CFS-Engineering Internship Report

Exploration of optimisation and machine learning methods in the fields of aircraft design

Student:
Vivien Riolo

Supervisors:
Aidan Jungo
Jan B. Vos

September 23, 2020

Abstract

Multi-disciplinary Analysis and Optimisation is a commonly used method in conceptual aircraft design. Combined with the power of computers an efficient tool can be developed to generate optimised or new designs for aircraft. The following report describes the implementation of two numerical tools within a conceptual design environment for the optimisation of an aircraft design and the use of surrogate models to approximate costly computations.

Contents

Nomenclature

Symbols

$f, \mathscr{F}$	Objective function
x	Design space variable
Ω	Design space
g	constraint function
C_l	Lift coefficient
C_d	Drag coefficient

Abbreviations

AGILE	Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts
CEASIOMpy	Computerised Environment for Aircraft Synthesis and Integrated Optimisation Methods in python
COBYLA	Constrained Optimisation by Linear Approximation
CPACS	Common Parametric Aircraft Configuration Schema
DLR	Deutsches Zentrum fuer Luft- und Raumfahrt/German Aerospace Center
MDA	Multi-discipplinary Analysis
MDAO	Multi-disciplinary Analysis and Optimisation
MDF	Multi-disciplinary Feasible
MDO	Multi-disciplinary Optimisation
SMT	Surrogate Modelling Toolbox
SQP	Sequencial Quadratic Programming
SLSQP	Sequencial Least-Square Programming
XDSM	eXtended Design Structure Matrix

1 Introduction

Though planes have been successfully flying around since more than a century for now, the effort to come up with ever increasing machine efficiencies and optimizing every subsystem of the aircraft has never stopped. As progress was made, the methodology in the domain of aircraft design also evolved to take into account an increasing complexity of the design processes and the strong coupling of various fields of engineering.

Nowadays the design of an aircraft and its subsystems is a combination of multiple disciplines, such as structural analysis or aerodynamics. Each discipline shall come up with a design that would best fit the aircraft requirements according to their analysis. However what is an optimal choice in one domain may be a drawback for another characteristic of the aircraft. Thus the modern design of an aircraft is a careful balance between the different disciplines, where the main goal is to reach a global optimum of the airplane requirements instead of focusing on local subsystem optima. Such approach is referred to as a multidisciplinary analysis and optimisation (MDAO) process. These processes, though used very early in aircraft design, are gaining a second wave of popularity within the fourth industrial revolution with the growing computational resources of modern computers, as they enable the solving of complex optimisation problems in a feasible amount of time.

Another domain which has drawn an increasing attention these past years, especially due to the possible applications of its tools in the areas of numerical computation, is machine learning, which allows to predict the output of a model without running costly calculations. This aspect therefore hints to a powerful tool that can be achieved if implemented in an optimisation routine to approximate the output of a computationally greedy discipline.

It is in light of these aspects and within the framework of the AGILE 4.0 project that an optimisation and a surrogate model building tool were implemented in the CEASIOMpy software as a tool to use for the conceptual design phase of an aircraft. The report is organized as follows: section 2 gives a brief refresh of the formulation of an optimisation problem and introduces the concept of MDAO. Sections 3 and 4 respectively describe the Optimisation and SM modules that were developed, and a small introduction on the combination of these tools is given in section 5.

2 Optimisation in MDAO systems

2.1 Optimization problem

The aim of such problems is usually to find the maximum or minimum value of a function, which depends on a set of parameters referred to as design variables and when some of these variables have a dependency between them the problem is said to be constrained. A mathematical formulation of an optimisation problem would be :

$$\begin{array}{cc} \text{Objective function} & \min\limits_{x\in\Omega}(\mathscr{F}(x)) \\ \text{Under constrains} & g(x) = 0 \end{array}$$

Where x represents the vector of design variables, Ω is the design space, $\mathscr{F}$ is the objective function and g the vector of constraint functions. $\mathscr{F}$ can also be a vector, in which case the optimisation problem is referred to as a multi-objective problem.

An important aspect of optimisation is whether or not the derivatives of the problem are known, as multiple search algorithm rely on this information to progress through the design space. Therefore two different families of algorithm exist :

- Gradient-based optimisation : These algorithm use the information of the derivatives between the parameters to conduct their search. Their use is relatively common and they are an efficient way to solve the problem if the objective function can be derived. A lot of algorithms have been developed based on this method [1].

- Gradient-free optimisation : In the case of an objective function which does not allow the derivatives to be computed, the algorithms must search through the design space without notion of the steepest descent, often using the evaluation of the function at some points in the design space to decide in which place to search next [2]. The current optimisation algorithm that was implemented in CEASIOMpy belongs to this class of algorithms.

In the case of an aircraft design, the optimisation problem is usually constrained and the objective functions are often complex non-linear functions, which may not be known. Objective functions that are of interest in the domain are usually part of the aircraft first level requirements [3, 4, 5]. Some recurrent physical quantities that can be cited are the lift-to-drag ratio $\frac{C_l}{C_d}$, the maximal take-off mass *mtom* or the range of the aircraft.

2.2 MDO Architecture

In multi-disciplinary optimisation, the objective function is often the result of one or multiple computations, which can be decomposed into smaller computation steps. One step can represent the outcome of one discipline analysis, and so one basic architecture for a multi-disciplinary system could be represented as seen in figure 1. This kind of representation is referred to as an XDSM diagram, which is a commonly used method of representation in MDO and will be used throughout this report to illustrate a workflow architecture. It is thus recommended to get acquainted with this representation by reading the paper of Lambe et al. [6] who developed this method.

