UNIVERSIDADE
BEIRA INTERIOR

MDO in Aeronautical Engineering

Otimização em Engenharia (15235)

2º Ciclo/Mestrado em Engenharia Aeronáutica

2024

Pedro V. Gamboa

Departamento de Ciências Aeroespaciais
Faculdade de Engenharia

0. Topics

- Complex engineering problems
- Problems formulation
- Models
- Multidisciplinary Analysis
- Decomposition Architectures (Unified Description of MDO Architectures, Monolithic Architectures, Distributed Architectures, Architecture Benchmarking Issues)
- Level of Fidelity
- Decision Support

1. Introduction

- As mentioned in Chapter 1, most engineering systems are multidisciplinary, motivating the development of multidisciplinary design optimization (MDO).
- The analysis of multidisciplinary systems requires coupled models and coupled solvers.
- We prefer the term component instead of discipline or model because it is more general.
- However, we use these terms interchangeably depending on the context.
- When components in a system represent different physics, the term multiphysics is commonly used.

1. Introduction

- All the optimization methods covered so far apply to multidisciplinary problems if we view the coupled multidisciplinary analysis as a single analysis that computes the objective and constraint functions by solving the coupled model for a given set of design variables.
- However, there are additional considerations in the solution, derivative computation, and optimization of coupled systems.
- In this chapter, we introduce models and solvers for coupled systems.
- Finally, we introduce various MDO architectures, which are different options for formulating and solving MDO problems.

2. The Need for MDO

- In Chapter 1, we mentioned that MDO increases the system performance, decreases the design time, reduces the total cost, and reduces the uncertainty at a given point in time.
- Modelling and optimizing a single discipline or component provide the same benefits, but broadening the modelling and optimization to the whole system has additional benefits.
- Even without performing any optimization, constructing a multidisciplinary (coupled) model that considers the whole engineering system is beneficial.
- Such a model ideally includes all the interactions between the components of the system, considering all the physics and beyond - other potential considerations include economics and human factors.

2. The Need for MDO

- The benefit of such a model is that it reflects the true state and performance of the system when deployed in the real world, as opposed to an isolated component with assumed boundary conditions.
- Using such a model, designers can quantify the true impact of proposed changes on the whole system.
- When considering optimization, the main benefit of MDO is that optimizing the design variables for the various components simultaneously leads to a better system than when optimizing the design variables for each component separately.
- Currently, many engineering systems are designed and optimized sequentially, which leads to suboptimal designs.
- This approach is often used in industry, where engineers are grouped by discipline, physical subsystem, or both.

2. The Need for MDO

- This might be perceived as the only choice when the engineering system is too complex and the number of engineers too large to coordinate a simultaneous design involving all groups.
- Sequential optimization is analogous to coordinate descent, which consists of optimizing each variable sequentially, as shown in Fig. 7.01.
- Instead of optimizing one variable at a time, sequential optimization optimizes distinct sets of variables at a time, but the principle remains the same.
- This approach tends to work for unconstrained problems, although the convergence rate is limited to being linear.

2. The Need for MDO

- Each variable is treated as independent in this case.
- One issue with sequential optimization is that it might converge to a suboptimal point for a constrained problem.

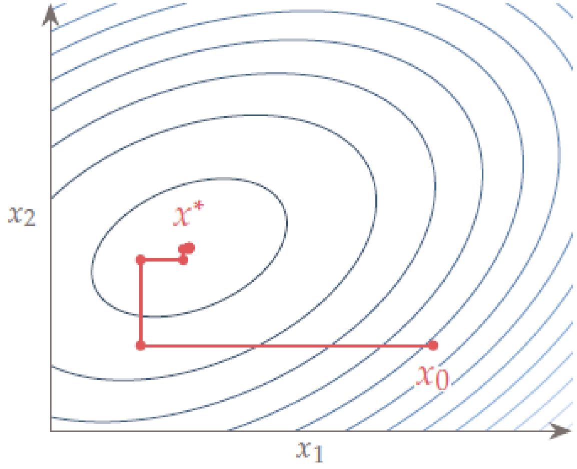


Figure 7.01 Sequential optimization is analogous to coordinate descent.

