh triangles.

![Mesh Editor window](image-url)

Figure 4.10. Mesh Editor window.
**Number of slices** field specifies the number of slices used by the multi-slice FEM during time-stepping FE simulation, i.e. the number of the machine’s cross sections in which the magnetic field is simultaneously calculated. For more details on the multi-slice FEM refer to section 2.4 of this manual. Note that during AC analysis only one slice is used (2D FEM) regardless of the **Number of slices** field.

You can apply periodic or antiperiodic boundary conditions choosing corresponding item from the **Boundary conditions** pop-up menu. By default, boundary conditions are not used (**None** item of the **Boundary conditions** pop-up menu is chosen). **Periodicity factor** is basically a number of pole pairs. For more details on using boundary conditions refer to section 2.2 of this manual.

**Mesh in shaft region** checkbox specifies whether or not the rotor shaft is included into analysis.

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 FEA significantly depends on the shape of finite elements especially within and close to the air gap region. The most accurate results are obtained provided that the shape of finite elements is as close as possible to the equilateral triangle shape. If the shape of finite elements is significantly distorted, try to change the **Mesh growth rate** and the **Air gap mesh quality** properties.

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 dimensions can only be seen on the enlarged scale as it is shown in Figure 4.11. In this case a slot corner radius is inconsistent with a slot width.

**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.

![Figure 4.11. Example of incorrect motor dimensions.](image-url)
5. EQUIVALENT CIRCUIT ANALYSIS

Equivalent circuit analysis allows the user to quickly estimate most commonly used motor parameters like voltage, current, power, torque, power factor, efficiency and losses for different operating points as well as compute the efficiency map. This type of analysis is the fastest but the accuracy is lower comparing with other analysis types.

5.1. General description.

The equivalent circuit analysis consists of No-load test and Load test and based on determination of equivalent circuit parameters of the induction machine. The equivalent circuit is shown in Figure 5.1.

![Equivalent circuit of the induction machine](image)

Figure 5.1. Equivalent circuit of the induction machine.

The No-load test is used to determine how the magnetizing inductance \( L_m \) depends on the magnetizing current \( I_m \). For induction machine this dependence is nonlinear. The Load test is used to calculate parameters of the equivalent circuit shown in Figure 5.1 for the specified operating point. Once parameters of the equivalent circuit are found the user can examine how motor parameters depend on different operating conditions.

5.2. Running Equivalent Circuit Analysis.

The view of the main window when Equivalent Circuit Analysis is chosen is shown in Figure 5.2. Note that No-load test and Load test are performed separately. No-load test should be performed first.

5.2.1. No-load Test.

No-load test is based on the nonlinear magnetostatic FEA assuming zero current in rotor bars during the test. The accuracy of the FEA solution is defined by the Convergence tolerance of FEA solution field; default value is 1e-005.

To run No-load test you should first specify stator current values used during the test in the Stator current values field. Since it is assumed that there is no rotor current during the no-load test \((I_2=0)\), the stator current is basically equal to the magnetizing current. You can use different MATLAB expressions
filling **Stator current values** field, separating the values with blanks or commas. Surround the entire list of values with square brackets, [ ].

![MotorAnalysis main window for Equivalent Circuit Analysis](image)

**Figure 5.2.** MotorAnalysis main window for Equivalent Circuit Analysis.

Examples of filling **Stator current values** field:

- [10 20 30 40 50 60 70 80 90 100]
- 10:10:100
- [1:2:10 15:5:50 60:10:100]

Note that for the second example using square brackets is not necessary.
To run the no-load test, click the **Run No-load Test** button. As the test is running the computed magnetizing inductance values will appear in the **Magnetizing inductance vs. magnetizing current** table. When the no-load test is completed you can plot the result with the **Plot Lm vs. Im** button. The example of \( L_m \) vs. \( I_m \) curve is shown in Figure 5.3. If there is no enough values in the table, add new values to **Stator current values** field and run No-load test once again. New values of the magnetizing inductance will be added to the table. Note that the stator current values for which the no-load test has been already performed will be ignored.

![Graph showing \( L_m \) vs. \( I_m \)](image)

Figure 5.3. Example of the magnetizing inductance vs. magnetizing current plot.

**Lm interpolation method** pop-up menu specifies the way how intermediate values of the magnetizing inductance are interpolated (linear or spline) during analysis. Be careful choosing spline interpolation since it can behave unexpectedly for incomplete no-load test data – always examine \( L_m - I_m \) plot for undesired distortion.

**Clear table** button cleans up all no-load data table cells.

### 5.2.2. Load Test.

Load test is based on the nonlinear time-harmonic FEA. The accuracy of the FEA solution is defined by the **Convergence tolerance of FEA solution** field; default value is 1e-005.

You should first specify the operating conditions for which the load test is performed: define RMS phase current, supply frequency and rotor speed (or slip). By default the speed is set to zero so the
standard locked rotor test can be performed. Depending on your needs and your experience you can compute equivalent circuit parameters for any arbitrary speed. It is recommended to choose the current value close to the nominal motor current. Keep in mind that $L_1$, $L_2$, $R_2$ and $R_m$ values (see Figure 5.1) and consequently the computed motor parameters will depend on the chosen speed and current values used for the load test. Then the load test is completed the parameters of the equivalent circuit will be displayed in corresponding fields of the window. **Input values** to the left indicate the operating conditions for which the equivalent circuit parameters are computed.

In some cases the load test fails to compute equivalent circuit parameters. It happens when rotor current $I_2$ is too small to correctly estimate $L_2$ value. Try to change the operating conditions for which the load test is performed, for example, decrease the supply current or increase slip, and run the load test again.

### 5.3. Viewing Equivalent Circuit Analysis results.

The view of the **PlotWizard** window when Equivalent Circuit Analysis is chosen is shown in Figure 5.4.

Use the top panel to plot standard motor parameters like those shown in Figure 5.5. To begin the analysis, specify the power supply properties: 3-phase voltage source or current source, RMS phase voltage (or current) and supply frequency, then choose variables you want to plot and click the **Plot** button, the MATLAB figure window with plots of the selected variables will appear (see Figure 5.5). The figure shows example for the 3-phase voltage source supply, therefore the RMS voltage plot is shown as a straight horizontal line.

You can adjust x-axis limits of the plot using controls to the left. **Number of points** field determines the number of points in which the chosen variables are computed. More points lead to smoother plots but increase computation time.

As it is shown in Figure 5.5, each variable has its own y-axis and different scales. Unselect **Multiple y-axes** checkbox if you want to use the same scale along y-axis to compare several variables.

Pay attention to the **Interpolation method** pop-up menu of the **Model setup** panel while doing the analysis. It determines the way intermediate values of the magnetizing inductance are interpolated (the same as the **L_m interpolation method** pop-up menu of the main window). Linear interpolation decreases the computation time while the nonlinear (spline in this case) interpolation is more accurate. Note that the nonlinear interpolation is prone to distortion of the magnetizing inductance vs. magnetizing current relationship in case of lack of no-load test data. To prevent that, examine the $L_m - I_m$ plot with spline interpolation chosen (see Figure 5.3) and use additional no-load test points if needed.

Another option of the Equivalent Circuit Analysis is creation of the efficiency map. The efficiency map shows the maximum efficiency which can be reached for the certain speed and load torque. The example of the plot of the efficiency map for motor and generator modes is shown in Figure 5.6. Along with the efficiency map some additional plots of the supply voltage, current, power, supply frequency,
slip as well as maximum power (power envelop) are available (see Figure 5.7). **Maximum RMS phase voltage**, **Maximum RMS phase current** and **Maximum input power** fields define supply voltage, current and power limitations applied while the efficiency map is computed. If the **Maximum input power** field is left empty then the input power is not limited (limited only by maximum voltage and current).

Figure 5.4. PlotWizard window for Equivalent Circuit Analysis.
Figure 5.5: Equivalent Circuit Analysis Plot.
Use the **Mechanical losses coefficients** field to apply mechanical losses while the efficiency map is computed. Mechanical losses coefficients are stored in a vector whose elements are the coefficients in descending powers of the mechanical loss equation. For mechanical losses coefficients shown in Figure 5.4 the mechanical loss equation is as follows:

\[ W_{\text{mech.loss}} = 0.0001\omega^2 + 0.0065\omega + 5.9, \]

where \( \omega \) – rotor speed.

Number of points (speed-torque pairs) at which the efficiency map is computed is defined by the **Map resolution** panel. Select **Add efficiency map for generator mode** checkbox if you need efficiency maps for both motor and generator modes; otherwise the efficiency map only for motor mode will be computed. Click the **Create Efficiency Map** button to compute the efficiency map. Make sure that the **Interpolation method** pop-up menu is properly setup as described above.

Note that in Equivalent Circuit Analysis the motor parameters are computed assuming that \( L_1, L_2, R_1, R_2 \) and \( R_m \) do not change while \( L_m \) changes depending on the magnetizing current according to no-load test data. In reality \( L_1, L_2, R_2 \) and \( R_m \) values more or less depend on rotor speed and saturation. AC analysis described in the next section provides more accurate results since it can take into account variations of \( L_1, L_2, R_2 \) and \( R_m \) values at different operating conditions.

![Efficiency map for motor and generator modes.](image)

Figure 5.6. Efficiency map for motor and generator modes.
Figure 5.7. Additional plots corresponding to the efficiency map shown in Figure 1.6.
6. AC ANALYSIS

AC Analysis is based on the time-harmonic FEA and the equivalent circuit but provides more accurate results comparing with Equivalent Circuit Analysis since it can take into account variation of equivalent circuit parameters at different operating conditions. Besides standard motor design parameters like voltage, current, power, torque, power factor, efficiency and losses, AC Analysis allows the user to obtain air gap and cross-section distribution plots of various quantities such as airgap flux and mmf, flux density, permeability, current density and etc.

6.1. Running AC Analysis.

The view of the main window when AC Analysis is chosen is shown in Figure 6.1. AC Analysis is performed for the specified operating points defined by the Power Supply (3-phase voltage source or 3-phase current source supply), RMS phase voltage (RMS phase current) and Supply frequency fields. The speed can be defined whether by the number of rotations per minute or by slip choosing corresponding item from the X-axis variable pop-up menu. You can define a set of speed (slip) values in the Speed values (Slip values) field (see Figure 6.1) as well as a single value. It is possible to use different MATLAB expressions filling this field, separating values with blanks or commas. Surround the entire list of values with square brackets [ ]. Note that only 2D FEA is used for AC analysis hence the Number of slices field of Mesh Editor has no effect on AC analysis (only one slice is used).

To start the analysis, click the Run AC Analysis button (green triangle) on the MotorAnalysis main window toolbar. The analysis will be gradually performed for all values defined in the Speed values (Slip values) field. To stop the analysis click the Stop AC Analysis button which is to the right of the Run AC Analysis button. The analysis will be stopped when the calculation for the current operating point is completed. It is not recommended to interrupt the analysis using Ctrl+C shortcut otherwise the simulation file may be corrupted and all data will be lost.

There is an option allowing the user to make an optimal choice of speed (slip) values automatically using results of Equivalent Circuit Analysis (Automatic speed (slip) values selection panel). To use this option, first choose the AC analysis target quantity, specify number of speed (slip) values and minimal and maximal values of the used speed (slip) range. Then click the Fill button right to the Speed values (Slip values) field. The program uses Equivalent Circuit Analysis results to plot the target quantity over the specified range and then chooses the given number of speed (slip) values to insure the best approximation of the target quantity curve. Figure 6.2 demonstrates how it works. In this case the torque is chosen as a target quantity and the picture shows how the program picked up 20 speed values to represent the torque plot behavior in the best way. Note that the chosen target quantity does not mean that other quantities are not included into analysis. It only means that generated speed (slip) values are best suitable to reflect variations of the target quantity.
Figure 6.1. MotorAnalysis main window for AC Analysis.