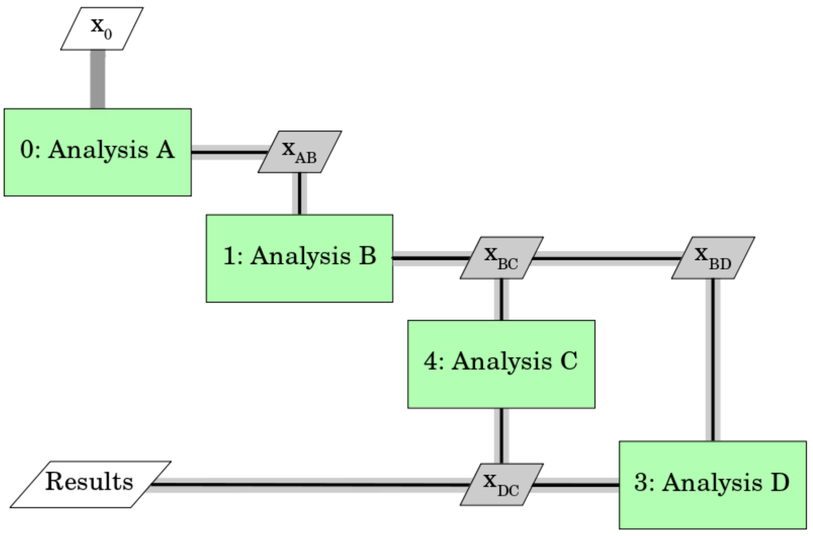


Figure 1: Basic architecture example for a multi-disciplinary analysis (MDA) system. Each analysis block represents a computation and the information flow between the blocks is represented by the gray lines, whereas the workflow connections is shown with the black line. The initial vector of inputs is indicated by the x_0 box and the resulting output vector is the *Results* box. The x_ij boxes show the vectors of variables that are passed from module i to module j.

Such a workflow can then be coupled to a solver which takes the results as an input and the first module inputs as design variables, and so a first optimisation routine can be obtained. This basic block can then be included as a component of a larger system, which can combine more of such components in a distributed or monolithic way [7], depending on the coupling of the components. This process allows for a wide range of possible architectures to chose from when setting up a MDO or MDAO problem.

Aircraft design was one of the first field in which MDO was used due to the multiple aspects to take into account during the process, starting with more simple sub-systems of the plane at first, such as wing design [7]. Nowadays a lot more of the aircraft design aspects can be taken into account within multi-level MDO frameworks [8, 9, 10], even taking disciplines which are not related to the aircraft object itself such as the economic aspects [11].

2.3 CEASIOMpy architecture

The CEASIOMpy software consists of multiple modules which run different analysis on a predefined CPACS file [12]. Some module inputs are the outputs of other modules, which creates a dependency between some disciplines of the analysis. The current analysis process is run as a monolithic bloc and no parallelisation is yet implemented. The architecture for the MDAO system was then chosen to remain the same as the MDA system, namely a multi-disciplinary feasible (MDF) architecture [7, 13], which does not take into account the consistency and discipline analysis constraints.

3 Optimisation module in CEASIOMpy

3.1 The OpenMDAO library

A specialised Python library was used to implement the Optimisation module in the software. OpenMDAO [14] is an open-source library that contains the functions to set up and solve an optimisation problem. Two types of routine were implemented using this library :

- Optimisation routine : The program solves an optimisation problem with user-defined parameters (constraints, design variables, objective function) and settings (choice of driver, modules to run, etc). OpenMDAO offers different search algorithms to solve a problem, however in CEASIOMpy the COBYLA algorithm is imposed as it is the only one to deal with gradient-free, constrained optimisation problems [15]. To modify this the user must make internal changes to the code.
- Design of Experiment (DoE) : The program evaluates an objective function at randomly chosen points in a user-defined design space. Several options exist to define the points at which the function shall be evaluated. The Design of Experiment process and implementation is further described in section 4.2.

An optimisation (or DoE) problem can be set up with OpenMDAO using its three basic components :

- Explicit Component : Used in the case of an explicit computation (output a is obtained by direct evaluation of some function $a = f(x)$).
- Implicit Component : Used to define an implicit computation (output a is computed by solving a system $f(x, a) = 0$).
- IndepVar Component : This component defines the variables of the problem whose values will be modified by the driver at each iteration. They represent the vector of design variables of the routine.

A workflow can be generated by defining the inputs and outputs of each problem components and connecting them together. The next step is then to indicate which inputs shall be considered as problem variables by assigning them to an IndepVar component and after choosing a driver (the algorithm to conduct the search through the design space) the routine can be launched. Figure 2 illustrates the final setup of the problem.

OpenMDAO also comes with options for recording the data and visualizing the problem architecture. Both of these features were implemented in the software and their output files are described in the outputs section of Table 1.

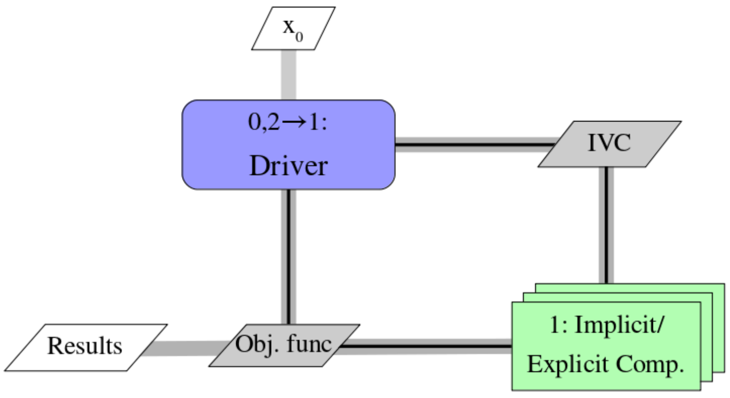


Figure 2: Example of a basic optimisation workflow with multiple components. The driver gives the inputs to the IndepVar component (IVC) which forwards them to the other component, Implicit or Explicit. The objective function is an output of the last component and is fed to the driver in order to move to the next iteration of the search algorithm.