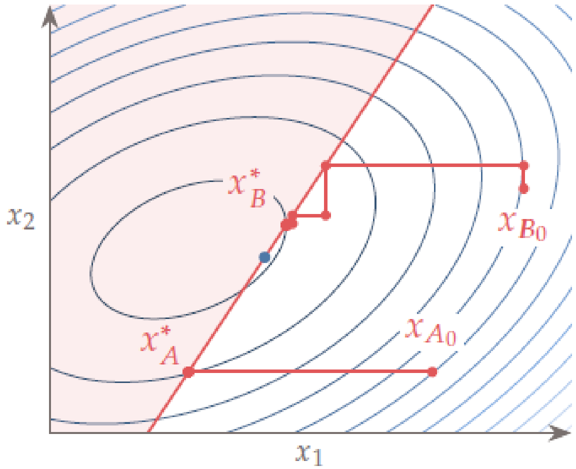


Figure 7.02 Sequential optimization can fail to find the constrained optimum because the optimization with respect to a set of variables might not see a feasible descent direction that otherwise exists when considering all variables simultaneously.

Source: Martins et Ning, 2021

2. The Need for MDO

- An example of such a case is shown in Fig. 7.02, where sequential optimization gets stuck at the constraint because it cannot decrease the objective while remaining feasible by only moving in one of the directions.
- In this case, the optimization must consider both variables simultaneously to find a feasible descent direction.
- Another issue is that when there are variables that affect multiple disciplines (called shared design variables), we must make a choice about which discipline handles those variables.
- If we let each discipline optimize the same shared variable, the optimizations likely yield different values for those variables each time, in which case they will not converge.

2. The Need for MDO

- On the other hand, if we let one discipline handle a shared variable, it will likely converge to a value that violates one or more constraints from the other disciplines.
- By considering the various components and optimizing a multidisciplinary performance metric with respect to as many design variables as possible simultaneously, MDO automatically finds the best trade-off between the components - this is the key principle of MDO.
- Suboptimal designs also result from decisions at the system level that involve power struggles between designers.
- In contrast, MDO provides the right trade-offs because mathematics does not care about politics.

2. The Need for MDO

Example 7.1: MDO applied to wing design.

