Script-based Control of GeckoMAGNETICS from MATLAB^{TM} or Another Application

GeckoMAGNETICS is a tool that enables fast, accurate and user-friendly modelling and pareto-optimal design of inductive power components.

The key modelling features of Gecko-MAGNETICS are:

1) An easy-to-use user interface that guides the user through the design process. Based on the user-specified desired inductance, application, and cooling conditions GeckoMAGNETICS finds an optimum design with respect to losses or volume.

2) Sophisticated and accurate models for loss, thermal and inductance calculation, including:

- A very accurate inductance cal-culation based on a novel air gap reluctance calculation approach.
- Different winding loss effects, such as Skin effect, Proximity effects (including the influence of an air-gap fringing field).
- Many core loss effects such as e.g. a DC premagnetization, re laxation effects, etc. Neglecting for instance a DC premagneti-zation, may lead to a loss underestimation by a factor of more than two.

3) A link to the circuit simulator GeckoCIRCUITS, enabling multi-domain modelling and extraction of waveforms from circuit simulations.

4) A material and core database (GeckoDB), which is a part of the GeckoMAGNETICS tool, for accurate core loss modelling.

5) The ability to use GeckoMAGNETICS in combination with another application used for modelling, simulation, and/or calculation, such as MATLAB^{TM}, via the GeckoMAGNETICS scripting feature.

This tutorial shows how to use the *GeckoMAGNETICS Script* feature in order to have GeckoMAGNETICS exchange data and be controlled from another application, that is, how the scripting feature can be used to automate and process a large number of GeckoMAGNETICS simulations.

GeckoMAGNETICS is a powerful simulation tool that allows for detailed modelling of inductors and transformers. Its Design Mode also allows the user to test out many different possible inductor or transformer designs for a given application and operating point and to select the best design for his needs.

In some cases however, more than that may be needed – it might be desirable to see how a given design performs with different waveforms (operating points) – e.g. at different loads or different switching frequencies in a switching converter. Advanced post-processing of the simulation results may be needed.

In other cases, it would be advantageous to combine the simulation and modeling capabilities of GeckoMAGNETICS with another application (e.g. MATLAB/Octave) which is modelling, simulating or evaluating other aspects of the overall system for which the magnetic components are being designed – in that case, there must exist a method of feeding input data into GeckoMAGNETICS and extracting in an automatic fashion the output data it produces.

For exactly such applications, the scripting-based control feature of GeckoMAGNETICS – *GeckoMAGNETICS Script* – has been developed. *GeckoMAGNETICS Script* allows you to use GeckoMAGNETICS as more than just a standard graphical user interface-controlled desktop application. With it, you can turn GeckoMAGNETICS into a component of your larger simulation, modeling or evaluation environment, as well as automate the execution of a large number of simulations under varying input parameters.

GeckoMAGNETICS provides the user with a set of easy-to-understand and straightforward to use scripting functions. Through the use of these scripting functions, all aspects of GeckoMAGNETICS modelling and simulation my be controlled and executed.

Be aware that *GeckoMAGNETICS Script* works with **Model Mode only**, as by using it, the user is essential creating his/her own "Design Mode", flexible and tailored to his/her specific needs and requirements.

Overview: How to use *GeckoMAGNETICS Script* with your custom Java application or MATLAB^{TM}/Octave

In order to have your **scripting client** application (whether a custom Java application which you wrote or MATLAB/Octave) use GeckoMAGNETICS Script, you must first point your application to the correct GeckoMAGNETICS **.jar* files on your system. In your GeckoMAGNETICS installation directory, there is a subdirectory named:

geckomagnetics/modules

which contains the **.jar* files. The file containing the scripting functions is called

gecko-geckomagnetics-script.jar

If using scripting from a Java application, you must add the above file to the application as a library. If using MATLAB or Octave for scripting, this file must be added to these programs' Java paths.

If you plan to use scripting to control a GeckoMAGNETICS instance running on the same machine as your custom application or MATLAB/Octave, you may simply point your scripting client to the above **.jar* file in the GeckoMAGNETICS installation directory. However, **if you wish to use scripting remotely, over a network, with GeckoMAGNETICS** and scripting client running on different machines, you must *copy* the above *.jar file to the machine on which the scripting client is located and appropriately point the client to it, as explained above. Be aware that some functionality is available only when both GeckoMAGNETICS and the scripting client are running on the same machine.

If you are using a custom Java application as a scripting client, once you have added the **.jar* file to your application's libraries, in your code you need the following import:

```
import gecko.geckomagnetics.script.*;
```

After doing this, you are now free to use the GeckoMAGNETICS Script functions in your Java program using standard Java syntax as you would with any other Java package.

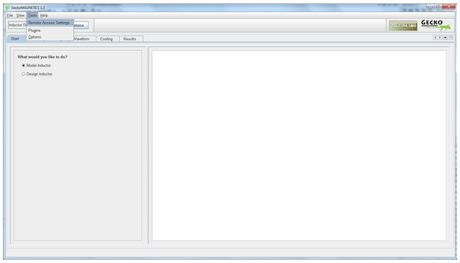
The remainder of this tutorial assumes the use of MATLAB or Octave, running, unless otherwise clearly stated, on the same machine as the GeckoMAGNETICS instance being controlled via scripting. All examples are given using MATLAB syntax (with Octave syntax, where different, being given in a footnote).

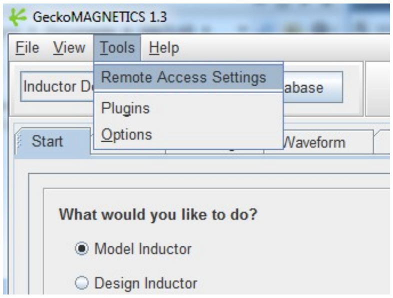
If you do not posses a MATLAB license, do note that you may download and install GNU Octave for free (the installation of a graphical front-end is also recommended). If MATLAB/Octave use is not possible for you, all examples given in this tutorial may also be executed inside a custom Java application (you may need to write additional code to properly display the output). Note that MATLAB syntax is slightly different from Java syntax (for example, MATLAB uses single quotes for strings, while Java uses double quotes, etc.).

