GEMS: A Python Library for Automation of Multidisciplinary Design Optimization Process Generation

François Gallard, Charlie Vanaret, Damien Guénot, Vincent Gachelin, Rémi Lafage, Benoit Pauwels, Pierre-Jean Barjhoux, Anne Gazaix

GEMS: A Python Library for Automation of Multidisciplinary Design Optimization Process Generation

Conference Paper · January 2018

DOI: 10.2514/6.2018-0657

8 authors, including:

François Gallard
IRT Saint Exupéry, Toulouse, France
14 PUBLICATIONS **79** CITATIONS
SEE PROFILE

Charlie Vanaret
Fraunhofer Institute for Industrial Mathematics ITWM
24 PUBLICATIONS **118** CITATIONS
SEE PROFILE

Damien Guénot
Altran, Toulouse
5 PUBLICATIONS **28** CITATIONS
SEE PROFILE

Rémi Lafage
The French Aerospace Lab ONERA
9 PUBLICATIONS **46** CITATIONS
SEE PROFILE

GEMS: A Python Library for Automation of Multidisciplinary Design Optimization Process Generation

François Gallard[*] Charlie Vanaret[*], Damien Guénot[*†], Vincent Gachelin[*‡], Rémi Lafage[*§], Benoit Pauwels[*], Pierre-Jean Barjhoux[*], Anne Gazaix[*¶]

This paper describes the GEMS software developed as part of the IRT Saint Exupéry MDA-MDO project[1] for supporting Multidisciplinary Design Optimization (MDO) capabilities. GEMS is a Python library for programing MDO simulation processes, built on top of NumPy[2], SciPy[3] and Matplotlib[4].

GEMS aims at pushing forward the limits of automation in simulation process development, with a particular focus on : i) the automatic programming of MDO processes; ii) distributed and multi-level MDO formulations (or MDO architectures); iii) the integration and development of state-of-the-art algorithms for optimization, design of experiments, surrogate models and coupled analyses; iv) the automation of MDO result analysis; v) the deployment of MDO processes in heterogeneous and distributed industrial simulation environments.

1. Introduction

MDO formulations, define a decomposition strategy to reformulate a design problem into one or multiple optimization problems[5]. Numerous formulations exist, and their respective performances depend on the addressed problem. For a given problem, their relative performances specially depend on the dimensions of the design variables, coupling variables, and constraints[6]. Besides, industrial applications invoke black box simulation codes that may not be modified or differentiated, or may involve discrete design variables, which restricts the set of potentially useful MDO formulations[1]. In addition, the deployment of MDO at industrial scale implies implementing numerous MDO processes, which can be very expensive if not supported by appropriate software.

The GEMS library was developed to tackle these industrial constraints. MDO processes should be generated automatically with a minimal amount of programming effort, based on a catalog of available MDO formulations which may be adapted to dedicated use cases.

In the aeronautical field, disciplinary optimization software, such as aerodynamic shape optimization or structural sizing, have typically more advanced capabilities than MDO capabilities involving both high-fidelity aerodynamics and structure simulations in a single process. In the past decades, considerable efforts were committed to the industrialization of such disciplinary problems solvers[7–9]. The methodologies used in these solvers take advantage of the disciplinary optimization problem structure, such as the linearity of the mass objective and the large amount of non-linear constraints in structural sizing problems, while the aerodynamic optimization problems have a non-linear objective function, the drag, with a small number of non-linear constraints such as the lift or only bounds on design variables. This makes them extremely efficient, but specialized. If one wants to reuse these capabilities in a MDO process, such a process shall involve sub-optimization problems and therefore the selected MDO formulation shall be distributed. Multi-level MDO formulations decompose the MDO problem into multiple optimization problems. For instance in

[*]Institute of Technology IRT Saint Exupéry, Toulouse, France
[†]Altran Technologies, Toulouse, France
[‡]Sogeti High Tech, Toulouse, France
[§]ONERA, Toulouse, France
[¶]Airbus Operations SAS, Toulouse, France

the BLISS98[10], the design variables that are shared by multiple disciplines are the unknowns of a system-level optimization, while the design variables that are only directly impacting a single discipline are the unknown of so-called disciplinary optimizations. These can therefore make direct use of existing disciplinary optimization software as opposed to monolithic MDO formulations, such as MDF, which involve a single optimization problem with coupled disciplines. Another advantage of this approach is that the MDO formulations are independent of the disciplinary tools, so a single implementation can be validated and shared for multiple use cases.

For these reasons, GEMS was developed with multi-level MDO formulations as an important use case, with the possibility to reuse existing black-box optimization software. This is a substantial difference with NASA's OpenMDAO[11], which mainly targets MDF and IDF since the rewriting of the code for version 1.0.0[12]. The focus is made in OpenMDAO on getting the maximal performance out of these formulations, taking advantage of parallel algorithms and the graph structure of the process. The KADMOS software[13], has MDO formulations capabilities, and is also mainly focused on MDF and IDF formulations as of 2017 in version 0.7. In comparison, GEMS is a generic library that supports both monolithic and multi-level MDO formulations. Another difference with the KADMOS software is that GEMS does not delegate the execution of the MDO process to a workflow engine but achieves itself the workflow and dataflow.

In Section 2, we propose a review of the existing paradigms for programming MDO processes, and conclude on the need for a new approach, implemented in GEMS. In Section 3, we describe GEMS' software architecture. In Section 4, we list some of the available features, among which the MDO formulations, the MDO results visualisations, and the MDA (Multidisciplinary Design Analysis) algorithms with coupled derivatives.

2. MDO processes programing : state of the art and a new paradigm

2.1. The basics of simulation process programming

In order to build an automated simulation process, the elementary bricks of the simulation processes shall be interfaced with a workflow manager, itself responsible for the overall execution and exchange of data. These interfaces are achieved through wrappers, which are a piece of software dedicated to the execution of other programs. Each program is encapsulated using a dedicated interface. A wrapper is an abstraction that is used to define input data, output data and an execution of the interfaced software independently of its own implementation (file formats and data structures for instance). The concept is used in all process integration software (OpenMDAO from NASA[11], ModelCenter™ from Phoenix Integration[14] or Isight™ from Dassault Systèmes[15]). It is a prerequisite to build any process.

The motivations of programming simulation processes using wrappers and a worfklow engine, compared to scripting it directly with a programming language, are the following.

- Easily create and maintain many similar processes using the same software wrappers.
- Make the process less dependent on the interfaced software version changes: input and output modification, internal methods implementation.
- Develop automated checks of the consistency of the process to anticipate errors before execution.
- Develop common preparation and analysis tools for multiple processes.

2.2. Data-driven versus workflow-driven

Several paradigms can be used to program a simulation process from wrapped software. The simulation process is defined by the execution sequence of the wrapped software, called "the workflow", and the data exchanges between them, called "the dataflow". There are many possible classifications of workflow engines: we propose here one particular view which is well suited to present the GEMS concepts in the next sections.

Currently existing workflow engines can be classified into three paradigms for programming the simulation processes.

1. Full process description: the process developer programs both the data exchanges between the wrapped software, and the execution sequence. The process is neither driven by the data nor the workflow, since both are inputs of the workflow engine. This is the case of ModelCenter™ from Phoenix Integration[14],

where the process developer shall first define the order of execution, and then the data exchanges between the boxes.

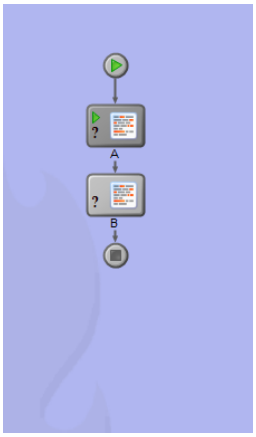
2. Data-driven: the process builder describes only the data exchanges of the process, and the workflow engine automatically generates the workflow. The process is then "data-driven" because the data exchanges drives the whole process. This is typically the case of MATLAB Simulink^{TM} processes.

3. Workflow-driven: the process builder describes only the sequence of execution of the process, and the workflow engine automatically generates the dataflow. The process is then "workflow-driven" because the execution sequence drives the whole process. This is typically the case of the WORMS workflow engine, developed at Airbus by Meaux et al. for aerodynamic shape optimization[8,16,17], and later eWORMS, based on Eclipse RCP.

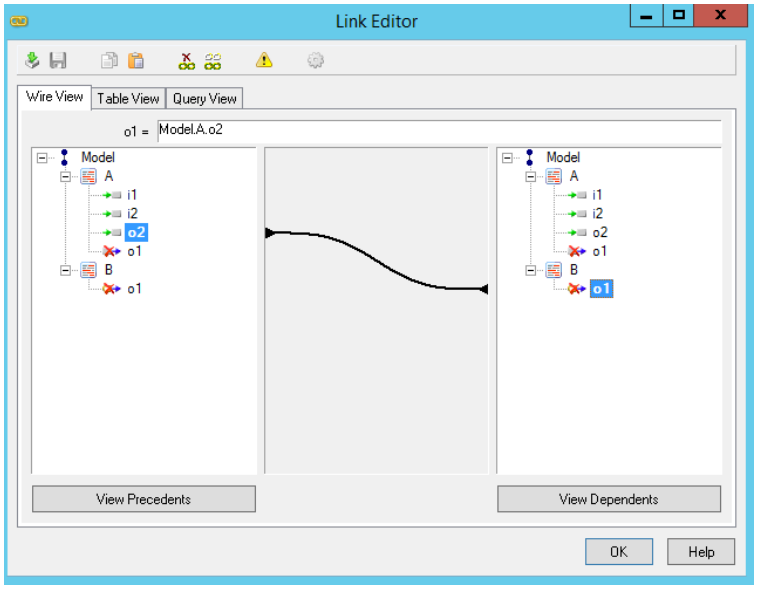
In the following paragraphs, we examine the pro and cons of data-driven versus workflow-driven engines in the perspective of MDO for aeronautical test cases.

2.2.1. Full process description

Figure 1 shows how the execution sequence and the data exchanges of the process are defined graphically by the builder of a ModelCenter process. The two wrapped software A and B are executed in the order A then B, as shown in Figure 1a. Then, the output "o2" of A, is connected to the input "o1" of B, as shown in Figure 1b.



(a) Execution sequence description.



(b) Data exchanges description.

Figure 1: Two steps for building a process in ModelCenter: defining the execution sequence (workflow), and then the data exchanges (dataflow).

In this case, the process builder programs both the workflow and the dataflow of the process. The workflow engine is therefore not automating a part of the process programming, but the process execution only. However, the builder has full direct control over the workflow and dataflow, which can be useful when the interfaced software do not use consistent names for their inputs and outputs, nor follow a standardized or generic process.

The advantages of workflow engines based on the full process description are:

- the process builder can fully customize the data exchanges and execution sequence to specific cases;
- there are no rules to be followed on the data names, nor generic process to be defined.

The main weaknesses of the approach are the following:

- When the interfaced software have many inputs and outputs, the process becomes very hard to build.
- The workflow is very sensitive to inputs and outputs changes. If, during a version change of a component, a new input appears, the whole process may be impacted. The process has a weak tolerance to inputs and outputs changes.
- The workflow is impacted by a change of interfaced software, changing one of them impacts the workflow execution sequence.

2.2.2. Data-driven

The data-driven paradigm is inspired from electrical wiring and discrete time simulation to automate the execution sequence of the process: the wrapped software are executed every time their inputs are ready.

