SWAT-EM Documentation

Release 0.5.0

Martin Baun

Aug 21, 2020

CONTENTS

1 Table of Contents **3**
- 1.1 Installation . . . 3
- 1.2 Theory . . . 4
- 1.3 Using the graphical user interface . . . 9
- 1.4 Using the api . . . 20
- 1.5 API Reference . . . 28

2 Indices and tables **35**

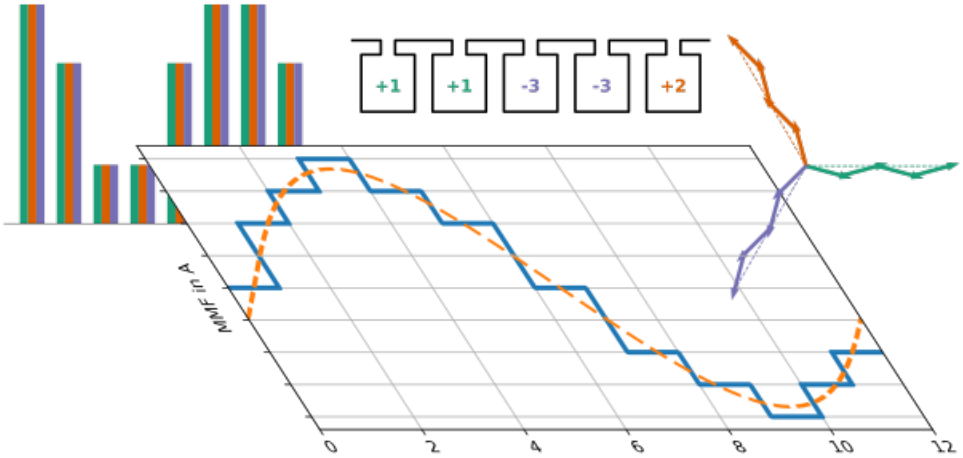
Bibliography **37**

Python Module Index **39**

Index **41**

SWAT-EM

Specific Winding Analyse Tool for Electrical Machines



SWAT-EM is a software for designing and analyzing winding systems for electrical machines. Currently supported are rotating field windings (permanent-magnet motors, induction motors, synchronout reluctance motors) with any number of phases. This can be distributed full pitch or fractional slot winding or tooth-coil winding. The design can be done by

- Generating with manual allocation of the coil sides to stator slots
- Defining individual number of turns for each coil
- Automatic winding generators
- Tables of possible winding systems for slot/pole combinations

Analyzing features

- Calculation of the winding factor based on the voltage star of slots
- Plot of the winding layout
- Plot of stator ampere-conductor distribution and the magnetomotive force (MMF)
- Plot of the slot voltage phasors
- Plot of the winding factor
- Max. possible number of parallel circuit connection of coils

There are two ways how to use SWAT-EM:

- Working with the graphical user interface (GUI)
- Working with python-API

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

There are different ways to install SWAT-EM.

1.1.1 Installer on Windows

SWAT-EM is based on python3 interpreter and some additional libraries. If you haven't this on your computer the easiest way is to download the SWAT-EM installer from: https://sourceforge.net/projects/swat-em/ Start the installer and follow the instruction. No further work is necessary. After installation the program can be startet by double-clicking the Desktop-Icon or with the entry in the start menu.

1.1.2 PIP

Use this install method if you are on LINUX or macOS or if you still have an python3 environment with pip on your computer. SWAT-EM is hosted on the Python Package Index (pip). To install open a terminal on your computer and type

```
pip install swat-em
```

pip will install all necessary dependencies.

1.1.3 From source

The source an be downloaded from https://gitlab.com/martinbaun/swat-em/ and installed from the project root directory:

```
$ python setup.py install
```

1.1.4 Run the programm

Run the programm with graphical user interface by starting it from the command line:

```
swat-em
```

or with:

```
swat-em.exe
```

under windows. If you have installed swat-em with the windows installer you can use the desktop icon to start.

1.2 Theory

1.2.1 Introduction

Designing, rating and choosing winding systems for electrical machines is a complicated task. Often some experience is necessary. This indroduction doesn't replace suitable education or a specialist book about this topic. This introduction only covers the basics to understand the following explanations.

1.2.2 Exemplary winding systems

Simple overlapping winding

For the beginning let's have a look how we can collect magnetic flux generated by a permanent-magnet rotor through a coil. The highest flux we get, if we define the coil width W equal to the pole pitch τ_p. However this in practise this often isn't the best choise because of the high harmonic content. Most windings have $W < \tau_p$.

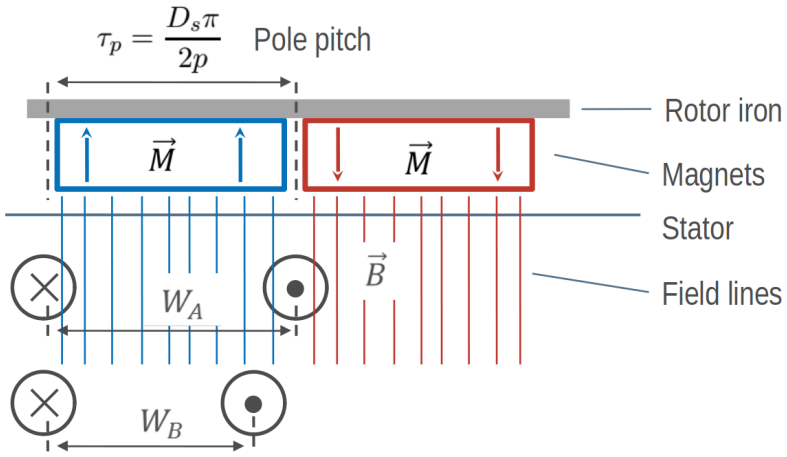


Fig. 1: How to get flux, based on the rotor, through a coil

For a symmetric three-phase winding we have to add two more coil which are shifted by 120°. For two poles this is one of the most simplest winding.

```
Number of slots:  6
Number of poles:  2
Number of phases: 3
```

(continues on next page)

(continued from previous page)

```
Number of layers: 1
Winding step    : 3
Number of slots per pole per phase: 1
Fundamental winding factor: 1.0, 1.0, 1.0
```

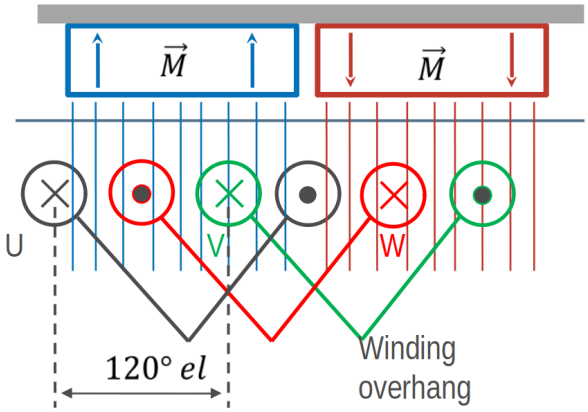


Fig. 2: Overlapping winding with 6 slots, 2 poles and 3 phases

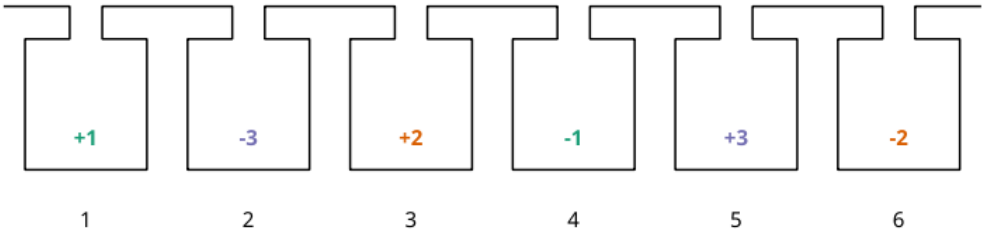


Fig. 3: Generated overlapping winding

Simple tooth-coil winding

Besides of the overlapping winding there is another winding winding systems - tooth coils. To get such a winding the winding step must be exactely $W = 1$. This means, that the distance between a wire and its reverse wire is one slot.

We can set the winding step explicite with the keyword 'stepwidth'. Compared to the overlapping winding we need only 3 slots for the two poles. To get a coil around every tooth, we need two winding layers:

```
Number of slots:  3
Number of poles:  2
Number of phases: 3
Number of layers: 1
Winding step    : 1
Number of slots per pole per phase: 1/2
Fundamental winding factor: 0.866, 0.866, 0.866
```

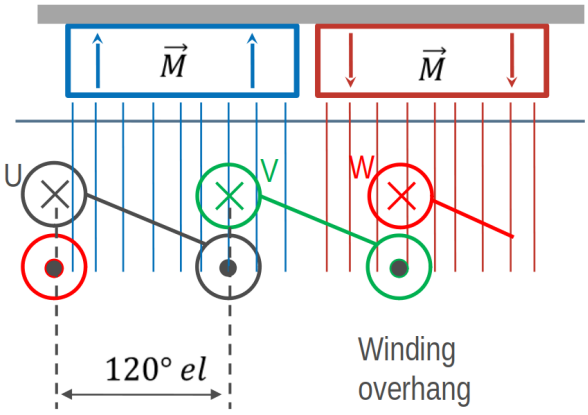


Fig. 4: Tooth-coil winding with 3 slots, 2 poles and 3 phases

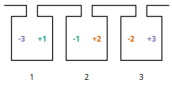


Fig. 5: Winding layout with 3 slots, 2 poles and 3 phases

A more complex winding

A more complex winding (overlapping full pitch winding with coil shortening)

```
Number of slots:  12
Number of poles:  2
Number of phases: 3
Number of layers: 2
Winding step    : 5
Number of slots per pole per phase: 2
Fundamental winding factor: 0.933, 0.933, 0.933
```

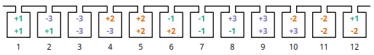


Fig. 6: Winding layout with 12 slots, 2 poles and 3 phases

1.2.3 Evaluation parameters

Winding factor

The winding factor k_w describes the coupling of the winding with the existing field in the stator. It depends on the ordinal number ν (electrical or mechanical ordinal number possible). There are many methods for calculating the winding factor, for example from the MMF ref{sec:MMF}.

Unfortunately there are limitations of this method because for three-phase windings the factor k_{w3} can't be determined. Further calculation methods derives specific equations based on the winding zones. However theses equation are not universal, so there are many equations for different winding systems. To be general SWAT-EM uses the phasors of the star of slots. The absolute value of the winding factor is defined by

$$|k_w| = \frac{|\sum E_i|}{\sum |E_i|}$$

and for all harmonics with

$|k_{w,\nu}| = \frac{|\sum E_{i,\nu}|}{\sum |E_{i,\nu}|}$

The winding factor gets the maximum value of 1 if all phasors of a phase have the same phase angle. Typically the winding factor is specified with a sign. This indicates the direction of the magnetic field wave, that is generated by the winding in the airgap. SWAT-EM determines the sign by generating the phasors plot for every harmonic number and detecting the sequence of the phases. The winding factor can referred to the mechanical ν or the electrical ν_{el} ordinal number.

Mechanical harmonics This representation is useful to detect all possible rotor pole numbers, which can be combined with the winding. Especially tooth-coil windings have many harmonics and so there are many pole-pairs per winding layout is possible.

Electrical harmonics If one have chosen a winding and a number of pole-pairs of the rotor it's a good idea to switch to the electrical ordinal numbers. Here the numbers describes influence of the winding of the waveform of the back-emf for permanent-magnet machines for example. If the winding factor for the harmonics is low, the waveform is more sinusoidal.

Double linked leakage

It's often a goal to reach a sinusoidal airgap field while designing windings for electrical machines. Harmonics could lead to noise and additional losses. Especially for induction machine there should as little as possible harmonics. The double linked leakage coefficient represents this harmonic content as the ratio of the magnetic energy of harmonics and subharmonics relative to the fundamental.

