

WEBINAR: PYLEECAN ADVANCED

PYthon Library for Electrical Engineering Computational ANalysis



6th November 2020

Pierre Bonneel – pierre.bonneel@eomys.com

Hélène Toubin – helene.Toubin@eomys.com

Emile Devillers – emile.devillers@eomys.com

Pr. Fabrizio MARIGNETTI from University of Cassino, AEIT



Ordine dei Periti Industriali e
dei Periti Industriali Laureati
della provincia di Frosinone



WEBINAR ORGANIZATION

1: How to use pyleecan (pyleecan basics, FEMM, GUI)

Friday 16 Oct 15:00-17:00 (GMT+2)

2: How to use pyleecan (advanced) (Optimization, mesh, plot)

Friday 30 Oct 15:00-17:00 (GMT+1)

3: How to contribute to pyleecan (Github, OOP)

Friday 6 Nov 15:00-17:00 (GMT+1)

WEBINAR ORGANIZATION

- 1. How to create a slot in pyleecan**
- 2. DXF import**
- 3. How the contribute to the models**
- 4. How to create SciDataTool objects**

OOP REMINDERS

The architecture of PYLEECAN is organized with « **Object Oriented Programming** »

The objective of the architecture is to encapsulate most of the code in « object / interfaces »

Every part of the code is a black box with:

- « properties » (values that defines the object)

- « methods » (function to interact with the object)

A slot object

With *comp_height*
method

A different slot
object

With a different
comp_height method

OOP REMINDERS

The classes are concept

a lamination, a slot, a machine...

They are some classes that are fully conceptual and can't be used:

In pyleecan **Machine** class is **Abstract** and can't be used

The class A can be a particular case of the class B: Then A inherit from B, A is the daughter of B

MachineDFIM inherit from **Machine**, **Machine SCIM** is the daughter of **MachineDFIM**

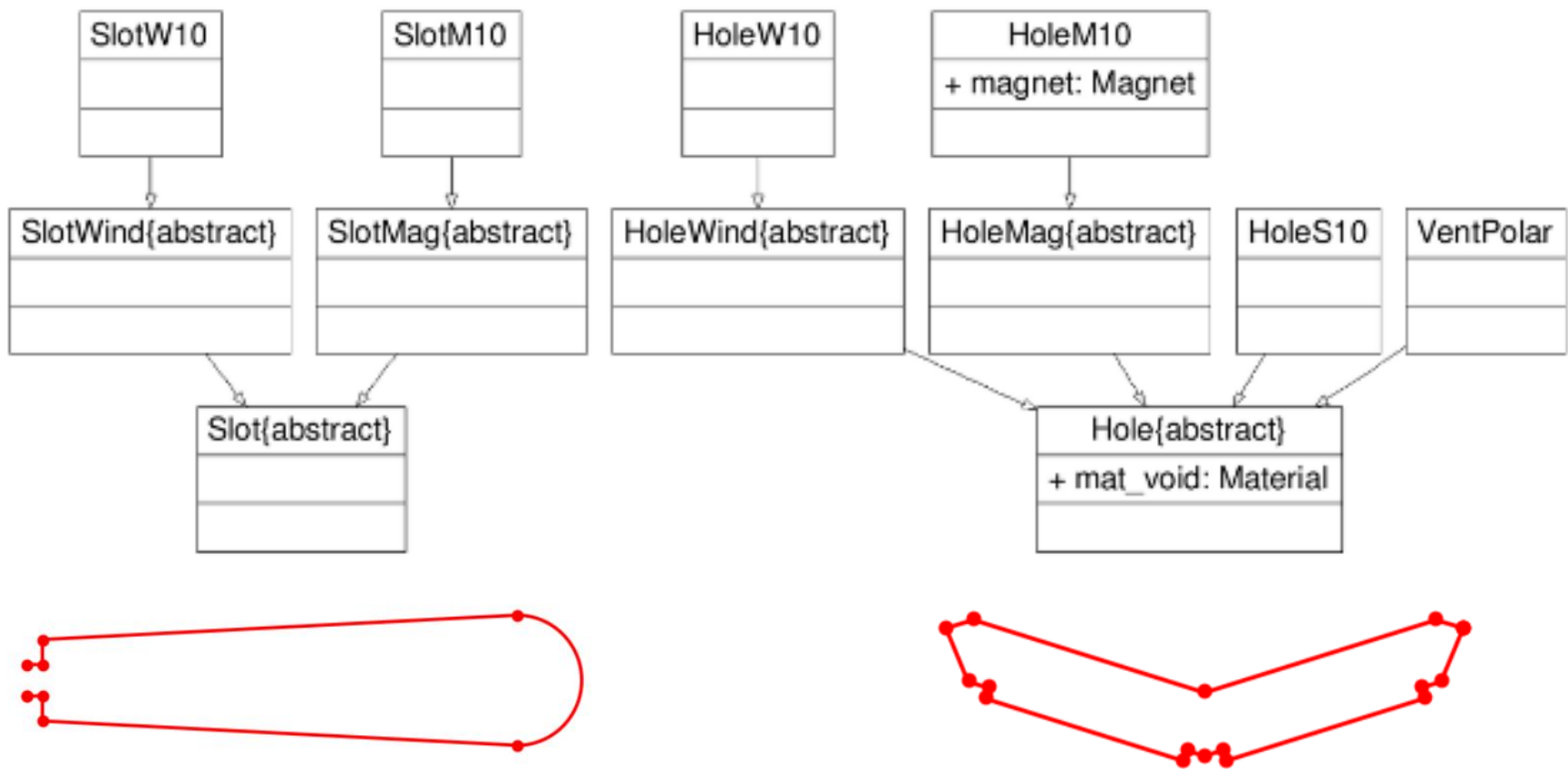
When a daughter has all the properties and method of the mother