Setting up GeckoMAGNETICS for Remote Access

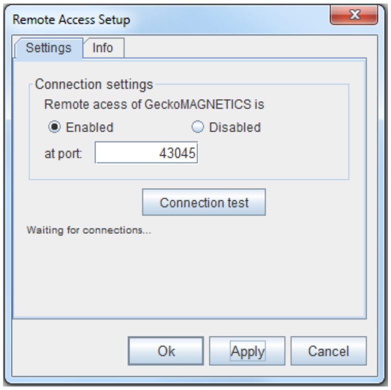
We will start by enabling a running instance of GeckoMAGNETICS for scripting control, that is, remote access. The basic idea behind GeckoMAGNETICS Script is that GeckoMAGNETICS can be set up as a server, listening on a network port for incomming client connections. By using the scripting functions, the client sends GeckoMAGNETICS messages with instructions via this network port, and GeckoMAGNETICS executes the corresponding action (e.g. loads a design file, simulates a component, returns the simulation results, etc.). This communication occurs over a network port regardless of whether the client and GeckoMAGNETICS are running on the same or on different machines – **therefore it is important that your machine have open ports and that programs can access them.**

To begin, start GeckoMAGNETICS. Once it is fully loaded, on the top menu bar, click "Tools" and then in the drop-down menu, click "Remote Access Settings".





This will open the remote "Remote Access Setup" window. By default, remote access is disabled. Click on the "Enabled" button. Now you will be able to enter a port number for scripting access. The default port number is 43045, which is usually an unused free port on most systems. You may use any such port. Now click "Apply". If the port has been successfully opened, a message will appear saying "Waiting for connections...".



Remote acess of GeckoMAGNETICS is [✓ Enabled] Disabled

at port: 43045

Waiting for connections...

If this message appears, GeckoMAGNETICS is ready for scripting access. If you see an error message instead, try a different port number, and then click "Apply" again. If you are unable to find a port which you can successfully open, please contact your IT/Network Systems Administrator for advice.

Connecting to GeckoMAGNETICS from MATLAB/Octave

Now open MATLAB or Octave. Once it loads, we must add the previously mentioned **.jar* file to its Java path. For the purpose of this tutorial, we assume that you are running on a Windows machine with GeckoMAGNETICS installed at C:\GeckoMAGNETICS – if this is not the case, adjust the examples given below accordingly. We do this by running the MATLAB command[1]:

```
javaaddpath('C:\GeckoMAGNETICS\geck
omagnetics\modules\gecko-
geckomagnetics-script.jar')
```

Another option would be to permanently add the path to the scripting **.jar* file to MATLAB by editing its *classpath.txt* file.[2] For MATLAB Release 2012a, this file is located in the *toolbox/local* subdirectory of the MATLAB installation directory. For other releases, it might be located in a slightly different location. Open this file, and add to its end the above path (without quotation marks). Save the file and restart MATLAB.

1 In Octave, the corresponding command would be `javaaddpath gecko-geckomagnetics-script.jar` under the assumption that the *.jar is located in your home directory.

2 Note that for some MATLAB installations, depending on your user access priviliges, using the `javaaddpath` command may not work. In this case you **must** edit classpath.txt.

Also, for the purposes of this tutorial, we need to point MATLAB to the location of the GeckoCIRCUITS **.jar* file. If you do not have a distribution of GeckoCIRCUITS separate from your GeckoMAGNETICS distribution, you can find the GeckoCIRCUITS.jar file in the *geckomagnetics/modules/ext* sub-directory of your GeckoMAGNETICS installation directory. Assuming the same installation directory as above, type into the MATLAB console:

```
javaaddpath('C:\GeckoMAGNETICS\geck
omagnetics\modules\ext\GeckoCIRCUIT
S.jar')
```

This path can also be added to MATLAB's *classpath.txt* file. Adding the path to the GeckoCIRCUITS **.jar* is necessary, and must be done at the beginning of your MATLAB session, if you want to use GeckoMAGNETICS Script together with GeckoCIRCUITS' GeckoSCRIPT, as we will do later in this tutorial.

Once you have added the **.jar* file to the Java path, you must then import it as a package by running the command[3]:

```
import('gecko.geckomagnetics.geckos
cript.*')
```

The scripting functions are now ready to use! First we must create a Java object through which we will execute the scripting functions. To do this, run the following MATLAB command[4]:

3 As of the writing of this tutorial, the `import` function has not yet been implemented in Octave, However, this is not an obstacle.

4 In Octave, the corresponding command would be

```
geckom =
javaObjectEDT('gecko.geckomagnetics
.script.GeckoMagRemoteClient')
```

and you should get an output in the MATLAB console similar to this:

```
geckom =

gecko.geckomagnetics.script.GeckoMa
gRemoteClient@6d560d8b
```

In this case, "geckom" is a name we chose for the object. You may use any other name. Now, to finally connect to GeckoMAGNETICS and start using *GeckoMAGNETICS Script*, type in the MATLAB console:

```
geckom.connect('localhost',43045)
```

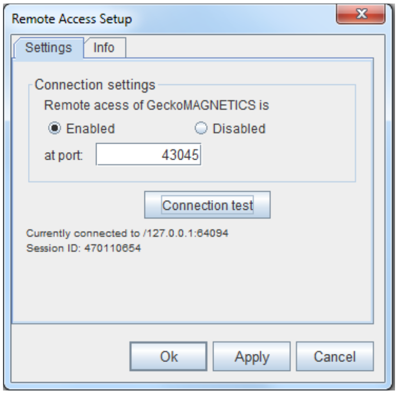
If you are using a port number different from the default, replace it in the above command. If the connection has been successful, MATLAB will display the following:

```
ans =

     1
```

Now, go back to GeckoMAGNETICS. In the Remote Acess Setup window, click "Connection Test". You will see a message displayed that starts with "Currently connected to..." with the IP number of the client (which should be 127.0.0.1 in this example, as both MATLAB and GeckoMAGNETICS are running on the same machine) and a session ID. Note that only one connection is allowed at a time. Click "OK" to close this window.

```
geckom =
javaObject('gecko.geckomagnetics.script.Ge
ckoMagRemoteClient')
```



Connection settings
Remote acess of GeckoMAGNETICS is
[✓ Enabled] Disabled
at port: 43045

Currently connected to /127.0.0.1:64094
Session ID: 470110654

Important Note: At this point, when attempting to establish a connection, your operating system may prompt you to allow network access for MATLAB/Octave and/or for GeckoMAGNETICS (or the Java platform). For example, under Windows, you might see a Windows Firewall pop-up window. You **must** click "allow access" or similar whenever prompted. If you are unable to do this, please contact you IT/Network Systems Administrator.

To disconnect MATLAB from GeckoMAGNETICS, call

```
geckom.disconnect()
```

Be sure to **always** disconnect from GeckoMAGNETICS when you are done with controlling it from your client application, in order to avoid errors in your code. If you have disconnected, reconnect as shown before to continue.