$\sigma_d = \frac{1}{k'^2_{w1}} \sum_{\nu \neq p}^{\infty} \left(\frac{k'_{w\nu}}{\frac{\nu}{p}} \right)^2$

By default SWAT-EM uses the star of slots for determining the winding factor. For calculating the double linked leakage this isn't useful because for the airgap field is generated by all phases and some of the harmonics cancel each other out. It is preferable to calculate the windingfactor k'_w from the MMF.

$k'_{w\nu} = \frac{C_\nu \pi \nu}{3\sqrt{2} I w}$

where C_ν are the amplitude of the fourier analyses of the MMF, I the current amplitude (for MMF plot $I = 1A$) and w the series number of turns per phase. More about this one can find in [Got07] and [Obe65] for example.

Magnetomotive force (MMF)

For evaluation of the winding the so called "Magnetomotive force" or short MMF is a useful tool. It is based on the the ampere-conductor distribution. This is shown for time $t = t_1$ with respect to the AC current system of m phases.

For every slot the winding direction ($d = \pm 1$), number of turns N_c the current i gets summed up

$$\Theta_{slot} = \sum d \cdot i N_c.$$

Therefor the distribution of ampere-turns is coupled with number of slots. The lower part in figure *Plot of the ampere-conductor distribution and the Magnetomotive force (MMF)* shows this for a winding example with $Q = 12$ slots, so there are 12 bars. In reality the distribution has a width per bar which corresponds to the slot opening. However in theory (in this program) the distribution can be interpreted as infinitely thin peaks. The integral of this leads to the MMF

$$MMF(\alpha) = \int_0^{2\pi} \Theta d\alpha,$$

which is shown in upper part in figure *Plot of the ampere-conductor distribution and the Magnetomotive force (MMF)*. The waveform of the MMF corresponds to the magnetic field, that is generated in the airgap by the winding. For further information consider the literature (eg cite{hendershot2010design}).

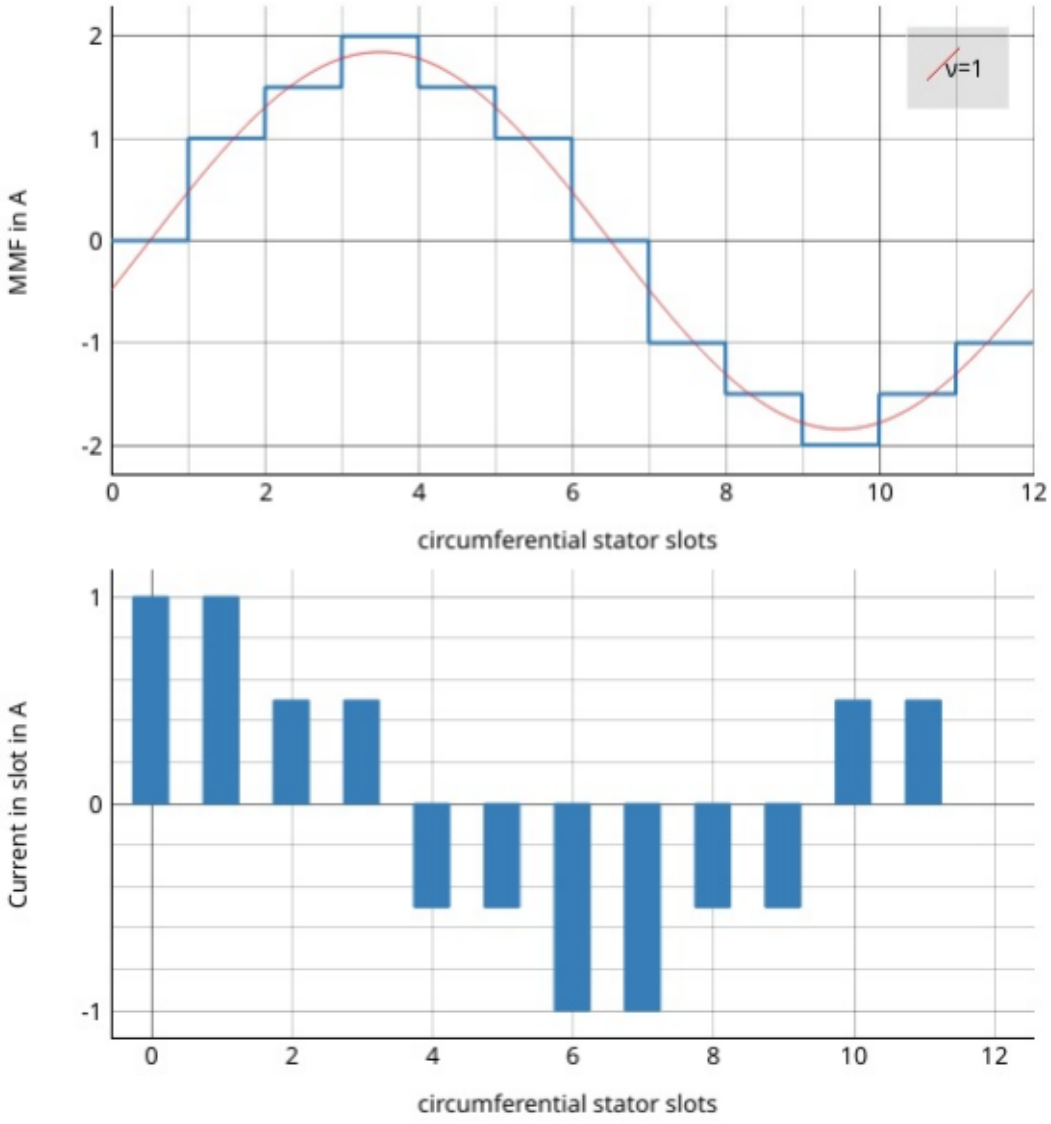


Fig. 7: Plot of the ampere-conductor distribution and the Magnetomotive force (MMF)

1.3 Using the graphical user interface

SWAT-EM comes with an QT based graphical user interface (GUI). The layout of the main window consists of the

- Workspace (1)
- Winding information's (2)
- Graphical analysis and report (3)

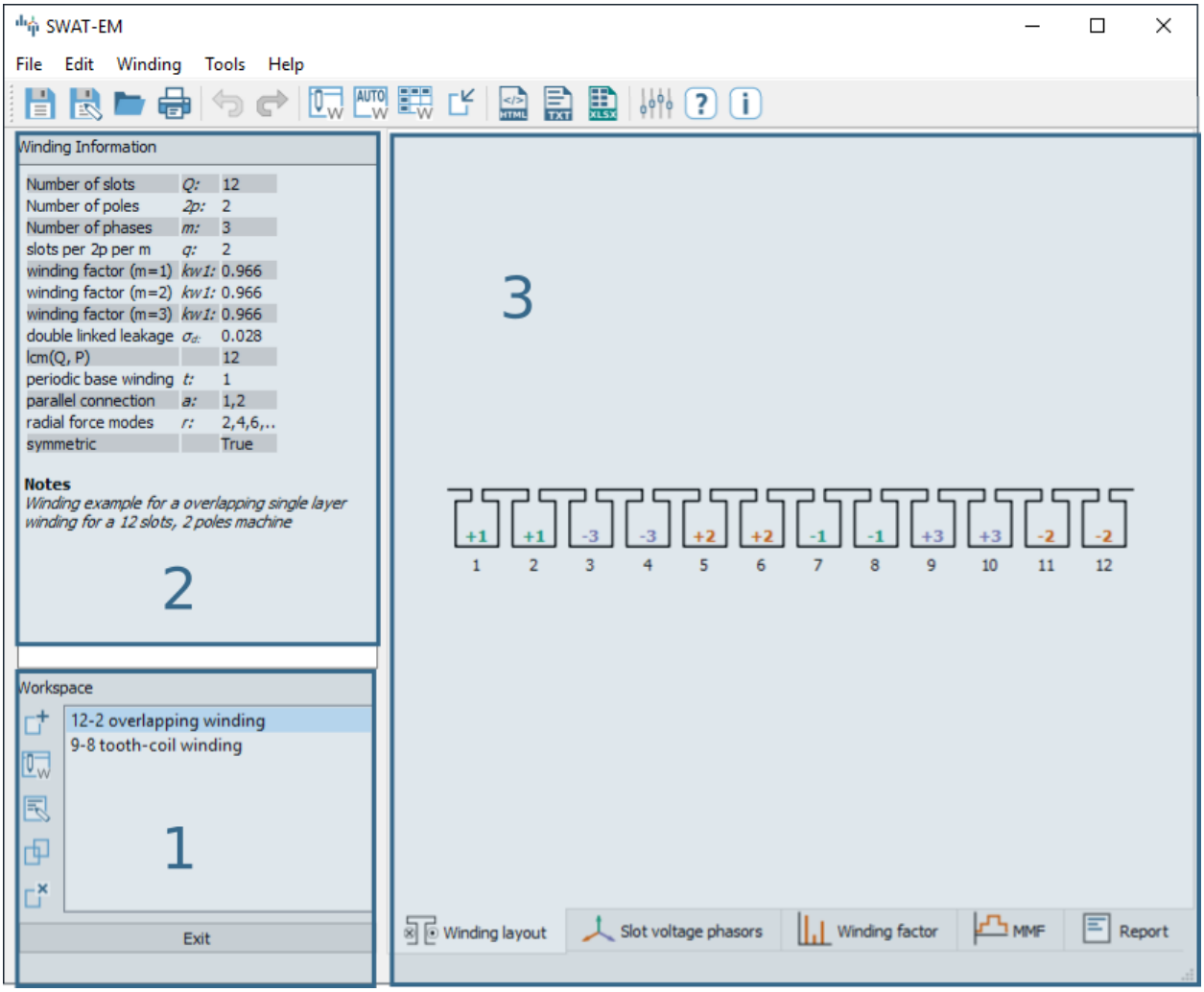


Fig. 8: Main-window

1.3.1 Workspace

A SWAT-EM project, that can be saved as *.wdg file can contain several different windings system. So, one can define and compare these windings in the same window. The workspace shows all the windings of the project. By clicking of the name all outputs (text and plots) gets updated. The buttons on the left of (1) in figure (*Main-window*) modifies the windings in the workspace

New winding Opens a dialog with all existing winding generator (see section winding generators). One can choose any of these generators to create a winding layout.

Manual winding layout Define the position of all coil sides by hand. (Not very comfortable but full control)

Auto winding layout Generates the winding automatically by number of slots, poles, ... (easy to use, almost every symmetric winding is possible)

Winding table Shows table slot/pole combinations (a good overview of possible combinations)

Notes If there a many windings in the project it might be a good idea to add some notes to the different layouts.

Clone For modifying windings one can clone/duplicate an existing one. So a switch-back to the initial state and a comparison is possible.

Delete Deletes the selected winding.

While saving the project to file (File $\rightarrow$ save) all windings of the workspace are saved. **Note:** Renaming of windings is possible by double-click or by pressing F2 on keyboard.

1.3.2 Winding information

The text field (2) in figure (*Main-window*) shows a summary of actual winding.

Q Number of stator slots

$2p$ Number of poles

m Number of phases

q Number of slots per pole per phase $q = \frac{Q}{2pm}$

$kw1$ Fundamental winding factor (for separate for each phase)

σ_d Double linked leakage (based on MMF)

$lcm(Q, P)$ Least common multiplier of number of slots an pole pairs. For permanent-magnet machines this is the first harmonic number of the cogging torque

t Periodicity of the base winding $t = gcd(Q, p)$.

a Number of possible parallel winding circuit. (In most cases a is equal to t)

$symmetric$ True, if all phases are identically and shifted by a constant angle

$Notes$ User defined description

1.3.3 Plotting

Many analyzing function results in plots which are shown on(3) in figure (*Main-window*). Every plot has a toolbar on the bottom for zooming, panning and saving the figure to file.