plot is a method of **Machine**, then one can call **MachineDFIM.plot()**

A daughter can have an alternative version of a method from the mother

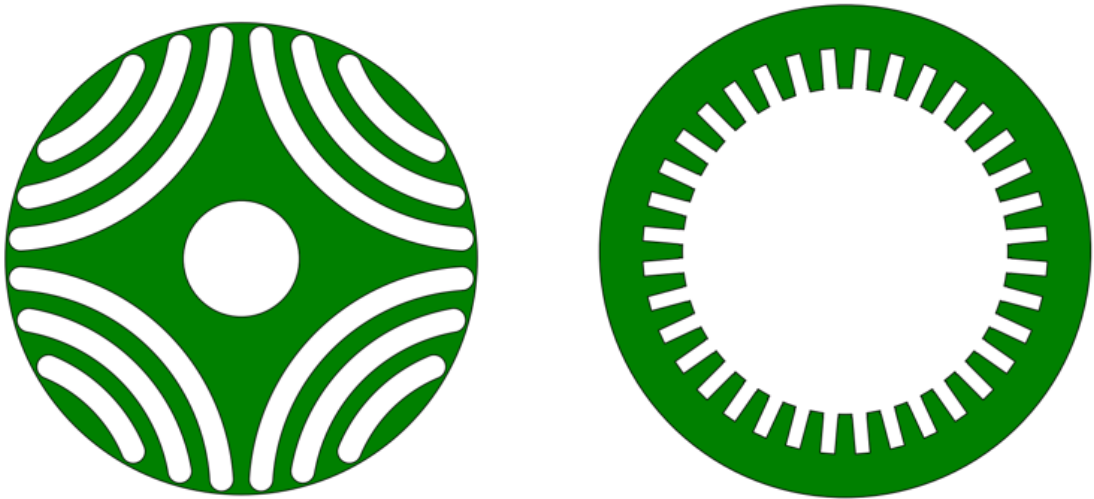
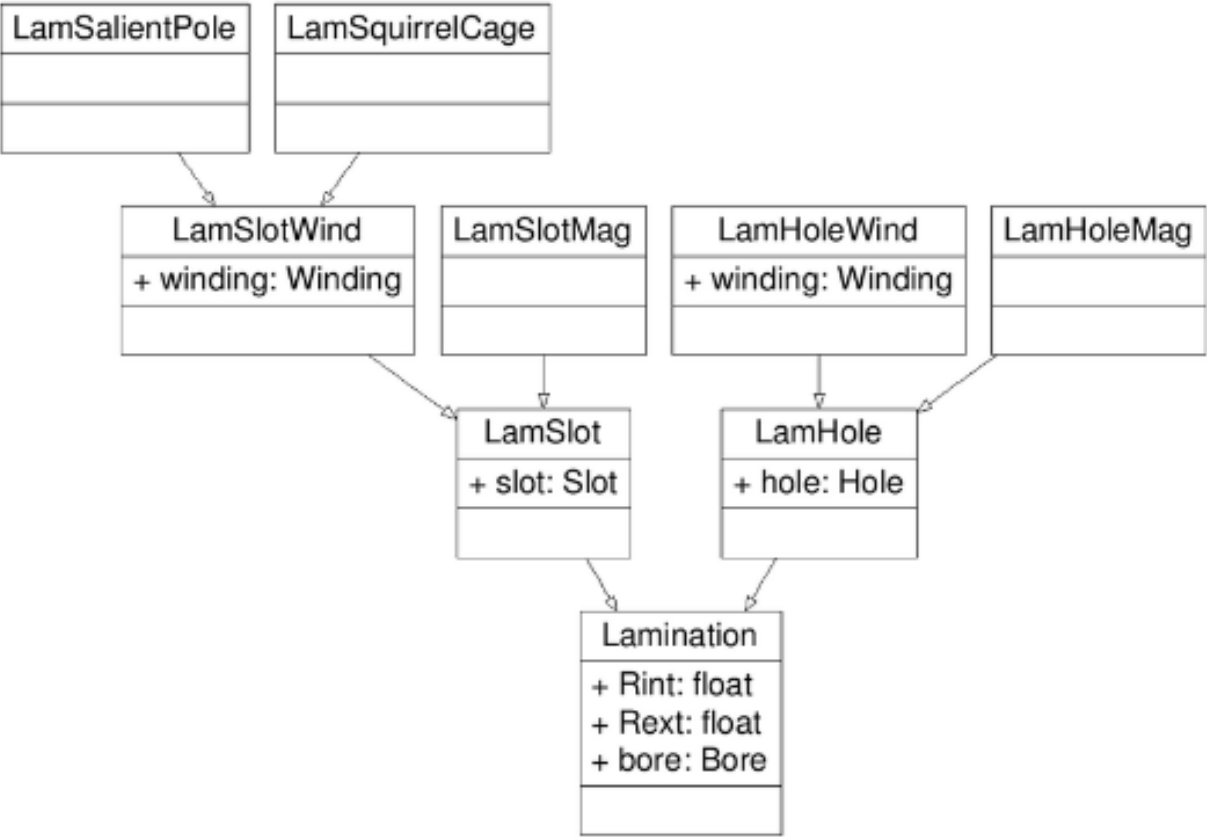
SLOT OBJECT ORGANIZATION

