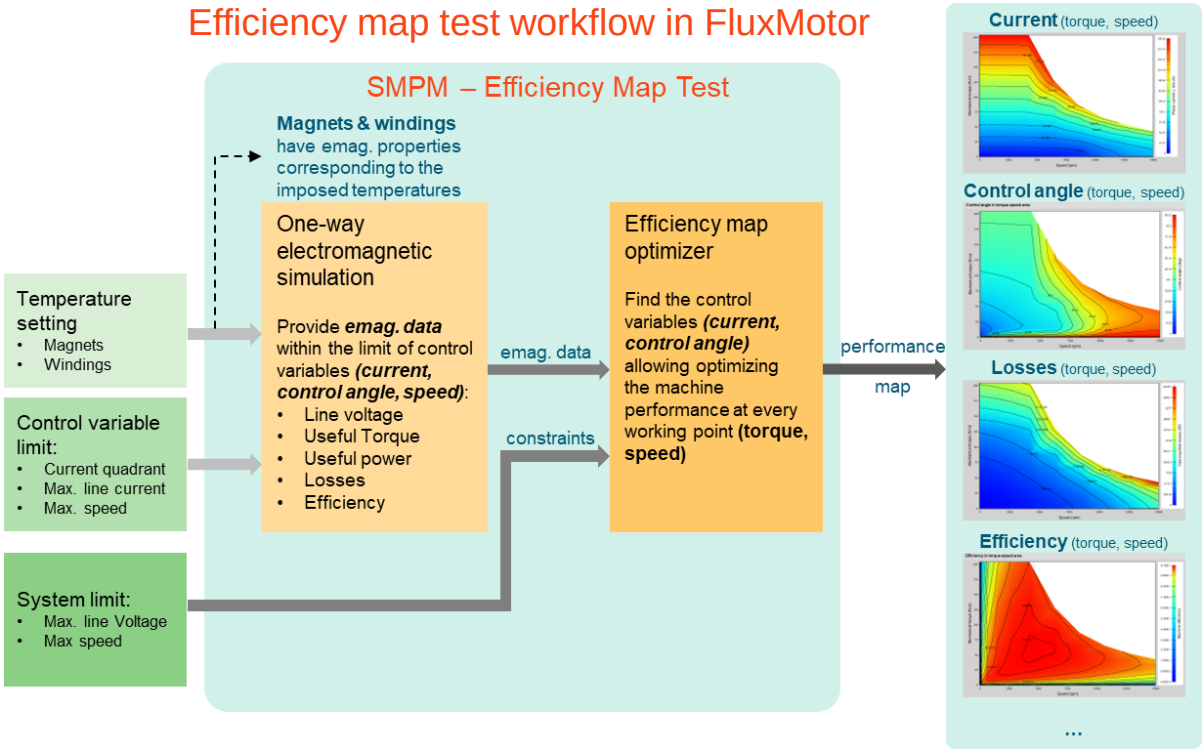
# TECHNICAL TUTORIAL FOR SCRIPT FACTORY

## EVALUATE THE REAL MACHINE TEMPERATURE IN AN EFFICIENCY MAP

Katalin Tamas, Anh-Tuan Vo - EM Solution / May 2024

# Context

## Efficiency map test workflow in FluxMotor




## How to evaluate the real temperature of SMPM at each working point in the efficiency map?

To address the real needs from the motor industry:

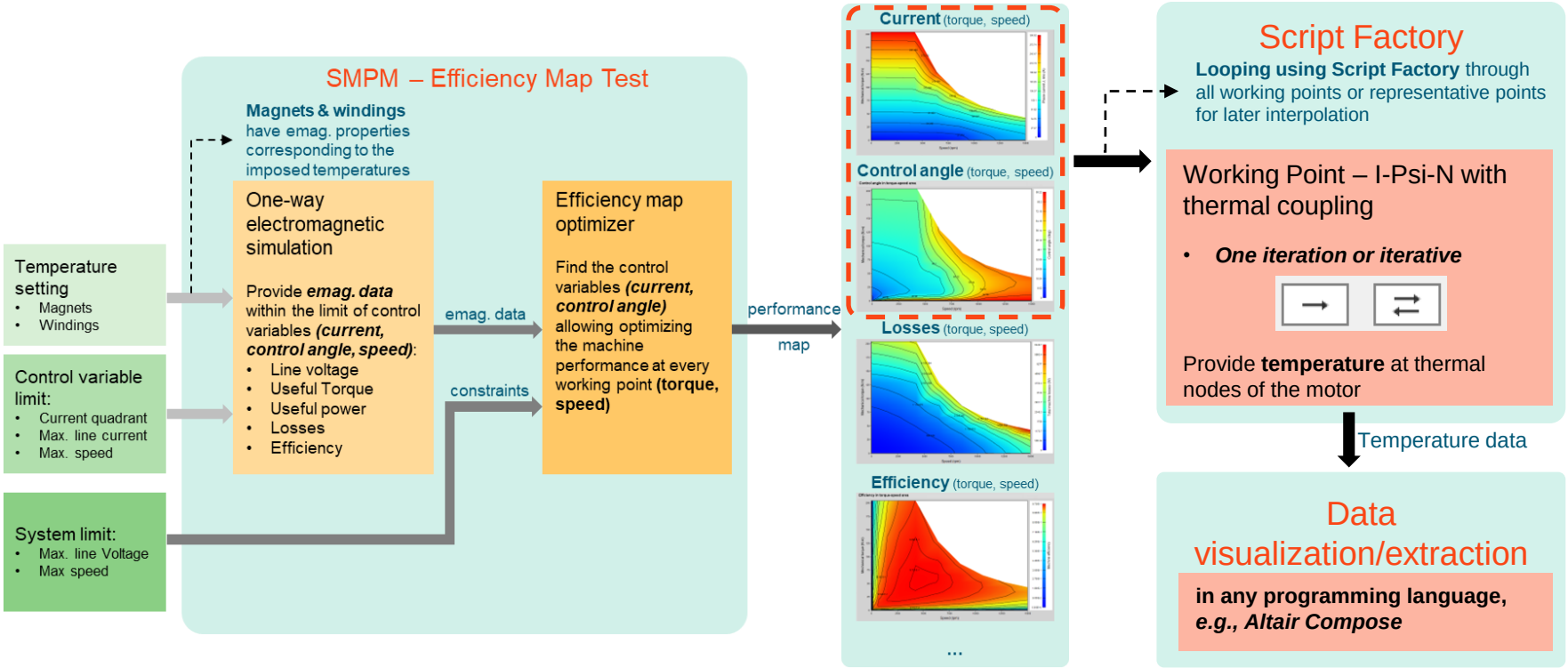
- Remove from the efficiency plot, all torque/speed points that would result in **temperatures over the pre-set thresholds**. In this way, a plot of all possible **safe operating points** is left.
- **Maximum torque and power outputs at a maximum acceptable temperatures**.