Winding layout

The winding layout plot shows sketched slots and coil sides. The number and color defines the number of phase the coil side belongs to. The sign (+ or -) defines the winding direction (+ means that the wire goes into the plain and - out of the plain)

Slot voltage phasors

The impact of the coils can be represented by the star of slot. The theory behind this is described in cite{mueller1996berechnung} for example. Every coil side S_i gets a phasor assigned with the angle

$$\alpha_i = \frac{2p\pi S_i}{Q} \tag{1.1}$$

The angle of the phasors can also be determined for the harmonics by adding the electrical ordinal number ν_{el}

$$\alpha_{i,\nu} = \frac{2\nu p\pi S_i}{Q}$$

with p pole pairs and the number of stator slots Q. If the coil side has a negative winding direction π is added to $alpha_i$ (turning down the phasor). With this the phasers E_i can be generated in the complex plane

$$E_i = e^{j\alpha_i}$$

All phasors of a phase are getting grouped a vectorial summed up which is shown as (1) in figure phasors plot. The dotted line represents the vectorial sum. The amplitude and the phase of this is shown in (2).

Options:

harmonic The star of slots can be drawn for any harmonic number by using eqn. (1.1).

force phase 1 on x-axis The angle of the sum of phasors depends on the location of the coil sides in the slots. If the whole winding is shifted by some slots the winding is still the same winding. However the phasors are getting a phase shift. To compare different windings and for having an unified diagram one can set this checkbox.

Winding factor

The winding factor k_w describes the coupling of the winding with the existing field in the stator (see theory section for further informations). Figure *Winding factor plot* shows the values in (1) as a table and the absolute values as a bar plot in (2). The sign in (1) gives information about the phase sequence of the corresponding harmonic.

Both can be displayed with respect to the mechanical ν or the electrical ν_{el} ordinal number by the radio buttons on the top of the table.

Mechanical harmonics This representation is useful to detect all possible rotor pole numbers, which can be combined with the winding. Especially tooth-coil windings have many harmonics and so there are many pole-pairs per winding layout is possible.

Electrical harmonics If one have chosen a winding and a number of pole-pairs of the rotor it's a good idea to switch to the electrical ordinal numbers. Here the numbers describes influence of the winding of the waveform of the back-emf for permanent-magnet machines for example. If the winding factor for the harmonics is low, the waveform is more sinusoidal.

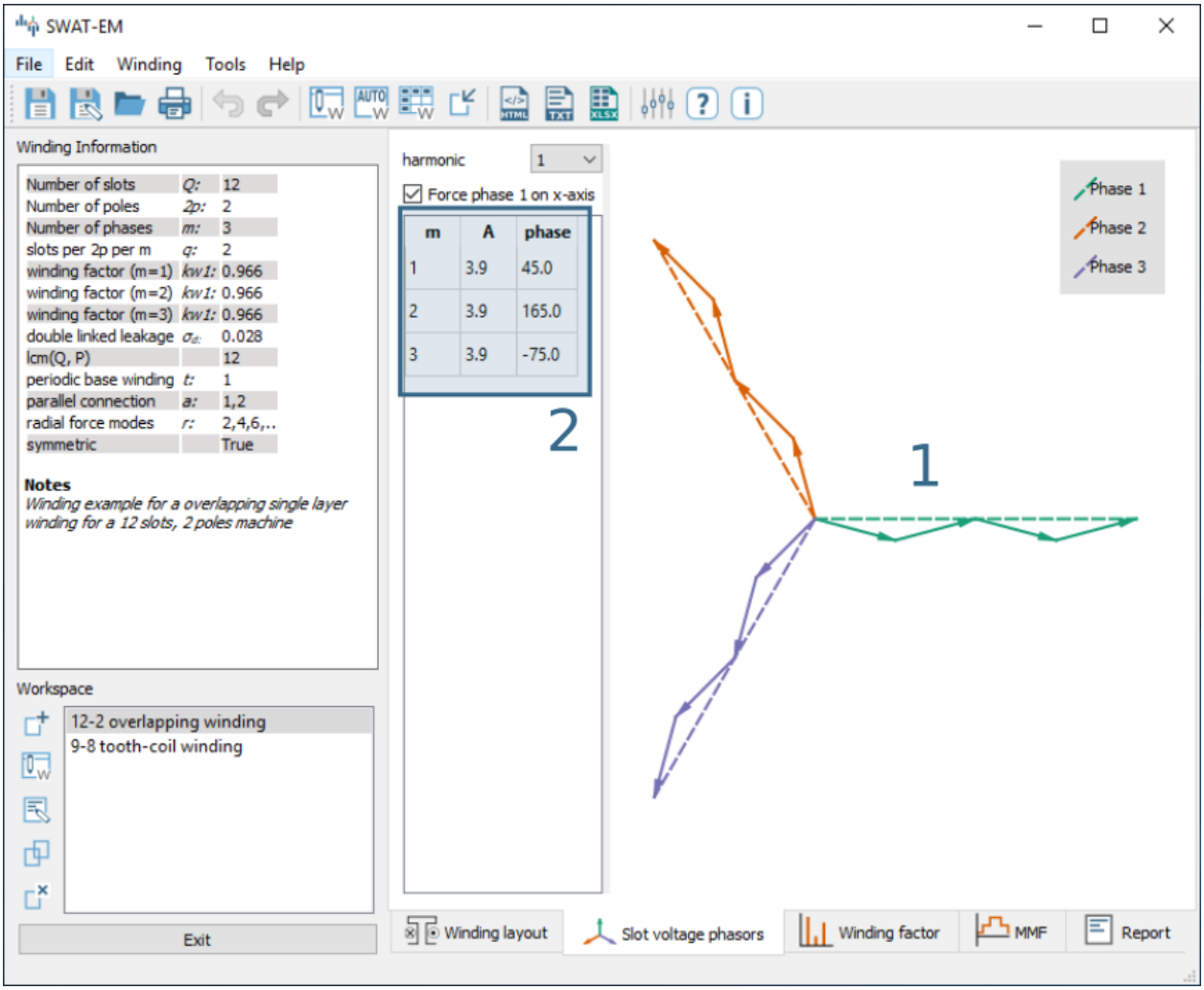


Fig. 9: Phasors plot

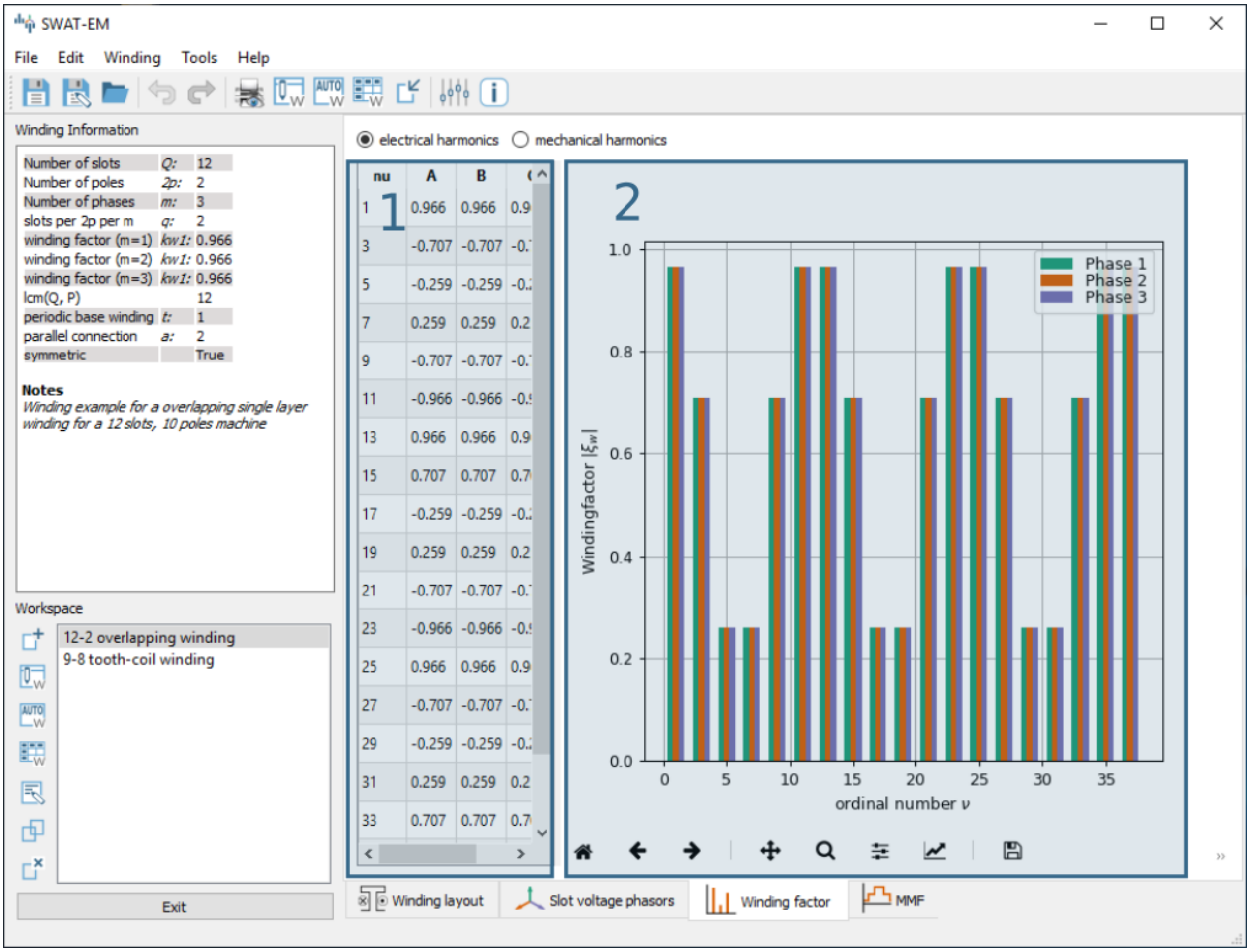


Fig. 10: Winding factor plot

Magnetomotive force (MMF)

For evaluation of the winding the so called "Magnetomotive force" or short MMF is a useful tool. It is based on the the ampere-conductor distribution. This is shown for time $t = t_1$ with respect to the AC current system of m phases.

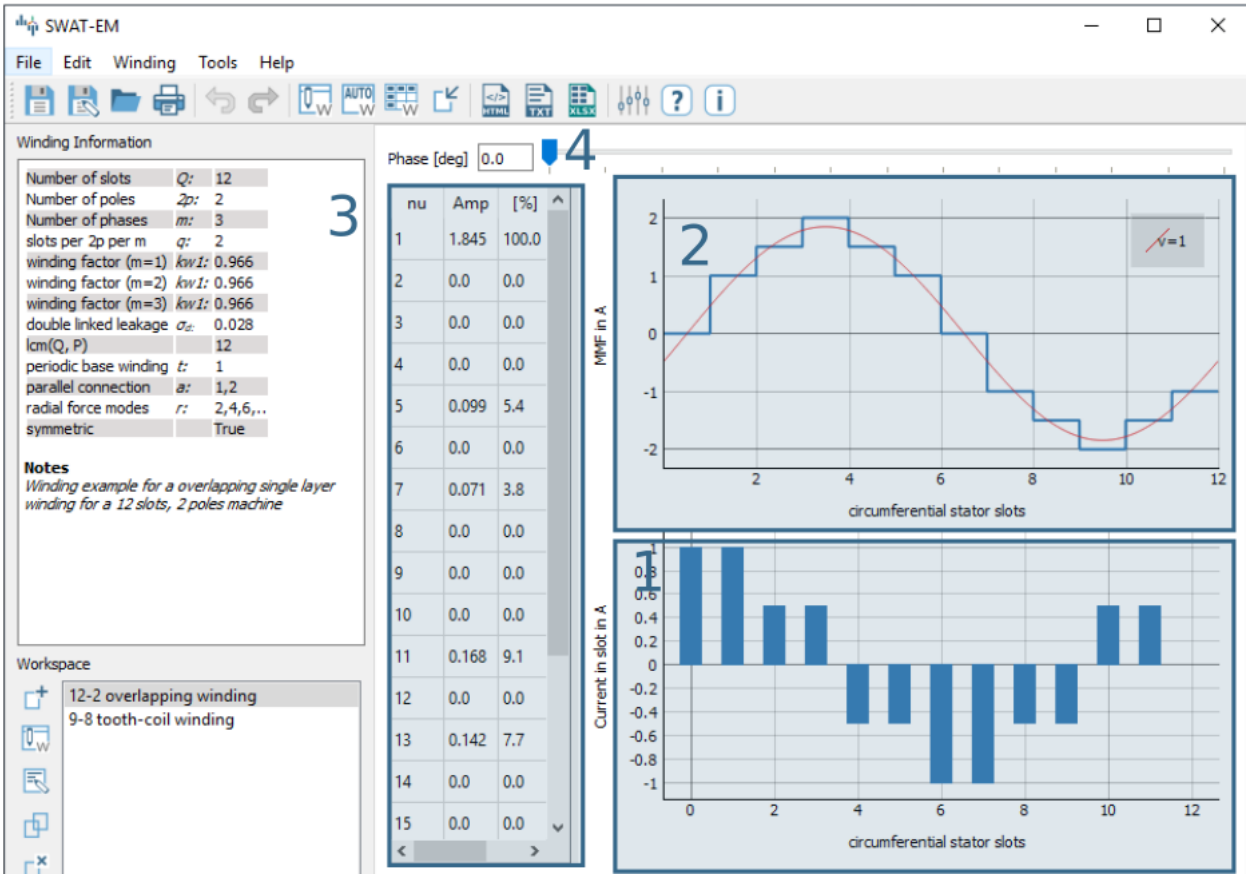


Fig. 11: Plot of the ampere-conductor distribution and the Magnetomotive force (MMF)