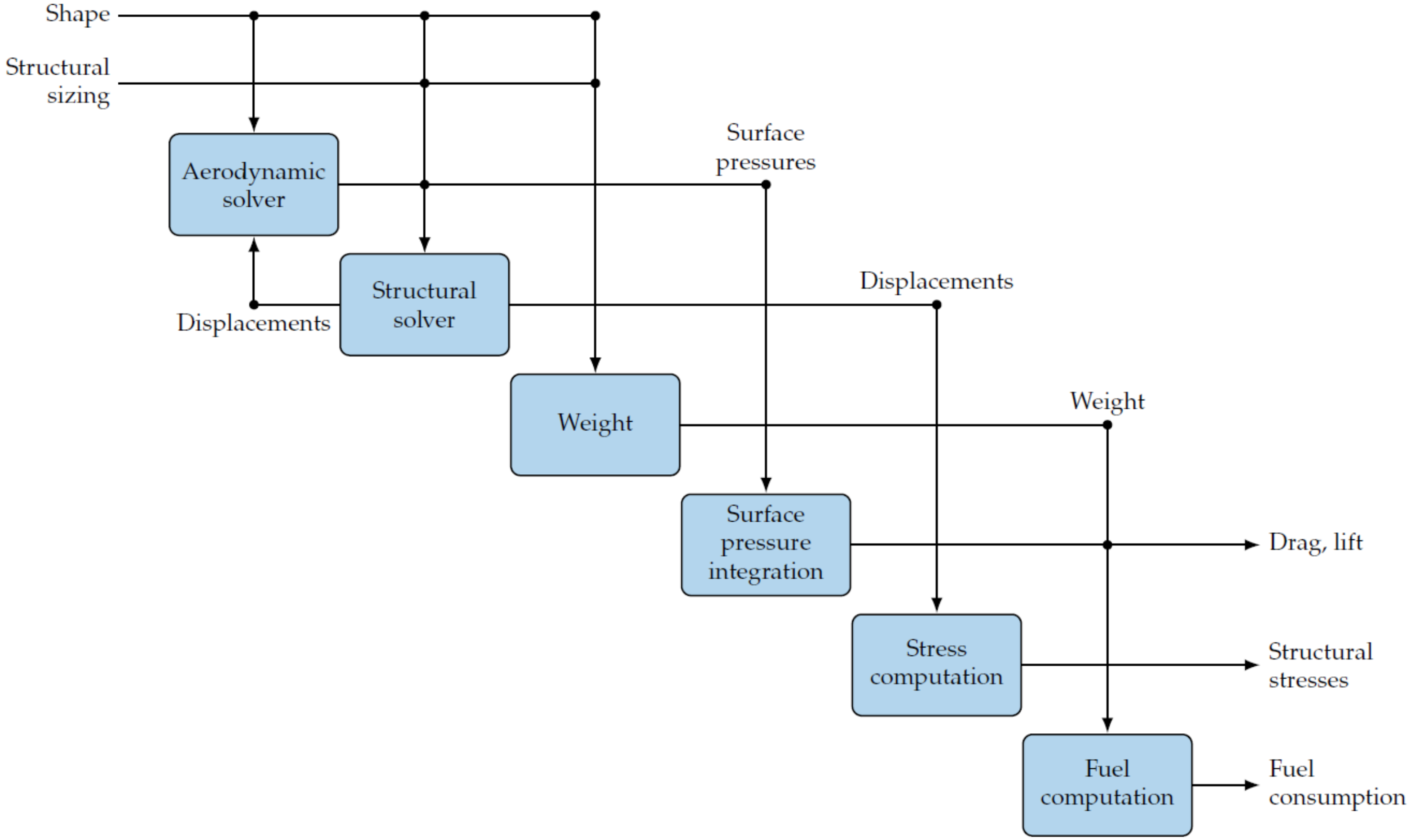


Figure 7.03 Multidisciplinary numerical model for an aircraft wing.

Source: Martins et Ning, 2021

3. Complex Engineering Problems

- The ability to develop and deploy complex engineered systems such as aircraft, automobiles, and space systems seems to be approaching a limit.
- Many large-scale engineering development programs are fraught with exorbitant cost overruns and delays due to a combination of technical, organizational, and political issues.
- There is no question that systems have become more complex and therefore increasingly more difficult to design and manage.
- While a consensus on a universal definition of complex engineered systems has not yet been reached, a defining characteristic of these systems is the emergent behaviour that current modelling and quantification methods fail to capture.
- This emergent behaviour arises from couplings in the system that we often do not understand and cannot model effectively, if at all.

3. Complex Engineering Problems

- Complex engineered systems also involve multiple disciplines, including disciplines that are still extremely difficult to quantify (e.g., disciplines involving human behaviour) and to integrate into mathematical models and optimization problems.
- When research into Multidisciplinary Design Optimization (MDO) began 35 years ago, the intention was to address these challenges and, in particular, the coupling within design hierarchies and between disciplines.
- MDO has evolved remarkably since then, and the focus of MDO has shifted dramatically, as new faculty and researchers are finding new ways to use MDO methods and tools on a wide array of problems.

3. Complex Engineering Problems

- There have been many advances to capture, represent, and propagate couplings in analysis, design, and even organizations, yet the design of complex engineered systems continues to be plagued by schedule delays and cost overruns.
- Consequently, many question the impact that MDO has had on industry and wonder what advances are needed to improve the design of complex engineered systems.