3.2 Module implementation

3.2.1 Routine setup

CEASIOMpy offers the possibility to run different analysis modules on an aircraft configuration, which is stored in a CPACS file. A workflow can be created to run different modules one after the other to make a complete analysis of the aircraft. An optimisation or DoE is implemented as follows :

Modules: First the optimisation module builds a problem model by creating an Explicit component for each CEASIOMpy module that will be used within the workflow and defining its inputs and outputs based on those found in the specification file of the module.

Geometry: Another Explicit component is created to enable the modifications of geometrical variables of the aircraft, which are not included in a CEASIOMpy module.

Objective: Here the inputs are the variables that are used to compute the objective function. The output is the result of the computation. For now only one objective function can be handled by the program.

Once these components are created, their inputs and outputs can be defined as parameters of the problem. The input and output list can be found in Table 1 and figure 3 illustrates the routine workflow with all the components.

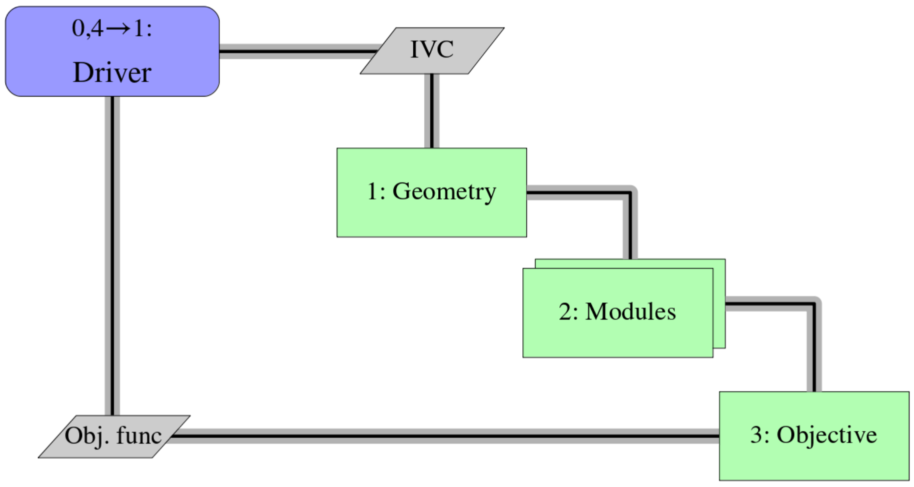


Figure 3: Architecture of a standard workflow within a routine.

3.2.2 Parameters

The choice of the parameters which can be set as design variables, constrains or components of the objective function is wide and varies with the list of modules that is used in the routine as not all modules have the same inputs and outputs.

Geometrical parameters : The first parameters to be integrated as design variables and that can be constrained in all cases are the geometric parameters of the aircraft, which can be called within any workflow analysis as they do not depend of any CEASIOMpy module. These can be handled with the functions of the TIGL library, but caution has to be taken as to the set of parameters to choose within one optimisation routine, as some variables may be geometrically related to each other (e.g the wing span and wing aspect ratio) and so, unless specified explicitly in the problem setup, it will cause major issues during the optimisation process.

Module parameters : For each module, the input parameters can be set as design variables, and the output parameters can be defined as components of the objective function, or they can be set as a constraint. This process is done automatically by reading the spec file of each module, which contain the information on the module inputs and outputs.

Aeromap parameters : For any CFD analysis (SU2 or PyTornado in CEASIOMpy), a set of special parameters must be given in order to characterize the flow around the aircraft. These are summed up in an aeromap, which contains 4 input parameters that may also serve as design variables:

- α : angle of attack.
- Ma : Mach number.
- h : Altitude.

- β : Side-slip angle.

And 6 output parameters will be generated which can be used as a constraint or as part of an objective function:

- C_l, C_d, C_s : Respectively the Lift, Drag, and Side-force coefficient.
- Cm_l, Cm_d, Cm_s : Moment coefficients.

For an optimisation routine, only the first entry of the aeromap parameters is taken into account, however in the case of a DoE, an entire aeromap can be taken to add multiple data to the sample within one single iteration. This option can be specified by enabling the "Use whole aeromap" option in the settings GUI.

3.3 Program implementation

3.3.1 Classes

The module uses 4 different classes that inherit from the ExplicitComponent class of OpenMDAO, whose role are described in 3.2.1. The fourth class (SmComp) is called in when a surrogate model is used in the routine, as the parameters that the model takes as inputs may not be the same than those that were selected for the routine and thus need to be specified.

3.3.2 Functions

The code was written using the Python 3 language. Figure 4 shows the program architecture for the optimisation module. For the sake of readability only the most important function calls are represented. The routine_launcher function is the main function of the program, which will call all the other functions that will construct the problem, solve it and process the data.