Basic Principle of Modelling and Simulating with *GeckoMAGNETICS Script*

As mentioned earlier, *GeckoMAGNETICS Script* works only with Model Mode in GeckoMAGNETICS. Additionally, **no** scripting functions are provided for directly setting the attributes of a modelled component (e.g. core, number of turns, air gap), or a standard waveform (e.g. peak-to-peak flux density, DC bias) directly. Instead, this is accomplished with the scripting functions **only by loading** GeckoMAGNETICS design files (**.gmd*) or waveform files (**.gmw).*

Such files may be created manually by the user, by defining a magnetic component or waveform manually via the GUI, and then clicking "File → Save Geometry" (or "File → Save Waveform") on the top menu bar. Also, these files may be edited and created through to use of the scripting functions. For this, knowledge of the structure of these files is required. This however is not difficult, as these are all human-readable plain-text files in easy-to-understand XML format. We will begin by using predefined **.gmd* and **.gmw* files supplied by this tutorial, and will explain how to modify and create new files using the scripting functions later on.

A list of all available *GeckoMAGNETICS Script* functions, with explanations, is given at the end of this document.

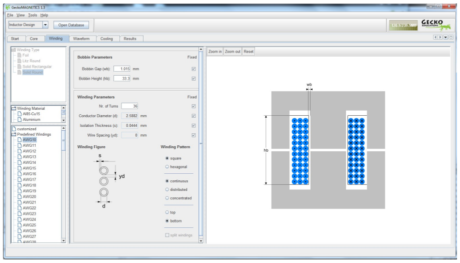
Loading a Design and Waveform and Running a Simulation

A design and a waveform file have been provided with this tutorial. In the following we assume the use of a Windows machine, and that the files have been placed in a directory called C:\tutorial. If this is not the case on your computer, please adjust the file paths in the commands below accordingly.

To load the supplied design file from MATLAB, type

```
geckom.loadDesignFile('C:/tutorial/
TutorialDesign1.gmd')
```

If you now go to GeckoMAGNETICS, you will see that the GUI has switched to the Winding tab, and that an inductor design consisting of an E-Core with 36 turns of AWG10 wire has been loaded.



Bobbin Parameters — Fixed
Bobbin Gap (wb): 1.015 mm ✓
Bobbin Height (hb): 33.3 mm ✓

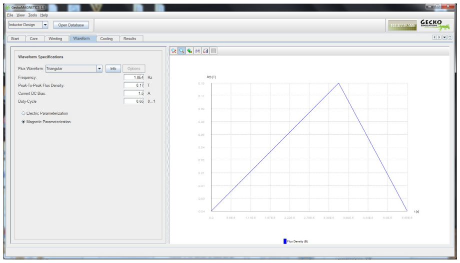
Winding Parameters — Fixed
Nr. of Turns: 36 ✓
Conductor Diameter (d): 2.5882 mm ✓
Isolation Thickness (s): 0.0444 mm ✓
Wire Spacing (yd): 0 mm ✓

Winding Pattern
[✓ square] hexagonal
[✓ continuous] distributed concentrated
top [✓ bottom]
split windings

Now, load the waveform file supplied with this tutorial. To accomplish this, type into the MATLAB console

```
geckom.loadWaveformFile('C:/tutoria
l/TutorialWaveform1.gmw')
```

Going back to GeckoMAGNETICS, you can see that now it has switched to the waveform tab, with a triangular waveform with a peak to peak flux density of 0.17 T and a DC bias of 1.5 A has been loaded.



Waveform Specifications

Flux Waveform: Triangular

Frequency: 1.8E4 Hz

Peak-To-Peak Flux Density: 0.17 T

Current DC Bias: 1.5 A

Duty-Cycle 0.65 0...1

( ) Electric Parameterization

(•) Magnetic Parameterization

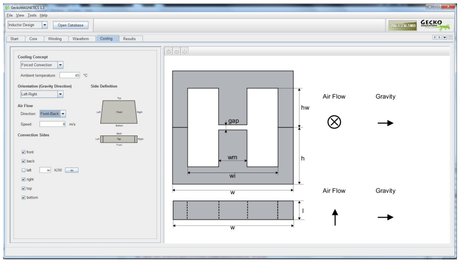
Next, we want to set the thermal properties and cooling method of the inductor which we have just loaded. For this, the `setCooling(...)` scripting function is provided. The first argument of the function tells GeckoMAGNETICS whether to perform any thermal modelling at all – so in this case, it should be `true`. The second argument defines the type of cooling – set it to `false` for natural convection, or `true` for forced convection. For this example, we will use forced convection. The third argument is the orientation of the core, i.e. the direction of gravity relative to the core shape. This argument is passed as a string, which must be exactly the same as the core orientation name in the "Orientation (Gravity Direction)" drop-down box of the GeckoMAGNETICS Cooling tab. In this case, we will define the orientation to be "Left-Right". The next argument is the ambient temperature, in this case 45°C. The last set of arguments are the sides of the core which are NOT exposed to any air flow. Again, the names must be given exactly as they are called in the GeckoMAGNETICS cooling tab. In this case, the left side of our inductor are not exposed to air flow. Therefore to set all these properties, type into the MATLAB command prompt

```
geckom.setCooling(true,true,'Left-
Right',45,'left')
```
[5]

Since forced convection has been selected as the cooling method, we must now additionally define the air speed and direction of air flow. Again, the direction is defined as a string that must be identical to the direction names in the "Direction" drop-down menu in the Cooling tab. To set an air flow direction of "Front-Back" and a speed of 8 m/s, type

```
geckom.setForcedCooling('Front-
Back',8)
```

In GeckoMAGNETICS, you will see now that the Cooling tab is active, with all of the above options and parameters set.



Cooling Concept

Forced Convection

Ambient temperature: 45 °C

Orientation (Gravity Direction)

Left-Right

Air Flow

Direction: Front-Back

Speed: 8 m/s

Convection Sides

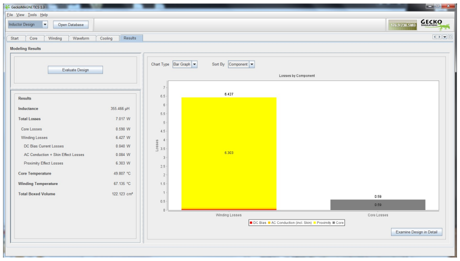
- [✓] front
- [✓] back
- [ ] left ∞ K/W ∞
- [✓] right
- [✓] top
- [✓] bottom

Now the component can be simulated and losses calculated. To do this, type

```
geckom.evaluateComponent()
```

5 To see how to set multiple sides of the component to be not exposed to convection, see the description of this function in the last section of this tutorial.