6.2. Selecting AC solution and viewing solution details.

Each solution resulted from AC Analysis is stored within a simulation file. When AC Analysis is completed you can view analysis results for each computed operating point. To select the desired AC solution use the Power supply, Supply frequency and Slip/Speed fields of the Selected solution panel. Note that ‘V’ at the end of the Power supply field item indicates that the motor is supplied from 3-phase voltage source, while ‘A’ implies the 3-phase current source supply. Choose the power supply first, then choose the supply frequency and after that choose the slip/speed value. Details of the selected solution such as torque, current, voltage and etc. are displayed at the bottom of the window. You can
also delete selected solution from a simulation file or even delete all solutions using buttons of the Selected solution panel. It is recommended to delete unnecessary solutions since they increase a size of the simulation file and influence the MotorAnalysis performance.

![Figure 6.2. Automatic speed values selection for the torque curve.](image)

6.3. Viewing AC analysis results.

The view of the PlotWizard window when AC Analysis is chosen is shown in Figure 6.3. Several plot types are available for AC analysis:

- plots of standard parameters like voltage, current, torque, efficiency and etc. versus speed/slip;
- air gap distribution plot;
- frequency spectrum of the air gap distribution;
- cross-section distribution plot;
Figure 6.3. PlotWizard window for AC Analysis.

6.3.1. General plot.

General plot panel allows the user to plot standard motor design parameters such as stator current, line to ground voltage, torque, input electrical power, power factor, efficiency, rotor cage, stator winding and iron core losses. If you have AC analysis results for different power supply or supply frequencies you should first select a desired AC-solution by the Power supply and Supply frequency fields of the MotorAnalysis main window (see Figure 6.1). Slip/Speed field has no effect on general plot. The supply frequency and the power supply of the selected AC-solution are displayed at the bottom of General plot panel. Select quantities to be plotted, specify line type and color if needed and click the Plot button, the MATLAB figure window with plots of selected quantities will appear (see Figure 6.4).
Figure 6.4. AC Analysis General plot.
Black circles on the plot indicate points at which the AC analysis has been performed. If Hold-on plot mode left to the Plot button is not chosen, every time when you click the Plot button selected parameters will be plotted in the new MATLAB window. Although, if you want to compare parameters for different power supply or supply frequencies you should activate Hold-on plot mode. In this case every time when you click the Plot button, selected parameters will be plotted in the same MATLAB window. When you choose Hold-on plot mode and click the Plot button, the hold-on plot mode will be active until you click the Quit hold-on mode button (see Figure 6.5).

![Hold-on plot mode](image)

Select the "Hold-on plot mode" checkbox and click the "Plot" button to activate hold-on plot mode.

Plot parameters you need by clicking the "Plot" button. When plotting is completed click the "Quit hold-on mode" button.

Figure 6.5. Using Hold-on plot mode.

### 6.3.2. Air gap distribution plot.

Air gap distribution plot panel allows the user to view the distribution of the different parameters such as flux, mmf and flux density components over the machine’s air gap as well as their frequency spectrum for the selected AC-solution. AC-solution is selected by the Power supply, Supply frequency and Slip/Speed fields of the MotorAnalysis main window (see Figure 6.1) and also displayed at the bottom of the PlotWizard window. A column of Subplot checkboxes allow you to control the number of axes (subplots) 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 6.6. Use Ctrl button to choose several variables.

![Air gap distribution plot](image)

Choose variables to plot from the list below:

- $Bm$ (Air gap flux density magnitude distribution, T)
- $Dl$ (Air gap flux density tangential component distribution, T)
- $Bn$ (Air gap flux density normal component distribution, T)
- $f1$ (Air gap magnetic flux distribution, kWs)
- $mmf$ (Air gap magnetostatic force distribution, A)

Add | Cancel

Figure 6.6. Choosing variables to be plotted.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bm (Air gap flux density magnitude distribution, T)</td>
<td>Defined as $B_m = \sqrt{B_n^2 + B_t^2}$, where $B_n$ and $B_t$ – normal and tangential components of the magnetic flux density in the center of the air gap, as it is shown in Figure 2.4.</td>
</tr>
<tr>
<td>Bt (Air gap flux density tangential component distribution, T)</td>
<td>$B_n$ and $B_t$ – normal and tangential components of the magnetic flux density in the center of the air gap, as it is shown in Figure 2.4.</td>
</tr>
<tr>
<td>Bn (Air gap flux density normal component distribution, T)</td>
<td>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 2.4 and the actual values are resulted as the average.</td>
</tr>
<tr>
<td>flux (Air gap magnetic flux distribution, Wb)</td>
<td>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 2.4 and the actual values are resulted as the average.</td>
</tr>
<tr>
<td>mmf (Air gap magnetomotive force distribution, A)</td>
<td></td>
</tr>
</tbody>
</table>

By clicking the Add button of the dialog (Figure 6.6), the selected variables are added to the corresponding line separating variables by commas as it is shown in Figure 6.7. Each line of the Variables column is editable, so you can you use MATLAB arithmetic operations to obtain desired plots. Example in Figure 6.7 produces a product of the normal and tangential components of the magnetic flux density in the air gap. According to Exp. 2.6 the $B_nB_t$ product provides an electromagnetic torque distribution over the machine air gap. **X-axis limits** and **Y-axis limits** fields allow you to set the x-axis and y-axis limits, respectively. **Time(s)** and **Slice** fields are not used in AC Analysis.
Figure 6.7. Example of using **Plot Wizard** for creating air gap plots.

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

Besides the air gap distribution, it is also possible to calculate the air gap harmonic components for the selected quantity. 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 is shown in Figure 6.8. Only first 100 harmonics are shown, since the value specified in the corresponding **X-axis limits** field is 100 (if a minimum limit equals to 0, it can be omitted).

Figure 6.8. Plotting of the air gap distribution of the magnetic flux and its spectrum.
6.3.3. Cross-section distribution plot.

Cross-section distribution plot is 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 axis. Figure checkboxes allow you to control 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 the Figure column. When the figure is active, 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 density, [J/m^3];
- Joule loss density, [W/m^3];
- Iron loss density, [W/m^3];
- Finite element mesh.

Chosen quantities are plotted for the selected AC-solution. AC-solution is selected by Power supply, Supply frequency and Slip/Speed fields of the MotorAnalysis main window (see Figure 6.1) and also displayed at the bottom of the PlotWizard window. Time(s) and Slice fields are not used in AC Analysis. Options field allows you to show the magnetic flux lines (if the flux lines item is selected) or magnetic flux arrows (if the flux arrows item is selected) on the corresponding plot. Flux arrows are plotted so 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 line levels field allows you to alter the flux lines density. If periodic/antiperiodic boundary conditions are used you can define whether a full cross-section or only a part of the cross-section will be displayed by changing the Full cross-section view checkbox.

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 are linearly mapped to the
used color scale (colormap). Data values less or greater than specified z-axis limits are mapped to the minimum limit or to the maximum limit, respectively.

An example of plotting the cross-section distribution of the magnetic flux density and the relative permeability is shown in Figure 6.9. 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 rotor current density is not plotted. Note that the relative permeability is not computed during AC Analysis (effective related permeability is used instead) so it takes some additional time to compute this quantity then it is chosen for plotting.

Figure 6.9. Example of using Plot Wizard for creating cross-section distribution plots.

6.4. Comparison of AC Analysis and Equivalent Circuit Analysis results.
Figure 6.10 compares results of Equivalent Circuit Analysis (dotted lines) with results of AC Analysis (solid lines). Equivalent circuit parameters are computed for 5A RMS phase current and zero speed (locked rotor). Node that comparison results can be significantly different for different motors.
Figure 6.10. Comparison of AC Analysis (solid lines) and Equivalent Circuit Analysis (dotted lines) results.
7. TRANSIENT ANALYSIS

Transient Analysis is based on the time-stepping FEA and has the highest accuracy among other analysis methods allowing to take into account rotation of the rotor, cogging torque, higher harmonics, PWM switching and etc. However, this type of analysis takes considerably more time comparing with previous two analysis types.

7.1. Running Transient Analysis.

The view of the main window when Transient Analysis is chosen is shown in Figure 7.1. To start Transient Analysis, click the Run Transient Analysis 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 Transient Analysis button which is to the right of the Run Transient Analysis 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.

Transient Analysis options shown in Figure 7.1 include the following parameters:

Simulation script file – MATLAB-function which is called on each simulation time step and allows the user to change all general simulation settings, compute and store user defined variables, control power supply sources and state of electronic switches, implement user’s motor control algorithms. Refer to chapter 10 for more details on using simulation script files.

Solver type – two solver types are available which are Nonlinear solver and Linear solver using fixed values of the magnetic permeability.

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 if the simulation is incorrectly stopped. Note that it takes some time to rewrite data – use reasonable value for this field not to significantly affect the computational speed.

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 computed 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 in seconds. You can use different time steps during Transient Analysis to reduce the analysis time.

**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 $\text{Exp}$. 2.5.

**Power supply** – by default the motor can be fed by a symmetrical sinusoidal 3-phase voltage source or 3-phase current course unless other is specified in the simulation script file.

---

**Figure 7.1.** MotorAnalysis main window for Transient Analysis.
**RMS phase voltage/current** and **Supply frequency** – power supply parameters by default unless other is specified in the simulation script file.

If **3-phase voltage source** supply is chosen, the default voltage applied to the motor will be as follows:

\[
\begin{align*}
    u_a &= U \cdot \sqrt{2} \cdot \sin(2\pi \cdot f \cdot t) \\
    u_b &= U \cdot \sqrt{2} \cdot \sin(2\pi \cdot f \cdot t + 2\pi/3) \\
    u_c &= U \cdot \sqrt{2} \cdot \sin(2\pi \cdot f \cdot t + 4\pi/3)
\end{align*}
\]

(7.1)

where \(U\) – value specified in the **RMS supply voltage** field, \(f\) - value specified in the **Supply frequency** field, \(t\) - value displayed in the **Simulation current time point** field.

If **3-phase current source** supply is chosen, the default current applied to the motor will be as follows:

\[
\begin{align*}
    i_a &= I \cdot \sqrt{2} \cdot \sin(2\pi \cdot f \cdot t) \\
    i_b &= I \cdot \sqrt{2} \cdot \sin(2\pi \cdot f \cdot t + 2\pi/3) \\
    i_c &= I \cdot \sqrt{2} \cdot \sin(2\pi \cdot f \cdot t + 4\pi/3)
\end{align*}
\]

(7.2)

where \(I\) – value specified in the **RMS supply current** field.

**Rotor speed dependency** – specifies whether the rotor speed is fixed (when the **Fixed speed simulation** item is chosen) or varies depending on the load and electromagnetic torque of the motor (when the **Variable speed simulation** item is chosen). If the **Fixed speed simulation** is chosen you should also specify the speed in the field appearing to the right. Positive speed value corresponds to clockwise rotation of the rotor. Make sure that direction of the rotor rotation is the same as direction of rotation of the magnetic field. You can use animation option of **Plot Wizard** to check it out.

**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 between stator and rotor.

**Initial speed** – initial speed of the rotor when Transient Analysis is first started.

**Simulation presetting** – specifies the way how the simulation is started. If the **General** is chosen, there is no presetting applied. When the **Time-harmonic FEA based initial conditions** is chosen, the simulation is started with the initial state (initial values of magnetic field, currents and voltages) computed using time-harmonic FEA (AC Analysis). When the **Steady state only** is chosen, the special technique is used to significantly reduce the computation time before the steady state is reached; use this option if only the steady state is of your interest. When the **Steady state only** is chosen, MotorAnalysis uses the specific built-in simulation script; refer to section 10.4 for more details.
7.2. Generating Transient Analysis report.
Transient Analysis report allows you to quickly estimate standard motor parameters like voltage, current, power, torque, torque ripple, power factor, efficiency and losses using Transient Analysis results for steady-state regime. To generate report click the Report button, motor parameters will be displayed in MATLAB command window. Note that to obtain correct parameters the steady state should be reached. By default, report parameters are averaged over the last simulated electrical period. Default report options can be changed from the Transient Analysis Report Options dialog available to the right of the Report button and shown in Figure 7.2. You can choose one, two or three electrical periods as averaging time for report generation as well as you can directly specify averaging time period selecting the User choice item from the Averaging time selection popup menu.