3.1. Potential for MDO

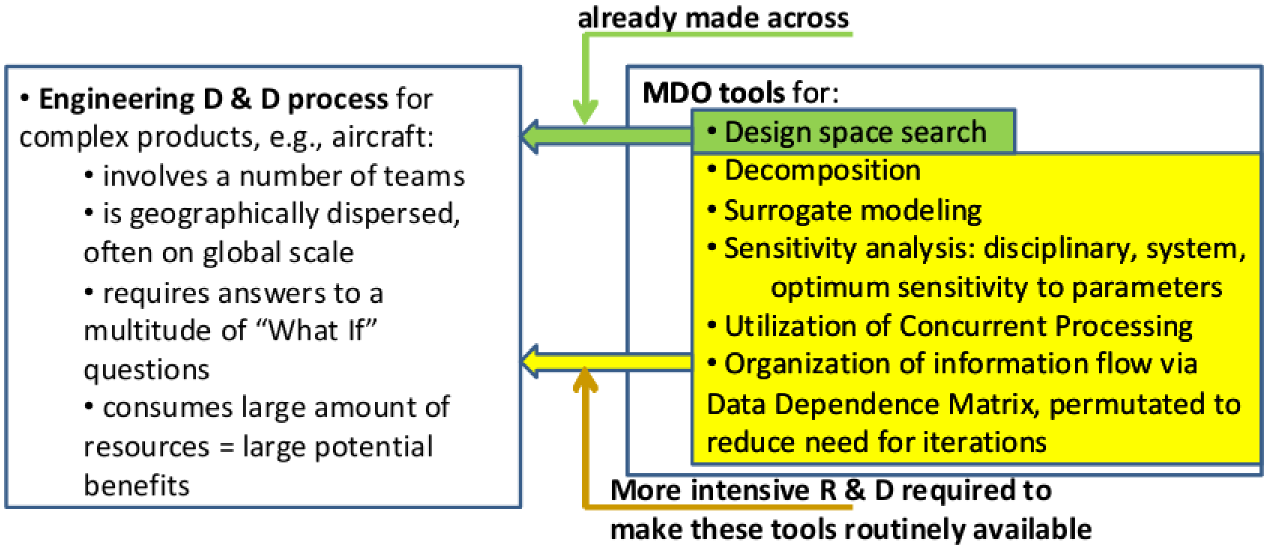
- While there are still many opportunities to improve MDO search algorithms, optimization takes up a relatively small fraction of time in the development of a project - analysis still consumes the majority of resources.
- Large gains can still be obtained by processing data concurrently through improvements in decomposition methods, surrogate modelling techniques, and taking advantage of advances in multi-processor computers, field-programmable gate arrays, and graphical processing units.
- However, inter-processor communication will limit the parallel adaption of many legacy codes that are still widely used today.
- Codes that now take several hours to run could be executed in fractions of a second, provided they are redeveloped and tailored to massively multiprocessor computers, thus rendering MDO qualitatively different.

3.1. Potential for MDO

- MDO can provide the core infrastructure for engineering design and development of complex engineered systems (see Fig. 7.04), but more research and development is needed to integrate engineering models with manufacturing and fabrication models, which still tend to be separate in many organizations.
- The benefit, however, would be unprecedented design agility within an organization, allowing it to return upstream to act on new information unimpeded by the cost of the already sunk effort.
- MDO needs to break out of its "gilded cage" and find ways to help conceive new designs, not just search the design space defined by the user.

3.1. Potential for MDO

- For example, topology optimization is a good example in that it often produces shapes that surprise designers and MDO needs to find ways to work in the function space so that it will help find innovative solutions like those often found in nature.



already made across

- **Engineering D & D process** for complex products, e.g., aircraft:
  - involves a number of teams
  - is geographically dispersed, often on global scale
  - requires answers to a multitude of "What If" questions
  - consumes large amount of resources = large potential benefits

MDO tools for:

- Design space search
- Decomposition
- Surrogate modeling
- Sensitivity analysis: disciplinary, system, optimum sensitivity to parameters
- Utilization of Concurrent Processing
- Organization of information flow via Data Dependence Matrix, permutated to reduce need for iterations

More intensive R & D required to make these tools routinely available

Figure 7.04 Potential for MDO in engineering design & development (D&D) process.

Source: Sobieski, 2010

3.2. Requirements vs Objective function

- There are challenges in the design of large-scale complex engineered systems, in particular those that arise within the hierarchical approach and requirements flow-down process that most companies use to design such systems.
- What are the advantages and disadvantages of using requirements in the flow-down process versus using objective functions (Figure 7.05)?