INPUTS	
Entry	**Description**
Objective function	Expression of the objective function, which can be a combination of multiple parameters.
Aeromap choice	Aeromap to be used within the routine if there are some aerodynamic modules included.
Optimisation goal	(OPTIM) Choosing between a minimisation or a maximisation problem.
Max iteration nb	(OPTIM)Criterion to stop the routine after a number of iteration.
Tolerance	(OPTIM) Criterion to stop the routine based on the convergence rate.
DoE Driver	(DOE) Choice of the driver for the DoE.
Number of samples	(DOE) Parameter to compute the number of samples for the DoE. This value is further described in the DoE section (see 4.2).
Use whole aeromap	(DOE) Indicate if an entire aeromap has to be saved at each iteration. This can be useful when one wants more than one aeromap point to be evaluated.
Saving geometry	(OPTIONAL) Save the CPACS file every n iteration.
Variable library	(OPTIONAL) CSV file containing all the parameters to setup the problem.
OUTPUTS	
Entry	**Description**
circuit.html	Visualization of the problem setup.
circuit.sqlite	Used to generate the N2 file.
Driver_recorder.sql	Contains all the information of the routine.
Variable_library.csv	Contains the initial information of all problem parameters.
Variable_history.csv	Contains the values of all problem parameters at each iterations.

Table 1: Input-Output summary of the optimisation module. An indication of when this entry is of use is given between brackets. The inputs will all be found in the GUI when launching a workflow with a routine.

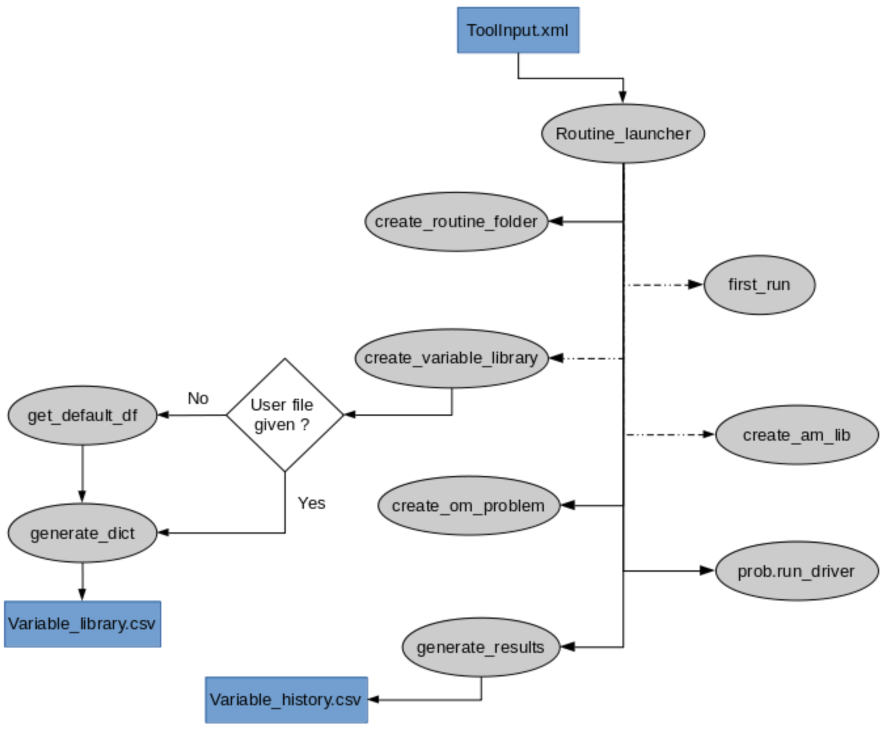


Figure 4: FFBD of the Optimisation module. The gray ovals represent a function of the program, and the blue boxes represent a file. The arrows indicate the direction of the information flow, and the dotted lines indicate that the function is called from another module.

Further technical information and details are available within the description of the code functions and classes. The standard structure for function description in CEASIOMpy is given as such :

```
"""Brief description of the function.

More detailed description.
An example may be added.

Args:
    List of inputs of the function.

Returns:
    List of outputs of the function.

"""
```

3.4 Program workflow

The optimisation module is called as soon as the modules are chosen in the optimisation routine. It is therefore not necessary to add it to the module list when choosing the modules to run in a workflow. The module parameters, listed in table 1, are then chosen in the SettingsGUI module.

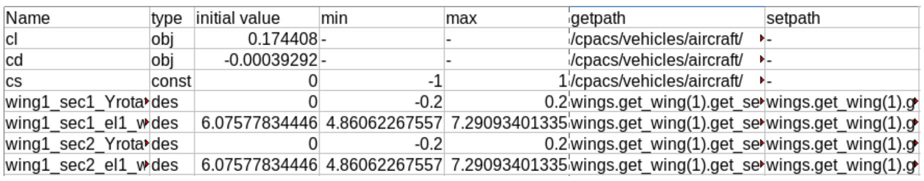
The CSV file that is either given by the user or generated by the program is presented in figure 5. It contains a set of parameters that will be passed on to the OpenMDAO library to setup the problem. The following rules apply when writing a parameter to the file :

- **Name:** Each parameter must have a different name. Usually this name corresponds to a module input or output that is specified in the *spec* file of said module.
- **type:** Three types can be assigned to a parameter : *obj*, *const* or *des*.

  obj This type marks the parameters as being one that is used to compute the objective function, if he appears in the objective function he must be added as such in the CSV file.

  const If a parameter has to be set as a constrain, this keyword is to be used. In this version of the optimisation module the only constraints available are fixed values.

  des For the parameters whose value will be changed by the driver this keyword must be indicated.
- **initial value:** The initial value that a parameter must have or has, based on the CPACS file.
- **min/max:** These are optional parameters to add for *des* or *const* parameters. They ensure a lower and/or upper boundary along the dimension of the parameter.
- **getpath/setpath:** The last two entries ensure for the correct parameter modification in the CPACS file. If the parameter can be changed via the Tixi library, then only an Xpath pointing to its location in the CPACS will be given and the setcommand will be left empty. Else, if the Tigl library is used to modify a parameter (usually related to aircraft geometry), the getpath will containt the Tigl command to compute the value of the parameter and the setpath will contain the command to modify that value.