- Abstract Slot and Hole classes provides numerical methods
- Easy to add a new slot with only the build_geometry method



LAMINATION OBJECT ORGANIZATION

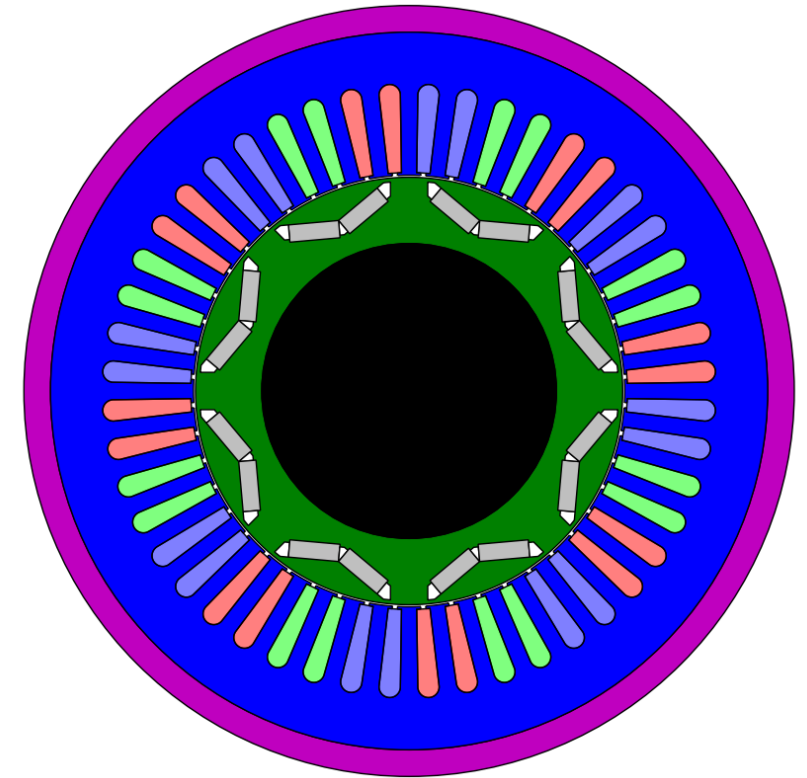
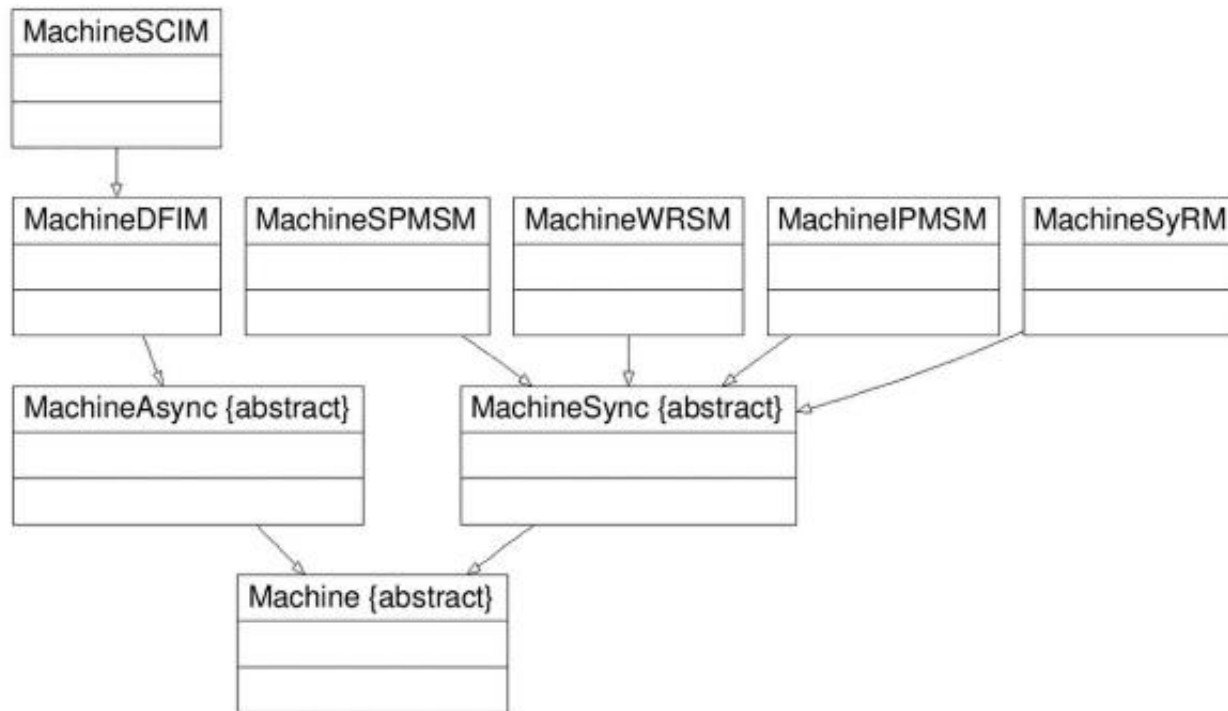
- Lamination split in LamSlot and LamHole according to bore radius shape
- Most machine use a LamSlotWind as a stator