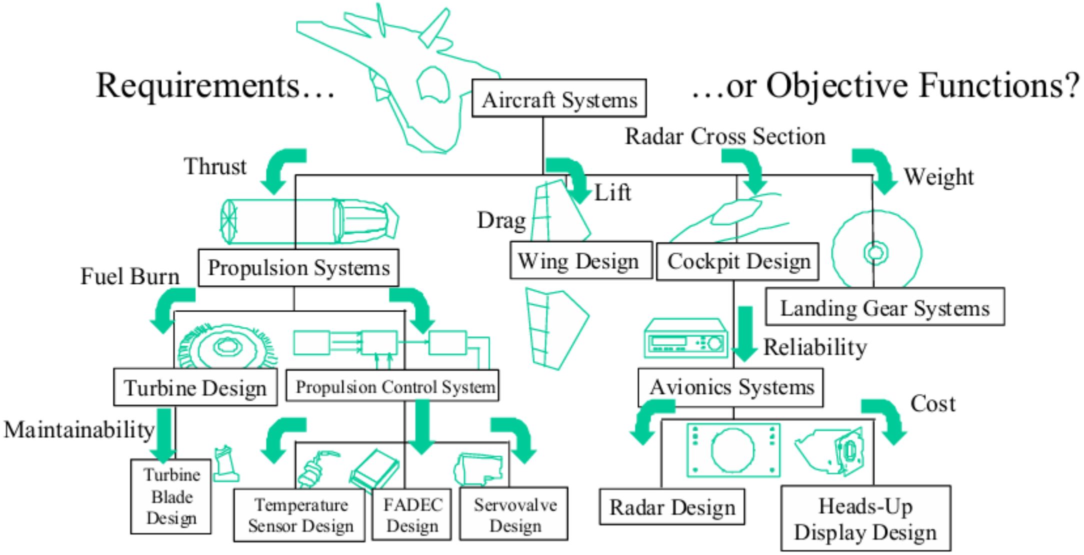


Figure 7.05 Example of requirements (left) versus objective function (right) flow down.

Source: Collopy, 2010

3.2. Requirements vs Objective function

- On the left, requirements for thrust, fuel burn, and maintainability are shown as they are typically “flowed down” in a design hierarchy.
- On the right, objective functions for weight, lift, cost, and reliability are flowed down instead, to provide more flexibility to designers.
- To further complicate matters, this hierarchy goes many layers deep (e.g., 8, 12 layers) for large-scale complex engineered systems.
- The result is that it is very difficult for engineers working at the lowest levels to see the “big picture” and understand how their decisions impact everything else.

3.3. Selecting the right tools

- In industry, teams of people are necessary to have expertise in different areas.
- Likewise, no single "tool" can be used to solve every problem; it is important to apply the right tool to the right problem at the right time.
- Understanding both of these aspects during the design of complex engineered systems is important and has led to MDO becoming a "state of mind" within their organization.
- MDO still relies on a solid foundation (e.g., aerodynamics, structures, weights, noise), but it now supports multiple needs and plays multiple roles within their organization.
- Figure 7.06, shows how the processes and tools must be matched to the level of analysis and fidelity, and be used in the right trade space at the appropriate program milestone.

3.3. Selecting the right tools

- With this capability, MDO has become valuable in industry due to its ability to integrate, automate, and explore trade spaces.
- However, there is still room for improvement.