Name	type	initial value	min	max	getpath	setpath
cl	obj	0.174408	-	-	/cpacs/vehicles/aircraft/	-
cd	obj	-0.00039292	-	-	/cpacs/vehicles/aircraft/	-
cs	const	0	-1	1	/cpacs/vehicles/aircraft/	-
wing1_sec1_Yrota	des	0	-0.2	0.2	wings.get_wing(1).get_se	wings.get_wing(1).g
wing1_sec1_el1_w	des	6.07577834446	4.86062267557	7.29093401335	wings.get_wing(1).get_se	wings.get_wing(1).g
wing1_sec2_Yrota	des	0	-0.2	0.2	wings.get_wing(1).get_se	wings.get_wing(1).g
wing1_sec2_el1_w	des	6.07577834446	4.86062267557	7.29093401335	wings.get_wing(1).get_se	wings.get_wing(1).g

Figure 5: Example of a CSV file containing the parameters for the optimisation module.

This file will be saved and a dictionary will be created by the program, which will serve as the collector for all new data that will be generated (the values for each parameter at each iteration). Once the routine has been completed, the dictionary will be saved as a CSV file called "Variable history", under the corresponding working directory folder. An N2 diagram called "circuit.html" (see 8.1) will also be generated, along with two other files ("circuit.sqlite" and "Driver_recorder.sql") that are generated by the OpenMDAO library. The .sqlite file is needed to generate the N2 diagram and the .sql file stores a history of the routine, which is used to read the results of the objective function.

A number of test cases can be found in the CEASIOMpy documentation [16]. They illustrate the various ways in which the software can be used to run a routine, use any of the CEASIOMpy modules and give indications on the different methods to setup and run a workflow, wether it is done via the GUI or the RCU software.

4 Surrogate Models in CEASIOMpy

4.1 Machine learning in Aircraft design

Machine learning is a sub-domain of artificial intelligence that focuses on the development of algorithms that can improve through the feeding of data. With the ever-increasing amount of data at disposal and the higher complexity of today's models for various system, such methods that can make a prediction for a given problem without directly solving a large system appear as interesting candidates to be used in industry software.

Although many methods of this domain are already being used in various sub-domains of aircraft design, one of the most commonly used machine learning technique in the domain of aircraft design appears to be surrogate models [17] . One approach that can be adopted is to make a first search using a surrogate model, which can be used to extrapolate a result based on a set of predefined data for which the output is already known. In the domain of CFD, such model could become particularly handy in order to guess an initial solution to a problem based on previous simulations results and so converging to a solution faster than without initial guess.

The implementation of surrogate models in CEASIOMpy is done by using the Surrogate Modelling Toolbox (SMT) which is an open-source Python package consisting of libraries of surrogate modeling methods [18].

4.2 Design of Experiment

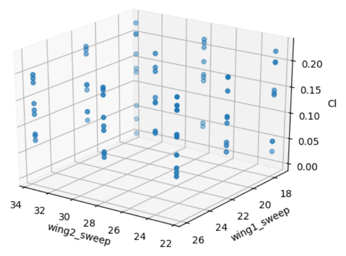
In order to generate the data that will be needed to train a surrogate model, a Design of Experiment must be conducted. The OpenMDAO library possesses the driver that can run such procedure. This procedure consists of running a specified workflow for a given number of time, by changing the design variables of the problem according to a sampling method. This procedure can be used to produce a first mapping of the objective function in the defined space of variable and so identify the parameters that have a significant influence on the objective function and those which are not relevant.

An example of DoE is made using only the vertical wings span and width of an aircraft as design variables, using the uniform distribution of experiment points. Figures 6a and 6b show the scattering of the sample points in two different variable space.

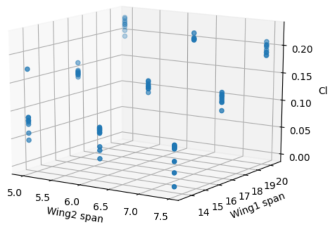
In the case of a DoE, the number of points to be evaluated within the design space can be generated in different ways. This can come in handy, for instance, when certain parts of the space are more of relevant than others, or if contrariwise points should randomly be taken in a uniform way over the whole space. In CEASIOMpy the sample of design points can be generated by using one of the following methods :

- FullFactorial : The full factorial method evaluates a combination of the X specified parameters for l number of different values of each variable, resulting in a total of l^X data points that will be generated. The level l of different values to be taken has to be chosen carefully, and this method is not recommended in the case of many parameters.

- Plackett-Burman : This method belongs to the fractional-factorial family,



(a) Sample points in the wing sweep space. It is observed that the sweep of each wing does not influence much the objective function, as for different combination of the two sweep values, the same magnitude of Cl is reached.



(b) Sample points in the wing span space. It is clearly seen that the span of the main wing influences Cl the most, and a small dependency can also be noticed for the span of the second wing.

its process to generate points is the same than the full-factorial one. However with this method not all combinations are made and thus less data is generated.

- Uniform : This method generates a number of sample points using a uniform distribution method for the variable values.
- LatinHypercube : This method takes random points in the design space by taking into account the location of the previously generated points, so the final sample of points is representative of the whole design space.
- CSV file : By giving a CSV file with predefined values, a DoE with an arbitrary set of values can be conducted.