![Transient Analysis Report Options dialog](image)

Figure 7.2. Transient Analysis Report Options dialog.

7.3. Viewing Transient Analysis results.
The view of the Plot Wizard window when Transient Analysis is chosen is shown in Figure 7.3. Several plot types are available for Transient Analysis:
- time-series data plot;
- air gap distribution plot and frequency spectrum of the air gap distribution;
- cross-section distribution plot;
- animation.

7.3.1. Time plot.
Time-series data 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 you to control 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 7.4. Use Ctrl button to select several variables.

Figure 7.3. Transient Analysis Plot Wizard window.
Figure 7.4. Choosing variables to be plotted.

The following variables are listed in the dialog:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isa (Total stator current, phase A, [A])</td>
<td>Current summarized over all parallel paths of a given phase.</td>
</tr>
<tr>
<td>Isb (Total stator current, phase B, [A])</td>
<td></td>
</tr>
<tr>
<td>Isc (Total stator current, phase C, [A])</td>
<td></td>
</tr>
<tr>
<td>Isa1 (Stator current of parallel path 1, phase A, [A])</td>
<td>Available for each parallel path of each phase.</td>
</tr>
<tr>
<td>Ir1 (Rotor current of bar 1, [A])</td>
<td>Available for each rotor bar.</td>
</tr>
<tr>
<td>Ua (Phase voltage, phase A, [V])</td>
<td>Measured directly on each stator coil</td>
</tr>
<tr>
<td>Ub (Phase voltage, phase B, [V])</td>
<td></td>
</tr>
<tr>
<td>Uc (Phase voltage, phase C, [V])</td>
<td></td>
</tr>
<tr>
<td>Pinput (Input apparent power, [VA])</td>
<td></td>
</tr>
<tr>
<td>Pcons (Consumed apparent power, [VA])</td>
<td></td>
</tr>
<tr>
<td>Pr (Apparent power (real and reactive) consumed by a rotor circuit, [VA])</td>
<td>For more details on these variables refer to Appendix A</td>
</tr>
<tr>
<td>Ps (Apparent power (real and reactive) consumed by a stator circuit, [VA])</td>
<td></td>
</tr>
<tr>
<td>Pmech (Mechanical power on the rotor shaft, [W])</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Pmf (Time derivative of the magnetic field energy, [VAR])</td>
<td></td>
</tr>
<tr>
<td>Torque (Electromagnetic torque, [N*m])</td>
<td>Equals to one of the variables Torque_maxwell or Torque_vwork depending on the selected torque calculation method.</td>
</tr>
<tr>
<td>Load (Load torque on the motor shaft, [N*m])</td>
<td></td>
</tr>
<tr>
<td>Speed (Rotor angular speed, [rad/s])</td>
<td></td>
</tr>
<tr>
<td>Rotang (Rotor angular position, [rad])</td>
<td></td>
</tr>
<tr>
<td>Torque_maxwell (Electromagnetic torque calculated using Maxwell’s stress tensor method, [N*m])</td>
<td>For more details on these variables refer to section 2.3.</td>
</tr>
<tr>
<td>Torque_vwork (Electromagnetic torque calculated using Virtual work method, [N*m])</td>
<td></td>
</tr>
<tr>
<td>Fx (Radial electromagnetic force acting between stator and rotor along x-direction, [N])</td>
<td></td>
</tr>
<tr>
<td>Fy (Radial electromagnetic force acting between stator and rotor along y-direction, [N])</td>
<td></td>
</tr>
<tr>
<td>User defined variable</td>
<td>All variables stored in the Userdata structure (see chapter 10 and example 2 of chapter 11) are available at the end of this list.</td>
</tr>
</tbody>
</table>

By clicking the **OK** button of the dialog (Figure 7.4), the MATLAB-expression is constructed to plot the selected variables appearing in the corresponding line as it is shown in Figure 7.5 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 7.5. Example of using Plot Wizard for creating time plots.](image-url)
When the **Plot** button is clicked, the MATLAB figure window with plots of the selected variables will appear (see Figure 7.6).

![MATLAB figure window with plots](image)

Figure 7.6. MATLAB figure window with time plots of the selected variables.

As it is seen, the plotting expression consists of the `plot` function with selected variables used as input arguments. If the **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 10.3) 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:

```matlab
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. If the Synchronize subplot time-axis limits checkbox is active, all subplots of the figure will have identical limits along the time-axis when you zoom or pan one of the subplots of the figure.

### 7.3.2. Air gap distribution plots.

Air gap distribution plots allow you to view 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. Subplot checkboxes allow you to control 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 7.7. Use Ctrl button to select several variables.

![Air gap distribution plot](image)

**Figure 7.7.** Choosing variables to be plotted.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bm (Air gap flux density magnitude distribution, [T])</td>
<td>Defined as $B_m = \sqrt{B_n^2 + B_t^2}$ where $B_n$ and $B_t$ – normal and tangential components of the magnetic flux density in the center of the air gap, as it is shown in Figure 2.4.</td>
</tr>
<tr>
<td>Bt (Air gap flux density tangential component distribution, [T])</td>
<td>$B_n$ and $B_t$ – normal and tangential components of the magnetic flux density in the center of the air gap, as it is shown in Figure 2.4.</td>
</tr>
<tr>
<td>Bn (Air gap flux density normal component distribution, [T])</td>
<td></td>
</tr>
</tbody>
</table>
Integration of the magnetic flux distribution over an air gap always gives zero:
\[ \Phi_{\text{airgap}} = l \oint_{\text{airgap}} B_n \cdot d\ell = 0, \]
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 2.4 and the actual values are resulted as the average.

Integration of the magnetomotive force distribution over an air gap always gives zero:
\[ F_{\text{airgap}} = \frac{1}{\mu_0} \oint_{\text{airgap}} B_t \cdot d\ell = 0. \]

Air gap magnetomotive force distribution is calculated over inner and outer contours as it is shown in Figure 2.4 and the actual values are resulted as the average.

By clicking the **Add** button of the dialog (Figure 7.7), the selected variables are added to the corresponding line separating them by commas as it is shown in Figure 7.8. 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 7.8 produce plots of the tangential air gap force density distribution and the radial air gap force density distribution, respectively, where \( \mu_0 \) – variable corresponding to the permeability of free space. According to Exp. 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 7.8. Example of using **Plot Wizard** for creating air gap plots.
**Time** fields of the **Air gap distribution plot** panel allow you to specify 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 the **Time plot** panel. **Slice** fields allow you to choose the machine’s cross-section which the selected variables are plotted for, if several slices are used (see section 2.4 for more details on multi-slice simulations).

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 7.9. 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 for the specified time is only possible, if the 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.

### 7.3.3. 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 you to control 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];
Figure 7.9. Plotting of the air gap distribution of the magnetic flux and its spectrum.

- Stator current density, [A/m^2];
- Rotor current density, [A/m^2];
- Squared flux density, [T];
- Magnetic field energy density, [J/m^3];
- Joule loss density, [W/m^3];
- Iron 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 line levels** field allows you to alter the flux lines density. If periodic/antiperiodic boundary conditions are used, by default, only part of the cross-section (as it appears in Mesh Editor) will be plotted. Select the **Full cross-section view** checkbox if you want to plot the whole cross-section.

**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 are linearly mapped to the used color scale (colormap). Data values less or greater than specified z-axis limits are mapped to the minimum limit or to the maximum limit, respectively.

An example of plotting the cross-section distribution of the magnetic flux density plots for first and second slices are shown in Figure 7.10. When the Plot button is clicked, two windows appear. As it is seen, the flux lines option is chosen for the both figures, so the flux lines are additionally shown. Since the third figure is not active, the flux density for third slice is not plotted. Note that only part of the machine’s cross-section is shown, since the **Full cross-section view** checkbox is not active.

![Figure 7.10. Example of using Plot Wizard for creating cross-section distribution plots.](image)
Plotting of the cross-section distribution for the specified time is only possible, if the 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.

### 7.3.4. 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 to be 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, for example, changing the scale and position of the plots.
7.4. Iron Loss Calculator.

Iron Loss Calculator tool shown in Figure 7.11 is used for iron loss estimation during Transient Analysis.

Iron Loss Calculator is available from MotorAnalysis main window when Transient Analysis is chosen. It estimates the average iron loss over the specified period of time defined by the **Period of time** panel. To use Iron Loss Calculator you should previously store simulation data for each time step of the specified period of time using **Extended data saving** option of the main window. Directory where these simulation data files were stored is defined by the **Data source folder** field.

**Flux density waveform/spectrum** button allows you to examine waveform and spectrum of orthogonal components of the magnetic flux density at several measuring points: stator tooth tip, stator tooth, stator back, rotor tooth tip, rotor tooth and rotor back. Example of flux density waveform and spectrum at stator tooth tip measuring point is shown in Figure 7.12.

**Maximum fft frequency of flux density** field defines the maximum frequency of flux density harmonic for which the iron loss is calculated. This value is restricted by the time-step as 1/timestep/2. Since the larger maximum fft frequency increase the computation time you can use flux density spectrum at different measuring points to choose the optimal frequency.

To start calculation click **Start Iron Loss Calculation** button. Result will be displayed at the bottom of the Iron Loss Calculator window.

![Iron Loss Calculator window](image)

Figure 7.11. Iron Loss Calculator window.
Figure 7.12. Waveform and spectrum of flux density components at stator tooth tip measuring point.
8. ADVANCED EQUIVALENT CIRCUIT ANALYSIS

Advanced Equivalent Circuit (EC) Analysis is used for the same purposes as Equivalent Circuit Analysis described above. The main goal behind Advanced EC Analysis is to provide the high accuracy and the high computational speed at the same time. In fact, this type of analysis shows the accuracy close to AC Analysis having the computational speed of the analytical model.


Similarly to Equivalent Circuit Analysis, Advanced EC Analysis is based on the determination of equivalent circuit parameters of the induction machine using no-load and load tests. The difference is that all circuit parameters are dependent on operating conditions as shown in Figure 8.1.

![Advanced equivalent circuit of the induction machine.](image)

As it is seen, the stator and rotor leakage inductances L1 and L2 as well as rotor resistance R2 are assumed to be functions of the stator current and magnetizing current (their RMS values), the magnetizing inductance is a function of the magnetizing current only. Iron losses are considered as an independent phenomenon (not influence the equivalent circuit) being a function of the stator current, magnetizing current and supply frequency.

8.2. Running Advanced Equivalent Circuit Analysis.

The view of the main window when Advanced Equivalent Circuit Analysis is chosen is shown in Figure 8.2. To determine parameters of the equivalent circuit click the Start model parameterization button. During the model parameterization, which usually takes several hours, no-load test and set of load tests are performed. The load test is performed for different operating conditions depending on the parameterization setup – six fields above the Start model parameterization button. Fields Maximum supply frequency, Maximum stator current (RMS) and Minimum stator current (RMS) determine the range of operating conditions of the motor. Number of no-load test current levels field (default value 35) defines the number of magnetizing inductance values computed for different values of magnetizing current. Fields Number of stator current levels (default value 10) and Minimum number of magnetizing current levels (default value 15) define number of operating points for which
the load test is performed to determine how the equivalent circuit parameters depend on the stator current and magnetizing current. More points enhance the accuracy of the model but also increase the parameterization time.

Once the parameterization is completed parameters of the equivalent circuit will be displayed as it is shown in Figure 8.2. You can examine how each circuit parameter changes depending on operating conditions using corresponding buttons to the right. Correct results of the model are guaranteed if the supply frequency does not exceed the value specified in the Maximum supply frequency field and the supply current is within the range specified in the Stator phase current effective range field.

![MotorAnalysis main window for Advanced Equivalent Circuit Analysis](image)

Figure 8.2. MotorAnalysis main window for Advanced Equivalent Circuit Analysis.
8.3. Verifying Advanced Equivalent Circuit model.

Once the model parameterization is completed you can verify the model by comparing its results with AC Analysis (time-harmonic FEA) using the Model verification panel at the bottom of the window. Setup a few operating point which will be used during the verification by filling the Torque values, Speed values and Slip values fields and choosing the Values selection method (“one by one” or “as 3d array”). For the example shown in Figure 8.2 the Values selection is “as 3d array” which means that 4*4*4=64 operating points will be used for the verification; for “one by one” chosen – only four operating points would be used. If you computed the efficiency map (see the next section) and you want to verify it, you can automatically setup verification points randomly chosen from the efficiency map using the Efficiency map verification setup panel: specify number of verification points and press the Setup Verification Points button - Torque values, Speed values and Slip values fields will be filled automatically. Use the Start Verification button to start the verification, verification results will be displayed in the MATLAB command window.

8.4. Viewing Advanced Equivalent Circuit Analysis results.

The view of the PlotWizard window when Advanced Equivalent Circuit Analysis is chosen is shown in Figure 8.3.

Use the top panel to plot standard motor parameters like those shown in Figure 8.4. To begin the analysis, specify the power supply properties: 3-phase voltage source or 3-phase current source, RMS phase voltage (or current) and supply frequency, then choose variables you want to plot and click the Plot button, the MATLAB figure window with plots of the selected variables will appear (see Figure 8.4).

You can adjust x-axis limits of the plot using controls to the left. The Number of points field determines the number of points in which the chosen variables are computed. More points lead to smoother plots but increase computation time.

As it is shown in Figure 8.4, each variable has its own y-axis and different scales. Unselect Multiple y-axes checkbox if you want to use the same scale along y-axis to compare several variables.

Pay attention to the Interpolation method pop-up menu of the Model setup panel while doing analysis. It determines the way how intermediate values of the equivalent circuit parameters are interpolated. Linear interpolation decreases the computation time while the nonlinear interpolation provides more accurate results.

Another option of Advanced Equivalent Circuit Analysis is creation of the efficiency map. The efficiency map shows the maximum efficiency which can be reached for the certain speed and load torque. The example of the plot of the efficiency map for motor and generator modes is shown in Figure 8.5. Along with the efficiency map some additional plots of the supply voltage, current, power, supply frequency, slip as well as maximum power (power envelop) are available (see Figure 8.6).
Maximum RMS phase voltage, Maximum RMS phase current and Maximum input power fields define supply voltage, current and power limitations applied while the efficiency map is computed. If the Maximum input power field is left empty then the input power is not limited (limited only by maximum voltage and current).

Use the Mechanical losses coefficients field to apply mechanical losses while the efficiency map is computed. Mechanical losses coefficients are stored in a vector whose elements are the coefficients in descending powers of the mechanical loss equation. For mechanical losses coefficients shown in Figure 8.3 the mechanical loss equation is as follows:

\[ W_{\text{mech.loss}} = 0.0001\omega^2 + 0.0065\omega + 5.9, \]

where \( \omega \) – rotor speed.

Number of points (speed-torque pairs) at which the efficiency map is computed is defined by the Map resolution panel. Select the Add efficiency map for generator mode checkbox if you need efficiency maps for both motor and generator modes; otherwise the efficiency map only for motor mode will be computed. Click the Create Efficiency Map button to compute the efficiency map. Make sure that the Interpolation method pop-up menu is properly setup as described above.

Once the efficiency map is computed for the first time it will be saved within the simulation file. Every time you compute the efficiency map once again you will be asked whether you want to replace previously computed efficiency map with the new one. Use the Plot Efficiency Map button to plot the efficiency map previously saved. There are also some additional options available to the right of the Plot Efficiency Map button as shown in Figure 8.3. For example, you can display the efficiency map as a table choosing the Display efficiency table item.
Figure 8.3. PlotWizard window for Advanced Equivalent Circuit Analysis.
Figure 8.4. Advanced Equivalent Circuit Analysis plot.
Figure 8.5. Efficiency map for motor and generator modes.
Figure 8.6. Additional plots corresponding to the efficiency map shown in Figure 8.5.
8.5. Adding your own analysis algorithms.

In the previous section different options to view analysis results has been discussed. Additionally, using the RunAdvancedEC function, you can add new options to view Advanced Equivalent Circuit Analysis results.

Syntax of the RunAdvancedEC function is as follows:

\[
[I1, U1, \text{torque}, f1, \text{Piron}, \text{niter}, \text{Im}, \text{I2}, \text{EC}] = \text{RunAdvancedEC} (\text{filename}, \text{param}, \text{paramvalue}, \text{speed}, \text{slip}, \text{interpmethod})
\]

\[
[I1, U1, \text{torque}, f1, \text{Piron}, \text{niter}, \text{Im}, \text{I2}, \text{EC}] = \text{RunAdvancedEC} (\text{filename}, \text{param}, \text{paramvalue}, \text{speed}, \text{slip}, \text{interpmethod}, f1)
\]

- \(I1, U1\) – supply current [Ampere] and voltage [Volt] in complex form; \(\text{torque}\) - torque in N*m; \(f1\) - supply frequency in Hz; \(\text{Piron}\) - iron losses in Watt; \(\text{niter}\) - number of iterations; \(\text{Im}, \text{I2}\) - magnetizing current and rotor current in complex form; \(\text{EC}\) – structure of equivalent circuit parameters;
- \(\text{filename}\) - name of the simulation file; \(\text{param}\) - can be one of the following values: ‘Is’, ‘Us’, ‘torque’; if \(\text{param} = \text{‘Is’}\), then \(\text{paramvalue}\) is a RMS value of supply current; if \(\text{param} = \text{‘Us’}\), then \(\text{paramvalue}\) is a RMS value of supply voltage, if \(\text{param} = \text{‘torque’}\), then \(\text{paramvalue}\) is a load torque; \(\text{speed}\) - speed of the motor in radians per second, \(\text{slip}\) - slip of the motor from 0 to 1; \(\text{interpmethod}\) – interpolation method, can be one of the following values: ‘linear’, ‘nonlinear’. Linear interpolation is faster, but nonlinear gives more accurate result.

Since speed and slip of the motor are defined, the supply frequency \(f1\), last input argument of the function, is implicitly defined as it follows \(f1 = \text{speed} \times \text{n PolePairs} / (2 \times \pi \times (1 - \text{slip}))\); use second syntax expression with \(f1\) as the last input argument for locked rotor (\(\text{speed}=0\)), otherwise \(f1\) will be ignored.

Example of using RunAdvancedEC to get plots of the supply voltage, efficiency, torque and supply frequency for the motor operating at 6000RPM and 1060A supply current for slip range from 0 to 10%:

```matlab
filename='example.mat';
param='Is';
paramvalue=1060; % supply current 1060A
speed=6000*2*pi/60; % from 6000RMP to rad/s
Slip=linspace(0,0.1,100);
U1=[]; Torque=[]; F1=[]; Efficiency=[];
for i=1:length(Slip)
    [i1, u1, torque, f1, Piron]=RunAdvancedEC(filename,param,paramvalue, speed,Slip(i), 'nonlinear');
    efficiency=100*torque*speed/(3*real(conj(i1)*u1)/2+Piron);
    U1=[U1 abs(u1)/sqrt(2)]; % supply voltage
    Torque=[Torque torque];
    F1=[F1 f1]; % supply frequency
    Efficiency=[Efficiency efficiency];
end
figure;plot(Slip,U1,Slip,Efficiency,Slip,Torque,Slip,F1);grid
```
9. BUILDING SIMULINK MODEL OF THE MOTOR

Once the parameters of the equivalent circuit of the motor are known (whether from Equivalent Circuit Analysis or from Advanced EC Analysis) you can simulate the dynamics of the motor using Simulink®. Two model configurations are available based on the standard equivalent circuit of the induction motor (Figure 5.1) as well as on the advanced equivalent circuit (Figure 8.1) described in the previous chapter. When the standard equivalent circuit configuration is used, only magnetizing inductance \( L_m \) is changed depending on the magnetizing current. For the advanced equivalent circuit configuration all parameters including \( L_m, L_1, L_2 \) and \( R_2 \) are dependent on operating conditions to better simulate the nonlinear behaviors. Iron losses are excluded from the Simulink model.

The IMECd block is available from file IMECtmpl.slx (IMECtmpl.mdl for older Simulink versions). Open IMECtmpl.slx and copy the block into your Simulink model. The IMECd block has four input parameters: three phase voltages and load torque; and five output parameters: three phase currents, electromagnetic torque and angular speed. Double-click the IMECd block to get the parameters tab (Figure 9.1). Enter the simulation file name corresponding to the motor; choose model type (Standard Equivalent Circuit or Advanced Equivalent Circuit), set up moment of inertia and initial speed of the motor. Parameters of the equivalent circuit will be taken from the specified simulation file.

![Function Block Parameters: IMECd](image)

Figure 9.1. Induction motor model block.
9.3. Example of using the motor model block.

The ECExample.slx (ECexample.mdl) simple example shown in Figure 9.2 demonstrates simulation of the direct starting of the motor. Block IMECd is set up to the standard equivalent circuit configuration. The motor is connected to a constant load of nominal value 9 N*m.

Run the simulation and observe the motor’s currents, torque and speed using the following commands:

```matlab
figure; subplot(211); plot(simout.Time, simout.Data(:,1:3)); grid;
xlabel('time (seconds)'); ylabel('Stator currents, A');
subplot(212); plot(simout.Time, simout.Data(:,4:5)); grid;
xlabel('time (seconds)'); legend('torque, N*m', 'speed, rad/s');
```

Figure 9.2. Example of using the induction motor model block and simulation results.
10. TRANSIENT ANALYSIS SIMULATION SCRIPT FUNCTIONS

A simulation script is a file with .m extension containing MATLAB-function of a specific format. While the transient simulation is in progress this function is called on each simulation time step and allows you to automatically change all simulation settings, compute and store your own variables, control power supply sources, implement motor control algorithms and etc. For example, the simulation script function can be used to control electronic switches (IGBT, for example) when the motor is fed by PWM inverter and etc.

To understand this chapter, some familiarity with MATLAB programming is required. Refer to MATLAB help documentation for more details on MATLAB programming.


MotorAnalysis offers you a variety of options to properly set up your simulation and in most cases you will not need to use simulation scripts. On the other hand, simulation scripts give you much more flexibility in using MotorAnalysis. Note that simulation scripts are used only for Transient Analysis. The chosen simulation script file is displayed in the Simulation script file field of the MotorAnalysis main window when Transient Analysis is chosen (see Figure 7.1). If the simulation script file is not chosen, No simulation script item will appear. To choose or change the 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. For example, if the 3-phase voltage source supply is selected, the default voltage applied to the motor will be defined by Exp. 7.1. But if the simulation script function is used, you can specify whether to use the default voltage defined by Exp. 7.1 or you can use your own expression to define voltage values.

10.2. Writing a simulation script function.

The definition of the simulation script function simscript and a minimum amount of code appearing in file simscript.m is as follows:

```matlab
function [ShaftPosition, CircuitControl, Settings, Enforce, Userdata]... = simscript (Outputs, Settings, Userdata, Circuit, Geometry, Mesh, ...
Windings, Core, A, cell_p, cell_t, cell_Nu, path)
ShaftPosition = [];
CircuitControl = [];
Enforce = [];
```
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 MATLAB-functions.

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.

**CircuitControl** – structure containing current and voltage values which are supplied by current and voltage courses as well as states of electronic switches. If CircuitControl = [] default voltage or current values (Exp. 7.1 and 7.2) are used. See chapter 11 for more details on using current and voltage sources and electronic switches.

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

**Enforce** – structure to control rotor speed. If Enforce=[], then rotor speed is computed depending on the load and electromagnetic torque (variable speed simulation). To set the rotor speed to some fixed value use the statement Enforce.speed=fixedspeed, where fixedspeed is the rotor speed in rad/s.

**Userdata** – structure to store user data, the same as input argument Userdata. By default Userdata is empty. See example 2 of chapter 11 to figure out how to use the Userdata structure to store your own time-series data using simulation script functions.

The function’s input arguments:

**Outputs** – structure containing transient simulation results.

**Settings** – structure of transient simulation settings.

**Userdata** – structure to store user data.

**Circuit** – structure containing stator and rotor 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.


**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 midpoint magnetic reluctivity values for every slice. Magnetic reluctivity is \( \nu = \frac{1}{\mu_0 \mu_r} \), where \( \mu_0 \) – permeability of the free space, \( \mu_r \) - relative permeability.

path – directory with application files.

10.3. Main data structures.

Outputs structure fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Size, type</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs.CurrentTime</td>
<td>1×1, double</td>
<td>second</td>
<td>Simulation current time point.</td>
</tr>
<tr>
<td>Outputs.Ua</td>
<td>1×n, double</td>
<td>Volt</td>
<td>Instantaneous values of voltages measured at stator winding terminals; n – number of computed time steps.</td>
</tr>
<tr>
<td>Outputs.Ub</td>
<td>1×n, double</td>
<td>Volt</td>
<td></td>
</tr>
<tr>
<td>Outputs.Uc</td>
<td>1×n, double</td>
<td>Volt</td>
<td></td>
</tr>
<tr>
<td>Outputs.Isa</td>
<td>Npp×n, double</td>
<td>Ampere</td>
<td>Current flowing in each parallel path of each phase; number of array rows corresponds to the number of parallel paths.</td>
</tr>
<tr>
<td>Outputs.Isb</td>
<td>Npp×n, double</td>
<td>Ampere</td>
<td>Npp – number of parallel paths.</td>
</tr>
<tr>
<td>Outputs.Isc</td>
<td>Npp×n, double</td>
<td>Ampere</td>
<td></td>
</tr>
<tr>
<td>Outputs.Ir</td>
<td>Nr×n, double</td>
<td>Ampere</td>
<td>Rotor bar currents; number of array rows corresponds to the number of rotor bars. Nr – number of rotor bars.</td>
</tr>
<tr>
<td>Outputs.time</td>
<td>1×n, double</td>
<td>second</td>
<td>Time points.</td>
</tr>
<tr>
<td>Outputs.Torque</td>
<td>1×n, double</td>
<td>N*m</td>
<td>Electromagnetic torque; equals to one of the variables Torque_maxwell or Torque_vwork depending on the selected torque calculation method.</td>
</tr>
<tr>
<td>Outputs.Load</td>
<td>1×n, double</td>
<td>N*m</td>
<td>Load torque on the motor shaft.</td>
</tr>
<tr>
<td>Outputs.Speed</td>
<td>1×n, double</td>
<td>rad/s</td>
<td>Rotor angular speed.</td>
</tr>
<tr>
<td>Outputs.Rotang</td>
<td>1×n, double</td>
<td>rad</td>
<td>Rotor angular position.</td>
</tr>
<tr>
<td>Outputs.Pinput</td>
<td>1×n, double</td>
<td>Watt</td>
<td>Input apparent power delivered by all current and voltage sources.</td>
</tr>
<tr>
<td>Outputs.Pcons</td>
<td>1×n, double</td>
<td>Watt</td>
<td>Consumed apparent power.</td>
</tr>
<tr>
<td>Outputs.Psi</td>
<td>nCircuitNodes×n, double</td>
<td>Volt</td>
<td>Stator and rotor electrical circuit.</td>
</tr>
<tr>
<td>Outputs</td>
<td>1×n, double</td>
<td>Watt</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>.Pr</td>
<td></td>
<td></td>
<td>Apparent power (real and reactive) consumed by the rotor electrical circuit.</td>
</tr>
<tr>
<td>.Ps</td>
<td></td>
<td></td>
<td>Apparent power (real and reactive) consumed by the stator electrical circuit.</td>
</tr>
<tr>
<td>.Pmf</td>
<td></td>
<td></td>
<td>Magnetic energy time derivative.</td>
</tr>
<tr>
<td>.Pmech</td>
<td></td>
<td></td>
<td>Mechanical power on the shaft.</td>
</tr>
<tr>
<td>.Piron.stator</td>
<td>1×1, double</td>
<td>Watt</td>
<td>Stator iron loss.</td>
</tr>
<tr>
<td>.Piron.rotor</td>
<td>1×1, double</td>
<td>Watt</td>
<td>Rotor iron loss.</td>
</tr>
<tr>
<td>.Torque_maxwell</td>
<td>1×n, double</td>
<td>N*m</td>
<td>Electromagnetic torque calculated with the Maxwell stress tensor method.</td>
</tr>
<tr>
<td>.Torque_vwork</td>
<td>1×n, double</td>
<td>N*m</td>
<td>Electromagnetic torque calculated with the Virtual work method.</td>
</tr>
<tr>
<td>.Fx</td>
<td></td>
<td>N</td>
<td>Radial electromagnetic force acting between stator and rotor along x-direction.</td>
</tr>
<tr>
<td>.Fy</td>
<td></td>
<td>N</td>
<td>Radial electromagnetic force acting between stator and rotor along y-direction.</td>
</tr>
</tbody>
</table>

**Settings structure fields:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Settings.script</td>
<td>'No simulation script'</td>
<td>Simulation script file; corresponds to the Simulation script file field of the main window.</td>
</tr>
<tr>
<td></td>
<td>Directory and/or file name</td>
<td></td>
</tr>
<tr>
<td>Settings.solvertype</td>
<td>'Nonlinear'</td>
<td>Solver type; corresponds to the Solver type field of the main window.</td>
</tr>
<tr>
<td></td>
<td>'Linear'</td>
<td></td>
</tr>
<tr>
<td>Settings.saveperiod</td>
<td>Number &gt; 0</td>
<td>Number of time steps in which the simulation data are being rewritten on hard drive; corresponds to the Simulation data saving period field of the main window.</td>
</tr>
<tr>
<td>Settings.extsave</td>
<td>0</td>
<td>Enables the extended data saving mode; corresponds to the <strong>Extended data saving</strong> field of the main window.</td>
</tr>
<tr>
<td>Settings.extsavefolder</td>
<td>Directory</td>
<td>Folder to store data when the extended data saving mode is enabled; corresponds to the <strong>Ext. data saving folder</strong> field of the main window.</td>
</tr>
<tr>
<td>Settings.timestep</td>
<td>Number &gt; 0</td>
<td>Simulation time step size; corresponds to the <strong>Time step</strong> field of the main window.</td>
</tr>
<tr>
<td>Settings.stopime</td>
<td>Number &gt; 0</td>
<td>Simulation stop time; corresponds to the <strong>Simulation stop time</strong> field of the main window.</td>
</tr>
<tr>
<td>Settings.motorload</td>
<td>Number ≥ 0</td>
<td>Load torque on the motor shaft, corresponds to the <strong>Load</strong> field of the main window.</td>
</tr>
<tr>
<td>Settings.momentinertia</td>
<td>Number ≥ 0</td>
<td>Combined rotor and load moment of inertia, corresponds to the <strong>Moment of inertia</strong> field of the main window.</td>
</tr>
<tr>
<td>Settings.tol</td>
<td>Number &gt; 0</td>
<td>Convergence tolerance; corresponds to the <strong>Convergence tolerance</strong> field of the main window.</td>
</tr>
<tr>
<td>Settings.fl</td>
<td>Number &gt; 0</td>
<td>Default supply frequency; corresponds to the <strong>Supply frequency</strong> field of the main window.</td>
</tr>
<tr>
<td>Settings.U</td>
<td>Number &gt; 0</td>
<td>Default supply voltage; corresponds to the <strong>RMS supply voltage</strong> field of the main window when “3-phase voltage source” is chosen.</td>
</tr>
<tr>
<td>Settings.I</td>
<td>Number &gt; 0</td>
<td>Default supply current; corresponds to the <strong>RMS supply current</strong> field of the main window when “3-phase current source” is chosen.</td>
</tr>
<tr>
<td>Settings.torquemethod</td>
<td>'Maxwell stress tensor'</td>
<td>Electromagnetic torque calculation method; corresponds to the <strong>Torque calculation method</strong> field of the main window.</td>
</tr>
</tbody>
</table>
Settings.force

Enable calculation of the radial force acting between stator and rotor; corresponds to the Rotor radial force calculation field of the main window.

Settings.report

Enable output of simulation data in the Command window; corresponds to the Display outputs in MATLAB Command Window field of the main window.

Settings.InitSpeed

Initial speed of the rotor; corresponds to the Initial Speed field of the main window when variable speed simulation is selected.

Geometry structure fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry.D1s</td>
<td></td>
<td>Machine’s cross section dimensions</td>
</tr>
<tr>
<td>Geometry.D2s</td>
<td></td>
<td>according to section 4.1 of this manual.</td>
</tr>
<tr>
<td>Geometry.lag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Dshaft</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Ods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Ows</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Tas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Sds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Rcs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Ws</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Odr</td>
<td>Number &gt;= 0</td>
<td></td>
</tr>
<tr>
<td>Geometry.Owr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Tar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Sdr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Rcr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Wr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Dch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Rch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Tach</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Rcs_ag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry.Rc_r_ag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Geometry.Ns</td>
<td>Number &gt; 0</td>
<td>Number of stator slots; corresponds to the Number of slots field of Geometry Editor.</td>
</tr>
<tr>
<td>Geometry.Nr</td>
<td>Number &gt; 0</td>
<td>Number of rotor bars; corresponds to the Number of bars field of Geometry Editor.</td>
</tr>
<tr>
<td>Geometry.statorslottype</td>
<td>'Parallel tooth'</td>
<td>Stator slot type; corresponds to the Slot type field of the Stator dimensions panel of Geometry Editor.</td>
</tr>
<tr>
<td></td>
<td>'Parallel slot'</td>
<td></td>
</tr>
<tr>
<td>Geometry.slotlayertype</td>
<td>'Single layer'</td>
<td>Number of stator winding layers; corresponds to the Number of winding layers field of Geometry Editor.</td>
</tr>
<tr>
<td></td>
<td>'Double layer'</td>
<td></td>
</tr>
<tr>
<td>Geometry.rotorslottype</td>
<td>'Parallel tooth'</td>
<td>Rotor slot type; corresponds to the Slot type field of the Rotor dimensions panel of Geometry Editor.</td>
</tr>
<tr>
<td></td>
<td>'Parallel slot'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'Round'</td>
<td></td>
</tr>
<tr>
<td>Geometry.statorslotcornertype</td>
<td>'General'</td>
<td>Stator slot bottom shape; corresponds to the stator Bottom corner type field of Geometry Editor.</td>
</tr>
<tr>
<td></td>
<td>'Round'</td>
<td></td>
</tr>
<tr>
<td>Geometry.rotorslotcornertype</td>
<td>'General'</td>
<td>Rotor slot bottom shape; corresponds to the rotor Bottom corner type field of Geometry Editor.</td>
</tr>
<tr>
<td></td>
<td>'Round'</td>
<td></td>
</tr>
<tr>
<td>Geometry.l</td>
<td>Number &gt; 0</td>
<td>Machine’s axial dimensions according to section 4.1 of this manual.</td>
</tr>
<tr>
<td>Geometry.RBskew</td>
<td>Number &gt; 0</td>
<td></td>
</tr>
</tbody>
</table>

Windings structure fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windings.Npp</td>
<td>Number &gt; 0</td>
<td>Number of parallel paths of the stator winding per phase; corresponds to the Number of parallel paths field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.Lsew</td>
<td>Number $\geq 0$</td>
<td>Leakage inductance of the stator winding end-turns per phase; corresponds to the <strong>End winding inductance</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Windings.Rs</td>
<td>Number $\geq 0$</td>
<td>Active resistance of the stator winding per phase; corresponds to the <strong>Winding phase resistance</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.layout</td>
<td>Structure with three cell arrays</td>
<td>Stator winding layout; corresponds to the <strong>Winding layout</strong> tables of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.kr</td>
<td>Number $&gt; 0$</td>
<td>Rotor bar material conductivity; corresponds to the <strong>Bar material conductivity</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.Rre</td>
<td>Number $\geq 0$</td>
<td>Active resistance of the arch of rotor end ring measured between two adjacent bars; corresponds to the <strong>End ring resistance</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.Lrew</td>
<td>Number $\geq 0$</td>
<td>Inductance of the arch of rotor end ring measured between two adjacent bars; corresponds to the <strong>End ring inductance</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.W</td>
<td>Number $&gt; 0$</td>
<td>Total number of conductors in one stator slot; corresponds to the <strong>Number of conductors per slot</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.fillfactor</td>
<td>0 $&lt; \text{Number} &lt; 1$</td>
<td>Stator slot fill factor; corresponds to the <strong>Slot fill factor</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.ks</td>
<td>Number $&gt; 0$</td>
<td>Stator winding material conductivity; corresponds to the <strong>Winding material conductivity</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Property</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Windings.statorcircuit</td>
<td>Function name</td>
<td>Name of the function specifying the stator electrical circuit; corresponds to the Stator circuit field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.Hsew</td>
<td>Number &gt; 0</td>
<td>Stator end winding axial overhang; corresponds to the End winding axial overhang field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.layoutmethod</td>
<td>'Automatic' 'Manual'</td>
<td>Stator winding layout input method; corresponds to the Layout input method field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.nPolePairs</td>
<td>Number &gt; 0</td>
<td>Number of pole pairs; corresponds to the Number of pole pairs field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.coilspan</td>
<td>Number &gt; 0</td>
<td>Stator winding coil span; corresponds to the Coil span in slot pitches field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.windingtype</td>
<td>'Lap' 'Concentric'</td>
<td>Stator winding type; corresponds to the Winding type field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.Lsew_inputmethod</td>
<td>'Automatic' 'Manual'</td>
<td>Stator end winding leakage inductance input method; corresponds to the End winding inductance input method field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.Rs_inputmethod</td>
<td>'Automatic' 'Manual'</td>
<td>Stator winding phase resistance input method; corresponds to the Winding phase resistance input method field of Windings Property Editor.</td>
</tr>
<tr>
<td>Windings.endringinputmethod</td>
<td>'Automatic' 'Manual'</td>
<td>Rotor end ring parameters input method; corresponds to the End ring parameters input method field of Windings Property Editor.</td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Windings.Hre</td>
<td>Number &gt; 0</td>
<td>Rotor end ring dimensions according to section 4.2 of this manual.</td>
</tr>
<tr>
<td>Windings.D1re</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windings.D2re</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windings.Ts</td>
<td>Number</td>
<td>Stator winding temperature; corresponds to the <strong>Stator winding temperature</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.Tr</td>
<td>Number</td>
<td>Rotor winding temperature; corresponds to the <strong>Rotor winding temperature</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.alfa_s</td>
<td>Number &gt;= 0</td>
<td>Stator winding temperature coefficient of resistance; corresponds to the <strong>Stator winding temperature coefficient of resistance</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
<tr>
<td>Windings.alfa_r</td>
<td>Number &gt;= 0</td>
<td>Rotor winding temperature coefficient of resistance; corresponds to the <strong>Rotor winding temperature coefficient of resistance</strong> field of <strong>Windings Property Editor</strong>.</td>
</tr>
</tbody>
</table>

**Mesh structure fields:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh.nAirGapLayers</td>
<td>3, 5, 7, 9</td>
<td>Number of mesh layers in the air gap; corresponds to the <strong>Number of layers in air gap</strong> field of <strong>Mesh Editor</strong>.</td>
</tr>
<tr>
<td>Mesh.Hgrad</td>
<td>1 &lt; number &lt; 2</td>
<td>Mesh growth rate; corresponds to the <strong>Mesh growth rate</strong> field of <strong>Mesh Editor</strong>.</td>
</tr>
<tr>
<td>Mesh.nrfmeshcalls</td>
<td>Number &gt; 0</td>
<td>Number of refinemesh function calls; corresponds to the <strong>Number of refinemesh calls</strong> field of <strong>Mesh Editor</strong>.</td>
</tr>
<tr>
<td>Mesh.nSlices</td>
<td>Number &gt; 0</td>
<td>Number of slices; corresponds to the <strong>Number of slices</strong> field of <strong>Mesh Editor</strong>.</td>
</tr>
<tr>
<td>Mesh.perbndcnd</td>
<td>'None', 'Periodic'</td>
<td>Periodic/antiperiodic boundary conditions; corresponds to the <strong>Boundary conditions</strong> field</td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mesh.nPolePairs</td>
<td>Number &gt; 0</td>
<td>Mesh periodicity factor; corresponds to the Periodicity factor field of Mesh Editor.</td>
</tr>
<tr>
<td>Mesh.shaftmesh</td>
<td>0</td>
<td>Specifies whether mesh in shaft region is used or not; corresponds to the Mesh in shaft region field of Mesh Editor.</td>
</tr>
<tr>
<td>Mesh.shaftmesh</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Mesh.p</td>
<td>2×np double</td>
<td>Initial mesh data; corresponding to p, t and e arrays used in MATLAB PDE Toolbox as mesh data.</td>
</tr>
<tr>
<td>Mesh.t</td>
<td>4×nt double</td>
<td></td>
</tr>
<tr>
<td>Mesh.e</td>
<td>7×ne double</td>
<td></td>
</tr>
<tr>
<td>Mesh.b</td>
<td>1×np double</td>
<td>Array of boundary conditions.</td>
</tr>
<tr>
<td>Mesh.Rotate</td>
<td>Structure of double arrays</td>
<td>Variable for internal use.</td>
</tr>
</tbody>
</table>

Core structure fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core.corematerial</td>
<td>String</td>
<td>Iron core material; corresponds to the Core material field of Core Property Editor.</td>
</tr>
<tr>
<td>Core.mu_lin</td>
<td>Number &gt; 0</td>
<td>Relative permeability used by linear solver; corresponds to the Linear relative permeability field of Core Property Editor.</td>
</tr>
<tr>
<td>Core.k_st</td>
<td>0 &lt; number &lt; 1</td>
<td>Iron core stacking factor; corresponds to the Stacking factor field of Core Property Editor.</td>
</tr>
<tr>
<td>Core.BHcurve</td>
<td>2×nBH cell array</td>
<td>Measured points of the BH-curve; corresponds to the BH-curve points table of Core Property Editor. First row corresponds to the magnetic flux density values, second row – to the magnetic field intensity values. nBH – number of the BH-curve points.</td>
</tr>
<tr>
<td>Core.ironlossdata</td>
<td>String</td>
<td>Name of the m-file with information about iron core losses; corresponds to the Iron loss data file field of Core Property Editor.</td>
</tr>
</tbody>
</table>
10.4. Built-in simulation script function for steady state analysis.

In section 7.1 it was mentioned that some specific built-in simulation script function is used to significantly reduce simulation time needed for Transient Analysis of steady state regimes. This simulation script function named simscript_steadystate is automatically selected when the Steady state only item of the Simulation presetting field is chosen. Some part of the code of the simscript_steadystate function will be discussed below. It can help you in writing your own simulation script functions. More examples on using simulation script functions can be found in the next chapter.

The general idea is to compute the initial conditions with time-harmonic FEA and then compute the initial transient regime with the larger time step and using linear solver. Once the steady state is reached the solver is set up to nonlinear and the time step is changed to that initially specified by user.

The following piece of code defines the necessary output structures. Note that the Enforce structure is defined depending on the Rotor speed dependency field value (Fixed speed simulation or Variable speed simulation, see section 7.1 of this manual for more details).