After this command has executed, you will see that GeckoMAGNETICS has switched to the Results tab with simulation results shown.



Modeling Results

Results	
Inductance	355.466 µH
Total Losses	7.017 W
Core Losses	0.590 W
Winding Losses	6.427 W
DC Bias Current Losses	0.040 W
AC Conduction + Skin Effect Losses	0.084 W
Proximity Effect Losses	6.303 W
Core Temperature	49.807 °C
Winding Temperature	67.135 °C
Total Boxed Volume	122.123 cm³

Now that the design has been simulated, we can extract the results back into MATLAB. For example, to get the core losses, you can type

```
geckom.getCoreLosses()
```

and in MATLAB, the following will be displayed:

```
ans =

    0.5902
```

which is the same amount of core losses in Watts as displayed in the GeckoMAGNETICS Results tab. All loss components, temperatures, boxed volume, and inductance can be retrieved using the corresponding scripting functions (see the list at the end of this tutorial). All the results can be retrieved also at once, by typing

```
geckom.getResults()
```

which in this case displays

```
ans =

    0.0004
    0.5902
    0.0395
    0.0841
    6.3033
   49.8071
   67.1349
    0.0001
```

The numbers above, are, from top to bottom, the inductance (in Henries), the core losses, DC bias winding losses, the AC conduction winding losses, the proximity effect winding losses, the core temperature, the winding temperature, and the boxed volume (in cubic metres).

You can also retrieve the results as nicely formatted string, showing all the details of the design (core, material, windings, etc.) by entering the commands

```
geckom.getResultsText()
```

which should then display the full design specifications and simulations results:

```
ans =

E-Inductor:          L = 355.4655 µH

Core:                            E55
Type:             EE with 3 air gaps
Core Material:                  N87+
Number stacked:                    1

Dimensions (mm):
Air gap:                         1.2
Height:                     h = 27.8
Width:                      w = 55.0
Inner width:               wi = 37.5
Mid-leg width:             wm = 17.2
Window height:             hw = 18.5
Length (one core):          l = 21.0
Length (total stack):       l = 21.0

***********************************
Winding:                       AWG10
Type:              Round solid wire
Material:        Copper (drawn wire)
Number of turns:              N = 36

Dimensions (mm):
Conductor diameter:        d = 2.588
Isolation thickness:      s = 0.0444
Wire spacing:               yd = 0.0

***********************************
Losses (W):
Core Losses:                    0.59
Winding losses DC:              0.04
Winding losses skin effect:     0.08
Winding losses prox. effect:     6.3
TOTAL:                          7.02

Winding temperature:         67.13 C
Core Temperature:            49.81 C
Inductor Orientation:
VERTICAL_LEFTRIGHT
Convection:                   FORCED
Air Flow Direction:        FRONTBACK
Air Speed:                   8.0 m/s

Total Boxed Volume:   122.1234 cm^3
***********************************
```

Working with GeckoMAGNETICS Design and Waveform files using *GeckoMAGNETICS Script*

As mentioned earlier, if you want to define magnetic components or standard waveforms using GeckoMAGNETICS Script, you must modify or create **.gmd* and **.gmw* files. Before trying this, open the supplied files TutorialDesign1.gmd and TutorialWaveform1.gmw in a text editor. If you are using Windows, to make sure that the file is formatted for easy reading, open it using *WordPad*, instead of the standard *Notepad*. Note the structure of the files. You can see that a magnetic component design is described using a series of nested XML tags. At the top of this hierarchy of tags are the *type categories* - "core" and "winding". Below that are the *property categories* – e.g. "coreType", "windingType", "coreMaterial", etc. Finally there is the *property* – e.g. a dimension ("h") or a component name (e.g. "name" of a core). First, try reading some elements of the supplied inductor design file. For example, to read the name of the predefined core used in the design, type

```
geckom.readFromParamFile('C:/tutori
al/TutorialDesign1.gmd','core','','
name')
```

and you should get the answer

```
ans =

E55
```

Note that in this case we leave the *property category* argument as an empty string, since the *property* we want to read ("name") is directly in the *type category* tag ("core"). To read the name of the core type used in the design, we need to qualify fully all arguments:

```
geckom.readFromParamFile('C:/tutori
al/TutorialDesign1.gmd','core','cor
eType','name')
```

which displays

```
ans =

EE 3 air gaps
```

To read the name of the core material, type

```
geckom.readFromParamFile('C:/tutori
al/TutorialDesign1.gmd','core','cor
eMaterial','')
```

Note that here the last argument (*property*) is left as an empty string, because the *property category* in this care ("coreMaterial") contains directly the data (there are no nested sub-tags – see the design file). The above command gives the result

```
ans =

N87+
```

To read the diameter of the wire used in the design, type

```
geckom.readDoubleFromParamFile('C:/
tutorial/TutorialDesign1.gmd','wind
ing','dimensions','d')
```

which returns

```
ans =

    0.0026
```

Note the difference between `readFromParamFile(...)` and `readDoubleFromParamFile(...)`. The former function will read any property in the design or waveform file, but will return the result always as a string. To get the result as a number (which is desired in the case of reading e.g. the number of turns or stacked cores), use the latter function. Note that if you try to read a non-numeric value with it (e.g. a winding type name) that it will throw an error.

Continue to try out different combinations with the above two functions and try to read more properties from the supplied **.gmd* and **.gmw* files. Once you have gotten a feel for the structure of these XML files, continue to the next step.

To modify and create new design and waveform files, you must use the `writeToParamFile(...)` and `setNewParamFile(...)` functions. Their syntax is like that of the functions used for reading, except that there is an additional argument at the end – the value to be written to the file. This value can be a string or a number (a double).

What we want to do is to create a new design file, called TutorialDesign2.gmd, which is identical to the supplied TutorialDesign1.gmd, except that the number of turns is 30 rather than 36. To accomplish this, type

```
geckom.setNewParamFile('C:/tutorial
/TutorialDesign1.gmd','C:/tutorial/
TutorialDesign2.gmd','winding','tur
ns','','30')
```

The `setNewParamFile(...)` function creates a new file (in this case, TutorialDesign2.gmd) without modifying the old file. Open this file, and you will see that it is identical to the original (TutorialDesign1.gmd) except that the number of turns is 30 instead of 36. Now we wish to continue modifying the design, but do not want to generate any more files. For this we use the `writeToParamFile(...)` function. To change the air gap to 0.5 mm from 1.2 mm, type

```
geckom.writeToParamFile('C:/tutoria
l/TutorialDesign2.gmd','core','core
Type','gap',0.0005)
```

If you open the file, you will see the air gap set to 5E-4. Now, load the modified design to GeckoMAGNETICS

```
geckom.loadDesignFile('C:/tutorial/
TutorialDesign2.gmd')
```

and you will see in the GUI the new design with the changed number of turns and air gap. Simulate the design

```
geckom.evaluateComponent()
```

and you will see that the results will change according to modifications made (e.g. different inductance and losses than before).