Figure 2 illustrates the data exchanges specified by a process builder for a typical simulation process built with a data-driven workflow engine such as CFD or CSM. The steps of the process are:

1. building the geometry,
2. generating the mesh from the geometry,
3. solving the equations of interests (RANS equations for instance in CFD) on the generated mesh,
4. post-processing the simulation (computing lift and drag for instance).

The data-driven way to describe this process is to describe all the exchanges between the wrapped software in terms of data. To illustrate the process execution in this particular case, when the user supplies a new set of parameters to the geometry, the geometry program generates a file that is passed to the meshing program, which triggers its execution, and produces a mesh. This mesh is then passed to the simulation program, along with geometrical information, which triggers the simulation. Finally, the solution, the mesh and the geometrical information are passed to the post-processing tool to compute quantities of interest.

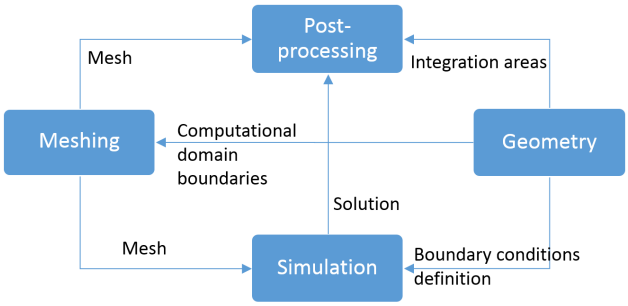


Figure 2: A data-driven description of a simulation process.

In data-driven workflow engines, the data drive the process: the builder specifies the data exchanges and the workflow is automated by the engine. Therefore, the workflow engine has specific rules to decide when the next wrapped program should be executed, that is when its inputs are ready. Each workflow engine has its own rules depending on the application. For signal processing for instance, fixed time steps may be used and all executions may be triggered simultaneously for each step.

The advantages of data-driven workflow engines are:

- the workflow is automated, the process developer does not have to care about the overall execution sequence;
- focus is on the data;
- it is a well suited approach for signal processing and electrical system designs, which have many components and low-dimensional data.

The main weaknesses of the approach are the following:

- When the interfaced software have many inputs and outputs, the process becomes very hard to build.
- The workflow has a weak tolerance to input and output changes. If a new input appears during a version change of a component, the whole process may be impacted.

2.2.3. Workflow-driven

In high-fidelity simulations, the inputs and outputs of the program are numerous, a CFD code for instance has typically hundreds of inputs and options. To overcome the limitations of data-driven paradigm and achieve a partial automation of the process development, workflow-driven engines have been developed.

The main idea of workflow-driven engines is that, in some processes such as CFD simulations, the execution order is constant. The geometry will always be generated before the mesh, which will then be used by the CFD solver, and the postprocessing will always come last. The predefined execution order, along with rules on the data exchanges, defines the dataflow. The workflow engine is in charge of managing the data exchanges accordingly. Namespaces as well as data privacy management between programs are important points to build complex processes.

Figure 3 illustrates the execution sequence specified by a process builder for a typical simulation process built with a workflow-driven engine.

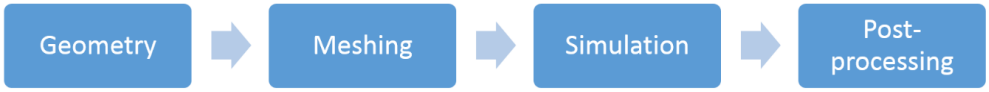


Figure 3: A workflow-driven description of a simulation process.

The advantages of workflow-driven workflow engines are:

- the dataflow is automated, the process developer does not have to care about the overall data exchanges, which is good for processes with a lot of data;
- focus is on the process execution order, which is well suited for some classes of processes such as CFD;
- it is a well suited paradigm for processes with a constant execution order, or programs whose inputs and outputs change over versions.

The main weaknesses of the approach are the following:

- The user has difficulties to have a clear representation of the process. Most people are used to boxes and wires, that is a data-driven way to display processes, which is not suited for workflow-driven engines, where the process is a tree of execution sequences.
- The process is sensitive to change in the interfaced software. For instance, if one wants to replace a simulation post processing tool by another, all processes that use the former will have to be reprogrammed.

2.2.4. Dataflow and workflow automation needs for MDO

From the previous sections, we can see that there is a duality between the dataflow and the workflow in the current way of programming simulation processes. In either case, the dataflow or/and the workflow have to be programmed. In high-fidelity MDO for aerospace applications we have disciplines with many inputs and outputs, which change over time and use cases. Therefore the process programmer can hardly care about all the data. In addition, for given disciplines, many different problems and associated processes are built in an industrial-scale development. Besides, MDO formulations exhibit different performances when the use case changes, since there is no "free lunch" for MDO formulations[6]. For given disciplines, changing the formulation completely changes the dataflow and the workflow.

Figure 4 shows some use cases addressed in the MDA-MDO project[1]. For each use case, multiple methods may be used, such as DOEs, RSMs, or MDO formulations, and for each methods, different algorithms may also be selected. Besides, the automation of the process shall be implemented in a workflow engine, and we display here three possible options (Isight, ModelCenter, WORMS). There are 99 paths in this graph:

each one represents a possible process, i.e. a combination of a use case, a formulation, an algorithm and a workflow engine. In addition, each of these items is implemented in a separate software package, which evolves in time. For three different versions of each software, this leads to $3^{99} \approx 10^{47}$ potential processes. This is a major issue for maintenance and industrial deployment of MDO methods, which illustrates why MDO process programming must be automated and reconfigurable.

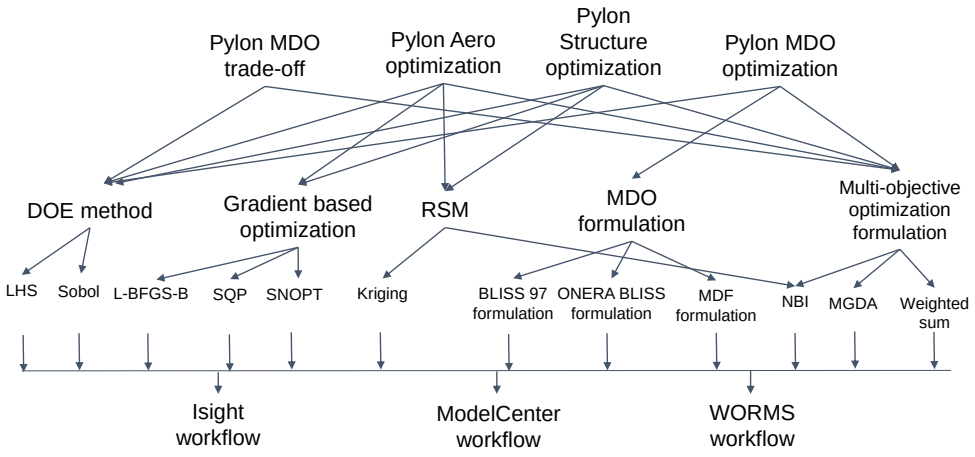


Figure 4: The combinatorial issue in MDO process development.

As a conclusion, we need automated dataflows as in workflow-driven engines and automated workflow management as in data-driven engines. Consequently, new paradigms to implement MDO simulation processes are required to overcome the current limitations.

2.3. GEMS' solution to address both dataflow and workflow automation

Some of the limitations of workflow-driven engines are solved by using the object oriented programming paradigm. If wrappers are classes, one can implement subclasses by inheritance, which then allows to substitute a subclass by another using polymorphism. This allows for instance to change one of the simulation tool by another with identical purpose in a process, but not to reconfigure the whole process with a completely different execution sequence. One cannot reconfigure in this way a MDF process, based on a single optimization calling a MDA, into a bi-level MDO process, based on multiple disciplinary optimizations.

With the objective of developing reconfigurable MDO processes in mind, generic processes have been developed in GEMS, which can be used for any wrapped program. MDAs (Multi-Disciplinary Analysis) for instance are generic processes, so are software chains that represent a linear execution sequence of a list of disciplines. Once instanciated with a list of disciplines as attributes, these generic process can be executed to achieve a workflow. They are programmed in a workflow-driven way to take advantage of this paradigm.

A second limitation in the current way of building the processes is the link with optimization algorithms. The definitions of the objective function and constraints depend on each test case, since the quantities of interest are actually computed by different programs. Either in workflow or data-driven engines, the optimization loops are programmed within the rest of the process. Therefore, the formulation is problem-specific, which is an issue for reconfiguring the process to use a different MDO formulation. Besides, current workflow engines have succeeded to support disciplinary optimization processes, but they are unadapted to multi-level MDO formulations, which involve multiple nested optimization loops. And above all, they are unable to support a complete process reconfiguration such as going from a monolithic formulation to a multi-level formulation.

MDO formulations are generic strategies that generate optimization problems on top of generic processes. Objective functions and constraints can be computed from the processes outputs. This is a generic approach that can be implemented independently of the test case. The dataflow is handled in a part by the optimization algorithm and in a part by the generic processes, as a consequence of the selected MDO formulation. The same applies for multi-level formulations such as BLISS98[10], since the decomposition strategy of the MDO problem into a system level and disciplinary optimization problems is also generic.

High-fidelity MDO processes take weeks to run. The traditional trial-and-error way to debug such a process is unrealistic. The GEMS way of building processes allows a multi-step for the validation and debugging of MDO processes:

- the generic processes and MDO formulations are independent of the test case, so they can be validated on well known test cases;
- the wrappers are independent of the processes, so the integration of the simulation programs can be validated separately;
- the performance of algorithms and formulations can be tuned on substitution problems thanks to the scalable methodology by Vanaret et al.[6]

3. Software architecture

3.1. General principles

In this section, some parts of GEMS' design are discussed, with focus on some key classes. GEMS has more than 100 classes, therefore the full class diagram is not shown here. The package has an intermediate size of 26,000 lines of Python code. Python was chosen because the main purpose of GEMS is to integrate multiple software, and Python is known to be efficient to glue together existing components in heterogeneous languages when compared to C++ and Java[18]. Besides, GEMS also contains numerical algorithms manipulating matrices and this can be programmed easily using he NumPy[2], SciPy[3] libraries. Many of the basic numerical algorithms for scientific computing (linear and non-linear systems, optimisation, signal processing) are provided in these libraries. Finally, optimization data plots can be easily programmed from the NumPy arrays produced by the scenarios execution using the Matplotlib library[4].

3.2. Main classes

The Scenario (DOE or MDO) is composed of disciplines, and the MDO formulation. The MDO formulation creates an optimization problem from the list of disciplines and a design space. This may require the intermediate creation of generic sub-processes such as MDAs, or other MDO scenarios. All generic processes inherit from disciplines and are composed of a list of disciplines. An optimization problem is defined from functions generated by the `FunctionGenerator`, on top of the generic processes or directly the disciplines as in the case of the IDF formulation. The drivers (optimization algorithm or DOE) solve or sample the optimization problem, and when started, trigger the execution of the process.

The high-level classes that are key in the architecture are the following.

- `MDOScenario` builds the process from a set of inputs, several disciplines and a formulation. It is one of the main interface classes for the MDO user. The `MDOScenario` triggers the overall optimization process when its `execute()` method is called. Through this class, the user provides an initial solution, user constraints and the design space. The user may also generate visualizations of the scenario execution, such as convergence plots of the algorithm.
- `DOEScenario` is similar to the `MDOScenario`, but is dedicated to DOE-based studies. Both `MDOScenario` and `DOEScenario` inherit from the `Scenario` class that defines common features.
- `MDODiscipline` represents a wrapped simulation program or a chain of wrapped software. It can either be a link to a discipline interfaced with a workflow engine, or can be inherited to interface a simulation program directly. Its inputs and outputs are represented in a Grammar.
- `MDOFormulation` describes the MDO formulation (e.g. MDF and IDF) used to solve the MDO problem.
- `OptimizationProblem` describes the mathematical functions of the optimization problem (objective function and constraints), along with the design variables. It is generated by the `MDOFormulation`, and solved by the optimization algorithm. It has an internal database that stores the calls to its functions by the optimization algorithm to avoid duplicate computations. It can be stored on disk and analyzed by the post-processing methods.