ALTAIR

# Solution

How to evaluate the real temperature of SMPM at each working point in the efficiency map?



# Objectives

- This tutorial allows users to evaluate the **steady-state temperature of a SMPM at all working point within its efficiency map via scripting in Script Factory**
  - An efficiency map will be built with imposed magnet and winding temperatures
  - Thermal-iterative working point test will be used to evaluate the steady state temperature of the machine at each (torque, speed) points
  - The user can have an estimation of temperature within the efficiency map and eliminate points that are not achievable compared to the imposed magnet temperature
  - Graphical representation can be done in Altair Compose
- Via this tutorial, users are desired to acquire the following technical skills using **Script Factory**:
  - Adjust the design of a motor
  - Execute tests in FluxMotor
  - Perform File operations such as export test data, or read text-based data
  - Perform loop operation to automate repetitive tasks

# SCRIPT FACTORY REMINDER

# Python command of a FluxMotor action

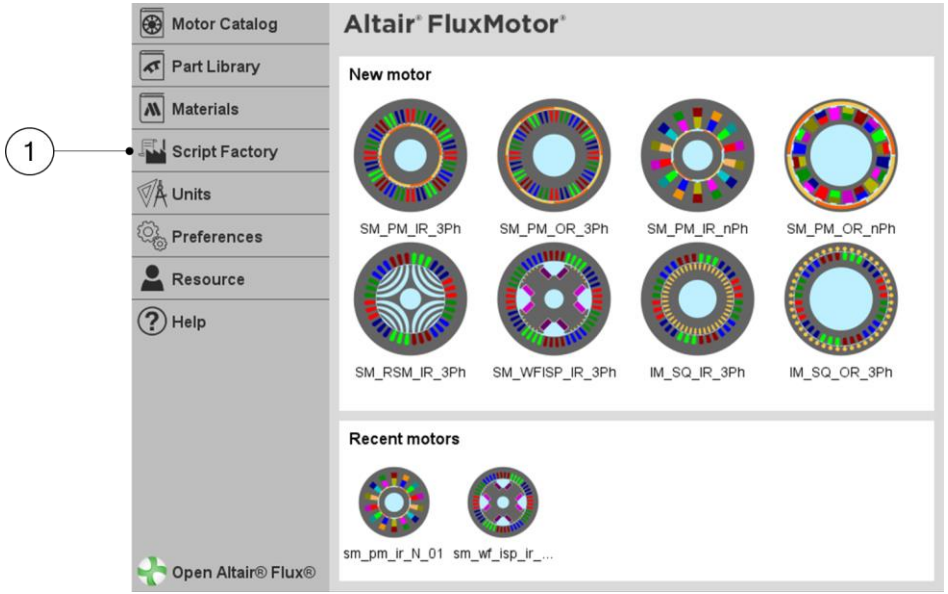
## And which ones are scripted?

- All

but really, all actions of FluxMotor are scripted, including:

  - Create / duplicate / save motor
  - Modify motor design
  - Modify input / setting of tests, launch them and export the obtained results
  - Export motor models to advanced tools of Altair

- Use Script Factory to run your FluxMotor script to automize your advanced studies.