```plaintext
if strcmp(Settings.SpeedDependency,'Fixed speed simulation')
    Enforce.speed=2*pi*Settings.FixedSpeed/60;
elseif strcmp(Settings.SpeedDependency,'Variable speed simulation')
    Enforce=[];
end
CircuitControl=[];
ShaftPosition=[];
```

The following piece of code is called only once when the simulation is first started (when variable CurrentTime is zero). Simulation time step is stored in the Userdata structure and the new time step is computed being five times less than electrical period 1/f1. Solver type is set to ‘Linear’.

```plaintext
CurrentTime=Outputs.CurrentTime;
if CurrentTime<eps
    Userdata.timestep=Settings.timestep;  % store time step
    Userdata.extsave=Settings.extsave;
    f1=Settings.f1;
    timestep=1/f1/5;
    i=0;
    while timestep<1
        timestep=timestep*10;
        i=i+1;
    end
    timestep=floor(timestep);
    timestep=timestep/10^i;
    Settings.timestep=timestep;
    Settings.solvertype='Linear';
    Settings.extsave=0;
End
```
The following piece of code determines whether the steady state is reached using the \texttt{issteadystate} function. When the steady state is reached, the time step is changed to the value previously stored in the \texttt{Userdata} structure (this value was initially specified by user). When the steady state is reached again with the new time step, the solver type is changed to ‘Nonlinear’.

\begin{verbatim}
X=Outputs.Torque;
time=Outputs.time;
ind=find(abs(diff(diff(time)))>eps);
if length(ind)==1
    X=X(ind+1:end);
end
if issteadystate(X,Settings.timestep,Settings.f1,...
    Settings.motorload,Enforce)
    if length(ind)==1
        Settings.solvertype='Nonlinear';
        Settings.simoption='General';
        Settings.extsave=Userdata.extsave;
        Userdata=[];
    else
        Settings.timestep=Userdata.timestep;
    end
end
\end{verbatim}
11. USING ELECTRICAL CIRCUITS

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

- resistors;
- capacitors;
- inductances;
- ideal diodes;
- electronic switches;
- 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 used for the 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 an example to write a function for your own electrical circuit.

It is worth mentioning that some electrical circuits cannot be used for Equivalent Circuit Analysis, Advanced Equivalent Circuit Analysis and AC Analysis or can give incorrect results. These are circuits containing diodes and electronic switches as well circuits causing unbalanced stator currents. Only Transient Analysis can be used for this kind of electrical circuits.

To understand this chapter some familiarity with MATLAB programming is required. Refer to MATLAB help documentation for more details on MATLAB programming.

11.1. Writing an electrical circuit function.

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

```
function Schematic = StarConnection(p,t,Subdomains,Circuit,l,nper,...
  PowerSupplyType,AC_RMSSupplyValue)
```

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

The input arguments of the function:

t – variable taken from Mesh.t,
See section 10.3 with the Mesh structure description for more details on Mesh.p and Mesh.t.
Subdomains – array of subdomains.
Circuit – structure containing stator and rotor electrical circuit parameters.
1 – lamination length; this variable is taken from the Lamination length field of the Geometry Editor window.
nper – number of periodicities to define periodic/antiperiodic boundary conditions.
PowerSupplyType – this variable corresponds to the Power supply field of the main window and determines whether motor is supplied by 3-phase voltage source or by 3-phase current source.
AC_RMSSupplyValue – RMS value of supply voltage or current (depending on the PowerSupplyType variable) used for AC Analysis, corresponds to the RMS supply voltage or RMS supply current field of the main window when AC Analysis is chosen.
The following fields of the Circuit structure can be used:
Circuit.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.
Circuit.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.
Circuit.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.
Circuit.layout – stator winding layout.
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.

11.1.1. Electrical circuit branches.
Building of an electrical circuit branch 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 Branch.
When all circuit components are added to the branch object (see section 11.1.2), the branch should be added to the circuit object using the following function:

```plaintext
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.