LamHole on the left, LamSlot on the right

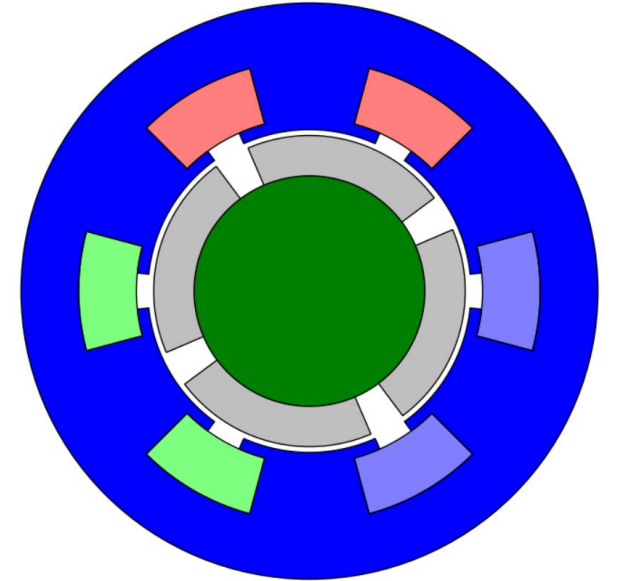
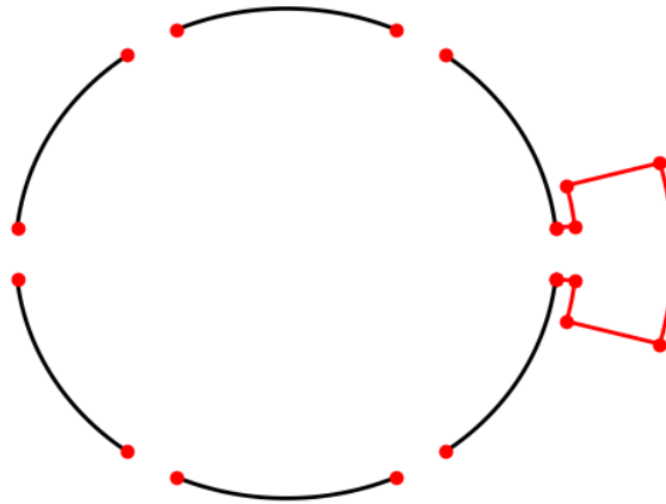
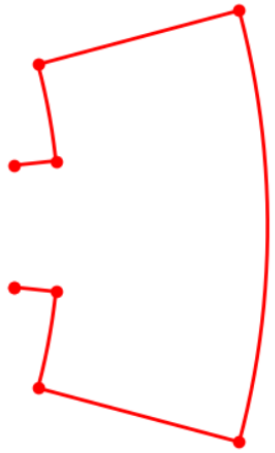
MACHINE OBJECT ORGANIZATION

- The Machine classes detail and gather all the parts of the machine (stator, rotor, shaft, frame...).
- MachineSCIM inherit from MachineDFIM – MachineSCIM is a particular case of MachineDFIM



BUILD_GEOMETRY PRINCIPAL

- Slot build_geometry return a list of Line
- Lamination build_geometry, copy rotate the Slot lines to create surfaces
- Machine gather the surfaces of each entity