4.3 SMTrain module

By giving a training data-set containing inputs and their corresponding outputs, a surrogate model can be trained and used to predict an output value for new input points. A surrogate model can be considered as a black box as the user can specify which parameters shall be taken as inputs and outputs. Thus one can create a model based on a workflow by including all desired modules. The methods that were implemented in CEASIOMpy are different forms of the Krigin model and a least-square approximation. These methods and others are described in detail at [19].

To work, the SMTrain module in CEASIOMpy only requires a csv file with a list of inputs and outputs, that will serve as the model data set for training and optional validation. In the case of parameters which can all be found in an aeromap the csv file does not need to be specified, the other entries of the module are summed up in table 2. The module generates a model which is saved as a binary file in the current working directory, along with the information of the inputs and outputs that the model takes.

INPUTS	
Entry	**Description**
Training Data-set	(DUAL) CSV or Aeromap containing the data to be used to train the surrogate model.
Data partition value	Select the percentage $\in [0,1]$ of data to be used to train the model, the other part serving for validation.
Model type	Choice of the Surrogate to create.
Use of Aeromap only	(OPTIONAL) Indicate if the model that is trained will only takes and predicts aeromap entries.
Name of Aeromap	(DUAL) Name of the aeromap whose entries shall be used to train model.

OUTPUTS	
Entry	**Description**
SM file	Binary file that contains the surrogate model and its input/output information

Table 2: Inputs and Outputs of the SMTrain module.

The ease of use of the toolbox however can become more complex when a closer look is taken at the hyper-parameters of the different models, i.e the coefficient that appear in the mathematical formulae of the model. For now the GUI only allows the user to choose from a list of different model, but the fine-tuning of their different parameters still remains within the code of the SMTrain module. The functional scheme of the module is represented in figure 7.

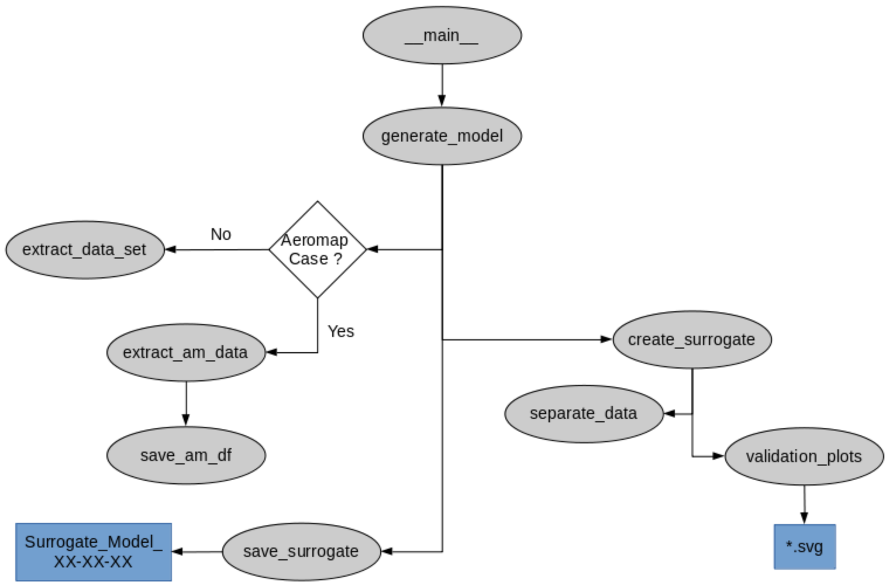


Figure 7: FFBD diagram of the SMTrain module

4.4 SMUse module

After a surrogate model has been generated, it can be used with the SMUse module to be run within a workflow. The particularity of this module comes from the entries that the surrogate will take, which will change depending on the model that is used. It is necessary to make a distinction here between the inputs and outputs of the SMUse module (tab. 3) and the inputs and outputs of the surrogate model, which is used by the module. This aspect is important to take into account as it implies that this module is a special case if one wants to create an automatic process for the modules using their inputs and outputs. Here they can be found in the file that has been saved during the creation of the model.

A practical case of adaptation of this module can be found within the optimisation module, where the inputs and outputs of each module appearing in the workflow are automatically added as variables. In this case a special class was created to deal with the specific inputs and outputs of any surrogate model as explained in section 3.3.1. The functional scheme of the module is represented in figure 8.

INPUTS	
Entry	**Description**
CPACS file	(DUAL) XML file from which the model inputs will be retrieved.
SM file	Binary file that contains the surrogate model and its input/output information.
Use of Aeromap	(OPTIONAL) Indicate if the model that is used only takes and predicts aeromap entries.
Name of Aeromap	(DUAL) Name of the aeromap whose entries shall be used by the model.

OUTPUTS	
Entry	**Description**
CPACS file	XML file in which the outputs of the model will be saved.

Table 3: Inputs and Outputs of the SMUse module.