(1) in Figure *Plot of the ampere-conductor distribution and the Magnetomotive force (MMF)* shows the distribution of ampere-turns. Because this winding example has $Q = 12$ slots, so there are 12 bars. In reality the distribution has a width per bar which corresponds to the slot opening. However in theory (in this program) the distribution can be interpreted as infinitely thin peaks. The integral of this leads to the MMF which is shown in (2). The plot also shows the fundamental and some of the harmonics. The number of harmonics which are plotted can be defined relative to the fundamental. Please consider the "Tools" → "Settings" dialog. Table (3) in the window displays the harmonic analyses of the MMF. With the slider (4) one can define the phase angle of the AC current system for the MMF plot. Note that the phase angle has no effect on the harmonic content of the MMF, so the harmonic analyses is independent from it.

1.3.4 Winding Generators

SWAT-EM comes with many different winding generators. Each of them have different features.

Manual layout

The manual layout generator (figure *Manual winding generator*) is the most basic generator in SWAT-EM. One can define the position and the number of turns for each coil side by hand. With this every winding layout can be sketched and analyzed. The price of this is the comparatively large manual effort.

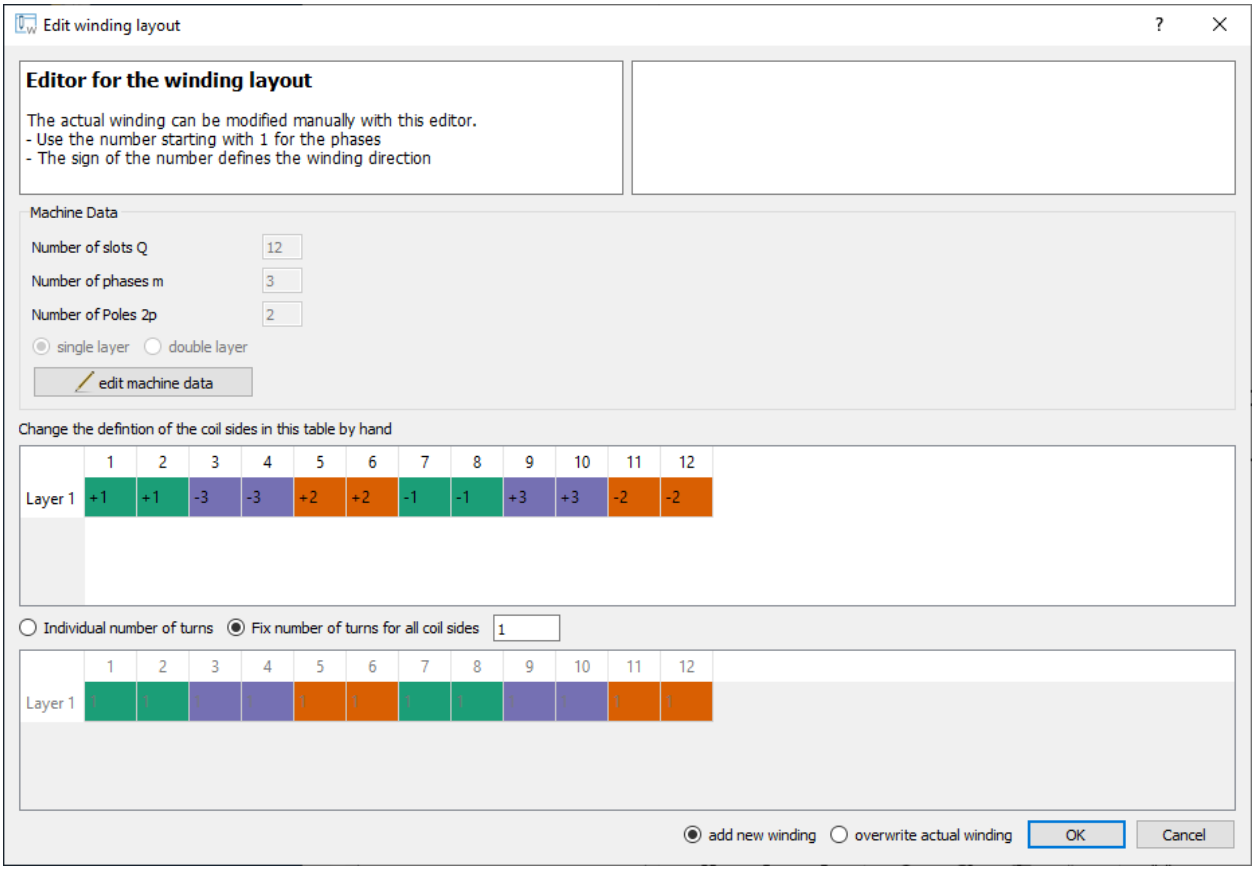


Fig. 12: Manual winding generator

Button "edit machine data" Use this dialog if you want to change the number of slots Q, of phases m, of poles $2p$ or layers.

definition of the coil sides Use the table to define the phase for the layers in each slot. The number describes the phase number. The color is added automatically for overview. The sign defines the winding direction (+ into the plane, - out of the plane)

number of turns If radio button is set to "fix number of turns for all coil sides" one can type the number of turns in the edit field apart from that. While choosing "individual number of turns" one can define this for each coil side. Use the table below

info On the upper right there is an info field. While the user defines the winding there is a live-analysis. If there is an unsymmetrical winding or if the sum of all winding turns is not zero for example, the user get an info.

overwrite winding There are two different possible action while exiting an generator dialog with the ok button. If the radio button "add new winding" is selected, the winding in the generator winding is added to the workspace

in the main window. If "overwrite" is selected, than the actual selected winding of the workspace getting overwritten. Be relaxed, if you have overwritten your winding accidentally, there is an undo function in the main window.

Automatic layout

With the automatic winding generator it is possible to generate almost every symmetric winding system. This includes

- overlapping full pitch winding
- overlapping fractional slot winding
- tooth coil winding
- dead coil windings (with empty slots)
- all above as single-layer or double-layer

This generator uses the star of slots to for defining the coil sides in the slots, based on the theory of [BianchiDaiPre06].

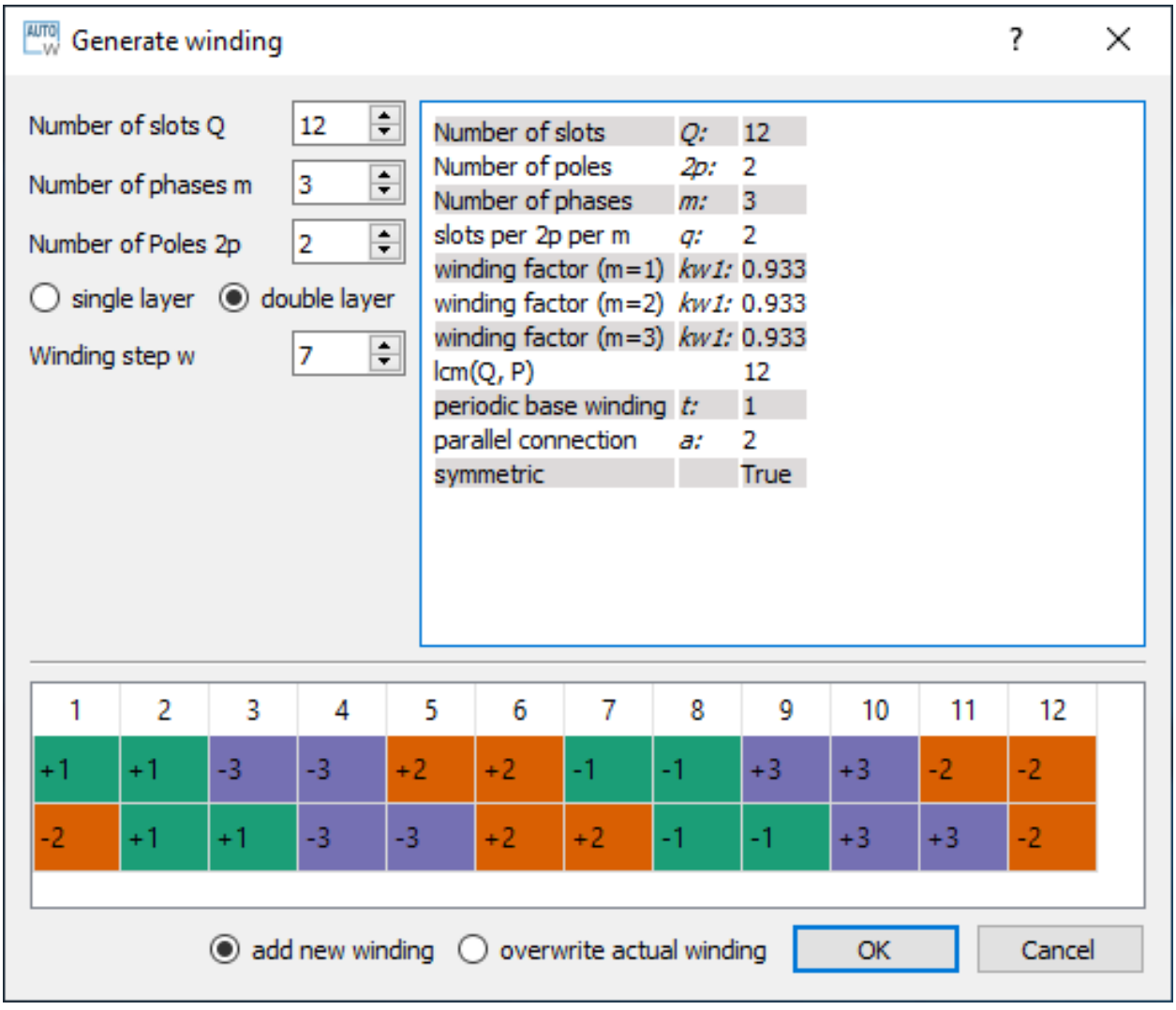


Fig. 13: Automatic winding generator

Machine data Number of slots Q, phases m and poles $2p$

layer Double layer winding means, that in every slot there are two coil sides (from the same or from different phases)

winding step Every coil has an "in" and an "out" conductor, which are connected via the winding overhang. The winding step defines the distance between "in" and "out" in slots. If winding-step is 1 a tooth-coil winding will be created. Note: For single layer windings there are some restriction to accommodate all coil sides, so in this case the winding step can't be influenced.

overwrite winding There are two different possible action while exiting an generator dialog with the ok button. If the radio button "add new winding" is selected, the winding in the generator winding is added to the workspace in the main window. If "overwrite" is selected, than the actual selected winding of the workspace getting overwritten. Be relaxed, if you have overwritten your winding accidentally, there is an undo function in the main window.

layout table The lower table shows the actual defined winding. Note, that layout can't changed here by hand. If you want to change, than accept the winding with OK to the workspace in the main window and use the manual generator (section Manual generator). The winding will be transmitted.