### 11.1.2. Adding circuit components.

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

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

<table>
<thead>
<tr>
<th>Branch</th>
<th>variable corresponding to the branch which the resistor is added to, name – unique circuit component name, value – resistance value in Ohm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example of adding resistor R1 of 1 kOhm to the branch defined by variable Branch:</td>
<td></td>
</tr>
<tr>
<td>Branch = CircuitAddR(Branch, 'R1', 1000);</td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Branch</th>
<th>variable corresponding to the branch which the inductance is added to, name – unique circuit component name, value – inductance value in Henries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example of adding inductance L1 of 100 mH to the branch defined by variable Branch:</td>
<td></td>
</tr>
<tr>
<td>Branch = CircuitAddL(Branch, 'L1', 0.1);</td>
<td></td>
</tr>
</tbody>
</table>

11.1.2.3. To add a capacitor to a branch, one of the following sentences can be used:

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

| Branch = CircuitAddC(Branch, name, value, Uinit) |
|--------|-------------------------------------------------------------------------------------------------------------------------------------|
| Branch = CircuitAddC(Branch, 'C1', 100*10^-6); |

Since in this case the input variable `Uinit` is not used the initial voltage is zero.
11.1.2.4. 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 the same parallel path.

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

\[
\text{Branch} = \text{CircuitAddCoil}(\text{Branch}, \text{name}, \text{phase}, \text{Subdomains}, \text{R}, \text{Lsew}, \ldots, \text{npath}, \text{Npp}, \text{coilorientation}, p, t, l, \text{nper});
\]

- \text{Branch} – variable corresponding to the branch which the coil object is added to, \text{name} – unique circuit component name, \text{phase} – phase accepts one of the following values: 'a', 'b', 'c';
- \text{Subdomains} – array of subdomains; \text{R} and \text{Lsew} – active resistance and end winding inductance of the coil component, respectively, \text{npath} – parallel path number (\text{npath}=1 if there are no parallel paths), \text{Npp} – number of parallel paths of the stator winding per phase; \text{coilorientation} = 1 \text{ if the coil component is oriented concordantly with the branch which the coil component is added to, otherwise } \text{coilorientation} = -1. \text{ 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 } p, t, l, \text{nper} \text{ are the same as described in section 11.1.}

Note that in order to prevent incorrect results each branch should include only one coil component, otherwise coil components must be divided by additional nodes (see example 2 of this chapter).

Example of adding the coil component to the branch defined by variable \text{Branch}:

\[
\text{Branch} = \text{CircuitAddCoil}(\text{Branch}, '\text{coil} \_\text{c}2', 'c', \text{Subdomains}, \ldots, \text{R} = \text{Rs}, \text{Lsew} = \text{Lsew}_2, \text{Npp} = 1, p, t, l, \text{nper});
\]

In this case the coil component corresponds to the second group (parallel path 2) of coils of phase “c” named as coil_c2, oriented concordantly with the branch, with the active resistance \text{Rs} and end winding inductance \text{Lsew}.

11.1.2.5. To add a diode to a branch, the following function is used:

\[
\text{Branch} = \text{CircuitAddDiode}(\text{Branch}, \text{name}, \text{Roff}, \text{Ron}, \text{direction});
\]
Branch – variable corresponding to the branch which the diode is added to, name – unique circuit component name, Roff and Ron – backward and forward resistance of the diode in Ohm; direction = 1 if the diode forward direction is oriented from node1 to node2 of the branch (see description of CircuitCreateBranch in section 11.1.1), otherwise direction = -1. Note that backward and forward resistances should be as follows:

\[ 0 < R_{off} < \infty \]
\[ 0 < R_{on} < \infty \]

Example of adding diode D1 to the branch defined by variable Branch:

\[
\text{Branch} = \text{CircuitAddDiode}(\text{Branch}, 'D1', 10^8, 10^{-6}, 1);
\]

11.1.2.6. To add a voltage source to a branch, one of the following sentences can be used:

\[
\text{Branch} = \text{CircuitAddE}(\text{Branch}, \text{name}, \text{data})
\]
\[
\text{Branch} = \text{CircuitAddE}(\text{Branch}, \text{name}, \text{data}, \text{Em}, \text{phi})
\]

Use the first sentence if you are planning to use the electrical circuit only for Transient Analysis. Second sentence is suitable for all four analysis types (Equivalent Circuit Analysis, AC Analysis, Transient Analysis and Advanced EC Analysis).

Branch – variable corresponding to the branch which the voltage source is added to, name – unique circuit component name, data – CircuitControl structure field used by a simulation script function to control the voltage source (see section 11.2). Last two input variables, Em and phi, – amplitude and phase of the sinusoidal voltage source to be used by time-harmonic FE solver (for AC Analysis, Equivalent Circuit Analysis and Advanced EC Analysis).

11.1.2.7. To add a current source to a branch, one of the following sentences can be used:

\[
\text{Branch} = \text{CircuitAddJ}(\text{Branch}, \text{name}, \text{data})
\]
\[
\text{Branch} = \text{CircuitAddJ}(\text{Branch}, \text{name}, \text{data}, \text{Im}, \text{phi})
\]

Use the first sentence if you are planning to use the electrical circuit only for Transient Analysis. Second sentence is suitable for all four analysis types (Equivalent Circuit Analysis, AC Analysis, Transient Analysis and Advanced EC Analysis).

Branch – variable corresponding to the branch which the current source is added to, name – unique circuit component name, data – CircuitControl structure field used by a simulation script
function to control the current source (see section 11.2). Last two input variables, $I_m$ and $\phi$, – amplitude and phase of the sinusoidal current source to be used by time-harmonic FE solver.

11.1.2.8. To add a switch to a branch, the following function is used:

```matlab
Branch = CircuitAddSwitch(Branch,name,data,Roff,Ron);
```

Branch – variable corresponding to the branch which the switch is added to, name – unique circuit component name, data – CircuitControl structure field used by a simulation script function to control the switch state (section 11.2), Roff and Ron – ‘off’ and ‘on’ resistance of the switch in Ohm.

Note that ‘off’ and ‘on’ resistances should be as follows:

$$0 < R_{off} < \infty$$
$$0 < R_{on} < \infty$$

11.2. Controlling the power sources and electronic switches using simulation script functions.

When Transient Analysis is in progress, the voltage and current values supplied by the voltage and current sources as well as the state of each electronic switch can be specified at each time step by the simulation script function. For this purpose the CircuitControl structure, output argument of a simulation script function, is used. Each voltage or current source and switch is associated with its field of the CircuitControl structure through the input argument data, when the CircuitAddE, CircuitAddJ or CircuitAddSwitch function is called. For the power sources, the value assigned to the corresponding field of the CircuitControl structure at the specified time step is the voltage or current output value of the voltage or current source associated with this field. Similarly, the state of the switch is determined by the value assigned to the corresponding field of the CircuitControl structure at the specified time step: 1 for state ‘on’, 0 – for state ’off’ (see example 3 for more details on using electronic switches).

The following example provides an explanation for the power source control principle. Assume that the voltage source named $U_1$ was added to the stator circuit calling the function

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

In this case the voltage source $U_1$ is associated with the field $u_1$ of the CircuitControl structure. The following piece of code can be used for the simulation script function to specify the voltage value supplied by voltage source $U_1$ at the given simulation time step:

```matlab
CurrentTime = Outputs.CurrentTime;
CircuitControl.u1 = 220*sqrt(2)*sin(2*pi*50*CurrentTime);
```
In this case we have the source of sinusoidal voltage with frequency 50Hz and RMS value 220V. Note that the power sources can have any desired voltage or current waveform.

Similarly, for the switch S1 created by calling the following function:

```plaintext
Branch = CircuitAddSwitch(Branch,'S1','CircuitControl.s1',Roff,Ron);
```

Writing simulation script function you can use the following command to set the switch S1 to ‘on’ state:

```plaintext
CircuitControl.s1 = 1;
```

The following command will set the switch S1 to ‘off’ state:

```plaintext
CircuitControl.s1 = 0;
```

If the simulation script function retrieves empty CircuitControl structure or if the simulation script function is not in use (No simulation script item is selected in the Simulation script file field), default voltage or current waveform defined by Exp. 7.1 or 7.2 will be used.

If you use default stator circuit (StarConnection or DeltaConnection item of the Stator circuit pop-up menu, see section 4.2) you can still use your own voltage or current waveform. CircuitControl structure fields `ua`, `ub`, `uc` are reserved to control voltage sources (fields `ia`, `ib`, `ic` for current sources) of default stator circuits when the user defined voltage or current waveform is required.

Example of the code which can be used for writing the simulation script function adding the 5-th and 7-th order higher harmonics to the supply voltage:

```plaintext
CurrentTime = Outputs.CurrentTime;
f1 = Settings.f1; U = Settings.U;
CircuitControl.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);
CircuitControl.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);
CircuitControl.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` (`ia`, `ib`, `ic`) for the CircuitControl structure fields associated with voltage (current) sources of your circuit. In this case, similarly to using the default circuit, if the simulation script function retrieves empty CircuitControl structure or if the simulation script function is not in use, default voltage or current waveform defined by Exp. 7.1 or 7.2 will be used. Otherwise, if other field names of the CircuitControl structure are used, the voltage or current waveforms must be specified through the simulation script function.
11.3. Examples of using stator electrical circuits.

**Example 1**

This example provides an explanation for the construction procedure of the electrical circuit with the star-connected stator winding which is one of the default stator circuits used in MotorAnalysis. Example of the star-connected stator winding with two parallel paths supplied by 3-phase voltage source is shown in Figure 11.1. Implementation of the function discussed in this example is given in file StarConnection.m.

StarConnection function is suitable for stator windings with any number of parallel paths supplied whether by 3-phase voltage source or by 3-phase current source.

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

```matlab
function Schematic = StarConnection(p,t,Subdomains,Circuit,l,nper,...
    PowerSupplyType,AC_RMSSupplyValue)
```

The function has input and output arguments as it was described in section 11.1.

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

```matlab
Npp=Circuit.Npp;   % number of parallel paths
Rs=Circuit.Rs;     % parallel path phase resistance
Rs = Npp*Rs;
Lsew=Circuit.Lsew; % parallel path end winding inductance
Lsew = Npp*Lsew;
layout = Circuit.layout;
if exist('AC_RMSSupplyValue','var')
    Em=sqrt(2)*AC_RMSSupplyValue; Im=sqrt(2)*AC_RMSSupplyValue;
else
    Em=[]; Im=[];
end

[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

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

\[
Rs_{\text{par}} = Npp \times Rs \\
Lsew_{\text{par}} = Npp \times Lsew
\]
Figure 11.1. Electrical circuit of star connected stator winding with two parallel paths and 3-phase voltage source.
where $Rs$ and $Lsew$ – values specified in the **Winding phase resistance** and **End winding inductance** fields of **Windings Property Editor**, respectively, $Npp$ – number of parallel paths per phase, specified in the **Number of parallel paths** field. Since there are two parallel paths in the circuit shown in Figure 11.1, according to Exp. 11.1 for each parallel path the active resistance is $2*Rs$ and the end winding inductance is $2*Lsew$.

Variables $Em$ and $Im$ are amplitudes of the voltage and current sources, respectively, used by time-harmonic FE solver (see chapters 11.1.2.6 and 11.1.2.7 for more details).