- `DesignSpace` is an attribute of the `OptimizationProblem` that describes the design variables, their bounds, their type (floating point or integer), and current value. This object can be read from a file.

Two low-level classes are crucial to understand the basic principles of GEMS:

- `MDOFunction` instances (objective function and possible constraints) are generated by the `MDOFormulation`. Depending on the formulation, constraints may be generated as well (e.g. consistency constraints in IDF).
- `MDOFunctionGenerator` handles the creation of `MDOFunctions` for a given `MDODiscipline`. It is a key class for the development of MDO formulations because these functions are passed to the optimizers, which drive the execution, and their values and gradients are computed from the disciplines. Therefore, the functions are a bridge between the MDO formulations and the generic processes part of GEMS.

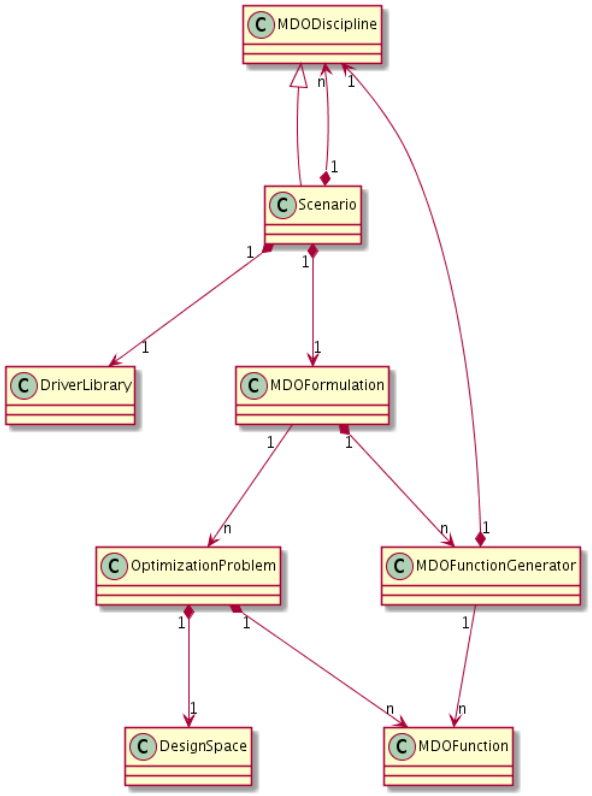


Figure 5: Core class diagram of GEMS.

Figure 5 shows the composite design pattern used for the `MDOScenario`, which is a subclass of `MDODiscipline` and is also composed of a list of `MDODisciplines`. It is a key point for multilevel MDO formulations because sub-`MDOScenarios` can be embedded into a system-level `MDOScenario` as its `MDODisciplines`.

Figure 7 shows the abstraction used to define `DOEScenario` and `MDOScenario` that both inherit from the main class `Scenario`. A `DOEScenario` can also be substituted with an `MDOScenario`, so trade-off processes can be built using a bi-level MDO formulation that mixes design of experiments and disciplinary sub-optimizations.

Figure 8 shows a part of the classes hierarchy for the generic processes. All generic processes are sub-classes of the `MDODiscipline` class. This composite design pattern is the key for building complex processes by blocks. The Gauss-Seidel algorithm for instance can be used to solve an MDA, and is built using a sequential execution ("chained") of the disciplines by the `MDOChain` class. The MDA is also a subclass of `MDODiscipline`, so can be embedded in any other generic process as a substitute of a `MDODiscipline`. Advanced MDAs as shown in Section 4.7.1 are composed of a `MDOChain`, taking parallel execution, `MDAs` and

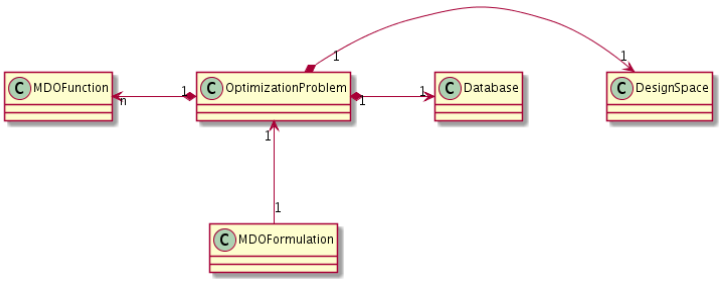


Figure 6: GEMS class diagram: optimization process.

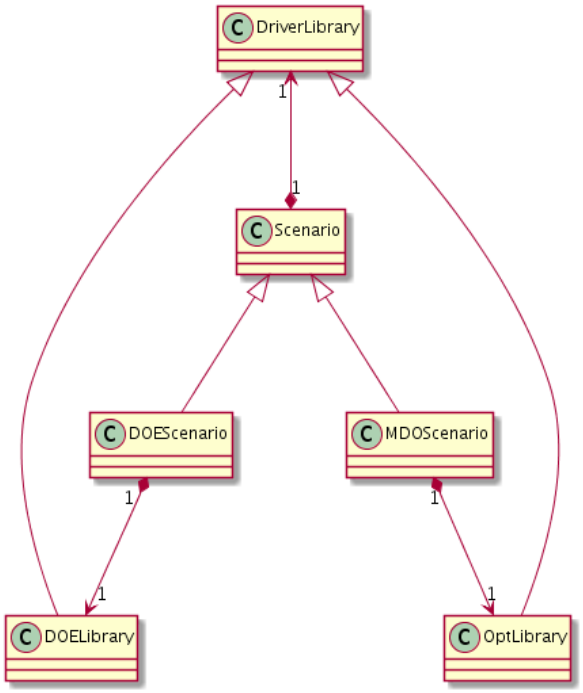


Figure 7: GEMS class diagram: driver and scenarios.

single `MDODiscipline` as its own input disciplines, in order to achieve the graph of execution plotted in Figure 22.

This building blocks design is also exploited for the automated derivatives computation. The computation of coupled derivatives is implemented once in the `MDA` base class and then shared by all MDA sub-classes. Besides, the classical chain rule for composing derivatives is available in the `MDOChain` class. Any process that is built on top of these classes will automatically benefit from these capabilities. Advanced MDAs, implemented `MDAChain`, can therefore either compute the derivatives of the overall coupled problem through its base `MDA` class, or by chain ruling the coupled derivatives of the sub-MDAs through the `MDOChain`.

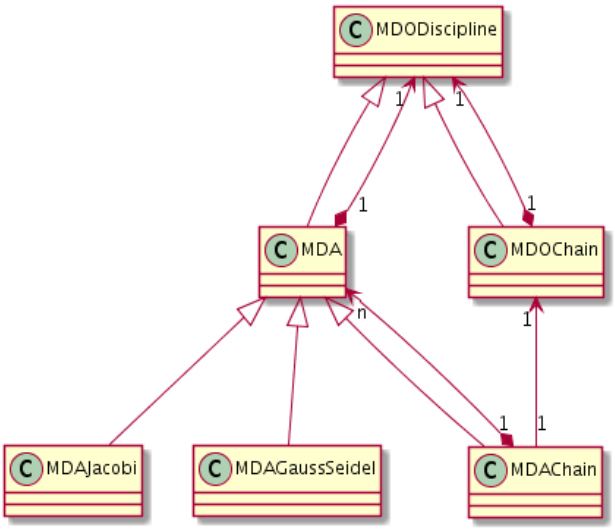


Figure 8: MDA and generic processes class diagram of GEMS.

3.3. Execution sequence

Figure 9 shows the execution sequence diagram of an MDF scenario. The scenario triggers the overall execution through the resolution of an optimization problem. The generated objective and constraints are passed to the optimizer, which calls them iteratively. At each call, a MDA is executed. A caching mechanism at the optimization problem level is also implemented to avoid duplicate executions, and is not illustrated here. Then, the algorithms may also call the gradient of the objective function and constraints, which triggers the "linearize" method of the MDA, which also needs the Jacobian matrices of the disciplines in order to compute the coupled derivatives. These Jacobian matrices are computed by the "linearize" methods of the disciplines.

A similar scheme applies to multi-level scenarios. At system level, the MDA of Figure 9 is replaced by multiple MDO scenarios.