Winding table

This generator gives an overview about possible slot/poles combinations. So it's generator with a broad but not very deep view on windings. It can be useful in the early state of designing electrical machine, for example to define the appropriate number of slots and poles.

While clicking on a item in the upper table, the winding characteristics shown on the left side and the winding layout is shown on the bottom table. As with the other generators the selected winding can be transferred to the workspace in the main window.

For some slot/pole combinations there are many winding system possible where this generator shows the winding with the highest fundamental winding factor $k_{w,1}$. At this time there is no way to modify the windings (changing winding steps for example). For more control you have to use other generators like sec:Manual generator or automatic generator.

Number of slots Defines the range of number the number of slots Q for the table. For symmetric windings the number of slots must be a integer multiple of the number of phases m. .. math:: Q = k cdot m, text{ with }k = 1, 2, 3...

For single layer windings (without dead coil windings) the number of slots must be doubled .. math:: Q = 2 cdot k cdot m, text{ with }k = 1, 2, 3...

Number of poles The number of poles $2p$. Only even integer values ≥ 2 are valid.

Number of phases The number of phases m in the machine. Every integer value > 1 is valid.

layers Defines the number of layers for the table. At this time only single layer and double layer windings are possible.

Force tooth coil winding In some cases you may want to realize tooth coil windings, even when the winding factor isn't very high. In this case the winding step is set to $w = 1$.

overwrite winding There are two different possible action while exiting an generator dialog with the OK button. If the radio button "add new winding" is selected, the winding in the generator winding is added to the workspace in the main window. If "overwrite" is selected, than the actual selected winding of the workspace getting overwritten. Be relaxed, if you have overwritten your winding accidentally, there is an undo function in the main window.

plot value Defines the number which is shown in the upper table.

kw1 The fundamental winding factor. A big number (near to 1) means a high-torque.

q The number of slots Q per pole $2p$ per phase m. It characterized the winding system. $p = \frac{Q}{2p \cdot m}$

t The number of the periodic sequence of identical "base-" windings.

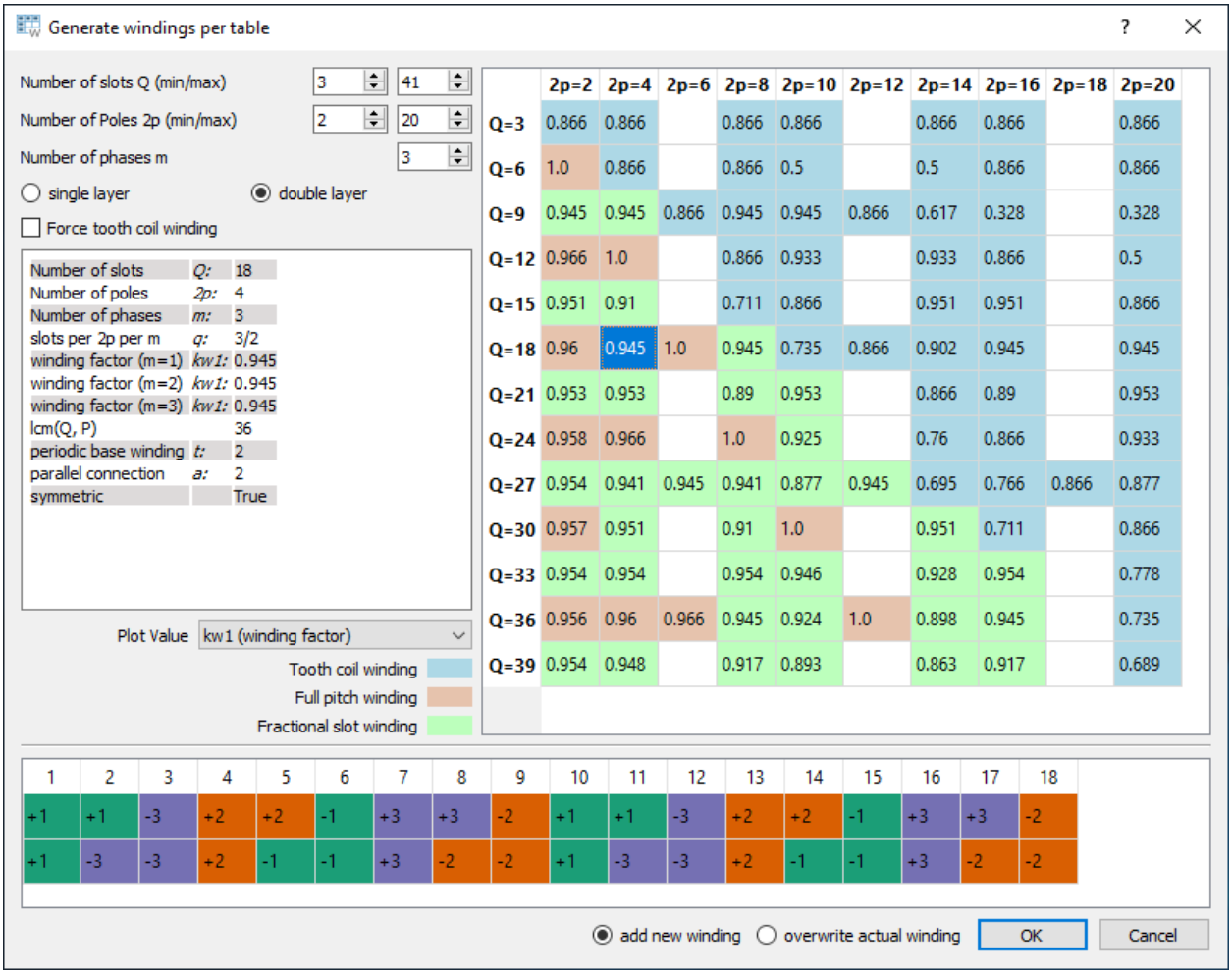


Fig. 14: Table of possible windings for diffrent slot/pole combinations

a The number of possible parallel circuits of coil groups in the winding. In most cases it`s the same as $t$. But for some windings it`s possible to connect coil groups in parallel while changing the start and end of the coils.

lcm(Q,2p) Means the least common multiple of the number of slots Q and number of poles $2p$. For permanent-magnet machines this is the first ordinal number of the cogging torque. Tends to be true: The higher the ordinal number the lower the amplitude of the cogging torque.

r1 This shows the ordinal numbers of the radial force mode caused by the winding.

sigma_d The coefficient of the double linkead leakage flux is a measure of the harmonic content of the MMF in the airgap caused by the winding. As higher the number as higher the harmonics.

1.3.5 Import winding

As in as in *Workspace* described you can have many winding system in the workspace. In some cases you may want to have a winding in your workspace which is saved as a *.wdg file on the hard disk. This can be done by the import function.

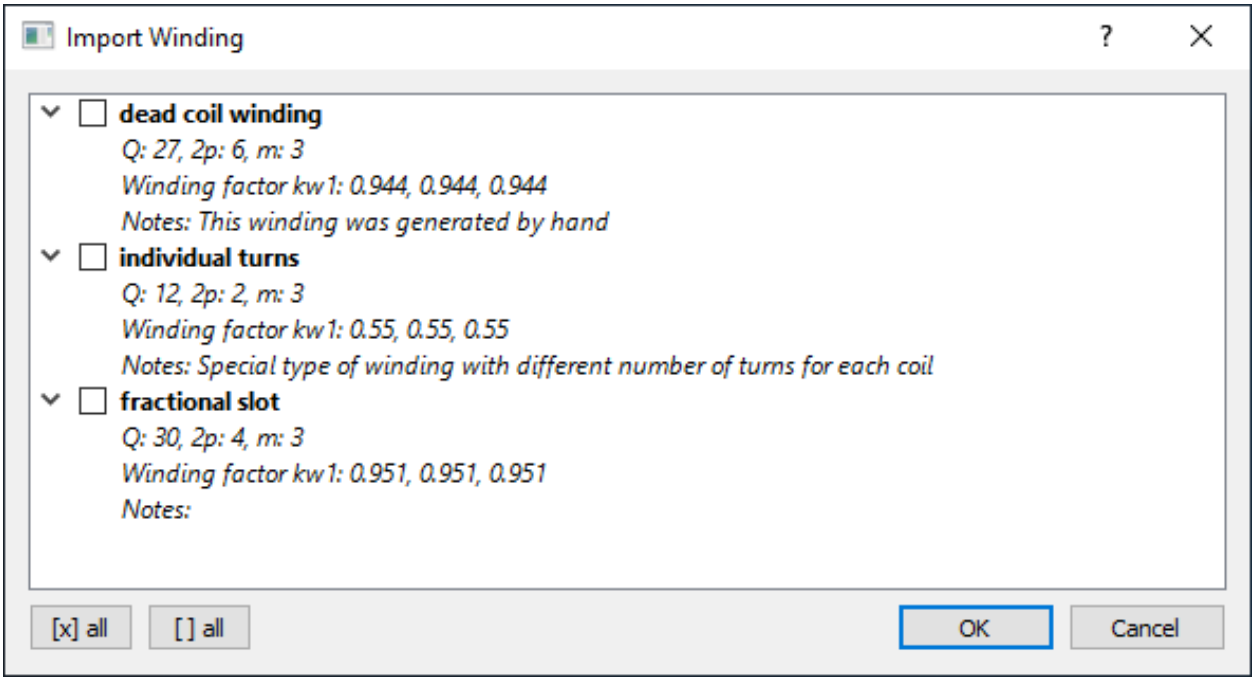


Fig. 15: Import winding from file

For import a window opens with the file dialog. Navigate to an existing *.wdg file. After that you get a list of all windings systems of the file (figure *Import winding from file*). Choose all windings you want to import into the workspace.

1.4 Using the api

1.4.1 Basic usage

Simple overlapping winding

In the theory section the following simple winding was shown.