Slot build_geometry

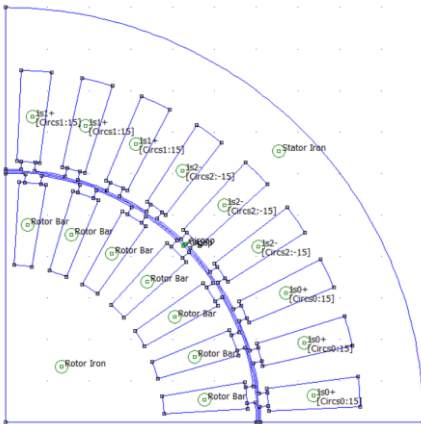
Lamination build_geometry

Machine build_geometry

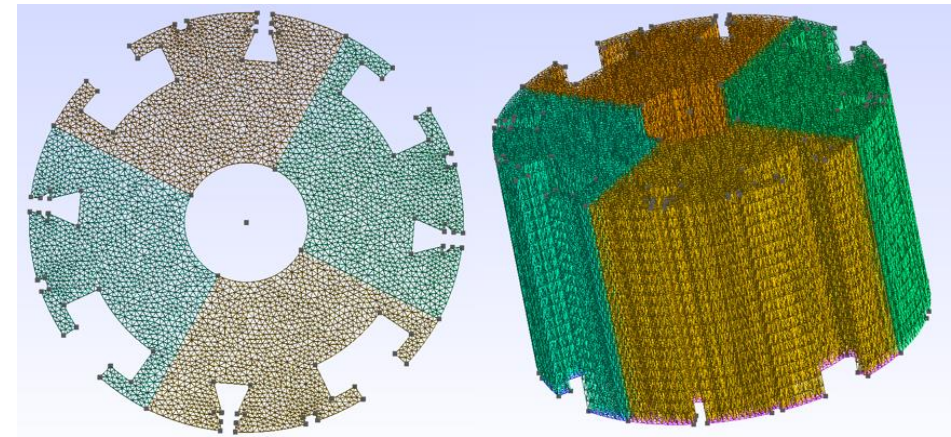
GENERIC COUPLING WITH FEA SOFTWARE

- Each Surface contains a list of line, a label and a reference point
- All the geometry complexity is handled in build_geometry, the coupling just need to draw the surfaces
- The same surface object are used for several coupling

```
for surface in machine.build_geometry():
    for line in surface.get_lines():
        line.draw_FEMM()
assign_surface(surface.ref_point, surface.label)
```



Electromagnetic module with FEMM coupling



3D mesh generation with gmsb as a first step for structural module

CLASS GENERATOR

What ?

Automatic generation of all the code of all the classes of Pyleecan

Why ?

Around 200 classes in pyleecan => no one want to maintain 200 classes files

How ?

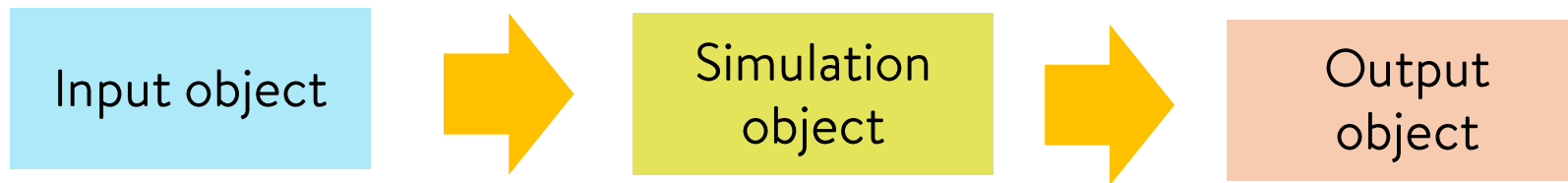
- All the classes are defined as .csv files
- All the methods are defined in a separate folder
- All the classes are defined on a generic template

Other benefits:

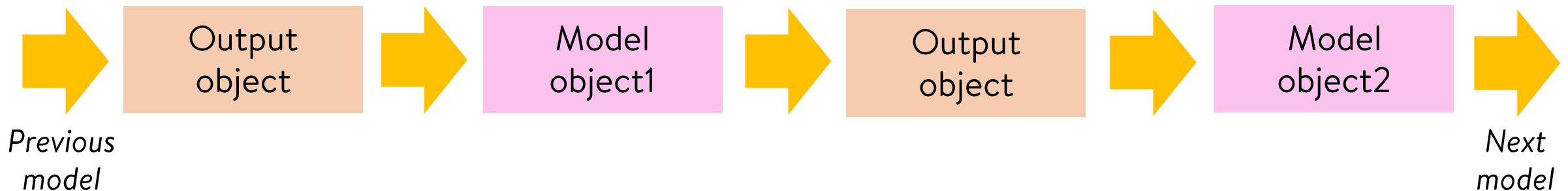
- The documentation is mandatory
- The class generator enable to add new features in all the classes
- We can automatically check if everything is Ok in the class

HOW TO CONTRIBUTE TO PYLEECAN MODELS

- In pyleecan everything is an object: topologies, simulation, input/output, model, post-processing...
- A pyleecan simulation starts from an input object and returns an output object