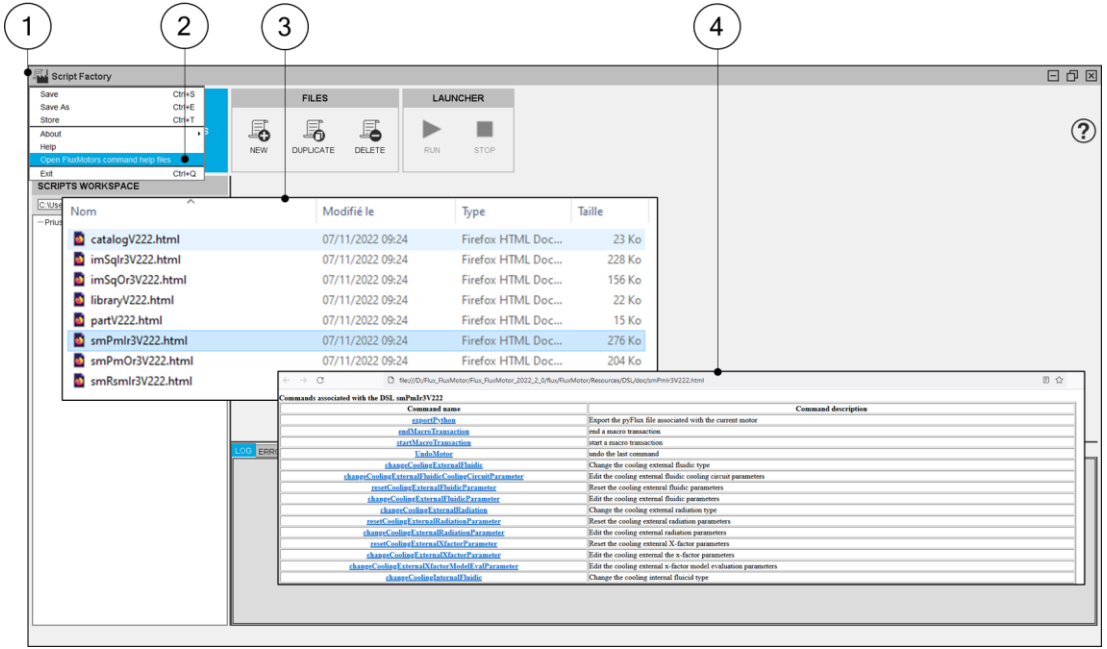


# Python command of a FluxMotor action

## And where to find them?

- A list of all available commands can be found via Script Factory Menu

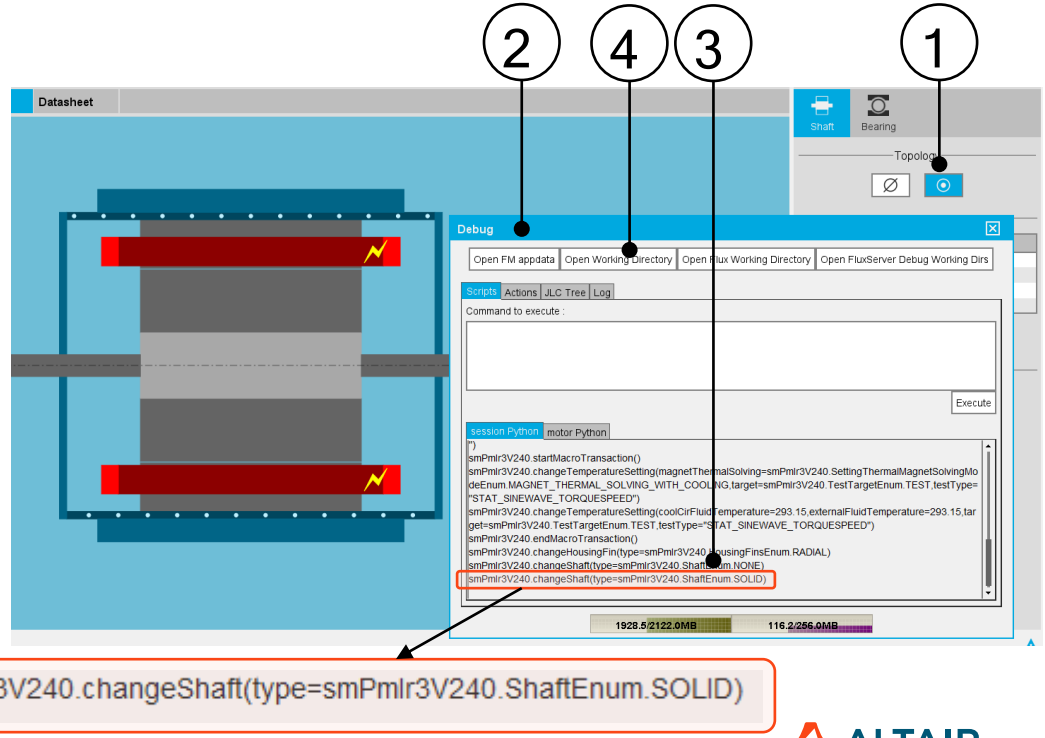
| | |
|---|---|
| 1 | Click on the icon “Script Factory” on the top left dropdown menu. |
| 2 | Select “Open FluxMotor command help files” option. |
| 3 | List of available commands dedicated to the main applications of FluxMotor. |
| 4 | Click on the command name to see the corresponding description. |



# Python command of a FluxMotor action

## And where to find them?

- Learn commands via practical examples, here are some tips:
  1. Run an action in FluxMotor (e.g., add a shaft to the motor)
  2. Enter the Debug Mode by the shortcut **Ctrl D**
  3. Copy the latest command and compare with the GUI action
  4. Via the Debug Dialog, select **Open Working Directory** to see all commands used for the motor in:
     - Session.py
     - MotorName.py (NISSAN_LEAF_1.py in the example)

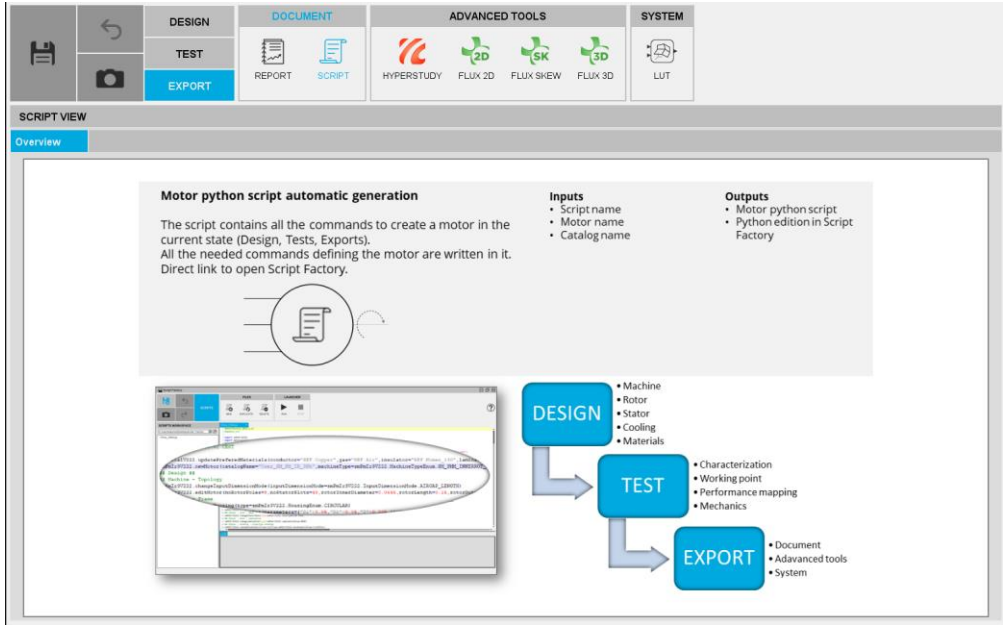



ALTAIR

# Python command of a FluxMotor action

## And where to find them?

- **Export >> Script** is also another way to see the commands to create the motor in the current state (Design, Tests and Exports)



# Python command of a FluxMotor action

## And Important notes about their inputs

- The unit system for inputs in FluxMotor GUI and its corresponding Python commands differs.
  - The FluxMotor GUI allows the selection of various unit systems, with the default being an adapted version of the Metric system, commonly preferred in the motor industry.
  - Conversely, in Python commands, all inputs are exclusively in the Metric system.