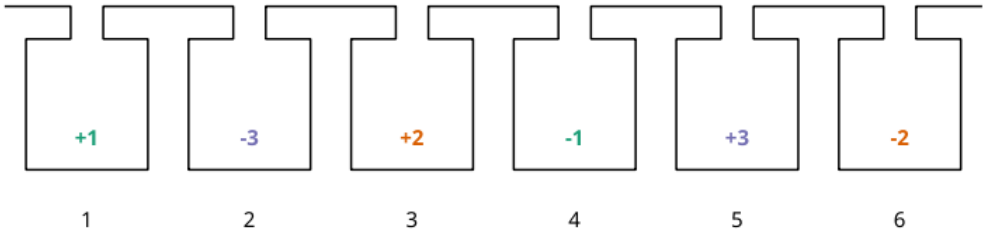


Fig. 16: Overlapping winding with 6 slots and 2 poles and 3 phases

Let's have a look how we can model this with swat-em. First of all we need to import swat-em. The relevant part is the datamodel() object. It includes all data and methods for the winding:

```
from swat_em import datamodel
```

The model has an built-in winding generator for almost every winding for rotating field motors such as permanent-magnet, synchronous or induction machines:

```
>>> wdg = datamodel()       # generate a datamodel for the winding
>>> Q = 6                    # number of slots
>>> P = 2                    # number of pole pairs
>>> wdg.genwdg(Q = Q, P = P, m = 3, layers = 1)
>>> print(wdg)              # print infos for the winding
WINDING DATAMODEL
=================

Title: Untitled
Number of slots:  6
Number of poles:  2
Number of phases: 3
Number of layers: 1
Winding step    : 3
Number of slots per pole per phase: 1
Fundamental winding factor: 1.0, 1.0, 1.0
```

Simple tooth-coil winding

In the same way we can create a tooth-coil winding

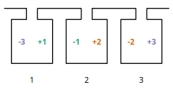


Fig. 17: Winding layout with 3 slots, 2 poles and 3 phases

We can set the winding step explicite with the keyword 'stepwidth'. Compared to the overlapping winding we need only 3 slots for the two poles. To get a coil around every tooth, we need two winding layers:

```
>>> wdg = datamodel()       # generate a datamodel for the winding
>>> Q = 3                    # number of slots
>>> P = 2                    # number of pole pairs
>>> w = 1                    # step width for the coil in slots

>>> # generate winding automatically
>>> wdg.genwdg(Q = Q, P = P, m = 3, layers = 2, w = w)
>>> print(wdg)              # print infos for the winding
WINDING DATAMODEL
=================

Title: Untitled
Number of slots:  3
Number of poles:  2
Number of phases: 3
Number of layers: 1
Winding step    : 1
Number of slots per pole per phase: 1/2
Fundamental winding factor: 0.866, 0.866, 0.866
```

A more complex winding

A more complex winding (overlapping full pitch winding with coil shortening)



Fig. 18: Winding layout with 12 slots, 2 poles and 3 phases

```
>>> wdg = datamodel()
>>> Q = 12
>>> P = 2
>>> w = 5     # without shortening w would be 6 for this winding
>>> wdg.genwdg(Q = Q, P = P, m = 3, layers = 2, w = w)
>>> print(wdg)
WINDING DATAMODEL
=================

Title: Untitled
Number of slots:  12
Number of poles:  2
Number of phases: 3
Number of layers: 2
Winding step    : 5
Number of slots per pole per phase: 2
Fundamental winding factor: 0.933, 0.933, 0.933
```

1.4.2 Results

Getting Results

After generating a winding, swat-em analyze it and provides the results:

```
>>> wdg = datamodel()
>>> wdg.genwdg(Q = 12, P = 2, m = 3, layers = 1)
>>> print('fundamental winding factor: ', wdg.get_fundamental_windingfactor())
fundamental winding factor:  [0.9659258262890683, 0.9659258262890683, 0.
↪9659258262890684]
>>> print('winding step: ', wdg.get_windingstep())
winding step:  6
```

Get the generated winding layout: For each phase there is a list of the 1st and the 2nd layer. In this example there is only 1 layer, so the second list is empty. An entry of the lists define the slot number in which is a coil-side of the phase is located. A negative number means, that the winding direction is reversed in the slot.

```
>>> print('winding layout:', wdg.get_phases())
winding layout: [[[1, 2, -7, -8], []], [[5, 6, -11, -12], []], [[-3, -4, 9, 10], []]]
```

The winding factor depends on the harmonic number. There are two possible interpretations for the harmonic number: The 'electrical' harmonic numbers the 'mechanical' ordinal numbers multiplyed with number of pole pairs 'p'. Use the 'mechanical' winding factor if you want du determine the possible number of poles your winding can drive and use the electrical winding factor if you know your number of pole pairs and if you want to analyze the harmonic content of the winding for example. Attention: The winding factor is calculated for each phase seperately.

```
>>> nu, kw = wdg.get_windingfactor_el()
>>> for k in range(len(nu)):
>>>     print(nu[k], kw[k])
1 [0.96592583 0.96592583 0.96592583]
3 [-0.70710678 -0.70710678 -0.70710678]
5 [-0.25881905 -0.25881905 -0.25881905]
7 [0.25881905 0.25881905 0.25881905]
9 [-0.70710678 -0.70710678 -0.70710678]
...
```

The datamodel() object stores the data in dictionaries. The user have direct access:

```
>>> print('Data for the machine: ', wdg.machinedata.keys())
Data for the machine:  dict_keys(['Q', 'p', 'm', 'phases', 'wstep', 'turns',
↪'phasenames'])
>>> # ... and all results:
>>> print('Data for the machine: ', wdg.results.keys())
Data for the machine:  dict_keys(['q', 'nu_el', 'Ei_el', 'kw_el', 'phaseangle_el',
↪'nu_mech', 'Ei_mech', 'kw_mech', 'phaseangle_mech', 'valid', 'error', 't', 'wdg_is_
↪symmetric', 'wdg_periodic', 'MMK', 'basic_char'])
```

For getting the results the get_* methods can be used:

```
>>> print('Is winding symmetric:', wdg.get_is_symmetric())
Is winding symmetric: True
>>> print('Fundamental winding factor:', wdg.get_fundamental_windingfactor())
Fundamental winding factor: [0.9659258262890683, 0.9659258262890683, 0.
↪9659258262890684]
>>> print('Number of turns in series:', wdg.get_num_series_turns())
Number of turns in series: 2.0
```

(continues on next page)

(continued from previous page)

```
>>> print('Excited radial force modes:', wdg.get_radial_force_modes())
Excited radial force modes: [2, 4, 6]
>>> print('Periodictiy:', wdg.get_periodicity_t())
Periodictiy: 1
>>> print('Possible parallel connections:', wdg.get_parallel_connections())
Possible parallel connections: [1, 2]
>>> print('Double linked leakage:', wdg.get_double_linked_leakage())
Double linked leakage: 0.02843683350047214
```

1.4.3 Plotting

Winding layout

SWAT-EM provides some possibilities for graphical representations. After creating a winding one would like to have a look on the layout, for example. This plot includes all coil sides of all phases in the slots:

```
>>> wdg = datamodel()
>>> wdg.genwdg(Q = 12, P = 2, m = 3, layers = 1)
>>> wdg.plot_layout('plot_layout.png')
```

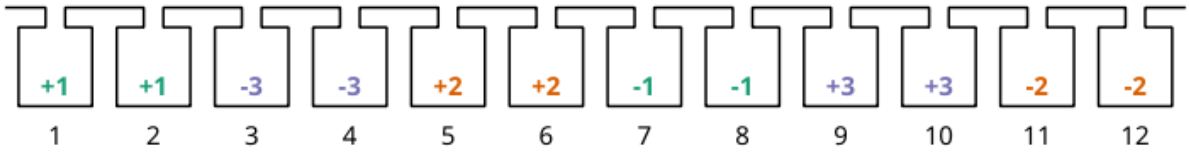


Fig. 19: Plot of the winding layout

Voltage phasors of the star of slot

SWAT-EM calculates the winding factor by the slot voltage phasors. The following is the corresponding visualization.

```
>>> wdg.plot_star('plot_star.png')
```

Winding factor

For the winding factor one have to decide between the mechanical or the electrical winding factor. Attention: For a 2-pole machine the electrical and mechanical winding factor is equal.

```
>>> wdg.plot_windingfactor('plot_wf.png', mechanical = False)
```

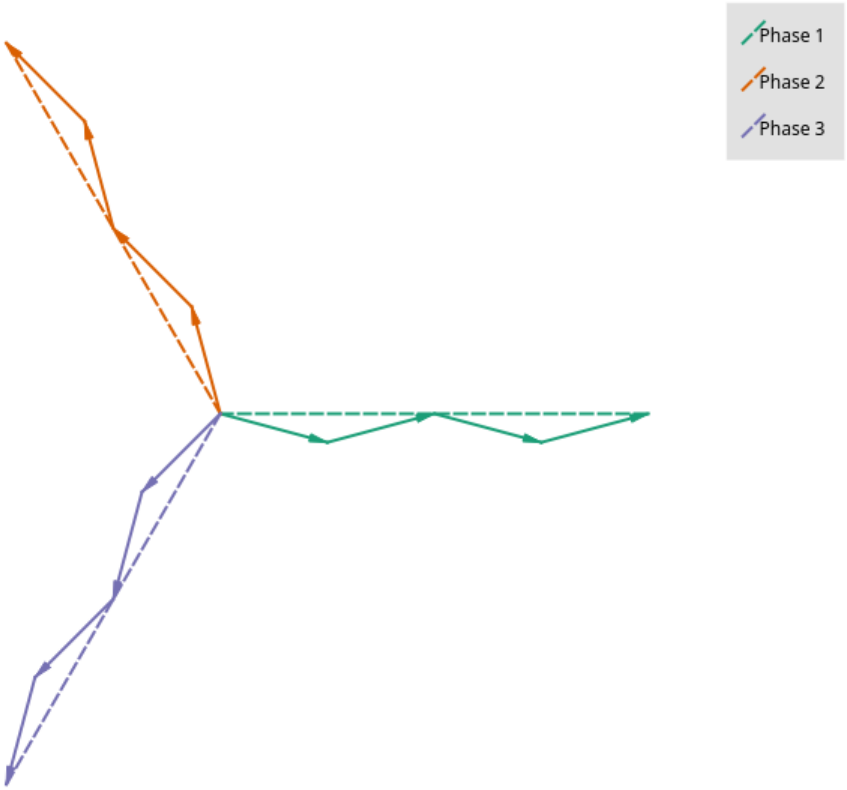


Fig. 20: Plot of the voltage phasors

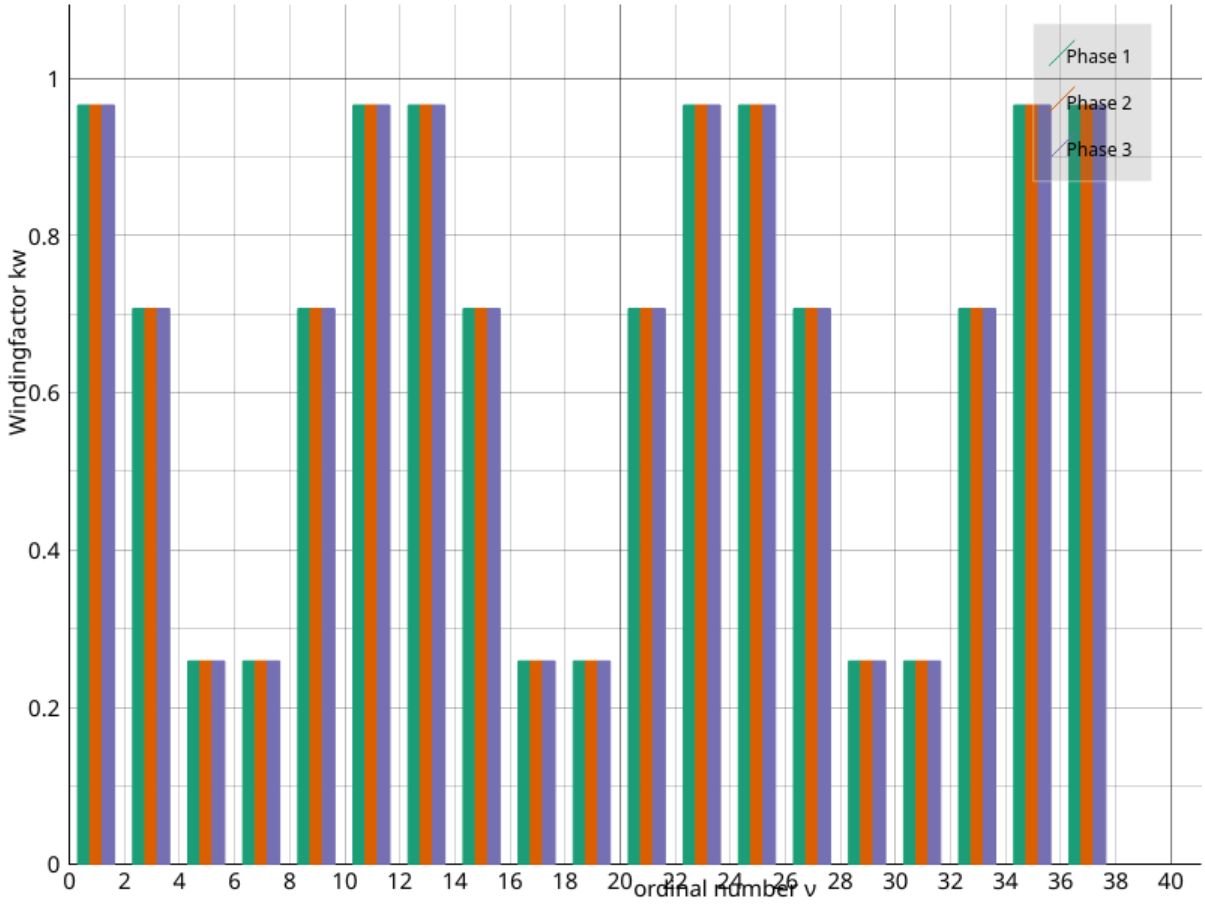


Fig. 21: Plot of the electrical winding factor

Magnetomotive force

The winding generates a current linkage in the slots. The integral of it leads to a magnetic field in the airgap, which is called the 'Magnetomotive force (MMF)'. It's a good indicator for the harmonic content of the winding. Also the resultion of the image can be definded:

```
>>> wdg.plot_MMK('plot_MMK.png', res = [800, 600], phase = 0)
```