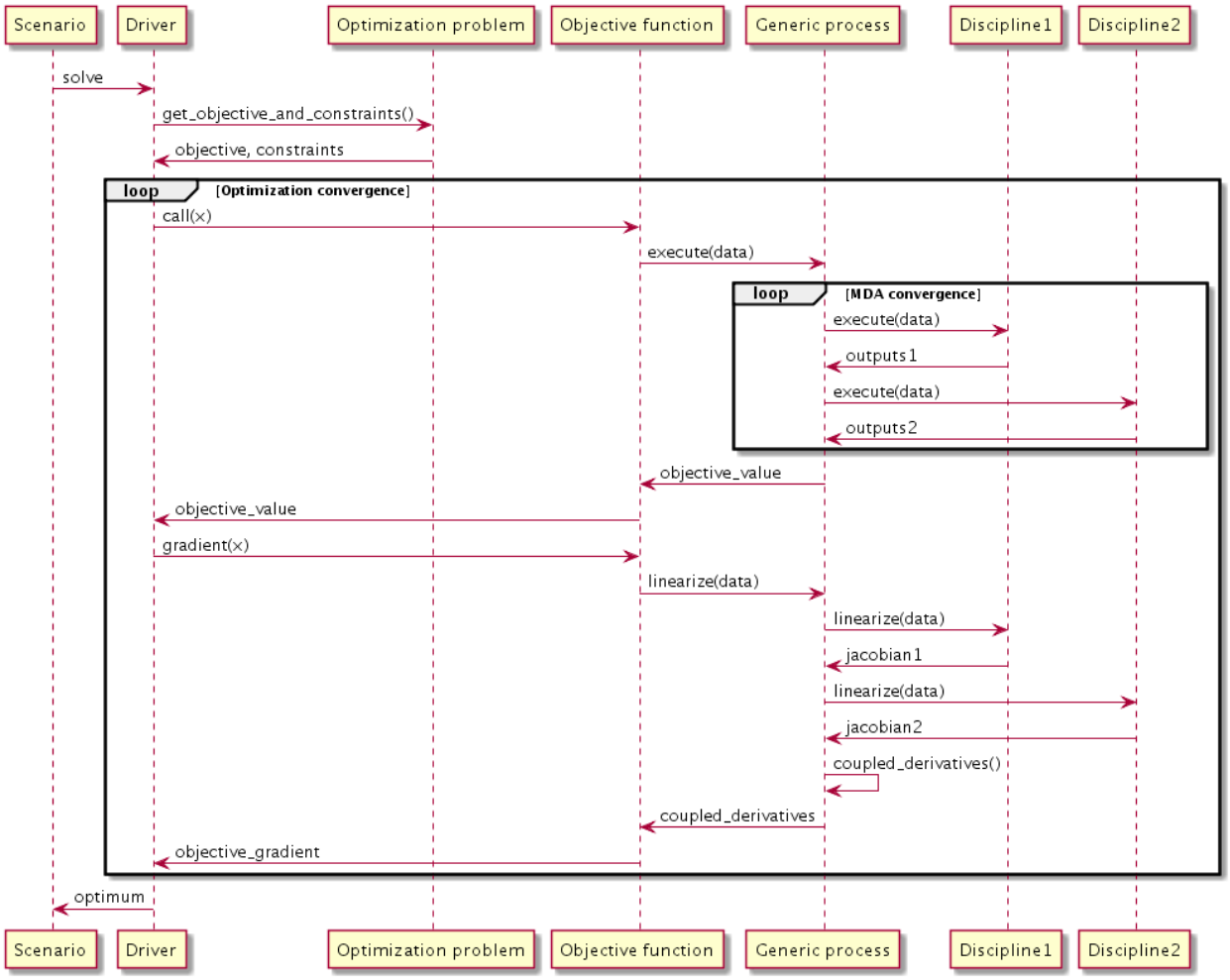


Figure 9: GEMS execution sequence diagram

3.4. GEMS in an MDO platform

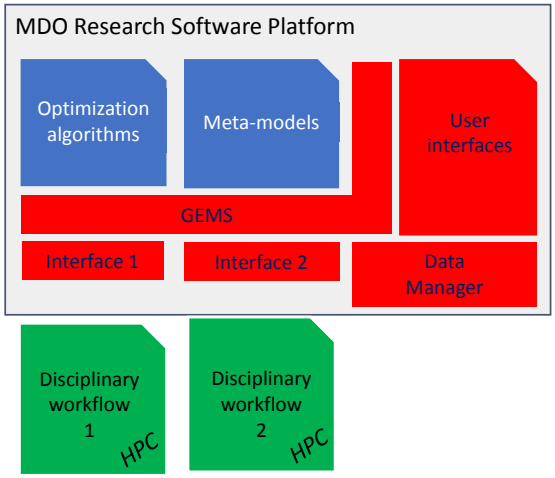


Figure 10: GEMS and its interfaces in an MDO platform.

Figure 10 shows how GEMS (the MDO formulations engine) is typically interfaced in an MDO platform. The platform provides standardized interfaces for disciplinary workflow engines that run on HPCs. This allows to build a multi-machine distributed MDO capability. GEMS has standard interfaces to optimization and DOE algorithms. A proof of concept of a distributed MDO process running on multiple machines under Windows or Linux and with different workflow engines has been performed[1]. The target platform for such a component-based architecture in the project is Eclipse RCP,[19] which allows graphical integration of multiple plugins. Such a graphical user interface is being developed on top of GEMS to provide user inputs, configure and monitor a MDO scenario.

4. Features overview

4.1. MDO formulations

The following MDO formulations have been developed with GEMS:

- Multidisciplinary Design Feasible (MDF).
- Individual Design Feasible (IDF).
- Disciplinary optimization for weakly coupled problems.
- Two variants of bi-level formulation using disciplinary design scenario[1], derived from ONERA's research on high-fidelity MDO for aeronautical applications[20].
- A bi-level formulation based on distributed MDF scenarios[1].
- A bi-level formulation based on asymmetric distributed MDF scenarios[1].
- A distributed MDO formulation with overlap on the shared design variables, based on a domain decomposition method with synchronization[21].
- A distributed formulation dedicated to structural optimization with sizing and categorical variables based on the branch and bound algorithm[22].

Once the MDO process is built (however not executed), GEMS can generate the XDSM diagram[23] associated to the selected MDO formulation using ONERA's XDSMjs library[24]. This feature is useful to the user to represent himself a simplified view of the process that is generated by GEMS. It also allows to check that all the connections (the dataflow and the workflow) are achieved properly before execution, and to debug the process.

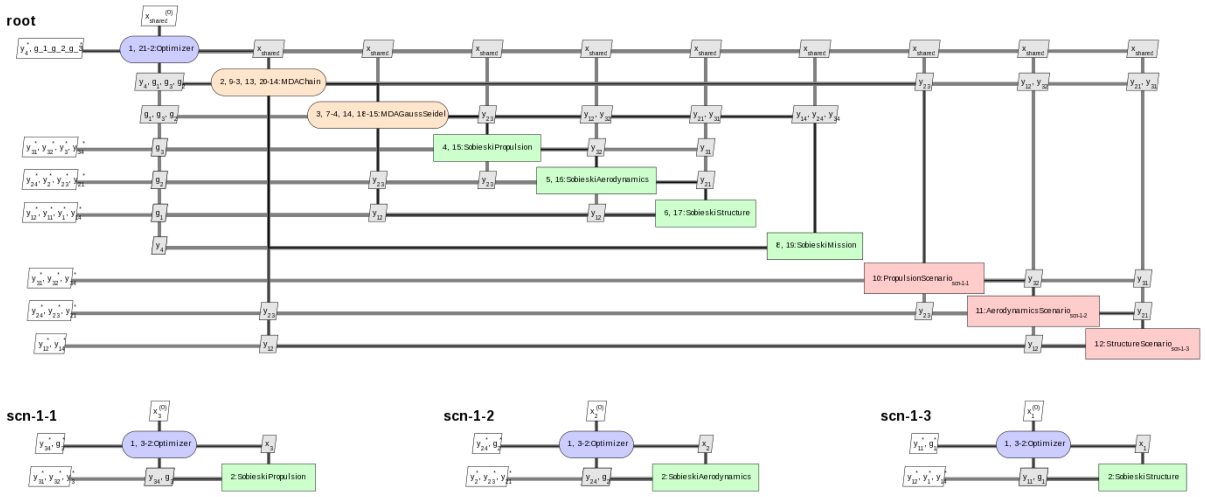


Figure 11: XDSM of a bilevel MDO process generated from GEMS.

4.2. GEMS scenario execution

Appendix A and Appendix B show the output log of two MDO scenarios executed with GEMS on Sobieski's SSBJ[10] design optimization problem. The initial design space is provided, as well as the generated optimization problem. Additional consistency constraints are generated by the IDF formulation. Note that the range objective function (y_4) and the design constraints (g_1, g_2, g_3) have different dependencies due to the different formulations. During the execution, a progress bar indicates the optimization progress as well as the number of iterations per second. The best feasible point is given as the solution of the problem. Finally the execution metrics can be printed, such as the total spent time and number of calls to each discipline.

Appendix C shows the first system iteration of the first bi-level process described by Gazaix et al.[1] whose XDSM[23] is also displayed in Figure 11. One can note the system optimization problem that handles the shared design variables, and the disciplinary optimizations. MDA histories are not shown here.

4.3. Interfaces to a range of DOE and optimization algorithms

Once GEMS has generated an optimization problem, its resolution is delegated to an optimization algorithm. A set of optimization and DOE algorithms are therefore interfaced with GEMS. The high performance of an optimization algorithm measured on a set of optimization problems necessarily comes with a lower relative performance on other sets of optimization problems, which is known as the "No free lunch theorem for optimization"[25]. This is particularly important in MDO since changing the coupling or design variables dimensions generates different optimization problems, which has an impact on the performance of the algorithms and in the end of the MDO process[6]. It is therefore key to allow the user to interface easily a dedicated optimization algorithm for solving its particular problem. GEMS provides a standardized interface to this aim. Currently 16 optimization algorithms are available from the NLopt[26], OpenOpt[27], SNOPT[28], as well as in-house optimization algorithms. 20 DOE algorithms are interfaced from the OpenTURNS[29] and pyDOE libraries. For a given MDO formulation, the driver used to execute the process may be either a DOE or optimization algorithm, and this can be reconfigured easily. The classes hierarchy used to support this feature is shown in Figure 7. This also allows to create trade-off scenarios based on bi-level formulations, where the system level optimization problem is sampled using a DOE algorithm and the sub-scenarios are solved by optimization algorithms.

4.4. Visualization of optimization results

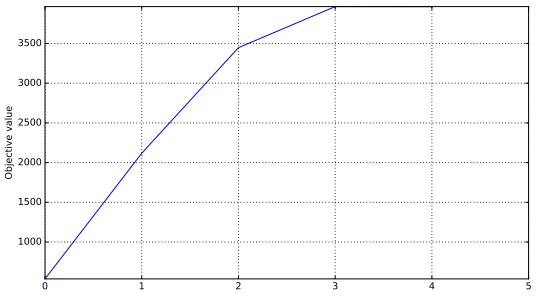
MDO results consist in the set of values for the objective function, design variables, coupling variables and constraints, for each iteration of each optimization problem. This data typically represents thousands or millions of scalar values, which cannot be analysed in a single plot. Therefore, a collection of dedicated plots must be used: each one represents a partial view on this data, or projections on adapted spaces. The typical use of these plots is to debug the MDO process, analyse the algorithms performance, or assist decision making in real applications.

In Figure 12, the first graph shows the evolution of the objective value during the optimization. The second graph shows the normalized values of the design variables with the index of the variable in the vector on the Y axis, with respect to the optimization iterations. The third graph shows the distance to the best optimization iteration, in log scale. The last graph shows an approximation of the second order derivatives of the objective function $\frac{\partial^2 f(x)}{\partial x^2}$, which is a measure of the sensitivity of the function with respect to the design variables, and of the anisotropy of the problem (differences of curvatures in the design space).

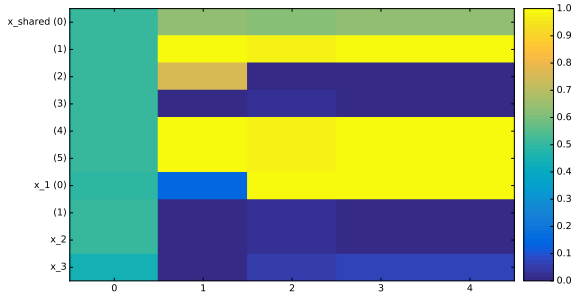
Figure 13 portrays the evolution of the values of the constraints. The components of g_1, g_2, g_3 are concatenated as a single vector of size 12. The constraints must be non-positive, that is the plot must be green or white in order to be satisfied (white = active, red = violated). At convergence, only two of them are active : g_2 and g_{3_3}.

One may also be interested in getting information on the constraints values while following the objective values iteration per iteration. Figure 14 plots all the constraints components using colors at each iteration while Figure 13 plots the objective function value superimposed with the maximum constraint violation. Therefore the first figure is dedicated to a fine analysis of the constraints values while the latter is used to analyse the best solution in terms of objective and constraints compromise.