# Python libraries

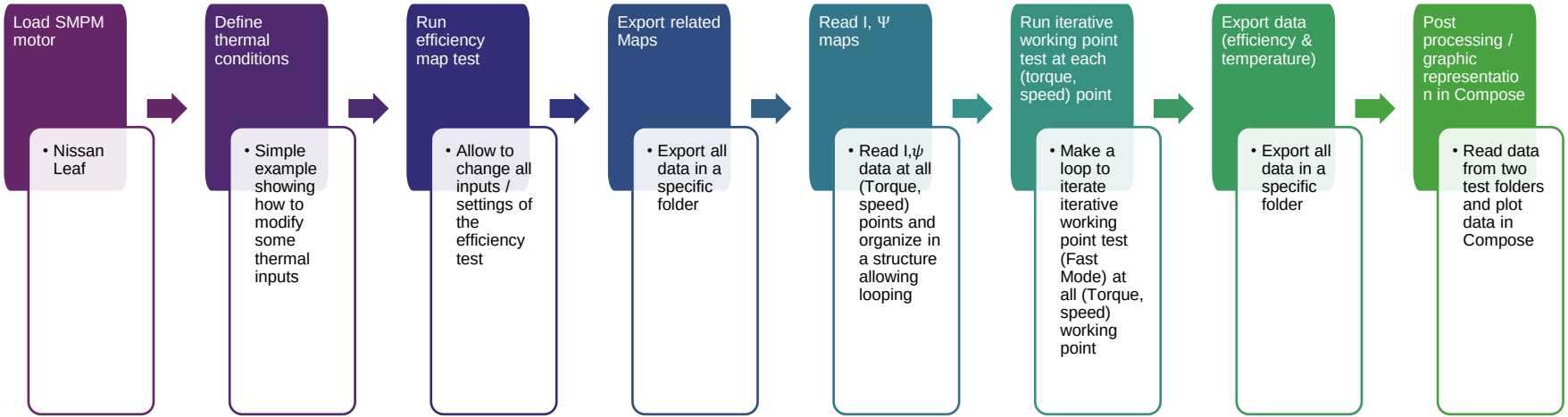
## And which ones are supported by Script Factory?

- Script Factory support all standard libraries of Python 2.7.

# STEPS TO BUILD THE SCRIPT

# Workflow

- The tutorial will outline the process of constructing a scripting solution, which involves the following steps:





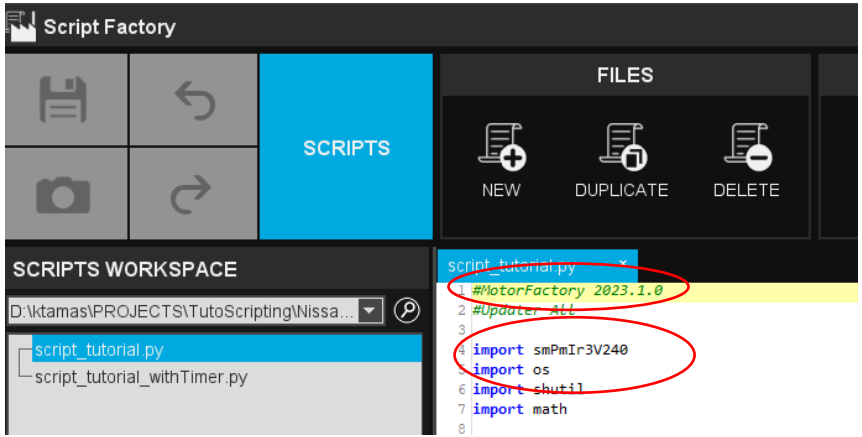

ALTAIR

# Script constraints

## Mandatory lines at the header part of the script

### Action

- Initialize the script file:
  - The first line is necessarily #MotorFactor <used version>
  - Before the first command, the library containing the FluxMotor scripting commands must be imported




### FluxMotor command

```
1  #MotorFactory 2023.1.0
2  #Updater_All
3
4  import smPmIr3V240
```

### Important note

- The second line assures the backward compatibility (if it will be executed later, with a newer version of FluxMotor).
- Other « import » commands (that we will use in the script later) can be gathered here, at the beginning of the script.



ALTAIR

# Load SMPM motor

Create *manually* a copy of Nissan_Leaf motor to the user catalog

## Important note

The scripting of the Motor Catalog application is not yet finished, so to have our motor in the user-catalog, we must copy manually the desired motor from the reference catalog.

Nevertheless, this operation can be replaced by the two commands bellow (openInternalMotor and saveAsMotor)

## Action

To create our motor to work with :

In the « Motor Catalog » application GUI, create a copy of Nissan_Leaf motor from the Automotive_Transport_1 reference catalog save it in the User_SM_PM_IR_3Ph user-catalog.

## FluxMotor command

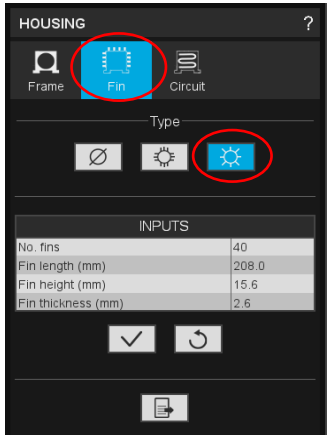
```
# 1) Open "Nissan_Leaf" motor from "Automotive_Transport_1" catalog
#    as this is a "REFERENCE" catalog, we use the "openInternalMotor" command
#   (to open motors in User catalogs, use the "openMotor" command)
smPmIr3V240.openInternalMotor(catalogName="Automotive_Transport_1",motorName="Nissan_Leaf")
# 2) Save the motor in the User catalog
smPmIr3V240.saveAsMotor(catalogName="User_SM_PM_IR_3Ph",motorName="Nissan_Leaf_tuto",eraseOld=True)
```

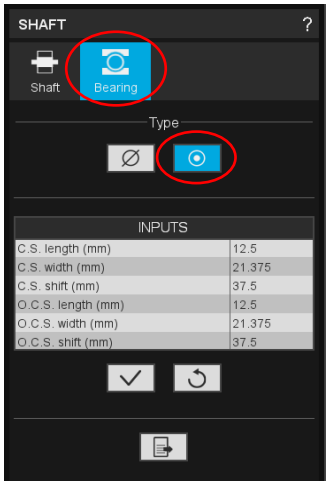
# Define thermal conditions

## Add necessary thermal details to the motor design

### Action

Thermal tests need a motor with housing and bearing, so in the GUI you can set :