Note that if you use the `setNewParamFile(...)` function and leave the first argument empty, a new empty file be created, containing only the property you defined in that function call. For example, if you execute

```
geckom.setNewParamFile('','C:/tutor
ial/TutorialDesign3.gmd','core','co
reMaterial','','N27+')
```

It will create a new file called TutorialDesign3.gmd which contains only the following:

```
<?xml version="1.0" encoding="UTF-
8"?>
<inductor>
    <core>
        <coreMaterial>N27+</coreMat
erial>
    </core>
</inductor>
```

Now that this file has been created, you can use the `writeToParamFile(...)` function to add further properties. In this way you can build up a design or waveform file, i.e. define a magnetic component design or standard waveform from scratch.

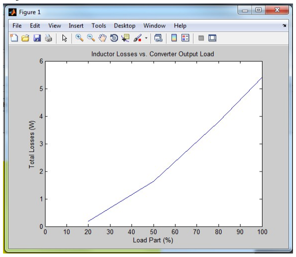
Scripting Example with MATLAB

In a real scripting application, you will not want to enter commands into the MATLAB console. Rather, you will want to build scripts in **.m* files that will automate your use of GeckoMAGNETICS. To demonstrate how this can be done, a file called `loadEvaluation.m` has been provided with this tutorial. Please open it and study it.

What we want to accomplish in this example is to evaluate the losses of an inductor at different load levels of the power converter in which it is to be used. We will use the inductor design we created earlier in the file TutorialDesign2.gmd. For this example, four different waveform files have been supplied with this tutorial – TutorialWaveform100pctLoad.gmw, TutorialWaveform80pctLoad.gmw, TutorialWaveform50pctLoad.gmw, and TutorialWaveform20pctLoad.gmw – each defining the inductor flux waveform at 100%, 80%, 50%, and 20% of the full converter output load, respectively.

The file `loadEvaluation.m` contains a script which loads the design file, then loads each waveform file, simulates the component with it, and stores the results in MATLAB, and then finally plots the total inductor losses vs. load level in a graph. Please note that the file paths for loading files in this script contain the previously assumed C:\tutorial directory – change them according to the actual file path on your computer, if necessary. The script also assumes that MATLAB is already connected to GeckoMAGNETICS, with all the libraries imported and a "geckom" object created, as was done in the preceding sections of this tutorial. If you write your own script, you may need to include in it the commands for adding to the MATLAB java path, importing the scripting libraries, creating a scripting object and connecting to GeckoMAGNETICS.

Now execute the script in MATLAB. GeckoMAGNETICS will perform four simulations, and then in MATLAB a graph of inductor losses vs. converter load will be displayed.



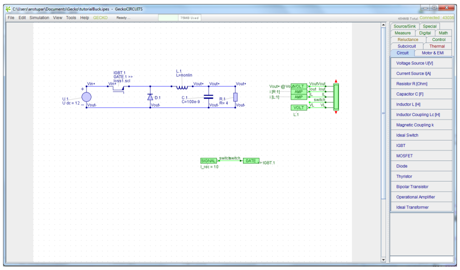
Working with GeckoCIRCUITS using *GeckoMAGNETICS Script*

Instead of using defined standard waveforms, GeckoMAGNETICS can extract simulation waveforms of magnetic components from a GeckoCIRCUITS circuit simulation. This can be done via *GeckoMAGNETICS Script* as well.

A file has been supplied with this tutorial called tutorialBuck.ipes. It contains a GeckoCIRCUITS simulation file of a simple buck converter with one inductor. By typing

```
geckom.loadCircuitsSimulationFile('
C:/tutorial/tutorialBuck.ipes')
```

you will switch the GeckoMAGNETICS waveform tab to "From GeckoCIRCUITS", GeckoCIRCUITS will start, a connection between GeckoCIRCUITS and GeckoMAGNETICS will be made, and the model of the buck converter will be opened in GeckoCIRCUITS.



Next, we must specify the simulation parameters – which component in the GeckoCIRCUITS model is our inductor, what are the names of the inductor's current and voltage signals, how long to simulate, which simulation step to use, and what is the frequency of our waveform. To do this, type

```
geckom.setCircuitsSimulationParamet
ers(15,'L.1','VL','IL',1e-9,1e-
3,200e3,200e4)
```

The arguments, from first to last, indicate the maximum current we expect through the inductor in the simulation (15 A), the inductor component name in GeckoCIRCUITS ("L.1"), the measurement signals showing the inductor voltage and current ("VL" and "IL" respectively), the circuit simulation time-step (1 ns), the total simulation time (1 ms), the fundamental frequency of the inductor waveform – in this case 200 kHz, which is the switching frequency of our buck converter – and the maximum expected frequency in the waveform – in this case set to 2 MHz, which means that GeckoMAGNETICS will take into account at least 10 harmonics of the current waveform when calculating losses. If you now look at the GeckoMAGNETICS Waveform tab, you will see that all of the above paremeters have been so set.

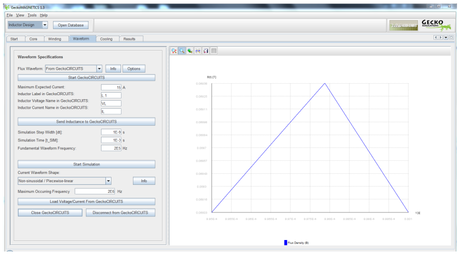
The last thing which must be specified when using GeckoCIRCUITS is the type of the waveform. Since this is a buck converter, we expect a triangular – piecewise-linear – waveform. To specify this, type

```
geckom.setWaveformType('PIECEWISE_L
INEAR')
```

You will see in the Waveform tab that the drop-down box now shows the waveform type as "Non-sinusoidal / Piecewise-linear". Now we can run the simulation. By calling

```
geckom.evaluateComponent()
```

the GeckoCIRCUITS simulation will be executed, the current and voltage waveforms transferred to GeckoMAGNETICS, and then GeckoMAGNETICS will calculate the losses. You will see the extracted flux waveform in the Waveform tab, as well as the calculated losses in the Results tab.