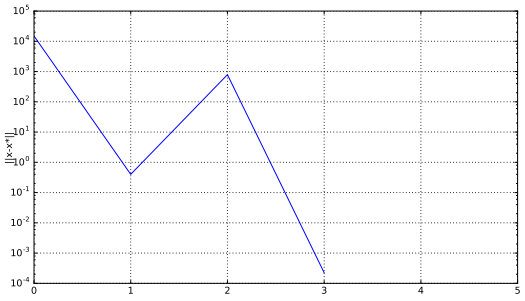
In other cases, one may prefer to plot all constraints values per iteration. In Figure 15, the constraints are plotted in separate figures, while colors in the background provide information on the magnitude of the



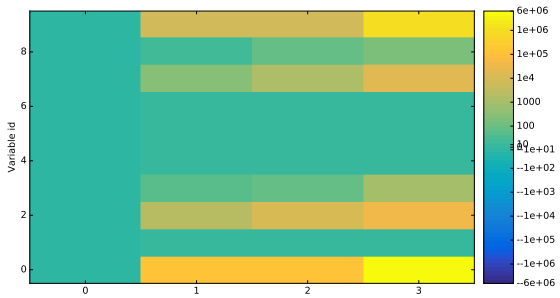
(a) Objective function



(b) Design variables



(c) Distrance to the optimum



(d) Hessian diagonal approximation

Figure 12: SSBJ MDF optimization history

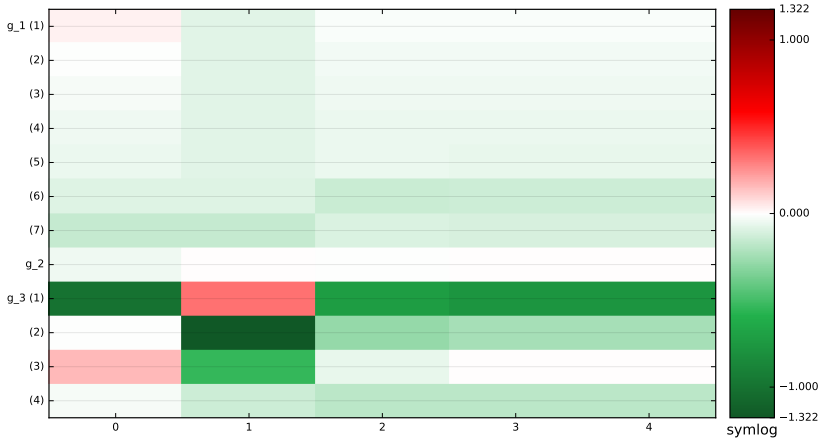


Figure 13: Optimization history: evolution of the inequality constraints in an MDF process.

values. Because of the plots scaling, in some cases it can be difficult to identify whether a constraint is slightly violated or not. This is why a vertical black line is plotted at the last iteration before the last sign change of the considered constraint.

All constraints may also be compared at a given iteration in Figure 16 on a radar chart. The scaling allows to compare the values of all constraints simultaneously while having different orders of magnitude. The radar shape makes the plot less sensitive to the number of constraints than classical line plots.

The parallel coordinates portray the design variables history during the scenario execution. In Figure 17, each horizontal coordinate is dedicated to a design variable, normalized by its bounds. A polyline joins all

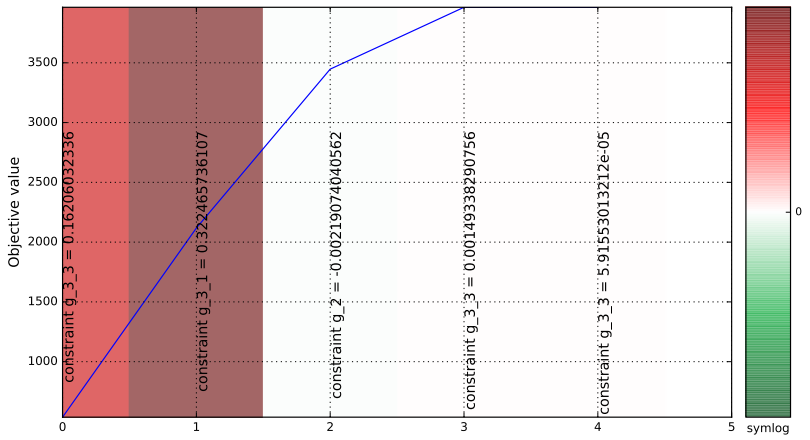


Figure 14: Objective function with respect to the iteration number, with constraint violation information in the background.

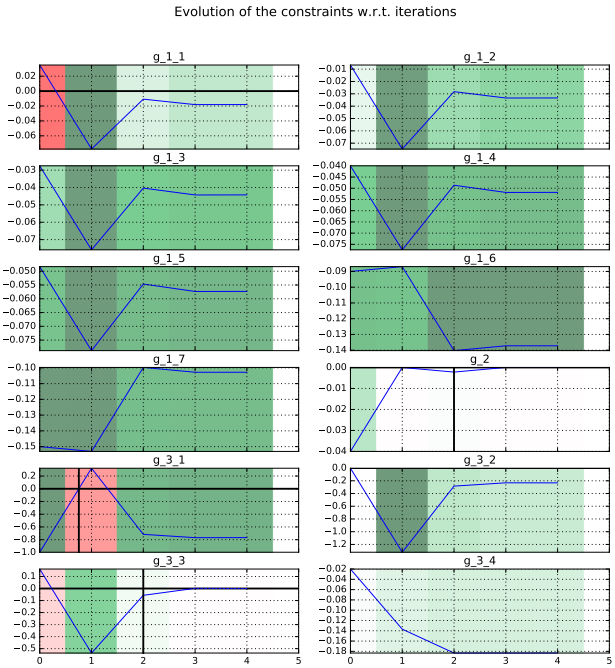


Figure 15: Constraints values with respect to iteration number, including magnitude of constraint violation information in the background.

components of a given design vector and is colored by objective function values. This highlights correlations between values of the design variables and the objective function. Figure 17 and Figure 18 show parallel coordinates respectively of the design variables and the constraints, colored by objective function value. The minimum objective value, in dark blue, corresponds to satisfied constraints.

In Figure 19, a Self-Organizing Map (SOM) is generated by using an unsupervised artificial neural network[30]. A map of size $n_x \times n_y$ is generated, where n_x is the number of neurons in the X direction and n_y is the number of neurons in the Y direction. The design space (whatever the dimension) is reduced to a 2D representation based on $n_x n_y$ neurons. Samples are clustered to a neuron when their design variables are close in terms of L2 norm. A neuron is always located at the same place on a map. Each neuron is colored according to the average value for a given criterion. This helps to qualitatively analyze if parts of the design space are good according to some criteria and not for others, and where compromises should be

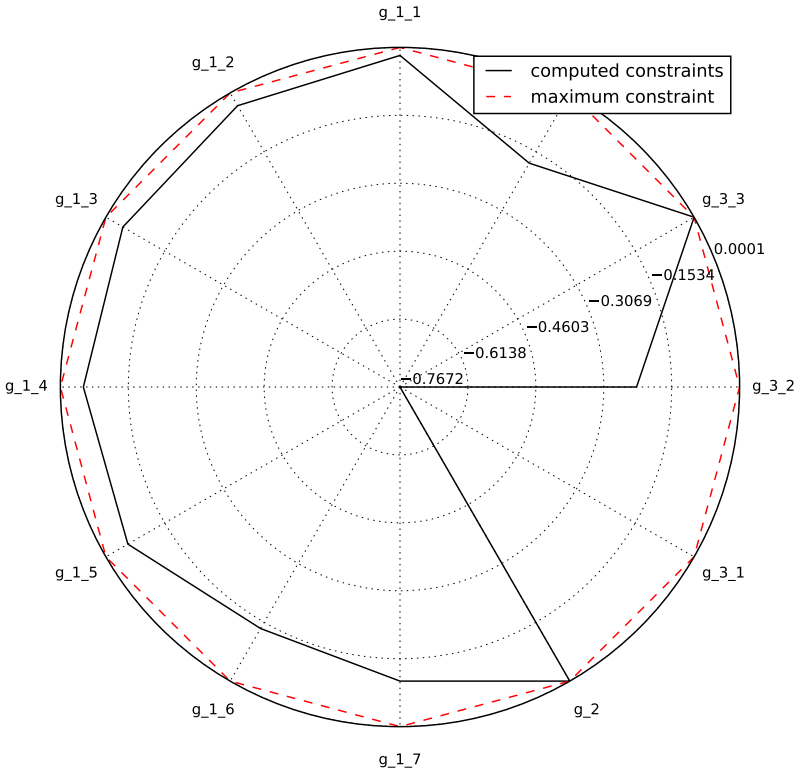


Figure 16: Constraints values at a given iteration number.

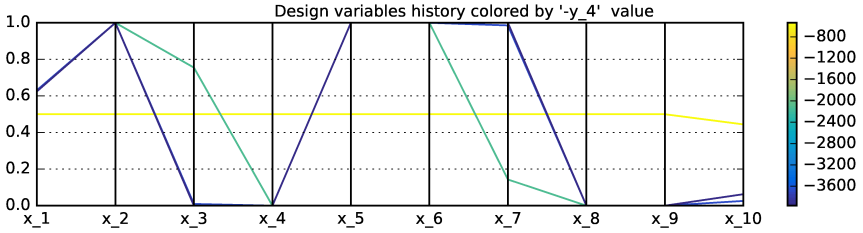


Figure 17: Optimization history: parallel coordinates of the design variables in an MDF process for the SSBJ problem.

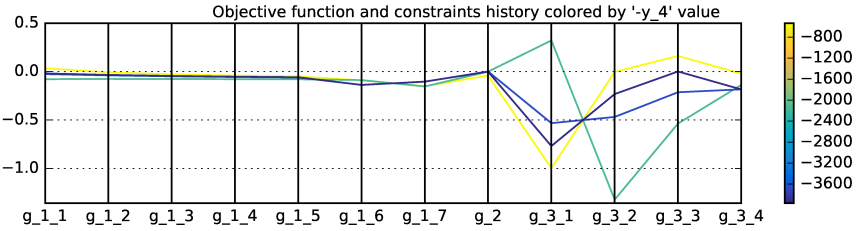


Figure 18: Optimization history: parallel coordinates of the objective function and constraints in an MDF process for the SSBJ problem.

made. SOM provide a qualitative view of the objective and the constraints, and of their relative behaviors. For more details, see the study by Kumano et al.[31] on wing MDO post-processing using SOM. Figure 19 features an SOM example on the Sobieski problem. A DOE of 10 000 samples using the MDF formulation is used to generate the data for the SOM. The process is created using the MDF formulation, that involves

an MDA; the objective function and constraints are then called by the DOE algorithm.

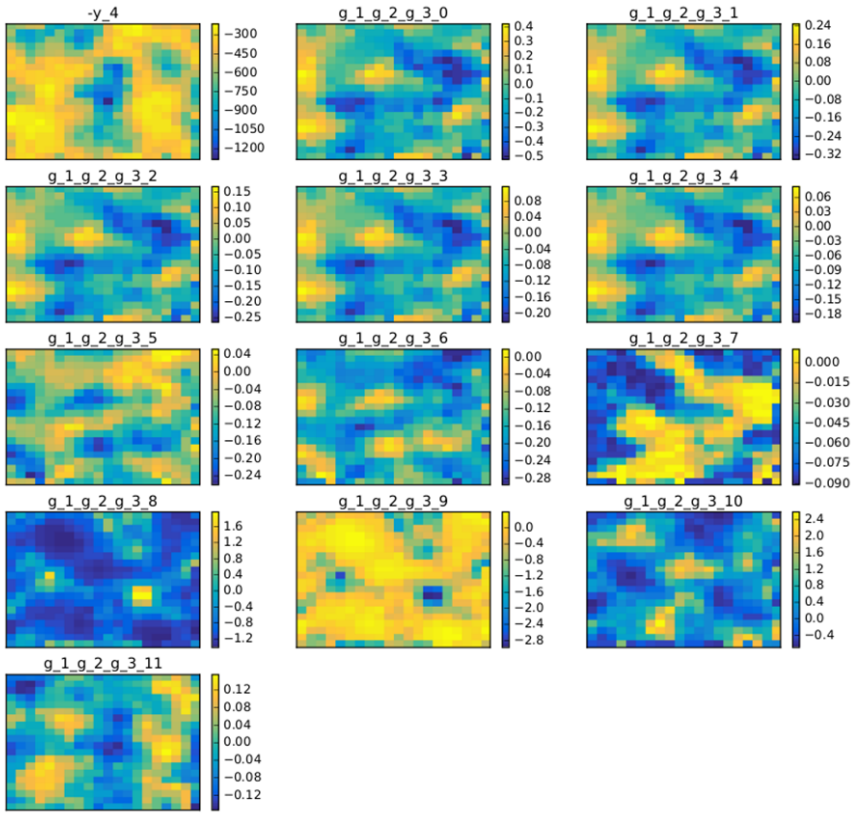


Figure 19: A Self-Organizing Map (SOM) in an MDF based process for the SSBJ problem.