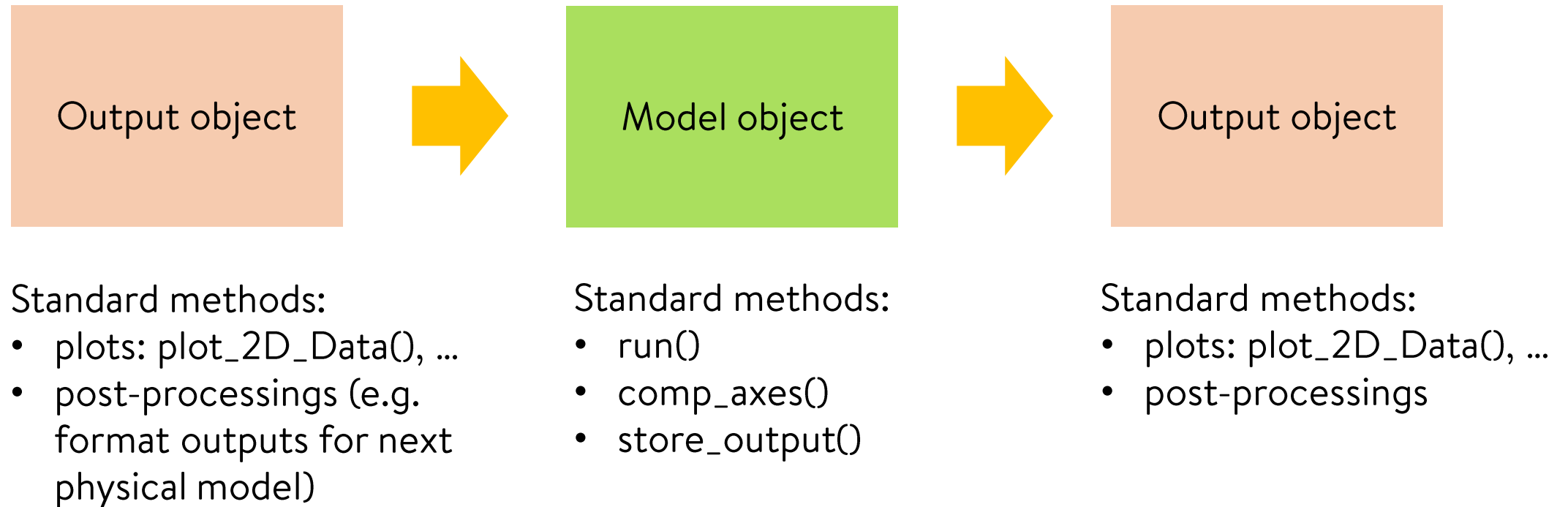
- A simulation object runs one or several physical model objects:



- A model object always takes and returns an Output object

HOW TO CONTRIBUTE TO PYLEECAN MODELS

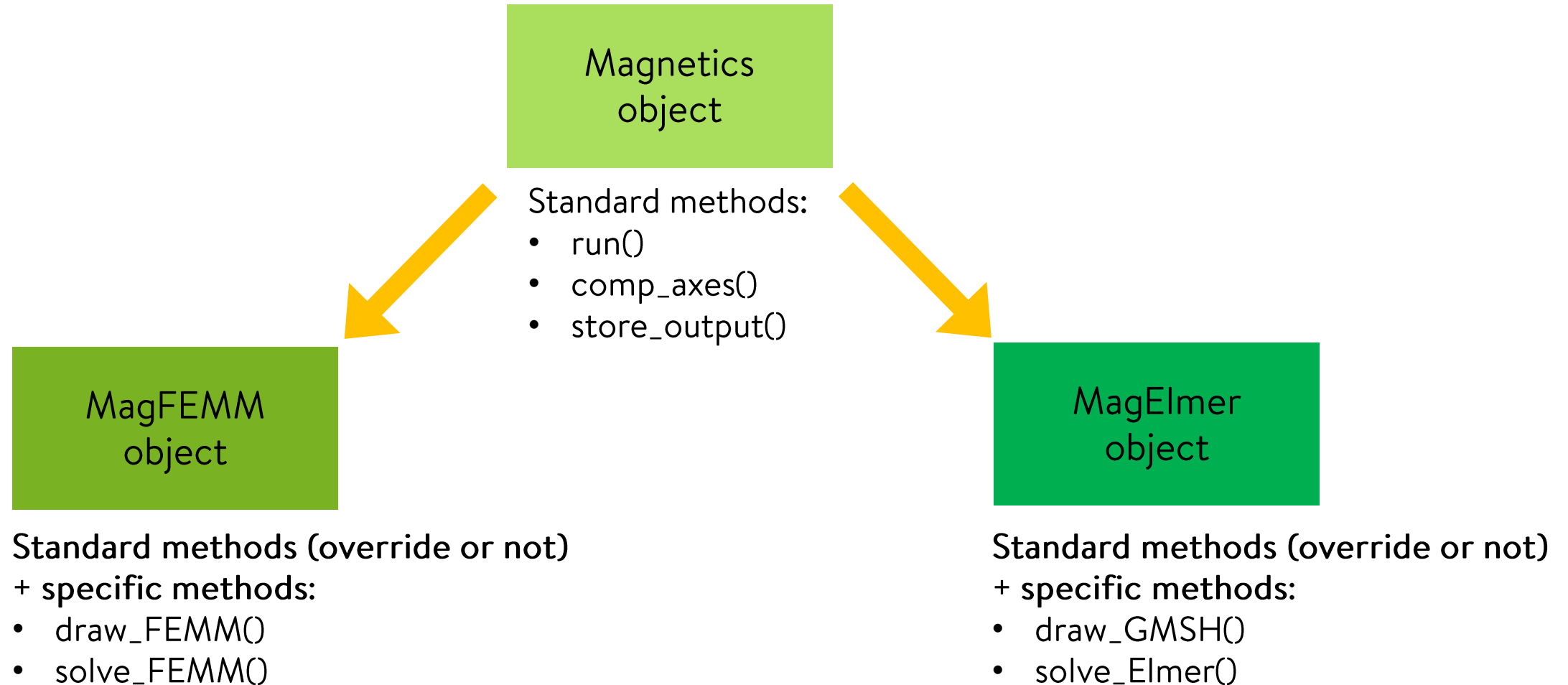
- The input object generates the output object required for the first model object