### Important note

Copy Python command from:

- Debug window, or
- session.py file

to your script editor

### FluxMotor command

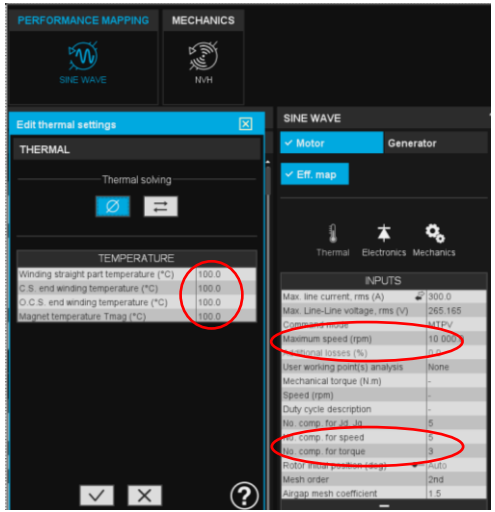
```
# 3) change the housing and the bearing to prepare the motor for thermal test
smPmIr3V240.changeHousingFin(type=smPmIr3V240.HousingFinsEnum.RADIAL)
smPmIr3V240.changeBearing(type=smPmIr3V240.BearingEnum.SOLID)
```

# Run the Efficiency-map test

## Thermal and speed settings of Efficiency Map test

### Action

- Modify some thermal and speed setting values.
- Run efficiency map test




### Important note

In Python commands:

- Temperature is converted from Celsius to Kelvin
- Speed is converted from revolution/min to radians/sec

### FluxMotor command

```
# 4) set temperature settings
smPmIr3V240.changeTemperatureSetting(connectionSideEndWindingTemperature=373.15,
                                     magnetTemperature=373.15,
                                     oppositeConnectionSideEndWindingTemperature=373.15,
                                     target=smPmIr3V240.TestTargetEnum.TEST,
                                     testType="STAT_SINEWAVE_TORQUESPEED",
                                     windingActiveLengthTemperature=373.15)

# 5) set max speed and the number of computation points
smPmIr3V240.changeStatSinewaveTorquespeedParameter(maximumSpeed=1047.1975511965977)   # in radians/sec
smPmIr3V240.changeStatSinewaveTorquespeedParameter(noComputationsForSpeed=5,noComputationsForTorque=3)
```

```
# 6) ===>  run test  STAT_SINEWAVE_TORQUESPEED
smPmIr3V240.runTest(testType="STAT_SINEWAVE_TORQUESPEED")
smPmIr3V240.saveMotor()
```

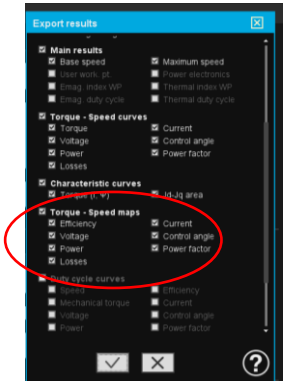
# Export results

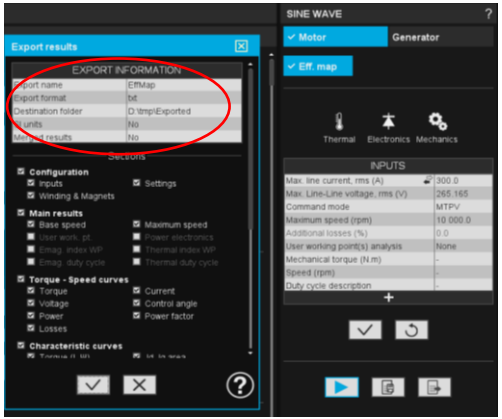
## Scripting

### Action

Export results obtained from efficiency map test to txt files saved:

- In a destination folder
- Under an export name




### Important note

- To automize the post-processing, destination folder and export name are assigned to the following variables:
  - *destinationFolder*
  - *effMapSubFolder*

### FluxMotor command

For better visibility / accessibility, we can move these two lines to the header part of our script

```
# 7) export test results in a new sub-folder
exportDestinationFolder="D:/tmp/Exported"
effMapSubFolder = "EffMap"

smPmIr3V240.exportTestResult(createExportFolder=smPmIr3V240.YesNoEnum.YES,
                             destinationFolder=exportDestinationFolder,
                             exportFormat=smPmIr3V240.ExportFileExtensionEnum.TXT,
                             exportName=effMapSubFolder,
                             mergedResults=smPmIr3V240.YesNoEnum.NO,
                             result=["controlAngleMap","currentMap","efficiencyMap",
                                     "inputData","currentCurve","settingData","basePointDat
                                     "powerCurve","powerFactorCurve","powerFactorMap","powe
                                     "voltageCurve","voltageMap","windingMagnetCharacterist
```

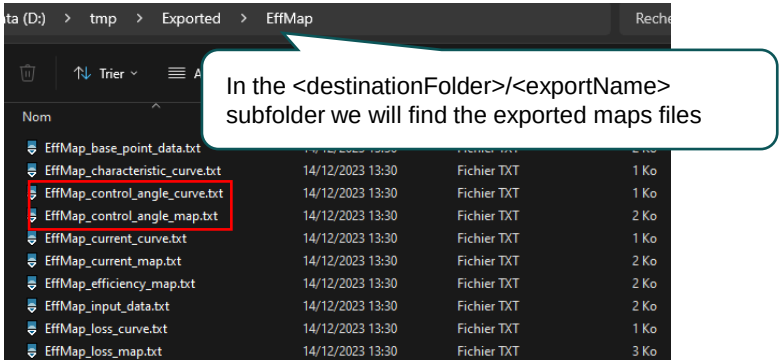
# Read current and control angle maps

## Path of data

### Action

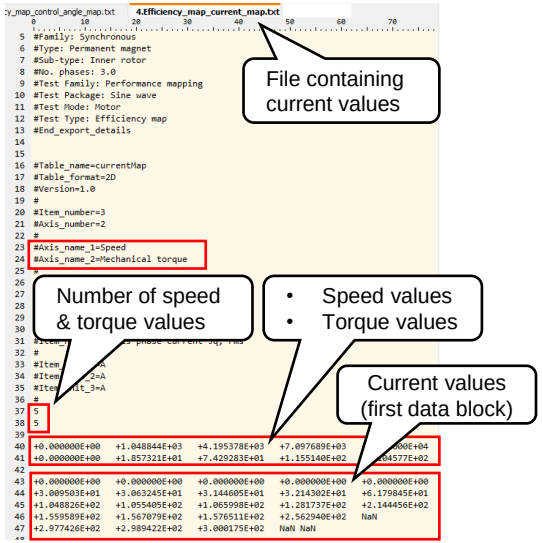
The exported files of Efficiency map test will be input data for Working point test. We need the two following map files:

- control_angle_map.txt
- current_map.txt




### Important note

Here is an example of data organization in txt files




### FluxMotor command

```
# 9) Exported data of the first test will be the input of the second test
#    control_angle_map file and path:
path_control_angle_map = os.path.join(effMapPath, effMapSubFolder + "_control_angle_map.txt")
#    current_map file and path:
path_current_map = os.path.join(effMapPath, effMapSubFolder + "_current_map.txt")
```

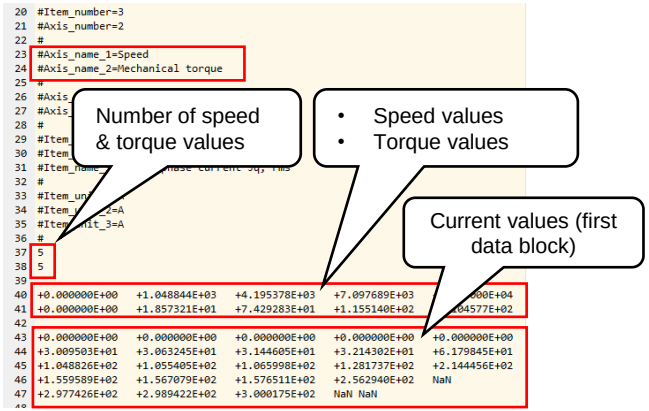
# Read current and control angle maps

## Method to extract needed data from txt files

### Action

In the « getMapFileValues » method, we:

- Open the map-files
- Read the needed information
- Store in variables




### Important note

- Script Factory support all standard libraries of Python 2.7.
- The getMapFileValues method can be found in the provided script.

### FluxMotor command

```
# 10) Method to get values from the exported data files
#-------------------------------------------------------------------------------
#    Get values from a map file
# From a file containing the exported results of the STAT_SINEWAVE_TORQUESPEED test,
# return the Speed, Torque or Map item number and a list with the values.
# Parameters: file_path: the full path of the result file
#             key: "Speed" / "Torque" / "Map" (if "Map" then return the first data block)
#-------------------------------------------------------------------------------
def getMapFileValues(file_path,key="Speed"):
    lines = []
    with open(file_path) as f:
        for line in f:
            lines.append(line)

    nb_speed = nb_torque = map_line_counter = 0
    speed_values = []
    torque_values = []
    map_values = []
    for i in range(len(lines)):
        line = lines[i]
```

# Read current and control angle maps

## Other methods

### Action

- Execute the « getMapFileValues » method to copy all the needed information to variables
- Convert speed and temperature to the required units of Python command:
  - Temperature: from Celsius to Kelvin
  - Speed: from revolution/min to radians/sec

### Important note

- Script Factory support all standard libraries of Python 2.7.
- The deg2rad and rpm2radPerSec methods can be found in the provided script.

### FluxMotor command

```
# 11) Get speed-, torque-, contorl-angle and current values
nb_speed, speed_values = getMapFileValues(path_control_angle_map,"Speed")
nb_torque, torque_values = getMapFileValues(path_control_angle_map,"Torque")
n, ctrl_angle_values = getMapFileValues(path_control_angle_map,"Map")
n, current_values = getMapFileValues(path_current_map,"Map")


# 12) Conversion of input values
#     Conversion of control angle (from degrees to radians)
def deg2rad(deg_value):
    return deg_value/180*math.pi

#     Speed conversion (from revolutions per minute to radians per second)
def rpm2radPerSec(rpm_value):
    return rpm_value/60*2*math.pi
```



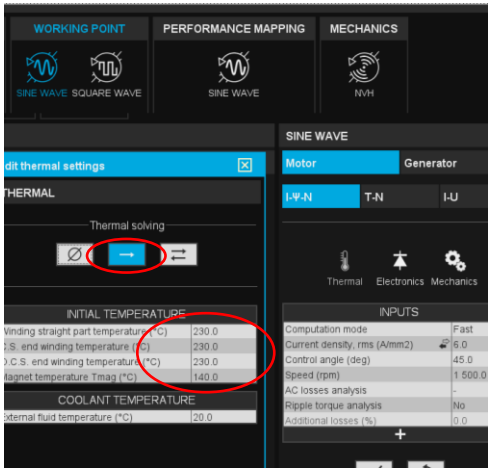

ALTAIR

# Working Point test

## Change thermal inputs

### Action

Change thermal inputs to the values corresponding to the ones used in efficiency map test




### Important note

- Choose types of thermal solving that you wish
  - One way: only one thermal simulation is run
  - Iterative: electromagnetic and thermal simulations are run iteratively until the machine temperatures converge
- Configure the test as you wish
  - Mode of computation: Fast / Accurate
  - AC losses analysis mode

### FluxMotor command

```
# 8) WP test settings
smPmIr3V240.changeTemperatureSetting(target=smPmIr3V240.TestTargetEnum.TEST,
                                     testType="WP_SINE_MOT_CURRENTPSISPEED",
                                     thermalSolving=smPmIr3V240.SettingThermalFullModeEnum.ONE_ITERATION)
smPmIr3V240.changeTemperatureSetting(connectionSideEndWindingTemperature=503.15,
                                     externalFluidTemperature=293.15,
                                     magnetTemperature=413.15,
                                     oppositeConnectionSideEndWindingTemperature=503.15,
                                     target=smPmIr3V240.TestTargetEnum.TEST,
                                     testType="WP_SINE_MOT_CURRENTPSISPEED",
                                     windingActiveLengthTemperature=503.15)
```