Figure 20 shows the total derivatives $\frac{\partial f}{\partial x_i}$ of the objective and constraints with respect to the design variables:

- a large value means that the function is sensitive to the variable,
- a null value means that, at the optimal solution, the function does not depend on the variable,
- a negative value means that the function decreases when the variable is increased.

Figure 20 displays the total derivatives of the MDA with respect to all the design variables at the optimum of the SSBJ test case. x_0 (wing taper ratio) and x_2 (Mach number) appear to be the most important for the gradient of the objective function. Constraints g_{1_0} to g_{1_4} are very similar, since they all quantify the stress in various sections. g_{1_5} and g_{1_6} correspond to the lower and upper bounds of the twist, therefore their sensitivities are of opposite signs. g_2 is a function of only x_0 since x_0 is the only variable that influences its gradient.

Figure 21 shows an approximation of the Hessian matrix $\frac{\partial^2 f}{\partial x_i \partial x_j}$ using the Symmetric Rank 1 method (SR1)[32] at the optimum of the SSBJ problem. The color map uses a symmetric logarithmic scale. The Hessian matrix is proportional to the curvature of the function, and non-diagonal terms are the cross-influence of the variables, which are useful information for understanding the design problem properties.

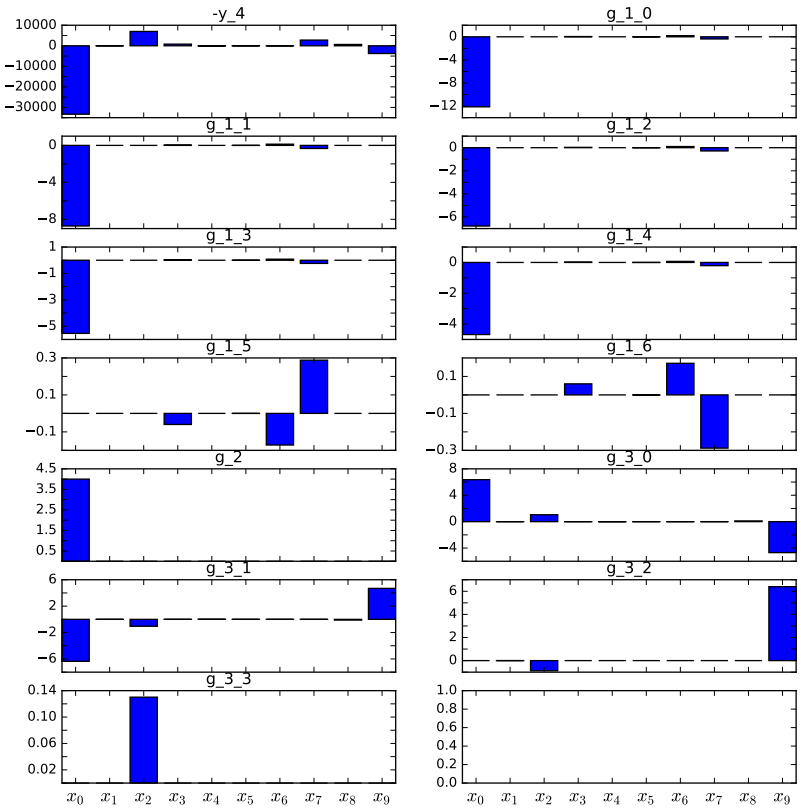


Figure 20: Coupled derivatives of the objective function and constraints at the optimum point of the SSBJ problem.

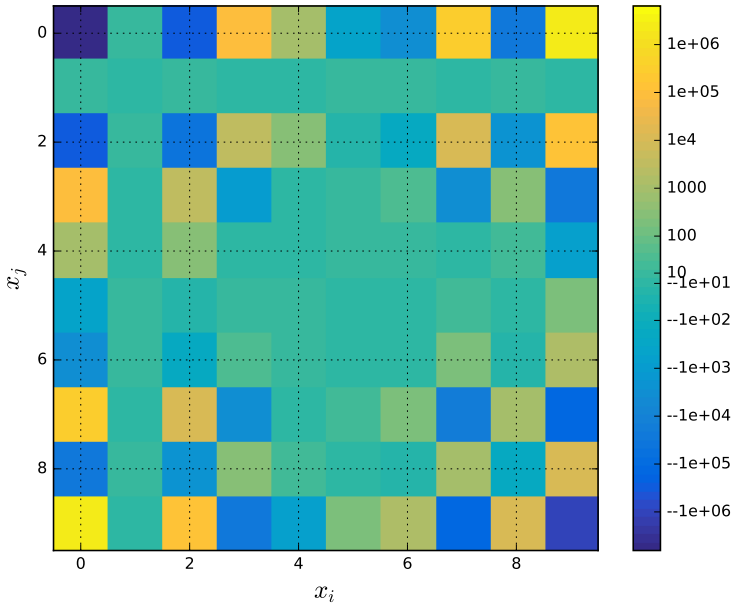


Figure 21: A Hessian SR1 approximation of the objective function at the optimum point of the SSBJ problem.

4.5. MDO formulations scalability benchmark

Multiple MDO formulations may be used to solve a given MDO problem. However, their performance vary depending on the addressed problem. The a priori selection of the most appropriate one is not straightforward. To this aim, GEMS supports the scalable MDO problem methodology proposed by Vanaret et al.[6].

This methodology takes a reference set of disciplines and builds surrogate versions of them, using a small DOE. These surrogates have a particular property that allows to change the dimensions of the inputs and output variables as desired. Combined with the automated MDO process generation capabilities of GEMS, one can then assess the relative scalability of the formulations with respect to the dimensions of the problem, without having to solve the real and computationally expensive MDO problem.

4.6. Wrapping the disciplines

To interface a simulation program with GEMS, two options are available.

1. Write a generic GEMS wrapper for any software interfaced with a given workflow engine. This option is the best if the workflow engine provides required services, such as HPC access, working directory management, post-processing, or if you want to easily chain multiple programs.

2. Write a specific wrapper for the software by directly inheriting from GEMS' `MDODiscipline` class in Python (see Section 3). This option is possible if you have no workflow engine, or if you have a light and pure in-memory code that does not require a lot of file processing or multi-machine access.

Both options can be mixed to build MDO processes.

JSON schemas[33] are used to describe the inputs, outputs and configuration parameters of most of GEMS components (disciplines, algorithms, visualizations). The JSON schema is a standard for the description of data structures (similar to the XML schema), typically used in web forms. This technology is implemented in most programming languages, which facilitates the development of GUIs to fill data forms and check the data provided by the user.

4.7. Advanced Multidisciplinary analyses

4.7.1. Available algorithms

GEMS has a series of MDA algorithms :

- Gauss-Seidel
- Jacobi with parallelism
- Newton and 10 quasi-newton variants from Scipy.
- A generic way to hybridize MDA methods
- A generic way of decomposing MDAs into sub-MDAs

In order to decompose a large coupled problem into multiple sub-coupling problems, the disciplines dependencies are analysed based on the inputs and outputs naming conventions. A directed graph of dependency is created, similar to the N2 matrix. On this graph, two algorithms are applied. First, the Tarjan algorithm[34] determines the strongly connected disciplines. Then, a Kahn's topological sort algorithm[35] determines the steps that may be run in parallel. The sub MDAs may be solved by any of the previously listed algorithms. Coupled derivatives may be computed using the discrete adjoint, as explained in the next section. The overall procedure is summarized in Figure 22.

4.7.2. Coupled derivatives

Gradient-based algorithms require the computation of the total derivatives of the output function $\boldsymbol{f} = (f_0, \dots, f_N)^T$ with respect to the design vector $\boldsymbol{x} = (x_0, \dots, x_n)^T$. These derivatives are computing while respecting the state equations of the system : $\boldsymbol{\mathcal{R}}(\boldsymbol{\mathcal{Y}}, \boldsymbol{x}) = 0$, where $\boldsymbol{\mathcal{R}}$ is the whole residual vector of the coupled system and $\boldsymbol{\mathcal{Y}}$ is the whole vector of coupling variables.

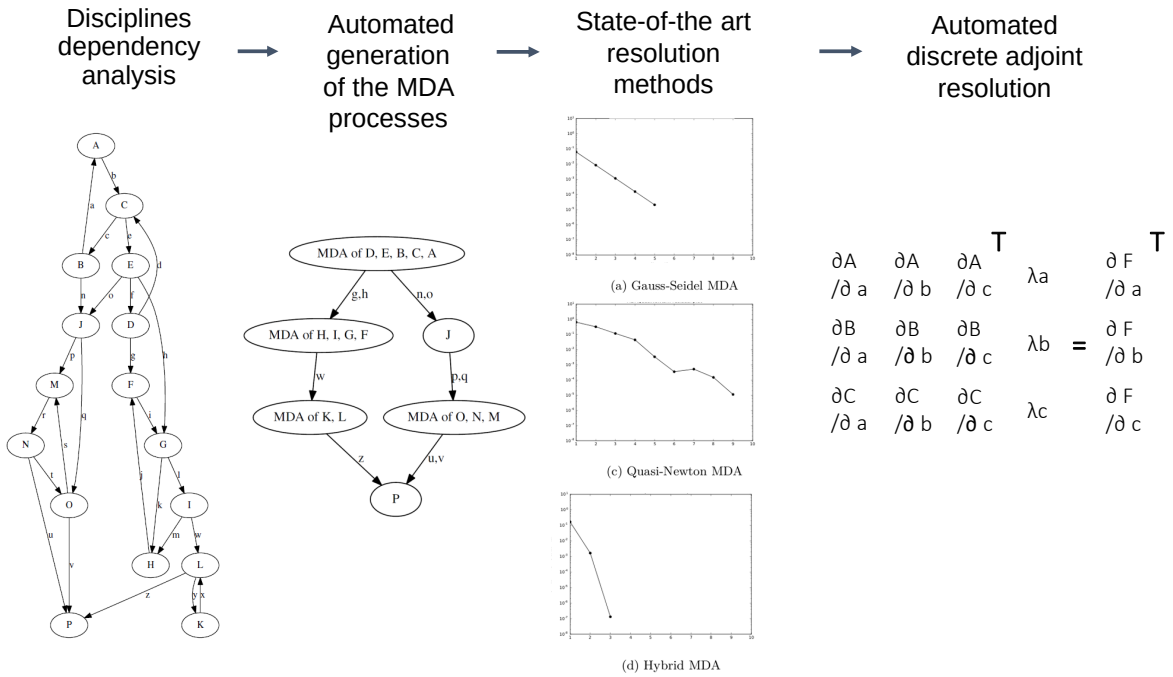


Figure 22: Generation and resolution of an MDA process.

$$\frac{d\boldsymbol{f}}{d\boldsymbol{x}} = \begin{pmatrix} \frac{df_0}{dx_0} & \cdots & \frac{df_0}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{df_N}{dx_0} & \cdots & \frac{df_N}{dx_n} \end{pmatrix}. \tag{1}$$

GEMS provides several features to compute the derivatives of the objective function and constraints: semi-analytical coupled derivatives, finite differences and complex step[36,37]. The semi-analytical derivatives allow significant time savings and higher precision compared to the other options, but it requires that the interfaced disciplines provide the derivatives of their outputs with respect to their inputs, which is not always the case.