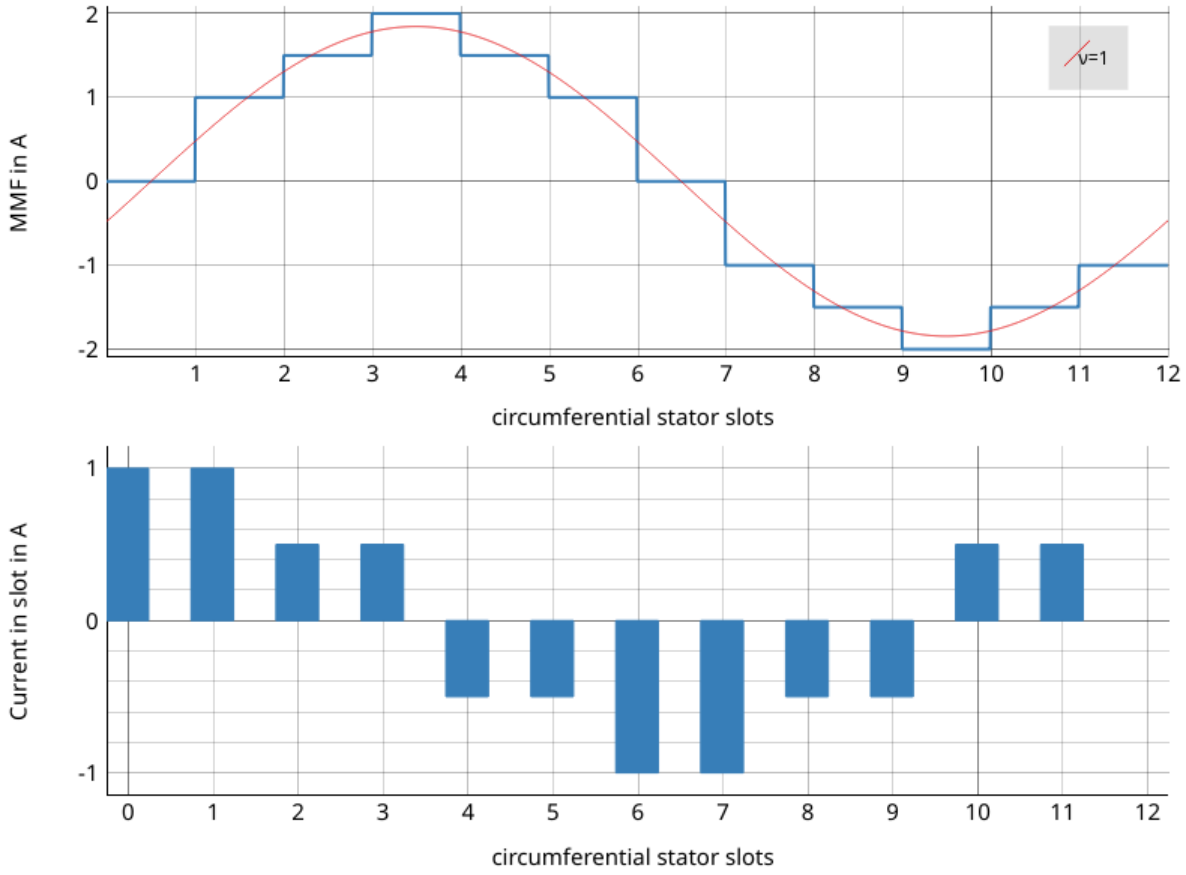


Fig. 22: Plot of the current linkage in the slots and the resulting Magnetomotove force

It also could be usefull to plot at different phase angles

```
>>> wdg.plot_MMK('plot_MMK_20deg.png', res = [800, 600], phase = 20)
```

1.4.4 File IO

Save/load a winding

After creating a winding we can save it as a *.wdg file This file can be used with the GUI for example. swat-em uses the "json" format for the *.wdg files.

```
>>> wdg = datamodel()
>>> wdg.genwdg(Q = 12, P = 2, m = 3, layers = 1)
>>> wdg.save_to_file('myfile.wdg')
```

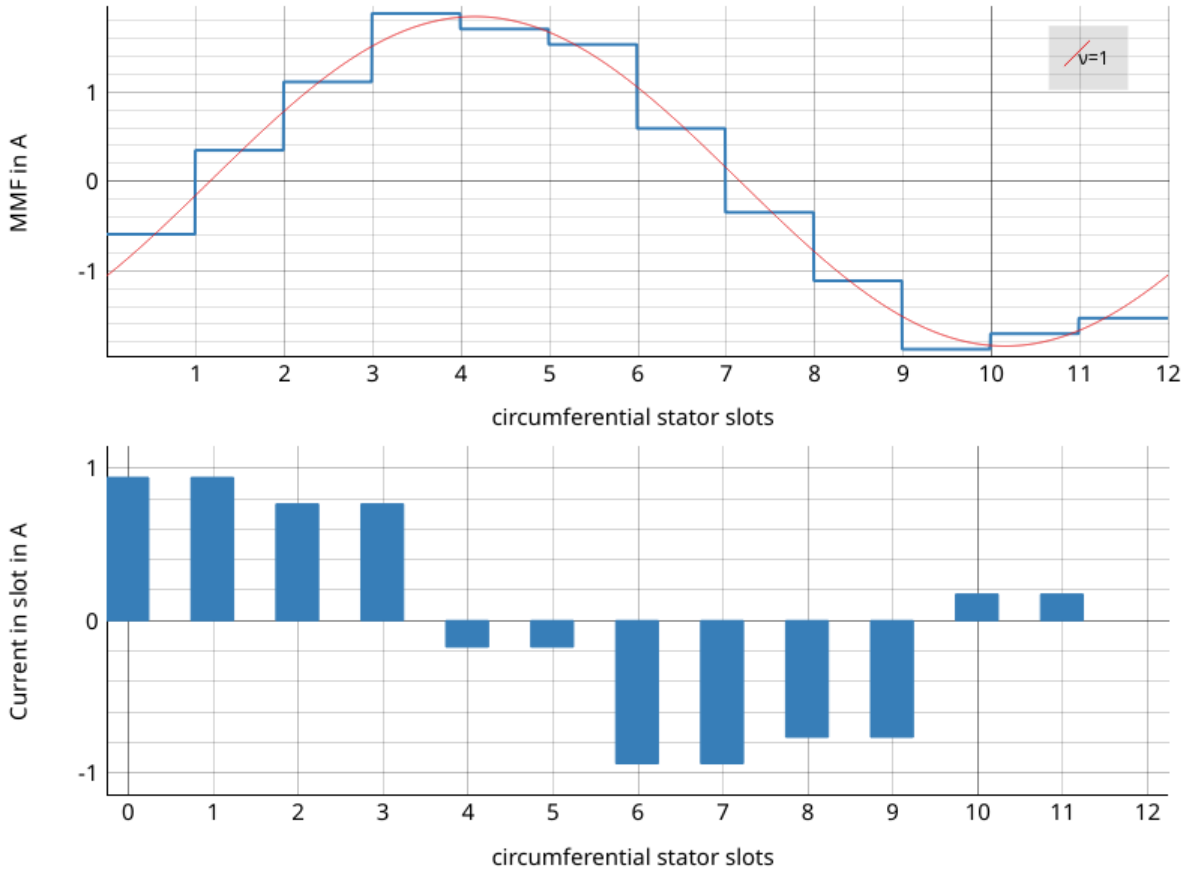


Fig. 23: Plot of the current linkage in the slots and the resulting Magnetomotove force with phaseangle = 20°.

We can also load an existing winding from file:

```
>>> wdg2 = datamodel()
```

Proof, that the data of the two objects is equal:

```
>>> print('same data?:', wdg.machinedata == wdg2.machinedata)
same data?: True
>>> print('same results?:', wdg.results == wdg2.results)
same results?: True
```

Export to Excel file

The data of an existing winding can exported to an Excel file (*.xlsx). Attention: The old *.xls format is not supported!

```
>>> wdg.export_xlsx('export.xlsx')
```

Text report

A summary of the winding can be exported as a text report:

```
>>> wdg.export_text_report('report.txt')
```

HTML report

Similar to the text report we can create a html report. This also includes the graphics.

```
>>> wdg.export_html_report('report.html')
```

1.5 API Reference

class swat_em.datamodel.**datamodel**

Provides a central place for all data. All analysis functions are connect with this class.

analyse_wdg()

Do a detailled analyses of the winding. This includes winding factors, detection of periodicity and symmetry, radial force modes and so on. Use the get_* functions for getting the results.

calc_num_basic_windings_t()

Calculates and returns the number of basic windings 't' for the actual winding layout

Returns **t** – Periodicity for the winding layout

Return type integer

export_html_report(*fname=None*)

Returns a winding report.

Parameters **fname** (*string*) – file name for html file. If not given a file is created in the temp dir of the file system (the file name ist returned by this function)

Returns **filnename** – The file name of the html-file which is stored in tmp directory

Return type string

export_text_report (*fname*)
Export winding report as a text file.

Parameters **fname** (*string*) – file name

export_xlsx (*fname*)
Export the results to Excel xlsx file.

Parameters **fname** (*string*) – file name

genwdg (*Q*, *P*, *m*, *layers*, *w=- 1*, *turns=1*, *empty_slots=0*)
Generates a winding layout and stores it in the datamodel

Parameters

- **Q** (*integer*) – number of slots
- **P** (*integer*) – number of poles
- **m** (*integer*) – number of phases
- **w** (*integer*) – winding step (1 for tooth coils)
- **layers** (*integer*) – number of coil sides per slot
- **turns** (*integer*) – number of turns per coil
- **empty_slots** (*integer*) –
- **the number of empty slots ("dead coil winding")** (*Defines*) –
- **-1** (*Choose number of empty slots automatically (the smallest*) – possible number is choosen)
- **0** (*No empty slots*) –
- **>0** (*Manual defined number of empty slots*) –

get_basic_characteristics ()
Returns the basic charactericits of the winding as dictionary and a html string

get_double_linked_leakage ()
Returns the coefficient of the double linkead leakage flux. This number is a measure of the harmonic content of the MMF in the airgap caused by the winding. As higher the number as higher the harmonics.

Returns **sigma_d** – coefficient of the double linkead leakage flux

Return type float

get_fundamental_windingfactor ()
Returns the fundamental winding factors for each phase

Returns **kw** – windings factors, (one entry for each phase)

Return type list

get_is_symmetric ()
Returns the symmetry of the winding

Returns **is_symmetric** – True if the winding is symmetric False if the winding is not symmetric

Return type Boolean

get_layers ()
Returns the definition of the winding layout alternative to the 'get_phases' function. For every layer (with the length of the number of slots) there is a sublist which contains the phase-number.

layers[0][0] contains the phase number for first layer and first slot layers[0][1] contains the phase number for first layer and second slot layers[0][Q-1] contains the phase number for first layer and last slot layers[1][0] contains the phase number for second layer and first slot

Returns

- **layers** (*numpy array*) – winding layout
- **slayers** (*numpy array*) – same as 'layers' but as string
- **layers_col** (*numpy array*) – phase colors

get_lcmQP()
Returns the lowest common multiple of the slot number Q and the number of Poles. For permanent-magnet machines this value represents the first ordinal number for the cogging torque.