Waveform Specifications

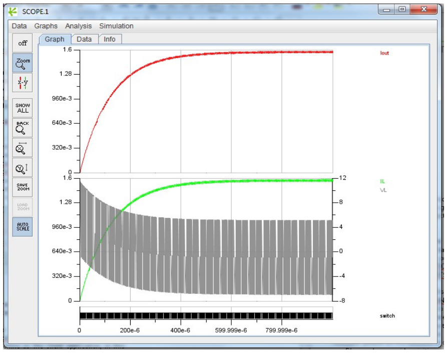
Flux Waveform: From GeckoCIRCUITS

Maximum Expected Current: 15 A
Inductor Label in GeckoCIRCUITS: L.1
Inductor Voltage Name in GeckoCIRCUITS: VL
Inductor Current Name in GeckoCIRCUITS: IL

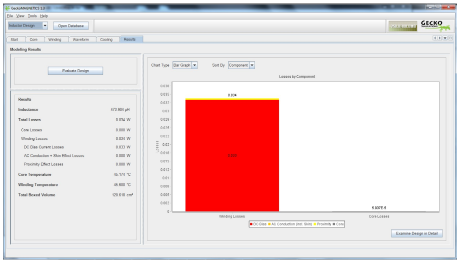
Simulation Step Width [dt]: 1E-9 s
Simulation Time [t_SIM]: 1E-3 s
Fundamental Waveform Frequency: 2E5 Hz

Current Waveform Shape: Non-sinusoidal / Piecewise-linear

Maximum Occuring Frequency 2E6 Hz



In GeckoCIRCUITS, you will see that the simulation has been completed and the inductor waveforms will be visible in SCOPE.1.



Modeling Results

Results	
Inductance	473.904 µH
Total Losses	0.034 W
Core Losses	0.000 W
Winding Losses	0.034 W
DC Bias Current Losses	0.033 W
AC Conduction + Skin Effect Losses	0.000 W
Proximity Effect Losses	0.000 W
Core Temperature	45.174 °C
Winding Temperature	45.600 °C
Total Boxed Volume	120.618 cm³

Using *GeckoSCRIPT* together with *GeckoMAGNETICS Script*

Through the use of GeckoMAGNETICS Script, you can load GeckoCIRCUITS simulation files, and set the simulation parameters, but you cannot make any changes to the actual circuit simulation model. However, GeckoCIRCUITS has its own scripting interface called *GeckoSCRIPT* which makes this possible. This interface is in fact what makes the exchange of data between GeckoCIRCUITS and GeckoMAGNETICS possible. Usually, you can make only one connection to GeckoCIRCUITS via *GeckoSCRIPT*. However, when you start GeckoCIRCUITS through GeckoMAGNETICS using *GeckoMAGNETICS Script*, one additional connection to the running GeckoCIRCUITS instance becomes possible. This allows you to connect **directly** to GeckoCIRCIUTS at the same time as to GeckoMAGNETICS, and to control, **in parallel**, GeckoCIRCUITS using GeckoSCRIPT, and GeckoMAGNETICS using GeckoMAGNETICS Script, while the two are, at the same time, connected to each other. For the remainder of this tutorial, we assume that you are familiar already with *GeckoSCRIPT*. If not, please consult the *GeckoSCRIPT* tutorial which came together with your distribution of GeckoCIRCUITS.

As we have previously added the GeckoCIRCUITS *.jar file to MATLAB's java path, what remains is to import the *GeckoSCRIPT* functions by typing

```
import('gecko.GeckoRemoteObject.*')
```

Then, create a *GeckoSCRIPT* object through which to control GeckoCIRCUITS by typing

```
geckoc =
gecko.GeckoRemoteObject.connectToEx
istingInstance(43035)
```

The above command is given under the assumption that GeckoMAGNETICS has connected to GeckoCIRCUITS via the default port – 43035. If you changed this port in the GeckoMAGNETICS settings, enter it instead of 43035 above. If the connection is successful, you will get output similar to

```
You are now connected to the
GeckoCIRCUITS instance at port
43035

geckoc =

gecko.GeckoRemoteObject@4cb0d314
```

Now, we can modify the buck converter model from MATLAB. If you look at the model, you will see that the converter load is a resistor called "R.1". To increase the load of the buck converter to 10 Ω, type

```
geckoc.setParameter('R.1','R',10)
```

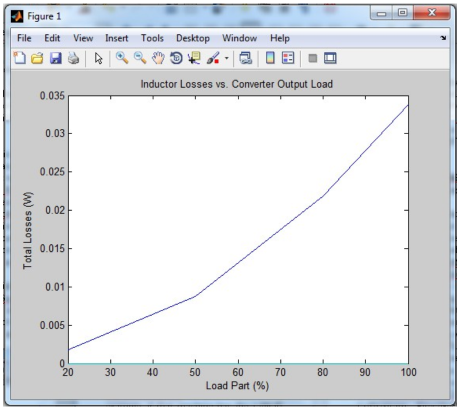
If you now go back to GeckoCIRCUITS, you will see that the resistance of R.1 is given as "R = 10".

Scripting Example with MATLAB and GeckoCIRCUITS

A second MATLAB **.m* file called loadEvaluationGeckoCIRCUITS.m has also been supplied with this tutorial. Please open it and study it. The purpose of this second scripting example is the same as the first – we want to evaluate the losses of an inductor design at different load levels of a converter. However, this time, instead of preparing waveform files for the different load levels before-hand, we will use the GeckoCIRCUITS simulation of the buck converter. By using *GeckoSCRIPT* as described in the previous section, we will modify the converter load directly in the simulation model and then evaluate the component in GeckoMAGNETICS – meaning that each time a different waveform will be extracted from the GeckoCIRCUITS simulation. In this way, we combine the power of both GeckoMAGNETICS and GeckoCIRCUITS scripting to use one inductor design file and one circuit simulation file to evaluate an inductor's losses at different operating points.

Please note that as in the previous example, this script assumes that a GeckoMAGNETICS scripting object "geckom" and a GeckoCIRCUITS scripting object "geckoc" have already been created and connections to GeckoMAGNETICS and GeckoCIRCUITS established. Also, if required, modify the file path to the design and model files according to their locations on your computer.

Now execute the script in MATLAB. GeckoCIRCUITS and GeckoMAGNETICS will each perform four simulations, and then in MATLAB a graph of inductor losses vs. converter load will be displayed.