When applying finite differences or the complex step method to an MDA (monolithic differentiation), an MDA has to be solved for each perturbed point $(\boldsymbol{x} + h_j \boldsymbol{e}_j)$:

$$\frac{df_i}{dx_j} = \frac{f_i(\boldsymbol{x} + h_j \boldsymbol{e}_j) - f_i(\boldsymbol{x})}{h_j} + \mathcal{O}(h_j). \tag{2}$$

If the size of the design vector is large, computing the sensitivity of the output $\boldsymbol{f}$ with respect to the design vector $\boldsymbol{x}$ becomes tedious.

The derivatives computations in GEMS is based on the discrete adjoint method.

- Direct method: $\dfrac{d\boldsymbol{\mathcal{Y}}}{d\boldsymbol{x}}$ is computed by solving a linear system:

$$\frac{d\boldsymbol{f}}{d\boldsymbol{x}} = -\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\mathcal{Y}}} \cdot \underbrace{\left[\left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\mathcal{Y}}} \right)^{-1} \cdot \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}} \right]}_{-d\boldsymbol{\mathcal{Y}}/d\boldsymbol{x}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}. \tag{3}$$

- Adjoint method: the computation of the adjoint vector $\boldsymbol{\lambda}$ is independent of the design variable vector $\boldsymbol{x}$:

$$\frac{d\boldsymbol{f}}{d\boldsymbol{x}} = -\underbrace{\left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\mathcal{Y}}} \cdot \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\mathcal{Y}}}\right)^{-1}\right]}_{\boldsymbol{\lambda}^T} \cdot \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = -\boldsymbol{\lambda}^T \cdot \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}. \tag{4}$$

The choice of the method (direct or adjoint) depends on how the number of design variables n compares to the size of the function (objective and constraints) vector N: if $N \ll n$, the adjoint method should be used, whereas the direct method should be preferred if $n \ll N$.
Both direct and adjoint methods are implemented in GEMS.

5. Conclusion

A Python library, GEMS, has been created to automatically build MDO processes, based on a range of MDO formulations, in particular bi-level formulations. The use of abstract concepts makes the process creation fully generic, which on one hand facilitates the construction of nested processes that are key in the implementation of multi-level MDO processes, and on the other hand makes possible to separate the MDO scenario from existing sub-workflows then managed through standard interfaces. Regarding the MDO workflow itself, no workflow engine is needed to implement it. Instead, Python itself is used to build the MDO workflow and dataflow. This provides a high level of flexibility to the process creation, and offers the capability to easily manipulate inputs and outputs, for instance in the purpose to assemble derivatives or to reconfigure the process.

This new software opens interesting perspectives for complex processes creation that embed a high number of disciplines and several levels of sub-processes, either for the needs of future projects, or for higher-fidelity processes.

Acknowledgments

The authors wish to acknowledge the PIA framework (CGI, ANR) and the members of Technology IRT Saint Exupéry for their support, financial funding and expertise.

References

[1]Gazaix, A., Gallard, F., Gachelin, V., Ambert, V., Pauwels, B., Sarouille, P., Guénot, D., Bartoli, N., Desfachelles, N., Hamadi, M., Druot, T., Lafage, R., Lefebvre, T., Brezillon, J., and Gurol, S., "Towards the Industrialization of New MDO Methodologies and Tools for Aircraft Design," AIAA AVIATION Forum, American Institute of Aeronautics and Astronautics, Jun 2017.

[2]Walt, S. v. d., Colbert, S. C., and Varoquaux, G., "The NumPy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, Vol. 13, No. 2, 2011, pp. 22–30.

[3]Jones, E., Oliphant, T., and Peterson, P., "{SciPy}: open source scientific tools for {Python}," 2014.

[4]Hunter, J. D., "Matplotlib: A 2D graphics environment," *Computing In Science & Engineering*, Vol. 9, No. 3, 2007, pp. 90–95.

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

[6]Vanaret, C., Gallard, F., and Martins, J., "On the Consequences of the "No Free Lunch" Theorem for Optimization on the Choice of an Appropriate MDO Architecture," AIAA AVIATION Forum, American Institute of Aeronautics and Astronautics, Jun 2017.

[7]Johnson, F. T., Tinoco, E. N., and Yu, N. J., "Thirty years of development and application of CFD at Boeing Commercial Airplanes, Seattle," *Computers & Fluids*, Vol. 34, No. 10, 2005, pp. 1115–1151.

[8]Meaux, M., Cormery, M., and Voizard, G., "Viscous aerodynamic shape optimization based on the discrete adjoint state for 3d industrial configurations," *4th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS), Jyväskylä, Finland, July*, 2004, pp. 24–28.

[9]Grihon, S., Krog, L., and Bassir, D., "Numerical Optimization applied to structure sizing at AIRBUS: A multi-step process," *International Journal for Simulation and Multidisciplinary Design Optimization*, Vol. 3, No. 4, 2009, pp. 432–442.

[10]Sobieszczanski-Sobieski, J., Agte, J. S., and Jr., R. R. S., "Bi-Level Integrated System Synthesis (BLISS)," Tech. Rep. TM-1998-208715, NASA, Langley Research Center, 1998.

[11]Gray, J., Moore, K., and Naylor, B., "OpenMDAO: An open source framework for multidisciplinary analysis and optimization," *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, 2010, p. 9101.

[12]Gray, J. S., Hearn, T. A., Moore, K. T., Hwang, J., Martins, J., and Ning, A., "Automatic evaluation of multidisciplinary

derivatives using a graph-based problem formulation in OpenMDAO," *15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2014, p. 2042.

[13]van Gent, I., La Rocca, G., and Veldhuis, L. L., "Composing MDAO symphonies: graph-based generation and manipulation of large multidisciplinary systems," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017, p. 3663.

[14]Phoenix Integration, "ModelCenter 9.0 User's Manual," 2009.

[15]Dassault Systèmes, "SIMULIA Isight 5.7 User's Guide," 2012.

[16]Meaux, M., *Amélioration d'une chaine d'optimisation numérique de formes aérodynamiques en vue du traitement de configurations complexes en écoulements visqueux*, Ph.D. thesis, MA Thesis, ENSICA, Toulouse, 2001.

[17]Gallard, F., *Optimisation de forme d'un avion pour sa performance sur une mission*, Ph.D. thesis, École Doctorale Aéronautique-Astronautique (Toulouse); 142618039, 2014.

[18]"Glue It All Together With Python," https://www.python.org/doc/essays/omg-darpa-mcc-position/.

[19]Eclipse Foundation, "Eclipse Rich Client Platform," http://www.eclipse.org, 2007.

[20]Defoort, S., Balesdent, M., Klotz, P., Schmollgruber, P., Morio, J., Hermetz, J., Blondeau, C., Carrier, G., and Bérend, N., "Multidisciplinary Aerospace System Design: Principles, Issues and Onera Experience." *AerospaceLab*, , No. 4, 2012, pp. p–1.

[21]Gratton, S., Pauwels, B., and Zhang, Z., "New multidisciplinary optimization approaches based on domain decomposition," presented at the Optimization 2017 conference in Lisbon, September 2017.

[22]Barjhoux, P.-J., Grihon, S., Morlier, J., Diouane, Y., and Bettebghor, D., "Mixed Variable Structural Optimization: toward an Efficient Hybrid Algorithm," *World Congress in Structure and Multidisciplinary Optimization, WCSMO*, Vol. 12, pp. 5–9.

[23]Lambe, A. B. and Martins, J. R., "Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes," *Structural and Multidisciplinary Optimization*, Vol. 46, No. 2, 2012, pp. 273–284.

[24]"XDSMjs," https://github.com/OneraHub/XDSMjs, Accessed: 2017-06-09.

[25]Wolpert, D. H. and Macready, W. G., "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, Vol. 1, No. 1, 1997, pp. 67–82.

[26]Johnson, S. G., "The NLopt nonlinear-optimization package," 2014.

[27]Kroshko, D., "OpenOpt: Free scientific-engineering software for mathematical modeling and optimization," *URL http://www. openopt. org*, 2007.

[28]Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM review*, Vol. 47, No. 1, 2005, pp. 99–131.

[29]Andrianov, G., Burriel, S., Cambier, S., Dutfoy, A., Dutka-Malen, I., De Rocquigny, E., Sudret, B., Benjamin, P., Lebrun, R., Mangeant, F., et al., "Open TURNS, an open source initiative to Treat Uncertainties, Risks' N Statistics in a structured industrial approach," *Proceedings of the ESREL'2007 Safety and Reliability Conference, Stavenger: Norway*, 2007.

[30]Kohonen, T., Schroeder, M., and Huang, T., "editors. Self-Organizing Maps," 2001.

[31]Kumano, T., Jeong, S., Obayashi, S., Ito, Y., Hatanaka, K., and Morino, H., "Multidisciplinary design optimization of wing shape for a small jet aircraft using kriging model," *44th AIAA Aerospace Sciences Meeting and Exhibit*, 2006, p. 932.

[32]Nocedal, J. and Wright, S. J., "Numerical optimization 2nd," 2006.

[33]Galiegue, F., Zyp, K., et al., "JSON Schema: Core definitions and terminology," *Internet Engineering Task Force (IETF)*, 2013.

[34]Tarjan, R., "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, Vol. 1, No. 2, 1972, pp. 146–160.

[35]Kahn, A. B., "Topological Sorting of Large Networks," *Commun. ACM*, Vol. 5, No. 11, Nov. 1962, pp. 558–562.

[36]Squire, W. and Trapp, G., "Using Complex Variables to Estimate Derivatives of Real Functions," *SIAM Rev.*, Vol. 40, No. 1, March 1998, pp. 110–112.

[37]Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262.

A. GEMS SSBJ MDF optimization log

```
*** Start MDO Scenario execution ***
MDOScenario:
Disciplines: SobieskiStructure SobieskiPropulsion SobieskiAerodynamics SobieskiMission
MDOFormulation: MDFFormulation
Algorithm: SLSQP

Optimization problem:
      Minimize: -y_4(x_shared, x_1, x_2, x_3)
With respect to:
    x_shared, x_1, x_2, x_3
Subject to constraints:
g_1(x_shared, x_1, x_2, x_3) <= 0
g_2(x_shared, x_1, x_2, x_3) <= 0
g_3(x_shared, x_1, x_2, x_3) <= 0
Design Space:
+----------+-------------+---------+-------------+---------+
| name     | lower_bound | value   | upper_bound | type    |
+----------+-------------+---------+-------------+---------+
| x_shared |      0.01   |  0.05   |     0.09    | float   |
| x_shared |     30000   | 45000   |    60000    | float   |
```

```
| x_shared |      1.4      |   1.6   |      1.8      | float |
| x_shared |      2.5      |   5.5   |      8.5      | float |
| x_shared |       40      |    55   |       70      | float |
| x_shared |      500      |   1000  |      1500     | float |
| x_1      |      0.1      |   0.25  |      0.4      | float |
| x_1      |      0.75     |    1    |      1.25     | float |
| x_2      |      0.75     |    1    |      1.25     | float |
| x_3      |      0.1      |   0.5   |       1       | float |
+----------+---------------+---------+---------------+-------+
Optimization: |               | 0/5   0% [0 iters/sec]
Optimization: |####           | 2/5  40% [14.16 iters/sec obj: 2117.85 ]
Optimization: |########       | 4/5  80% [14.52 iters/sec obj: 3964.45 ]
Optimization: |##########| 5/5 100% [14.51 iters/sec obj: 3963.44 ]
Optimization result:
Objective value = 3963.43637476
The result is feasible.
Optimizer message: Maximum number of iterations reached, GEMS stopped the solver

Design Space:
+----------+---------------+--------------------+---------------+-------+
| name     | lower_bound   |       value        | upper_bound   | type  |
+----------+---------------+--------------------+---------------+-------+
| x_shared |      0.01     | 0.06000064798049083|      0.09     | float |
| x_shared |     30000     |       60000        |     60000     | float |
| x_shared |      1.4      |        1.4         |      1.8      | float |
| x_shared |      2.5      |        2.5         |      8.5      | float |
| x_shared |       40      |         70         |       70      | float |
| x_shared |      500      |        1500        |      1500     | float |
| x_1      |      0.1      |        0.4         |      0.4      | float |
| x_1      |      0.75     |        0.75        |      1.25     | float |
| x_2      |      0.75     |        0.75        |      1.25     | float |
| x_3      |      0.1      |  0.1562539883512961|       1       | float |
+----------+---------------+--------------------+---------------+-------+
*** MDO Scenario run terminated in 0:00:00.355815 ***
Total number of executions 155
Total number of linearizations 20
```