# Working Point test

## Run the test iteratively

### Action

- Create a loop that iterates torque and speed values of the efficiency map
- Read the corresponding current and control angle values from txt files
- Change the speed, current & control angle input of working point test
- Run the working point test
- Export the temperature results of each iteration (see next slide)

### FluxMotor command

```
# 13) Loop organisation for WP tests
#=====================================

# Loop on torque values (ordiata)
for j in range(nb_torque):
    # Loop on speed values (abscissa)
    for i in range(nb_speed):
        # conversion of speed value to radians/sec
        speed = rpm2radPerSec(speed_values[i])
        # map index
        index = j*nb_speed + i
        # conversion of control-angle (map value) value to radians
        ctrl_angle = deg2rad(ctrl_angle_values[index])
        # current (map value)
        current = current_values[index]
        if not math.isnan(current) and not math.isnan(ctrl_angle) and current!=0 and speed!=0:
            print "\n=======================================\n"+"i = "+str(i)+", j = "+str(j)+"\nctrl_angle = "+str(ctrl_angle)+", current = "+str

            # Set WP-test parameters
            smPmIr3V240.changeStatSinewaveWorkingPointCurrentSpeedPsiParameter(acLossesAnalysis=smPmIr3V240.AcLossesComputationMode.FE_ONE_PHASE,
                                                                               computationMode=smPmIr3V240.ComputationModeEnum.FAST,
                                                                               controlAngle=ctrl_angle,
                                                                               maximumLineCurrent=current,
                                                                               speed=speed)

            #  ===>  run test  WP_SINE_MOT_CURRENTPSISPEED
            smPmIr3V240.runTest(testType="WP_SINE_MOT_CURRENTPSISPEED")

            # Export WP test results
            exp_filename = "WP_I"+str(i)+'_N'+str(i)
            smPmIr3V240.exportTestResult(createExportFolder=smPmIr3V240.YesNoEnum.YES,
                                         destinationFolder=exportDestinationFolder_WP,
                                         exportFormat=smPmIr3V240.ExportFileExtensionEnum.TXT,
                                         exportName=exp_filename,
                                         mergedResults=smPmIr3V240.YesNoEnum.NO,
                                         mode=smPmIr3V240.ActionModeEnum.FORCED,
                                         result=["inputData","settingData","airgapFluxDensityCurve","temperatureData","workingPointData","winding
                                         siUnits=smPmIr3V240.YesNoEnum.NO,testType="WP_SINE_MOT_CURRENTPSISPEED")
```

# Working Point test

## Export the working point test result iteratively

### Action

- Export the temperature results of each iteration

You can specify here the format and the desired contains of the exported data :

### FluxMotor command

```
# Export WP test results
exp_filename = "WP_T"+str(j)+'_N'+str(i)
smPmIr3V231.exportTestResult(createExportFolder=smPmIr3V231.YesNoEnum.YES,
                             destinationFolder=exportDestinationFolder,
                             exportFormat=smPmIr3V231.ExportFileExtensionEnum.TXT,
                             exportName=exp_filename,
                             mergedResults=smPmIr3V231.YesNoEnum.NO,
                             result=["inputData","settingData","airgapFluxDensityCurve","workingPointData","windingMagnetCharacteristicData"],
                             siUnits=smPmIr3V231.YesNoEnum.NO,testType="WP_SINE_MOT_CURRENTPSISPEED")
```

| | |
|---|---|
| createExportFolder | Create a new folder for the result files? |
| destinationFolder | Destination folder name |
| exportFormat | txt or Excel format |
| exportName | Prefix of the exported files |
| mergedResults | in one file or each result in a separated file |
| mode | Erase or not old results |
| result | List of result types to export |
| siUnits | Convert (if needed) to SI units? |
| testType | "WP_SINE_MOT_CURRENTPSISPEED" |

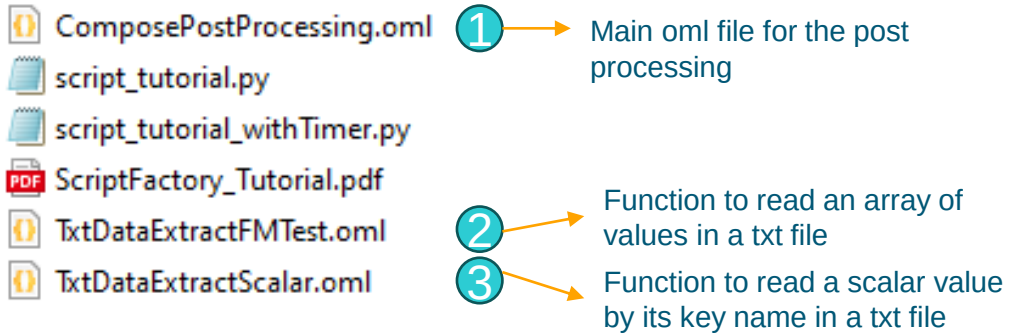
# Post-processing in Compose

## Workflow

### Action

1. Read results of the efficiency map test
2. Read results of the iterative working point test
3. Plot maps of efficiency, temperatures, …

### Compose files provided






Load SMPM motor → Define thermal conditions → Run efficiency map test → Export related Maps → Read I, Ψ maps → Run iterative working point test → Export data (efficiency & temperature) → Post processing in Compose

# Post-processing in Compose

## How to read curve/map data of FluxMotor

```
function [outputDimension, outputData] = TxtDataExtractFMTest(filePath, dataDim, keySize, keyData)
```

- Use **TxtDataExtractFMTest.oml** to read curve / map of FluxMotor

```
#Export_details
#Version: 2024.0.10
#Motor Name: Nissan_Leaf_tuto
#Catalog Name: User_SM_PM_IR_3Ph
#Family: Synchronous
#Type: Permanent magnet
#Sub-type: Inner rotor
#No. phases: 3.0
#Test Family: Performance mapping
#Test Package: Sine wave
#Test Mode: Motor
#Test Type: Efficiency map
#End_export_details


#Table_name=efficiencyMap
#Table_format=2D
#Version=1.0
#
#Item_number=1
#Axis_number=2
#
#Axis_name_1=Speed
#Axis_name_2=Mechanical torque
#
#Axis_unit_1=rpm
#Axis_unit_2=N.m
#
#Item_name_1=Machine
#
#Item_unit_1=1
#
11
11
```