- The model object generates the output object required for the next model object

HOW TO CONTRIBUTE TO PYLEECAN MODELS

- All physical modules have a standard model object whose all models inherits from:



OBJECTIVE

When implementing a new model in PYLEECAN -> need to use computation axes + store output data

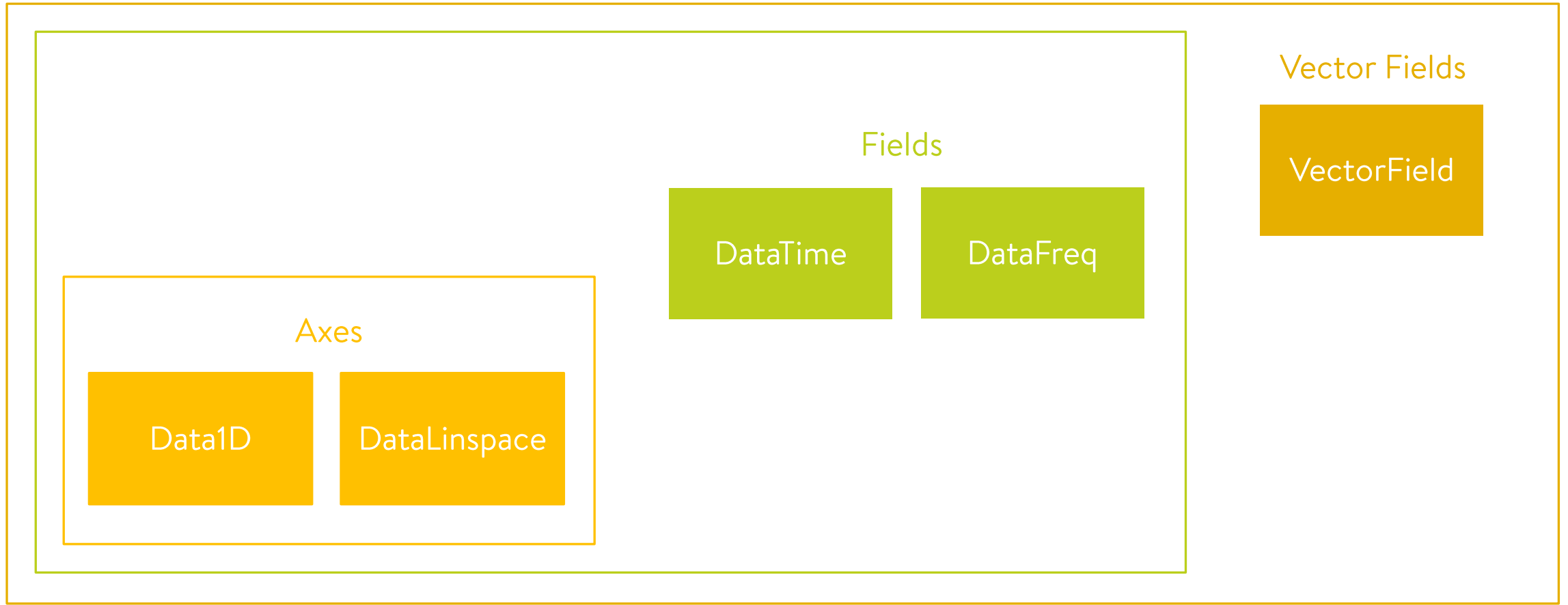
- > understand the structure of SciDataTool objects