**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:

```matlab
Schematic = CircuitCreateSchematic();
if strcmp(PowerSupplyType,'3-phase voltage source')
    if Npp==1
        Branch = CircuitCreateBranch(1,0);
        Branch = CircuitAddE(Branch,'Ea','CircuitControl.ua',Em,0);
        Branch = CircuitAddCoil(Branch,'coil_a','a',Subdomains,...
            Rs,Lsew,1,Npp,1,p,t,l,nper);
        Schematic = CircuitAddBranch(Schematic,Branch);
    end
    Branch = CircuitCreateBranch(1,0);
    Branch = CircuitAddE(Branch,'Eb','CircuitControl.ub',... Em,2*pi/3);
    Branch = CircuitAddCoil(Branch,'coil_b','b',Subdomains,...
            Rs,Lsew,1,Npp,1,p,t,l,nper);
    Schematic = CircuitAddBranch(Schematic,Branch);
    Branch = CircuitCreateBranch(1,0);
    Branch = CircuitAddE(Branch,'Ec','CircuitControl.uc',... Em,4*pi/3);
    Branch = CircuitAddCoil(Branch,'coil_c','c',Subdomains,...
            Rs,Lsew,1,Npp,1,p,t,l,nper);
    Schematic = CircuitAddBranch(Schematic,Branch);
elseif Npp>1
    Branch = CircuitCreateBranch(0,1);
    Branch = CircuitAddE(Branch,'Ea','CircuitControl.ua',Em,0);
    Schematic = CircuitAddBranch(Schematic,Branch);
    Branch = CircuitCreateBranch(0,2);
    Branch = CircuitAddE(Branch,'Eb','CircuitControl.ub',... Em,2*pi/3);
    Schematic = CircuitAddBranch(Schematic,Branch);
    Branch = CircuitCreateBranch(0,3);
    Branch = CircuitAddE(Branch,'Ec','CircuitControl.uc',... Em,4*pi/3);
    Schematic = CircuitAddBranch(Schematic,Branch);
end
```
for i=1:Npp
    Branch = CircuitCreateBranch(1,3+i);
    Branch = CircuitAddCoil(Branch, ['coil_a', num2str(i)], ...
        'a', Subdomains, Rs, Lsew, i, Npp, p, t, l, nper);
    Schematic = CircuitAddBranch(Schematic, Branch);

    Branch = CircuitCreateBranch(2,3+i);
    Branch = CircuitAddCoil(Branch, ['coil_b', num2str(i)], ...
        'b', Subdomains, Rs, Lsew, i, Npp, p, t, l, nper);
    Schematic = CircuitAddBranch(Schematic, Branch);

    Branch = CircuitCreateBranch(3,3+i);
    Branch = CircuitAddCoil(Branch, ['coil_c', num2str(i)], ...
        'c', Subdomains, Rs, Lsew, i, Npp, p, t, l, nper);
    Schematic = CircuitAddBranch(Schematic, Branch);
end

elseif strcmp(PowerSupplyType, '3-phase current source')
    if Npp==1
        Branch = CircuitCreateBranch(1,0);
        Branch = CircuitAddJ(Branch, 'Ja', 'CircuitControl.ia', Im, 0);
        Branch = CircuitAddCoil(Branch, 'coil_a', 'a', Subdomains, ... Rs, Lsew, 1, Npp, 1, p, t, l, nper);
        Schematic = CircuitAddBranch(Schematic, Branch);

        Branch = CircuitCreateBranch(1,0);
        Branch = CircuitAddJ(Branch, 'Jb', 'CircuitControl.ib', ... Im, 2*pi/3);
        Branch = CircuitAddCoil(Branch, 'coil_b', 'b', Subdomains, ... Rs, Lsew, 1, Npp, 1, p, t, l, nper);
        Schematic = CircuitAddBranch(Schematic, Branch);

        Branch = CircuitCreateBranch(1,0);
        Branch = CircuitAddJ(Branch, 'Jc', 'CircuitControl.ic', ... Im, 4*pi/3);
        Branch = CircuitAddCoil(Branch, 'coil_c', 'c', Subdomains, ... Rs, Lsew, 1, Npp, 1, p, t, l, nper);
        Schematic = CircuitAddBranch(Schematic, Branch);

    elseif Npp>1
        Branch = CircuitCreateBranch(0,1);
        Branch = CircuitAddJ(Branch, 'Ja', 'CircuitControl.ia', Im, 0);
        Schematic = CircuitAddBranch(Schematic, Branch);

        Branch = CircuitCreateBranch(0,2);
        Branch = CircuitAddJ(Branch, 'Jb', 'CircuitControl.ib', ... Im, 2*pi/3);
        Schematic = CircuitAddBranch(Schematic, Branch);

        Branch = CircuitCreateBranch(0,3);
    end
Branch = CircuitAddJ( Branch,'Jc','CircuitControl.ic', ...  
  Im,4*pi/3);
Schematic = CircuitAddBranch( Schematic,Branch);

for i=1:Npp
  Branch = CircuitCreateBranch(1,3+i);
  Branch = CircuitAddCoil( Branch,['coil_a' num2str(i)], ...  
    'a',Subdomains,Rs,Lsew,i,Npp,l,p,t,l,nper);
  Schematic = CircuitAddBranch( Schematic,Branch);

  Branch = CircuitCreateBranch(2,3+i);
  Branch = CircuitAddCoil( Branch,['coil_b' num2str(i)], ...  
    'b',Subdomains,Rs,Lsew,i,Npp,l,p,t,l,nper);
  Schematic = CircuitAddBranch( Schematic,Branch);

  Branch = CircuitCreateBranch(3,3+i);
  Branch = CircuitAddCoil( Branch,['coil_c' num2str(i)], ...  
    'c',Subdomains,Rs,Lsew,i,Npp,l,p,t,l,nper);
  Schematic = CircuitAddBranch( Schematic,Branch);

  Branch = CircuitCreateBranch(0,3+i);
  Branch = CircuitAddR( Branch,['Rn' num2str(i)],10^7);
  Schematic = CircuitAddBranch( Schematic,Branch);
end
else
  error( ' ' );
end

Depending on the chosen power supply (whether the PowerSupplyType variable is  
3-phase voltage source or 3-phase current source) different part of the code is  
executed. In case of the 3-phase voltage source, default voltage sources are used associated with ua,  
ub, uc fields of the CircuitControl structure (ia,ib,ic fields for the 3-phase current source).  
Using the simulation script function for this circuit function is not required (though still possible). If the  
stator winding has parallel paths, each parallel path (branch) is constructed within the for loop.  
As one can notice, when 3-phase current source is used the addition branch with resistor 10MOhm is  
connected to each star neutral point. Branches with current sources have infinite resistance that  
destabilizes the FE solver performance, so the branch with finite resistance connected in parallel is  
required. Since the resistor value is large enough, the influence of these additional branches on the  
circuit is negligible.
**Example 2**

This example deals with a three-phase induction motor connected to a single-phase supply. The electrical circuit for the single phase connection of the stator winding used in the example is shown in Figure 11.2.

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

![Electrical circuit for the single phase connection of the stator winding.](image)

**Figure 11.2.** Electrical circuit for the single phase connection of the stator winding.

**SinglePhaseConnection function source code:**

```matlab
function Schematic = SinglePhaseConnection(p,t,Subdomains,Circuit,...
  l,nper,PowerSupplyType,AC_RMSSupplyValue)

Npp=Circuit.Npp;       % number of parallel paths
Rs=Circuit.Rs;
Rs = Npp*Rs;           % parallel path phase resistance
Lsew=Circuit.Lsew;
Lsew = Npp*Lsew;       % parallel path end winding inductance
layout = Circuit.layout;
if exist('AC_RMSSupplyValue','var')
  Em=sqrt(2)*AC_RMSSupplyValue;
else
  Em=[];
end
if Npp>1
```

```
error('SinglePhaseConnection function does not support parallel coil connection. You can provide your own code to fulfill this requirement');
end

Schematic = CircuitCreateSchematic();

Branch = CircuitCreateBranch(0,1);
Branch = CircuitAddE(Branch,'U1','CircuitControl.u1',Em,0);
Schematic = CircuitAddBranch(Schematic,Branch);

Branch = CircuitCreateBranch(1,0);
Branch = CircuitAddCoil(Branch,'coil_a','a',Subdomains,...
    Rs,Lsew,1,Npp,1,p,t,l,nper);
Schematic = CircuitAddBranch(Schematic,Branch);

Branch = CircuitCreateBranch(0,2);
Branch = CircuitAddCoil(Branch,'coil_b','b',Subdomains,...
    Rs,Lsew,1,Npp,1,p,t,l,nper);
Schematic = CircuitAddBranch(Schematic,Branch);

Branch = CircuitCreateBranch(1,2);
Branch = CircuitAddC(Branch,'c',30*10^-6,0);
Branch = CircuitAddCoil(Branch,'coil_c','c',Subdomains,...
    Rs,Lsew,1,Npp,1,p,t,l,nper);
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 11.1.2.4, 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 11.2, node 2 was added to the circuit to fulfill this requirement.

Implementation of the function for single phase connection of the 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 ul field of the CircuitControl structure. Besides voltage source control, the function also stores the capacitor voltage values at each time step in the Userdata structure.

The source code of the simulation script function:

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

ShaftPosition=[];
Enforce = []; 
CircuitControl = []; 
CurrentTime = Outputs.CurrentTime; 
U = Settings.U; 
f1 = Settings.f1; 
ul = U*sqrt(2)*sin(2*pi*f1*CurrentTime); 
CircuitControl.ul = ul; 

% 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

This example also demonstrates how to use the Userdata structure to store your own time-series data. Capacitor voltage data are stored in this case. When the simscript_SinglePhase function is called for the first time the empty array Userdata.Uc=[] is created. Every subsequent call the capacitor voltage value for the current time step is added to the array Userdata.Uc. You can plot time-series data stored in the Userdata structure using Plot Wizard the same way as it is described in section 7.3.1.

Example 3
This example demonstrates how to use MotorAnalysis to study induction motors with PWM supply. The electrical circuit used in the example is shown in Figure 11.3. Implementation of the circuit function of this example is given in file InverterCircuit.m. This circuit function supports star-connected stator windings with different number of parallel paths. The circuit shown in Figure 11.3 has two parallel paths. Simulation script function related to the example can be found in file simscript_pwm.m, example of simulation file - example_pwm.mat. The simulation script function generates switching signals for the inverter to get the desired stator current waveform.
Figure 11.3. Example of inverter electrical circuit.
Though PWM inverter circuit is not suitable for the time-harmonic FEA you can still use the time-harmonic FEA based initial conditions to reduce simulation time before the steady state is reached (see section 7.1). When you start simulation of the current example for the first time the following dialog box will appear:

Click the Yes button and choose file StarConnection.m from the appearing list of files. The initial conditions will be computed assuming standard sinusoidal voltage sources and star-connected stator windings and then the transient simulation will be continued with inverter supply.
12. ADDITIONAL SETTINGS

MotorAnalysis additional settings stored in 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 \( 0 < a < 1 \). 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:**
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 NaN values.

**Equivalent circuit analysis:**
HarmonicLeakage – determines whether to include the harmonic leakage inductance into equivalent circuit analysis as a part of stator leakage inductance \( L1 \) or not.
APPENDIX

Appendix A. Power balance and accuracy of the Transient Analysis 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:

\[ P_{\text{input}} = P_{\text{cons}} \]

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

\[ P_{\text{input}} = \sum_{n=1}^{m} i_n u_n \]

The consumed power is defined as follows:

\[ P_{\text{cons}} = P_r + P_s + P_{\text{mech}} + \frac{\Delta W_{mf}}{\Delta t} \]

where \( P_r \) – apparent power (real and reactive) of the rotor electrical circuit, \( P_s \) – apparent power (real and reactive) of the stator electrical circuit, \( P_{\text{mech}} \) – 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 the user to estimate the reliability of the transient simulation results.

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

MotorAnalysis has been currently tested with MATLAB 2009b and MATLAB 2014a. MATLAB 2009b demonstrated better performance comparing with MATLAB 2014a in terms of number of finite elements and speed. MATLAB 2009b worked stably with 10 slices and the mesh in each slice consisting of \(~17000\) nodes and \(~34000\) elements, so the total amount of mesh nodes and finite elements were \(~170000\) and \(~340000\), respectively. The same mesh with MATLAB 2014a caused the “out of memory” error.

“Out of memory” errors can occur when using the mesh of too large size. According to the 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.