B. GEMS SSBJ IDF optimization log

```
*** Start MDO Scenario execution ***
MDOScenario:
Disciplines: SobieskiPropulsion SobieskiAerodynamics SobieskiMission SobieskiStructure
MDOFormulation: IDFFormulation
Algorithm: SLSQP

Optimization problem:
      Minimize: -y_4(x_shared, y_14, y_24, y_34)
With respect to:
    x_shared, x_1, x_2, x_3, y_14, y_32, y_31, y_24, y_34, y_23, y_21, y_12
Subject to constraints:
y_31_y_32_y_34(x_shared, x_3, y_23) = y_31(x_shared, x_3, y_23) - y_31 = 0
                                      y_32(x_shared, x_3, y_23) - y_32 = 0
                                      y_34(x_shared, x_3, y_23) - y_34 = 0
y_24_y_23_y_21(x_shared, x_2, y_32, y_12) = y_24(x_shared, x_2, y_32, y_12) - y_24 = 0
                                            y_23(x_shared, x_2, y_32, y_12) - y_23 = 0
                                            y_21(x_shared, x_2, y_32, y_12) - y_21 = 0
y_12_y_14(x_shared, x_1, y_31, y_21) = y_12(x_shared, x_1, y_31, y_21) - y_12 = 0
                                       y_14(x_shared, x_1, y_31, y_21) - y_14 = 0
g_1(x_shared, x_1, y_31, y_21) <= 0
g_2(x_shared, x_2, y_32, y_12) <= 0
g_3(x_shared, x_3, y_23) <= 0
Design Space:
+----------+---------------+--------------------+---------------+-------+
| name     | lower_bound   |       value        | upper_bound   | type  |
+----------+---------------+--------------------+---------------+-------+
| x_shared |      0.01     |        0.05        |      0.09     | float |
| x_shared |     30000     |       45000        |     60000     | float |
| x_shared |      1.4      |        1.6         |      1.8      | float |
| x_shared |      2.5      |        5.5         |      8.5      | float |
| x_shared |       40      |         55         |       70      | float |
| x_shared |      500      |        1000        |      1500     | float |
| x_1      |      0.1      |        0.25        |      0.4      | float |
| x_1      |      0.75     |         1          |      1.25     | float |
| x_2      |      0.75     |         1          |      1.25     | float |
| x_3      |      0.1      |        0.5         |       1       | float |
| y_14     |     24850     |    50606.9741711   |     77100     | float |
| y_14     |     -7700     |    7306.20262124   |     45000     | float |
| y_32     |     0.235     | 0.5027962499999999 |     0.795     | float |
| y_31     |      2960     |    6354.32430691   |     10185     | float |
| y_24     |      0.44     |      4.15006276    |     11.13     | float |
| y_34     |      0.44     |      1.10754577    |      1.98     | float |
| y_23     |      3365     |    12194.2671934   |     26400     | float |
```

```
| y_21      |    24850    |   50606.9741711    |    77250    | float |
| y_12      |    24850    |     50606.9742     |    77250    | float |
| y_12      |     0.45    |        0.95        |     1.5     | float |
+-----------+-------------+--------------------+-------------+-------+
Optimization: |           | 0/17   0% [0 iters/sec]
Optimization: |#####      | 8/17  47% [ 76.90 iters/sec obj: 4010.11 ]
Optimization: |##########| 16/17  94% [86.66 iters/sec obj: 3963.39 ]
Optimization result:
Objective value = 3963.38769233
The result is feasible.
Status: 8
Optimizer message: Positive directional derivative for linesearch
Number of calls to the objective function by the optimizer: 17

Design Space:
+-----------+-------------+--------------------+-------------+-------+
| name      | lower_bound |       value        | upper_bound | type  |
+-----------+-------------+--------------------+-------------+-------+
| x_shared  |     0.01    | 0.05999996023996285|     0.09    | float |
| x_shared  |    30000    |       60000        |    60000    | float |
| x_shared  |     1.4     |        1.4         |     1.8     | float |
| x_shared  |     2.5     |        2.5         |     8.5     | float |
| x_shared  |     40      |         70         |     70      | float |
| x_shared  |     500     |  1499.999526920426 |     1500    | float |
| x_1       |     0.1     |  0.3999980739053585|     0.4     | float |
| x_1       |     0.75    |  0.7500001939672997|     1.25    | float |
| x_2       |     0.75    |        0.75        |     1.25    | float |
| x_3       |     0.1     |  0.1562445549478446|      1      | float |
| y_14      |    24850    |  44749.79881384457 |    77100    | float |
| y_14      |    -7700    |  19350.53402189886 |    45000    | float |
| y_32      |     0.235   |  0.7328134577326895|     0.795   | float |
| y_31      |     2960    |  9437.36650481347  |    10185    | float |
| y_24      |     0.44    |  8.057487174891243 |    11.13    | float |
| y_34      |     0.44    |  0.9239326841166937|     1.98    | float |
| y_23      |     3365    |  5553.812561801911 |    26400    | float |
| y_21      |    24850    |  44749.79881384457 |    77250    | float |
| y_12      |    24850    |  44749.79881384457 |    77250    | float |
| y_12      |     0.45    |  0.9027888895408732|     1.5     | float |
+-----------+-------------+--------------------+-------------+-------+
*** MDO Scenario run terminated in 0:00:00.203204 ***
Total number of executions calls 64
Total number of linearizations 24
```

C. GEMS SSBJ Bi-Level optimization log

```
*** Start MDO Scenario execution ***
MDOScenario:
Disciplines: MDOScenario MDOScenario MDOScenario SobieskiMission
MDOFormulation: BiLevelMDOFormulation
Algorithm: NLOPT_COBYLA

Optimization problem:
      Minimize: -y_4(x_shared)
With respect to:
    x_shared
Subject to constraints:
g_1_g_2_g_3(x_shared) <= 0
Design Space:
+-----------+-------------+---------+-------------+-------+
| name      | lower_bound |  value  | upper_bound | type  |
+-----------+-------------+---------+-------------+-------+
| x_shared  |     0.01    |   0.05  |     0.09    | float |
| x_shared  |    30000    |  45000  |    60000    | float |
| x_shared  |     1.4     |   1.6   |     1.8     | float |
| x_shared  |     2.5     |   5.5   |     8.5     | float |
| x_shared  |     40      |   55    |     70      | float |
| x_shared  |     500     |  1000   |     1500    | float |
+-----------+-------------+---------+-------------+-------+
Optimization: |           | 0/40   0% [0 iters/sec]

*** Start MDO Scenario execution ***
MDOScenario:
Disciplines: SobieskiPropulsion
MDOFormulation: DisciplinaryOptFormulation
Algorithm: SLSQP

Optimization problem:
      Minimize: y_34(x_3)
With respect to:
    x_3
Subject to constraints:
g_3(x_3) <= 0
```

```
Design Space:
+-------+-------------+-------+-------------+-------+
| name  | lower_bound | value | upper_bound | type  |
+-------+-------------+-------+-------------+-------+
| x_3   |     0.1     |  0.5  |      1      | float |
+-------+-------------+-------+-------------+-------+
Optimization: |          | 0/20   0% [0 iters/sec]
Optimization: |##        | 4/20  20% [385.28 iters/sec obj:  1.11 ]
Optimization result:
Objective value = 1.10878747393
The result is feasible.
Status: 0
Optimizer message: Optimization terminated successfully.
Number of calls to the objective function by the optimizer: 5

Design Space:
+-------+-------------+--------------------+-------------+-------+
| name  | lower_bound |       value        | upper_bound | type  |
+-------+-------------+--------------------+-------------+-------+
| x_3   |     0.1     | 0.4302702621790957 |      1      | float |
+-------+-------------+--------------------+-------------+-------+
*** MDO Scenario run terminated in 0:00:00.014960 ***

*** Start MDO Scenario execution ***
MDOScenario:
Disciplines: SobieskiAerodynamics
MDOFormulation: DisciplinaryOptFormulation
Algorithm: SLSQP

Optimization problem:
      Minimize: -y_24(x_2)
With respect to:
    x_2
Subject to constraints:
g_2(x_2) <= 0
Design Space:
+-------+-------------+-------+-------------+-------+
| name  | lower_bound | value | upper_bound | type  |
+-------+-------------+-------+-------------+-------+
| x_2   |     0.75    |   1   |     1.25    | float |
+-------+-------------+-------+-------------+-------+
Optimization: |          | 0/20   0% [0 iters/sec]
Optimization: |##        | 3/20  15% [308.36 iters/sec obj:  4.28 ]
Optimization result:
Objective value = 4.28334355888
The result is feasible.
Status: 0
Optimizer message: Optimization terminated successfully.
Number of calls to the objective function by the optimizer: 4

Design Space:
+-------+-------------+-------+-------------+-------+
| name  | lower_bound | value | upper_bound | type  |
+-------+-------------+-------+-------------+-------+
| x_2   |     0.75    |  0.75 |     1.25    | float |
+-------+-------------+-------+-------------+-------+
*** MDO Scenario run terminated in 0:00:00.013896 ***

*** Start MDO Scenario execution ***
MDOScenario:
Disciplines: SobieskiStructure
MDOFormulation: DisciplinaryOptFormulation
Algorithm: SLSQP

Optimization problem:
      Minimize: -y_11(x_1)
With respect to:
    x_1
Subject to constraints:
g_1(x_1) <= 0
Design Space:
+-------+-------------+-------+-------------+-------+
| name  | lower_bound | value | upper_bound | type  |
+-------+-------------+-------+-------------+-------+
| x_1   |     0.1     |  0.25 |     0.4     | float |
| x_1   |     0.75    |   1   |     1.25    | float |
+-------+-------------+-------+-------------+-------+
Optimization: |          | 0/20   0% [0 iters/sec]
Optimization: |####      | 7/20  35% [137.86 iters/sec obj:  0.16 ]
Optimization result:
Objective value = 0.156318238104
The result is feasible.
Status: 0
```

```
Optimizer message: Optimization terminated successfully.
Number of calls to the objective function by the optimizer: 8

Design Space:
+------+-------------+--------------------+-------------+-------+
| name | lower_bound |       value        | upper_bound | type  |
+------+-------------+--------------------+-------------+-------+
| x_1  |     0.1     | 0.1000000000000061 |     0.4     | float |
| x_1  |     0.75    |  0.985712151728832 |     1.25    | float |
+------+-------------+--------------------+-------------+-------+
```