- > be able to correctly create them

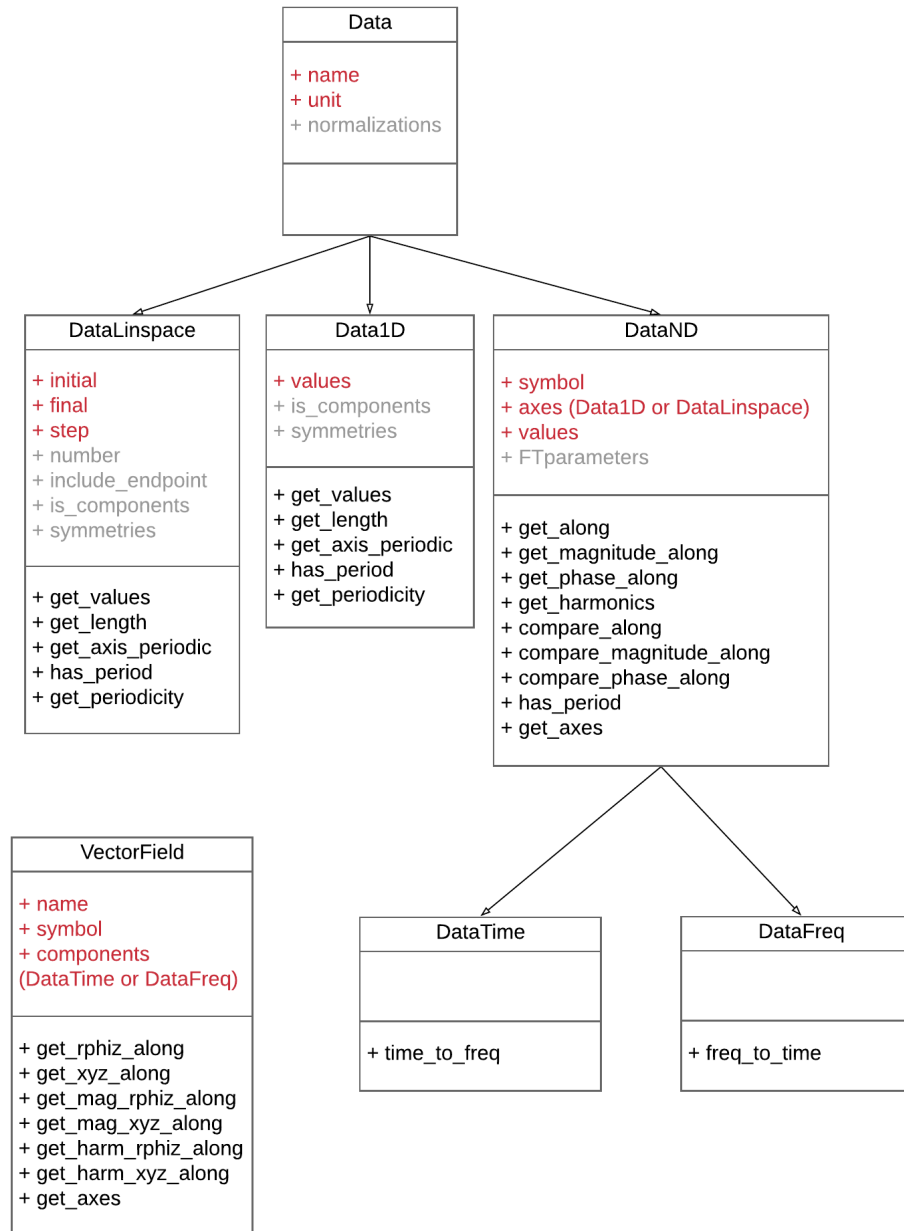
- > use their full potential

SCIDATATOOL OBJECTS

SciDataTool has 5 main objects to store scientific data:



SCIDATATOOL OBJECTS



HOW TO CREATE SCIDATATOOL AXES

```
Time = Data1D(
    name="time",
    unit="s",
    values=time,
    symmetries={"antiperiod": 8},
    normalizations={"elec_order": 8, "mech_order": 2, "angle_rotor": 0.3}
)
```

```
Angle = DataLinspace(
    name="angle",
    unit="rad",
    initial=0,
    final=2*np.pi/12,
    number=2016/12,
    include_endpoint=False,
    symmetries={"period": 12},
    normalizations={"space_order": 12, "distance": 0.4}
)
```



```
Angle_full = DataLinspace(
    name="angle",
    unit="rad",
    initial=0,
    final=2*np.pi,
    number=2016,
    include_endpoint=False,
    normalizations={"space_order": 12, "distance": 0.4}
)

Angle = Angle_full.get_axis_periodic(12, is_antiperiod=False)
```

```
Phase = Data1D(
    name="phase",
    unit="",
    values=gen_name(qs),
    is_components=True,
)
```

AXES IN PYLEECAN

In PYLEECAN, computation axes are automatically computed given:

- the discretizations
- the machine topology (periodicities are detected -> can be used if discretization divisible by number of periods + periodicities activated in model)
- the rotation speed

-> see `comp_axes` method in each module

HOW TO CREATE SCIDATATOOL FIELDS

-> see `store_output` method

```
Is = DataFreq(
    name="Stator current",
    unit="A",
    symbol="Is",
    axes=[Phase, Freqs],
    values=transpose(Is),
)
```

```
output.mag.B = VectorField(
    name="Airgap flux density",
    symbol="B",
)
# Radial flux component
if "Br" in out_dict:
    output.mag.B.components["radial"] = DateTime(
        name="Airgap radial flux density",
        unit="T",
        symbol="B_r",
        axes=axis_list,
        values=out_dict.pop("Br"),
    )
# Tangential flux component
if "Bt" in out_dict:
    output.mag.B.components["tangential"] = DateTime(
        name="Airgap tangential flux density",
        unit="T",
        symbol="B_t",
        axes=axis_list,
        values=out_dict.pop("Bt"),
    )
```

EXTRACT IN PYLEECAN


A field can be defined on various axes:

- computation axes
- axes from a previous module
- axes from an import

-> need to **interpolate** field on computation axes

```
Brphiz = output.mag.B.get_rphiz_along(  
    "time=axis_data",  
    "angle=axis_data",  
    axis_data={"time": time, "angle": angle},  
)  
Br = Brphiz["radial"]  
Bt = Brphiz["tangential"]  
Bz = Brphiz["axial"]
```

CONCLUSIONS

- Join our development effort on  <https://github.com/Eomys/pyleecan>
- Webinar videos will be made available at <https://pyleecan.org/media.html#>
- Subscribe to our newsletter on <https://pyleecan.org/>