Once this script has finished executing successfully, you can terminate the connection between GeckoCIRCUITS and MATLAB by typing

```
geckoc.disconnectFromGecko()
```

which should give you the following output

```
You have been disconnected from the
GeckoCIRCUITS instance at port
43035
GeckoREMOTE session closed.
```

Now, you can disconnect MATLAB from GeckoMAGNETICS by typing

```
geckom.disconnect()
```

If successful, you will get the output

```
ans =

     1
```

If you now go to the GeckoMAGNETICS Remote Access Settings window, you will see that indicates no current connection.

Connecting to GeckoMAGNETICS via the network

The previous examples all had GeckoMAGNETICS runnning on the same machine as the client application, in this case MATLAB. If you have access to two machines, you can try the following. Start MATLAB on one machine, and GeckoMAGNETICS on the other. Once you've followed all of the steps from the initial sections of this tutorial (including copying the correct **.jar* file to the machine running MATLAB!) to setup MATLAB and GeckoMAGNETICS to use scripting, call the connect(...) function in MATLAB but replace 'localhost' with the address of the machine running GeckoMAGNETICS on your network. For example, if that machine has the LAN IP address of 192.168.0.107, type

```
geckom.connect('192.168.0.107',4304
5)
```

assuming you've opened port 43045 on the other machine. If the connection is successful, you will get

```
ans =

     1
```

Now you can try any of the previous examples, but now truly remotely, over a network. Beware, the files you are manipulating (**.gmd*, **.gmw*, **.ipes*) **must** be on the machine running GeckoMAGNETICS, NOT on the machine running MATLAB!

Starting GeckoMAGNETICS using *GeckoMAGNETICS Script*

All the previous examples assumed that you had already started GeckoMAGNETICS before running you client application and that you manually enabled it for remote access. Using *GeckoMAGNETICS Script*, you can also start GeckoMAGNETICS, with the started instance immediately being enabled for scripting access once it finishes loading. Note that starting GeckoMAGNETICS using the scripting functions works **only** when the scripting client (e.g. MATLAB) and the GeckoMAGNETICS installation are located on the **same** machine. For the following we assume the use of a Windows machine, with GeckoMAGNETICS being installed at C:\Program Files\GeckoMAGNETICS. If this is not the case on your computer, please modify the file paths in the example commands accordingly. To try starting GeckoMAGNETICS using a scripting function, once you have opened

MATLAB and set it up for the use of *GeckoMAGNETICS Script*, type

```
geckom.startGeckoMAGNETICS('C:/Prog
ram
Files/GeckoMAGNETICS/bin/geckomagne
tics.exe')
```

GeckoMAGNETICS should start, and once it is loaded, if you open the Remote Access Settings window, you will see that GeckoMAGNETICS is waiting for being connected with MATLAB. Once you are done with GeckoMAGNETICS, you can also shut it down completely by typing

```
geckom.shutDown()
```

and GeckoMAGNETICS will close.

This concludes our introduction to *GeckoMAGNETICS Script*. To find out more about the scripting functionality of GeckoMAGNETICS, please take a look at the complete list of available scripting functions on the next page.

GeckoMAGNETICS Script Functions

Below, a list of all available scripting functions is given, with explanations and/or examples of their use. The list given uses standard Java syntax. Examples using MATLAB/Octave syntax were given in the previous sections. Note that when using the scripting methods in a custom Java application, to create a scripting object through which you can call the functions, you need to do

```
GeckoMagRemoteClient geckom = new GeckoMagRemoteClient();
```

where "geckom" is the name we chose for the object in this example. Once you create this object, you can execute the following methods (functions) of it:

`boolean connect(``String host`**`)`**

Connects to an instance of GeckoMAGNETICS open for remote access via the default port (43045). The "host" string should be the address of the machine on which GeckoMAGNETICS is runnning (e.g. "localhost" if the client and GeckoMAGNETICS are on the same machine, or something like "192.168.1.2" if GeckoMAGNETICS is running on another machine on the local network, etc.). Returns true if a connection is successfully established, false otherwise.

`boolean connect(``String host, int port`**`)`**

Same as above, except that the port through which to establish a connection with GeckoMAGNETICS must be explicitly specified via the second argument.

`boolean disconnect()`

Disconnects from GeckoMAGNETICS. Returns true if disconnection is successful.

`boolean startGeckoMAGNETICSNoCallback(``String path, int port`**`)`**

This method attempts to start GeckoMAGNETICS and automatically configure it for remote access at the given port. The "path" string must must be an absolute path to the GeckoMAGNETICS executable (e.g. "C:\Program Files\GeckoMAGNETICS\bin\geckomagnetics.exe"). This method returns once the command to run the executable is sent to the operating system (returns true if it is passed successfully, false otherwise) and makes **NO GUARANTEE** as to how long it will take for GeckoMAGNETICS to be up and running and ready for connections. Therefore this method should be used **FOR TESTING ONLY**, to see if you can start GeckoMAGNETICS via the scripting function on your system. **NOTE:** the client and the GeckoMAGNETICS server **MUST** be on the same machine!

boolean startGeckoMAGNETICSNoCallback()
boolean startGeckoMAGNETICSNoCallback(`String path`**)**
boolean startGeckoMAGNETICSNoCallback(`int port`**)**

Variations of the previous method using the default port and/or having the scripting client automatically try to figure out the location of the GeckoMAGNETICS executable in the system. All limitations as above apply to these three methods as well.

boolean startGeckoMAGNETICS()

This method tries to find the location of the GeckoMAGNETICS executable automatically, start GeckoMAGNETICS, set it up for remote access via the default port (43045) and waits (at the default port, 43055) for GeckoMAGNETICS to confirm that it is ready. When GeckoMAGNETICS is started this way, once it is completely finished loading and open for remote access, it will send a message back to the scripting client stating that it is ready. This method therefore blocks until this message is received or the default timeout (10 minutes) is exceeded. If this method returns true, then it is completely safe to immediately afterward call `connect("localhost")`. **NOTE:** the client and the GeckoMAGNETICS server **MUST** be on the same machine!

boolean startGeckoMAGNETICS(`String path`**)**

Same as above, except that the user must here explicitly specify the path to the GeckoMAGNETICS executable, e.g. "C:\Program Files\GeckoMAGNETICS\bin\geckomagnetics.exe".

boolean startGeckoMAGNETICS(`int remoteAccessPort, int waitForReplyPort, int waitForReplyTimeout`**)**

Tries to find the location of the GeckoMAGNETICS executable automatically, starts GeckoMAGNETICS, sets it up for remote access via the specified port (`remoteAccessPort`) and waits (at `waitForReplyPort`) for GeckoMAGNETICS to confirm that it is ready. When GeckoMAGNETICS is started this way, once it is completely finished loading and open for remote access, it will send a message back to the scripting client stating that it is ready. This method therefore blocks until this message is received or the specified timeout (`waitForReplyTimeout`) is exceeded. If this method returns true, then it is completely safe to immediately afterward call `connect("localhost",remoteAccessPort)`. **NOTE:** the client and the GeckoMAGNETICS server **MUST** be on the same machine!