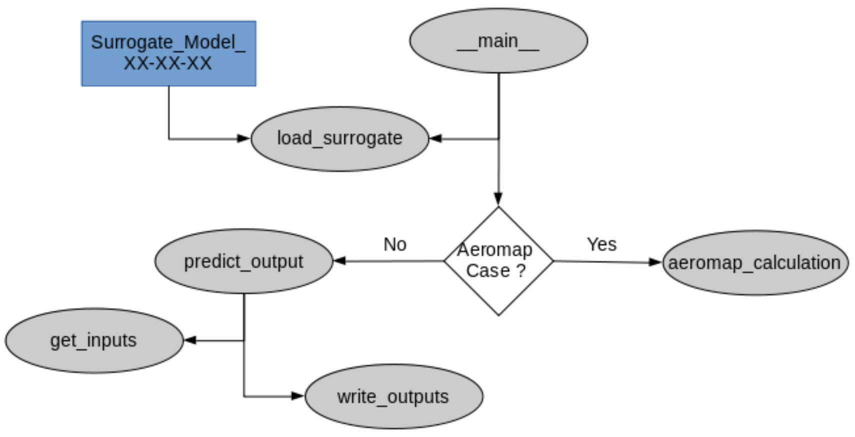


Figure 8: FFBD diagram of the SMUse module

5 Combination of Optimisation and surrogate model

5.1 Computational time gain

The main advantage of using a surrogate model within an optimisation process lies in the drastic reduction of computational resources needed to run the process. However this gain in time comes with a drawback in the fidelity of the solution as the surrogate model guesses a solution based on its experience. This must be taken into account and a validation of the final solution of an optimisation routine using a model should always be verified by a higher fidelity CFD analysis.

A simple optimisation routine using a surrogate model take less than 10 seconds to be completed on one processor only, whereas a routine with a SU2 computation that must be converged at every iteration can take up to a few days. The main difference lies in the fidelity of the results, where the surrogate has only made a prediction, the SU2 computation has given a result that is trustworthy up to a certain level.

5.2 Foreseeable usage

An iterative process can be developed to obtain an optimisation routine that would give a satisfying solution in terms of fidelity and use the surrogates to reduce the computational cost. The global scheme would be to use the surrogate model and guess a first optimal solution based on the available data. This solution would then be validated with a medium or high-fidelity CFD software. Based on the CFD analysis the new data could be either fed towards an optimisation cycle that would try to find the local optimum around the obtained solution, or it could be added to the previous set of points to train a new surrogate model and repeat the previous steps [20].

6 Conclusion

In the framework of the Agile 4.0 project, two functionalities were added to the CEASIOMpy software. First, an optimisation module which can solve non-linear constrained problems or run a Design of Experiment has been implemented. The second addition consists of two tools, one that creates a surrogate model and another one that uses a surrogate to approximate the outputs of a module or a combination of modules. The combination of both tools aims to provide a rapid and efficient mean of optimisation.

Future work which could be foresighted are the implementation of a more advanced GUI that could allow the user to tune the optimisation and model generation on a lower level, e. g. by adding specific parameters to the driver or the surrogate model or implementing new search algorithms for the optimisation routine, such as genetic algorithms which may be used for multi-objective optimisation.

A more complex and consequent task would be the implementation of the components on a lower level of the code, i.e defining each variable in a module as a component and specify the relation between each of them. Such improvement would allow for more robust and efficient problem setting and solving. Plus the derivatives could be implemented and could then lead to the use of gradient-based methods for the optimisation such as SLSQP. However this would imply to rewrite entire pans of code which may result as being more time-consuming than using the present implementation.

7 Acknowledgement

The work presented in this report has been performed in the framework of the AGILE 4.0 project (TowardsCyber-physical Collaborative Aircraft Development) and has received funding from the European Union Horizon 2020 Programme under grant agreement nr°815122.

References

[1] M. Bierlaire, *Optimisation: Principles and Algorithms.* Presses Polytechniques Universitaires Romandes, 2015.

[2] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization.* MOS-SIAM Series on Optimization, Society for Industrial and Applied Mathematics, 2009.

[3] W. H. Mason, *Configuration Aerodynamics*, ch. 4. AIAA Education Series, -, 6 ed., 2018.

[4] D. P. Raymer, *Aircraft Design : A Conceptual Approach*, ch. 2. -, 2019.

[5] V. Komarov, *Conceptual Aircraft Design.* -, 2011.

[6] A. B. Lambe and J. R. R. A. Martins, "Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes," *Structural and Multidisciplinary Optimization*, vol. 46, pp. 273–284, 2012.

[7] J. R. R. A. Martins and A. B. Lambe, "Multidisciplinary Design Optimization: A Survey of Architectures," *AIAA Journal*, vol. 51, no. 9, pp. 2049–2075, 2013.

[8] Pat Piperni, "MDO for Aircraft Design at Bombardier Aerospace." Symposium on Collaboration in Aircraft Design, 10 2015.

[9] A. Sgueglia, P. Schmollgruber, N. Bartoli, E. Benard, J. Morlier, J. Jasa, J. R. R. A. Martins, J. T. Hwang, and J. S. Gray, "Multidisciplinary Design Optimization Framework with Coupled Derivative Computation for Hybrid Aircraft," *Journal of Aircraft*, vol. 0, no. 0, pp. 1–15, 0.

[10] J. T. Hwang and A. Ning, "Large-scale Multidisciplinary Optimization of an Electric Aircraft for On-demand Mobility," *AIAA*, jan 2018.

[11] S. Roy, W. A. Crossley, K. T. Moore, J. S. Gray, and J. R. R. A. Martins, "Monolithic Approach for Next-Generation Aircraft Design Considering Airline Operations and Economics," *Journal Aircraft*, vol. 56, July 2019.

[12] DLR, "Common Parametric Aircraft Configuration Schema."

[13] R. M. Lewis, G. R. Shubin, E. J. Cramer, J. E. Denis, P. D. Franck, R. Michael, and L. Gregory, "Problem formulation for multidisciplinary optimization," *SIAM Journal on Optimisation*, february 1997.