General data of the motor & the test (string type)

#Table_format=2D

Name of the map and **number of dimensions/axis (dataDim)**

Name of the map axis

Unit of the axis

Name of the data

Number of elements per axis

**keySize**

**keyData**
If you want to extract axis values

**keyData**
If you want to extract data of the map

Axis of the map

Axis 1

Axis 2



Load SMPM motor → Define thermal conditions → Run efficiency map test → Export related Maps → Read I, Ψ maps → Run iterative working point test → Export data (efficiency & temperature) → Post processing in Compose

ALTAIR

# Post-processing in Compose

## How to read scalar data of FluxMotor

```
function outputData = TxtDataExtractScalar(filePath, key)
```

- Use **TxtDataExtractScalar.oml** to scalar values of FluxMotor by their keys

```
1   #Export_details
2   #Version: 2024.0.10
3   #Motor Name: Nissan_Leaf_tuto
4   #Catalog Name: User_SM_PM_IR_3Ph
5   #Family: Synchronous
6   #Type: Permanent magnet
7   #Sub-type: Inner rotor
8   #No. phases: 3.0
9   #Test Family: Working point
10  #Test Package: Sine wave
11  #Test Mode: Motor
12  #Test Type: Current-Control angle-Speed
13  #End_export_details
14
15
16  #Table_name=temperatureData
17  #Table_format=KeyValue
18  #Version=1.0
19  #
20  #Item_number=26
21  #
22  #----Machine----
23  Shaft_(C)=+2.212086E+01
24  Shaft_extension_C.S._(C)=+2.204016E+01
25  Shaft_extension_O.C.S._(C)=+2.203993E+01
26  Bearing_inner_C.S._(C)=+2.199172E+01
27  Bearing_outer_C.S._(C)=+2.196666E+01
28  Bearing_inner_O.C.S._(C)=+2.199145E+01
29  Bearing_outer_O.C.S._(C)=+2.196637E+01
30  Frame_(C)=+2.217080E+01
31  End_cap_C.S._(C)=+2.193772E+01
32  End_cap_O.C.S._(C)=+2.193741E+01
33  #----Rotor----
```

General data of the motor & the test (string type)

key → outputData

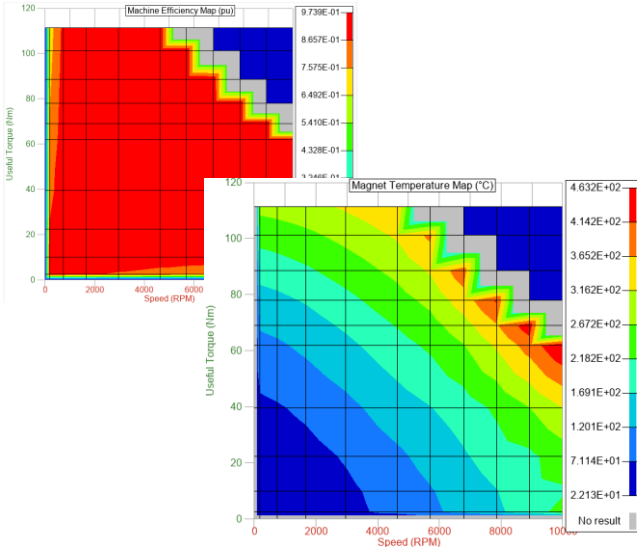


Load SMPM motor → Define thermal conditions → Run efficiency map test → Export related Maps → Read I, Ψ maps → Run iterative working point test → Export data (efficiency & temperature) → Post processing in Compose

# Post-processing in Compose

## How to read scalar data of FluxMotor

- Use **ComposePostProcessing.oml** for the post processing, the script can be described by 3 following steps.




**Step 1**

```
$////////////////////////////////////////////////////////////////////////////
$$ 1111111111111111111111111111111111111111111111111111111111111111111111
$$ READ FLUXMOTOR EFFICIENCY MAP TEST RESULT
$////////////////////////////////////////////////////////////////////////////
```

**Example**

```
% read Efficiency map_efficiency_map.txt
dataDim = 2;
keySize = 33;
keyData = 39;
FMFilePath = strcat(EMPath, '/', [EMFolder,'_efficiency_map.txt']);
[EMCurveFMDimension, EM.effMachine] = TxtDataExtractFMTest(FMFilePath, dataDim, keySize, keyData);
```

**Step 2**

```
$////////////////////////////////////////////////////////////////////////////
$$ 2222222222222222222222222222222222222222222222222222222222222222222222
$$ READ THERMAL DATA
$////////////////////////////////////////////////////////////////////////////
```

**Example**

```
% Read temperature data using TxtDataExtractScalar function
WP.tempShaft(i,j)          = TxtDataExtractScalar(filePath, 'Shaft_(C)');
WP.tempShaftCS(i,j)        = TxtDataExtractScalar(filePath, 'Shaft_extension_C.S._(C)');
```

**Step 3**

```
$////////////////////////////////////////////////////////////////////////////
$$ 3333333333333333333333333333333333333333333333333333333333333333333333
$$ PLOT MAP
$////////////////////////////////////////////////////////////////////////////
```

**Example**

```
% Efficiency Map
figure
contour(EM.N, EM.torqueUseful, EM.effMachine)
colorbar
xlabel('Speed (RPM)')
ylabel('Useful Torque (Nm)')
title('Machine Efficiency Map (pu)')
```

# Notes for usage

- The whole script (« script_tutorial.py ») can be found in the folder of this tutorial.
  - (You can also find another script with time-measuring : creates a file with the execution time for each test.)
- You can find a lot of information, explanations in the comment lines of this script.
- Before executing, do not forget to fill the destination folder, etc. at the header part of the script.

# THANK YOU

altair.com

#ONLYFORWARD



ALTAIR