boolean startGeckoMAGNETICS(`String path, int remoteAccessPort, int waitForReplyPort, int waitForReplyTimeout`**)**

Same as the previous one, except that the user must here explicitly specify the path to the GeckoMAGNETICS executable, e.g. "C:\Program Files\GeckoMAGNETICS\bin\geckomagnetics.exe".

boolean shutDown()

Disconnects from GeckoMAGNETICS and shuts it down. Returns true if successful.

void loadDesignFile(`String gmdFileName`**)**

Loads a **.gmd* file containing core and winding information.

void writeToParamFile(`String fileName, String typeCategory, String propertyCategory, String property, double value`**)**
void writeToParamFile(`String fileName, String typeCategory, String propertyCategory, String property, String value`**)**

Modifies a **.gmd* or **.gmw* file using the tags used in these XML files. The arguments would be for example,
`typeCategory`: "core", "winding", "waveform"
`propertyCategory`: "coreType", "windingType", "bobbin", "parameters", "stackedCores", "windingMaterial"
`property`: "h", "name", "turns", "d", "windingMaterial", "dcBias"
`value`: 0.0278, 2, "E55", 25, 0.002, "Annealed Copper", 2.3

NOTE: this function overwrites the contents of the given file.

void setNewParamFile(`String oldFileName, String newFileName, String typeCategory, String propertyCategory, String property, double value`**)**
void setNewParamFile(`String oldFileName, String newFileName, String typeCategory, String propertyCategory, String property, String value`**)**

Same as the previous, except that the original file is not modified, but instead copied into a new file at the specified location (`newFileName`), with the new file then modified accordingly. If the first argument (`oldFileName`) is left null or empty, the newly created file contains only the property defined in the method call.

double readDoubleFromParamFile(`String fileName, String typeCategory, String propertyCategory, String property`**)**
String readFromParamFile(`String fileName, String typeCategory, String propertyCategory, String property`**)**

Reads from a **.gmd* or **.gmw* file using the tags used in these XML files. Arguments as above.

`void loadWaveformFile(``String gmwFileName`**`)`**

Loads a **.gmw* file containing an ideal waveform definition.

`void loadWaveformsFromFile(``int skipLines, String regex, int harmonics, String fileName`**`)`**

Loads the waveform from a text file (equivalent to "from file" option in the GUI). Arguments:
`skipLines`: the number of lines in the file to skip before starting to read data.
`regex`: defines the format in the file – e.g. "t,i,v" denotes data in three columns, time, current and voltage, separated by commas; "t v i" - time, voltage, current separated by white space.

`void loadCircuitsSimulationFile(``String ipesFileName`**`)`**

Starts GeckoCIRCUITS (if not started already), and opens the given **.ipes* file.

`void setCircuitsSimulationParameters(``double maxCurrent, String componentLabel, string voltageLabel, stringCurrentLabel, double dt, double tSim, double fundFreq, doubleMaxFreq`**`)`**

Sets the GeckoCIRCUITS simulation parameters.

`void setWaveformType(``String type`**`)`**

Sets the waveform type. This method should be used when using GeckoCIRCUITS or loading a waveform from a text file. Possible waveform types are "PIECEWISE_LINEAR", "SINUSOIDAL_IDEAL", "RECTIFIED_SINUSOIDAL_IDEAL", "SINUSOIDAL_HFRIPPLE", and "RECTIFIED_SINUSOIDAL_HFRIPPLE".

`void setCooling(``boolean doThermal, boolean isForced, String orientation, double ambientTemp, String… nonExposedSides`**`)`**

Sets the cooling type, ambient temperature, core orientation, and non-exposed sides. In a Java application, you may simply list as many sides as you wish to set to be not exposed to air flow at the end. For example, to set the left, right, and bottom sides to be non-exposed, call:

```
setCooling(true, false, "Top-Down", 
25, "left", "right", "bottom");
```

However, in MATLAB or Octave, for multiple sides, you MUST first construct an array of side names. The equivalent to the above would then be:

```
sides = [{'left'}, {'right'}, 
{'bottom'}]
setCooling(true,false,'Top-
Down',25,sides)
```

`void setForcedCooling(``String direction, double airSpeed`**`)`**

If forced cooling has been set as the cooling method using the previous function, sets the flow direction and the air speed for forced air flow.

void setNonExposedSideConstantRth(String side, double rth**)**

Sets a side of the component to be not exposed to air flow and to have the given constant thermal resistance.

void evaluateComponent()

Runs the calculation for model mode. If GeckoCIRCUITS is started, this command sends the inductance value to GeckoCIRCUITS, starts the simulation and reads the voltages and currents back.

String getResultsText()

Returns the results of a model mode calculation as a String (in the format of the “Examine in Detail” window).

double[] getResults()

Gets all the separate result components and returns them as an array. The value stored at each index is, by index value:
0 – inductance
1 – core losses
2 – winding losses due to DC bias
3 – winding losses due to AC conduction and skin effect
4 – winding losses due to proximity effect
5 – core temperature
6 – winding temperature
7 – total boxed volume

In addition to the above function, a series of functions exist to get all of the above results separately:

```
double getInductance()
double getTotalLosses()
double getCoreLosses()
double getWindingLosses()
double getWindingDCLosses()
double getWindingACLosses()
double getWindingProximityLosses()
double getCoreTemperature()
double getWindingTemperature()
double getBoxedVolume()
```

In this tutorial, a step-by-step introduction to the tool scripting feature GeckoMAGNETICS – *GeckoMAGNETICS Script* - was given. It has been shown how GeckoMAGNETICS can be controlled via the scripting functions from a client application (e.g. MATLAB/Octave) and how scripting can be used to automate tasks which would otherwise require the user's manual input.

If you have any further questions, or you would like to have a feature which is not available at the moment, please do not hesitate to contact us. If you experience problems while following this tutorial or using GeckoMAGNETICS in general, please also consider visiting our bug reporting platform at

www.bugs.gecko-simulations.org

Thank you.

Contact Information / Feedback

Andrija Stupar

Gecko-Simulations AG
Physikstrasse 3
CH-8092 Zürich, Switzerland

Phone +41-44-632 6576
Fax +41-44-632 1212

www.gecko-simulations.com
contact@gecko-simulations.com

Document version: December 2014