Returns **lcmQP** – Lowest common multiple lcm(Q, P)

Return type integer

get_notes()
Get notes for the winding

Returns **notes** – Some notes

Return type string

get_num_layers()
Returns the number of layers of the actual winding layout

get_num_phases()
Returns the number of phases m

Returns **m** – number of phases

Return type integer

get_num_polepairs()
Returns the number of pole-pairs p

Returns **p** – number of pole-pairs

Return type integer

get_num_series_turns()
Returns the number of turns in series per phase. If the number of coil sides per phase or number of turns per phase is not identically than a mean value of turns of all phases is returned.

Returns **w** – number of turns in series per phase

Return type number

get_num_slots()
Returns the number of slots Q

Returns **Q** – number of slots

Return type integer

get_parallel_connections()
Returns all possible parallel connections of the winding.

Returns **a** – Number of possible parallel connections

Return type list

get_periodicity_t()
Returns the periodicity of the winding. In most cases t = gcd(Q,p). For user defined windings this may be different.

Returns **t** – Number of periodic base windings

Return type integer

get_phasenames()
Returns the names of the phases as a series of characters 'A', 'B', 'C', ... with length of the number of phases

Returns **phasenames** – names of the phases

Return type list

Examples

if there are m = 3 phases: >>> data.get_phasenames() ['A', 'B', 'C']

get_phases()
Returns the definition of the winding layout. For every phase there is a sublist which contains the slot number which are allocated to the phase. phases

phases[0] contains the slot numbers for the first phase phases[1] contains the slot numbers for the second phase phases[m-1] contains the slot numbers for the last phase

Returns **phases** – winding layout

Return type list of lists

get_q()
Returns the number of slots per pole per phase.

Returns **layers** – number of slots per pole per phase

Return type Fraction

get_radial_force_modes(*num_modes=None*)
Returns the radial force modes caused by the winding. The results includes also the modes with a multiple of the phase-number (which aren't there if the machine is star-connected).

Parameters **num_modes** (*integer*) – Max. number of modes. If not given the default value from the config file is used

Returns **MMK** – radial force modes

Return type list

get_text_report()
Returns a winding report.

Returns **report** – Report

Return type string

get_title()
Get the title of the winding

Returns **title** – title

Return type string

get_turns()
Returns the number of turns. If all coil sides has the same number of turns, the return value is a scalar. If every coil side has an individual number of turns, the return value consists of lists with the same shape as the winding layout (phases)

Returns **turns** – number of turns

Return type integer, float or list of lists

get_windingfactor_el()
Returns the windings factors with respect to the electrical ordinal numbers

Returns

- **nu** (*numpy array*) – ordinal numbers
- **kw** (*2D numpy array*) – windings factors, (one column for each phase)

get_windingfactor_mech()
Returns the windings factors with respect to the electrical ordinal numbers

Returns

- **nu** (*numpy array*) – ordinal numbers
- **kw** (*2D numpy array*) – windings factors, (one column for each phase)

get_windingstep()
Returns the winding step

Returns **w** – winding step

Return type integer

load_from_file(*fname*, *idx_in_file=0*)
Load data from file.

Parameters **fname** (*string*) – file name

plot_MMK(*filename*, *res=None*, *phase=0*, *show=False*)
Generates a figure of the winding layout

Parameters

- **filename** (*string*) – file-name with extension to save the figure
- **res** (*list*) – Resolution for the figure in pixes for x and y direction example: res = [800, 600]
- **phase** (*float*) – phase angle for the current system in electical degree in the range 0..360°
- **show** (*Bool*) – If true the window pops up for interactive usage

plot_layout(*filename*, *res=None*, *show=False*)
Generates a figure of the winding layout

Parameters

- **filename** (*string*) – file-name with extension to save the figure
- **res** (*list*) – Resolution for the figure in pixes for x and y direction example: res = [800, 600]
- **show** (*Bool*) – If true the window pops up for interactive usage

plot_overhang(*filename*, *res=None*, *show=False*, *optimize_overhang=False*)
Generates a figure of the winding layout

Parameters

- **filename** (*string*) – file-name with extension to save the figure
- **res** (*list*) – Resolution for the figure in pixes for x and y direction example: res = [800, 600]
- **show** (*Bool*) – If true the window pops up for interactive usage

plot_star(*filename*, *res=None*, *ForceX=True*, *show=False*)
Generates a figure of the star voltage phasors

Parameters

- **filename** (*string*) – file-name with extension to save the figure
- **res** (*list*) – Resolution for the figure in pixes for x and y direction example: res = [800, 600]
- **ForceX** (*Bool*) – If true the voltage phasors are rotated in such way, that the resulting phasor of the first phase matches the x-axis
- **show** (*Bool*) – If true the window pops up for interactive usage

plot_windingfactor(*filename*, *res=None*, *mechanical=True*, *show=False*)
Generates a figure of the winding layout

Parameters

- **filename** (*string*) – file-name with extension to save the figure
- **m** (*list*) – Resolution for the figure in pixes for x and y direction example: res = [800, 600]
- **mechanical** (*Bool*) – If true the winding factor is plotted with respect to the mechanical ordinal numbers. If false the electrical ordinal numbers are used
- **show** (*Bool*) – If true the window pops up for interactive usage

reset_data()
resets all data of the datamodel

reset_results()
Remove all existing results

save_to_file(*fname*)
Saves the data to file.

Parameters **fname** (*string*) – file name

set_machinedata(*Q=None*, *p=None*, *m=None*, *Qes=None*)
setting the machine data

Parameters

- **Q** (*integer*) – number of slots
- **p** (*integer*) – number of pole pairs
- **m** (*integer*) – number of phases

set_notes(*notes*)
Set additional notes for the winding

Parameters **notes** (*string*) – Some notes

set_num_phases(*m*)
Sets the number of phases m

Parameters **m** (*integer*) – number of phases

set_num_polepairs(*p*)
Sets the number of pole pairs p

Parameters **p** (*integer*) – number of pole pairs

set_num_slots(*Q*)
Sets the number of slots Q

Parameters **Q** (*integer*) – number of slots

set_phases(*S*, *turns=1*, *wstep=None*)
setting the winding layout

Parameters

- **S** (*list of lists*) – winding layout for every phase, for example: S = [[1,-2], [3,-4], [5,-6]]. This means there are 3 phases with phase 1 in in slot 1 and in slot 2 with negativ winding direction. For double layer windings there must be additional lists: S = [[[1, -4], [-3, 6]], [[3, -6], [-5, 2]], [[-2, 5], [4, -1]]] Hint: [[[first layer], [second layer]], ...]
- **wstep** (*integer*) – winding step (slots as unit)

set_title(*title*)
Set the title of the winding

Parameters **title** (*string*) – title

set_turns(*turns*)
Sets the number of turns. If all coil sides has the same number of turns, the parameter should be an scalar. If every coil side has an individual number of turns, the parameter value have to consist of lists with the same shape as the winding layout (phases)

Parameters **turns** (*integer, float or list of lists*) – number of turns

set_windingstep(*w*)
Sets the winding step w

Parameters **w** (*integer*) – winding step

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Got07] R. Gottkehaskamp. *Optimal gefertigt - Systematischer Entwurf von dreisträngigen Zahnspulenwicklungen bürstenloser Motoren*. Antriebstechnik 10/2007, S. 30-35, 2007.

[Obe65] Kurt Oberretl. Die oberfeldtheorie des käfigmotors unter berücksichtigung der durch die ankerrückwirkung verursachten statoroberströme und der parallelen wicklungszweige. *Archiv für Elektrotechnik*, 49:343–364, 10 1965. doi:10.1007/BF01587916.

[BianchiDaiPre06] N. Bianchi and M. Dai Pre. Use of the star of slots in designing fractional-slot single-layer synchronous motors. *IEE Proceedings - Electric Power Applications*, 153(3):459–466, May 2006. doi:10.1049/ip-epa:20050284.

[BianchiDaiPre06] N. Bianchi and M. Dai Pre. Use of the star of slots in designing fractional-slot single-layer synchronous motors. *IEE Proceedings - Electric Power Applications*, 153(3):459–466, May 2006. doi:10.1049/ip-epa:20050284.

PYTHON MODULE INDEX

S

swat_em, 28

INDEX

A

analyse_wdg() (*swat_em.datamodel.datamodel method*), 28

C

calc_num_basic_windings_t() (*swat_em.datamodel.datamodel method*), 28

D

datamodel (*class in swat_em.datamodel*), 28

E

export_html_report() (*swat_em.datamodel.datamodel method*), 28
export_text_report() (*swat_em.datamodel.datamodel method*), 28
export_xlsx() (*swat_em.datamodel.datamodel method*), 29

G

genwdg() (*swat_em.datamodel.datamodel method*), 29
get_basic_characteristics() (*swat_em.datamodel.datamodel method*), 29
get_double_linked_leakage() (*swat_em.datamodel.datamodel method*), 29
get_fundamental_windingfactor() (*swat_em.datamodel.datamodel method*), 29
get_is_symmetric() (*swat_em.datamodel.datamodel method*), 29
get_layers() (*swat_em.datamodel.datamodel method*), 29
get_lcmQP() (*swat_em.datamodel.datamodel method*), 30
get_notes() (*swat_em.datamodel.datamodel method*), 30
get_num_layers() (*swat_em.datamodel.datamodel method*), 30
get_num_phases() (*swat_em.datamodel.datamodel method*), 30
get_num_polepairs() (*swat_em.datamodel.datamodel method*), 30
get_num_series_turns() (*swat_em.datamodel.datamodel method*), 30
get_num_slots() (*swat_em.datamodel.datamodel method*), 30
get_parallel_connections() (*swat_em.datamodel.datamodel method*), 30
get_periodicity_t() (*swat_em.datamodel.datamodel method*), 30
get_phasenames() (*swat_em.datamodel.datamodel method*), 31
get_phases() (*swat_em.datamodel.datamodel method*), 31
get_q() (*swat_em.datamodel.datamodel method*), 31
get_radial_force_modes() (*swat_em.datamodel.datamodel method*), 31
get_text_report() (*swat_em.datamodel.datamodel method*), 31
get_title() (*swat_em.datamodel.datamodel method*), 31
get_turns() (*swat_em.datamodel.datamodel method*), 31
get_windingfactor_el() (*swat_em.datamodel.datamodel method*), 32
get_windingfactor_mech() (*swat_em.datamodel.datamodel method*), 32
get_windingstep() (*swat_em.datamodel.datamodel method*), 32

L

load_from_file() (*swat_em.datamodel.datamodel method*), 32

M

module
    swat_em, 28

P

plot_layout() (*swat_em.datamodel.datamodel method*), 32
plot_MMK() (*swat_em.datamodel.datamodel method*), 32
plot_overhang() (*swat_em.datamodel.datamodel method*), 32
plot_star() (*swat_em.datamodel.datamodel method*), 33
plot_windingfactor() (*swat_em.datamodel.datamodel method*), 33

R

reset_data() (*swat_em.datamodel.datamodel method*), 33
reset_results() (*swat_em.datamodel.datamodel method*), 33

S

save_to_file() (*swat_em.datamodel.datamodel method*), 33
set_machinedata() (*swat_em.datamodel.datamodel method*), 33
set_notes() (*swat_em.datamodel.datamodel method*), 33
set_num_phases() (*swat_em.datamodel.datamodel method*), 34
set_num_polepairs() (*swat_em.datamodel.datamodel method*), 34
set_num_slots() (*swat_em.datamodel.datamodel method*), 34
set_phases() (*swat_em.datamodel.datamodel method*), 34
set_title() (*swat_em.datamodel.datamodel method*), 34
set_turns() (*swat_em.datamodel.datamodel method*), 34
set_windingstep() (*swat_em.datamodel.datamodel method*), 34
swat_em
    module, 28