[14] J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore, and B. A. Naylor, "OpenMDAO: An Open-Source Framework for Multidisciplinary Design, Analysis, and Optimization," *Structural and Multidisciplinary Optimization*, vol. 59, pp. 1075–1104, 2019.

[15] M. J. D. Powell, "A view of Algorithms for Optimization without Derivatives." William Benter Distinguished Lecture, City University of Hong Kong, 2007.

[16] "CEASIOMpy Documentation." `https://ceasiompy.readthedocs.io/en/latest/user_guide/getting_started.html`. Accessed: 2020-08-14.

[17] C. Jacob, J. Bieler, and A. Bardenhagen, "Introducting surrogate models to the structural preliminary aircraft design phase," in *Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V.*, 2018.

[18] M. A. Bouhlel, J. T. Hwang, N. Bartoli, R. Lafage, J. Morlier, and J. R. R. A. Martins, "A python surrogate modeling framework with derivatives," *Advances in Engineering Software*, p. 102662, 2019.

[19] "Smt: Surrogate modeling toolbox." `https://smt.readthedocs.io/en/latest/index.html#documentation-contents`. Accessed: 2020-07-14.

[20] P. I. Frazier, "A Tutorial on Bayesian Optimization." arXiv:1807.02811v1, 2018.

8 Appendix

8.1 N2 Diagram

The OpenMDAO library allows for n2-diagram generation, which can be a useful tool to vizualise all the connections between the parameters of the problem. An example is illustrated in figure 9 where one module and some geometric components are used within an optimisation routine. The N2 diagram can be opened in an explorer window and is interactive. By default the routine generates such a diagram and saves it in the corresponding CEASIOMpy_Run folder in the WKDIR directory.

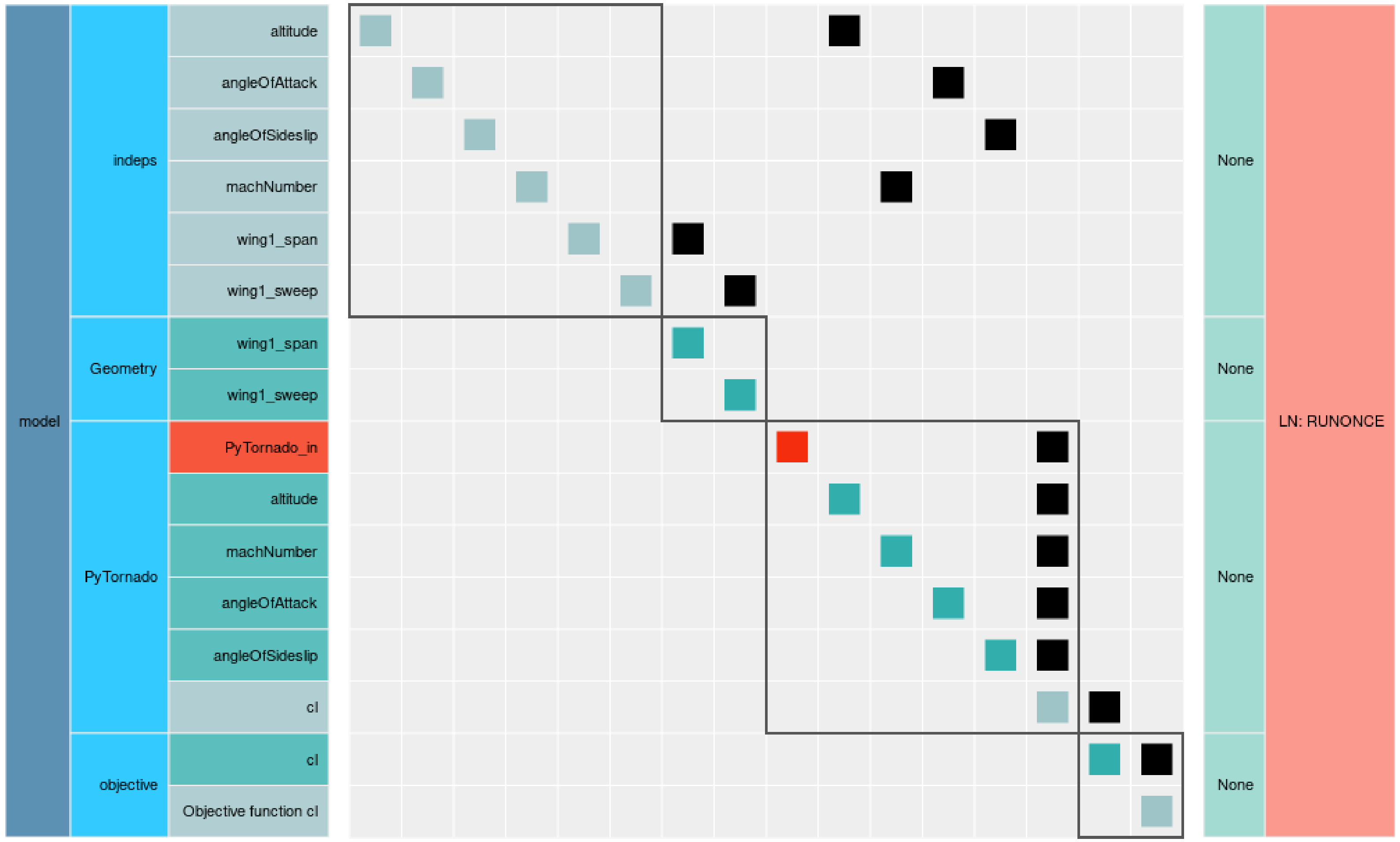


Figure 9: N2 diagram of an optimisation workflow with the *PyTornado* and the geometric components modules.