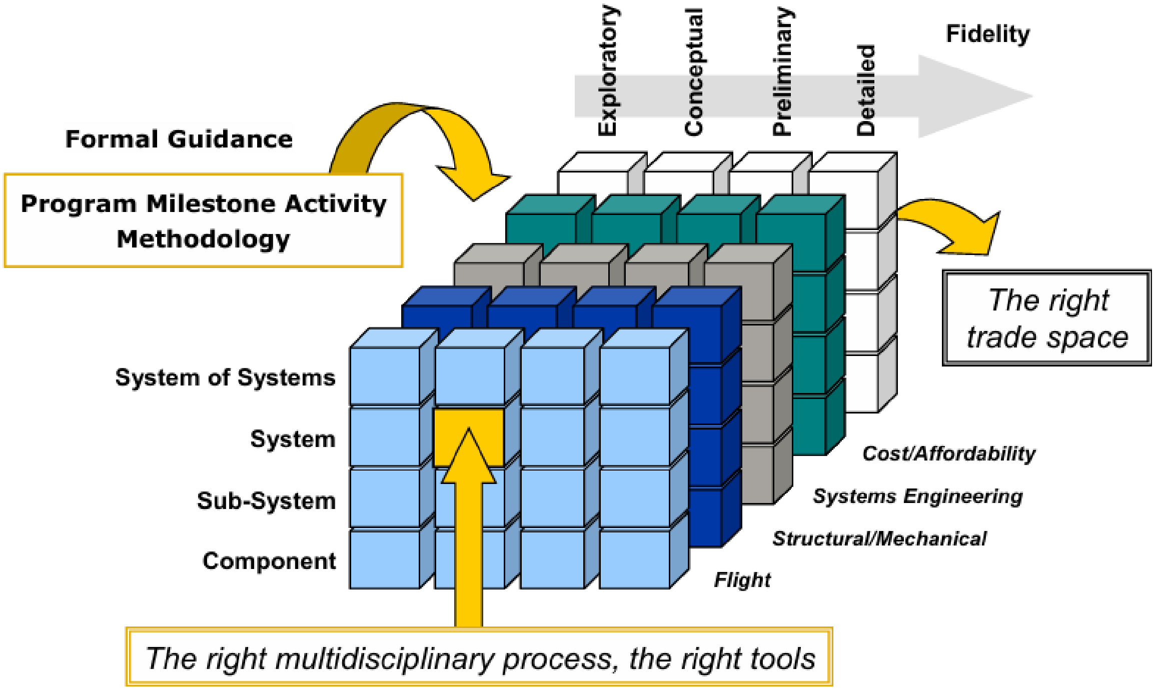


Figure 7.06 Selecting the right tool for the right problem at the right time.

Source: Cramer, 2010

3.3. Selecting the right tools

- Two areas have received a great deal of attention:
  1. Optimization, and
  2. multi-fidelity modelling.
- For optimization, there is research in problem formulation, process steps, mixed integer nonlinear programming, and multi-objective programming.
- For multi-fidelity modelling, many issues still need better understanding (e.g., getting the right fidelity for the right application, mixing and matching models at different levels of fidelity, and using several multi-fidelity models at once) and the methods are not yet mature to the point of being “commoditized”, making industry hesitant to adopt them.

3.4. Feasibility

- In the automotive industry, for example, MDO problems are typically characterized as being large-scale and multi-physics, requiring multiple models and simulation codes, having large numbers of design variables (continuous and discrete) and constraints, having highly nonlinear responses, and having computationally intensive high-fidelity models (e.g., a crash simulation).
- Vehicle disciplines are coupled through common design variables and entail analyses related to weight, safety, noise, vibration, and harshness, body structure, chassis and vehicle durability, vehicle dynamics, and thermal and aerodynamics systems engineering and climate control.
- MDO allows to search the design space and find the optimum of the feasible region among the various disciplines' domains (Fig. 7.07).

3.4. Feasibility

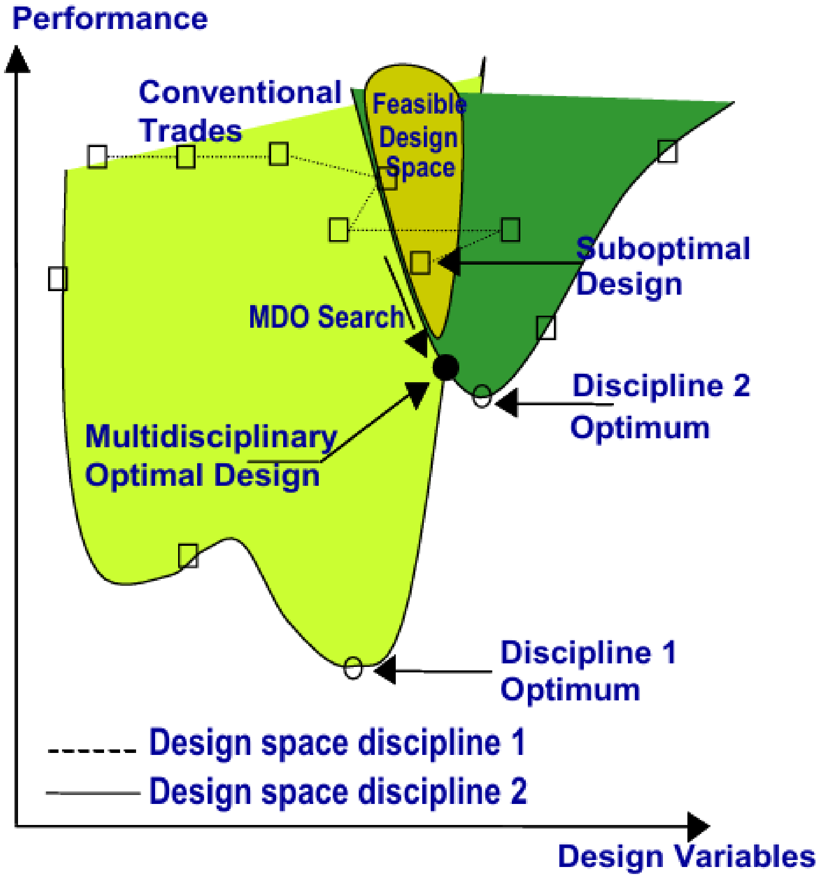


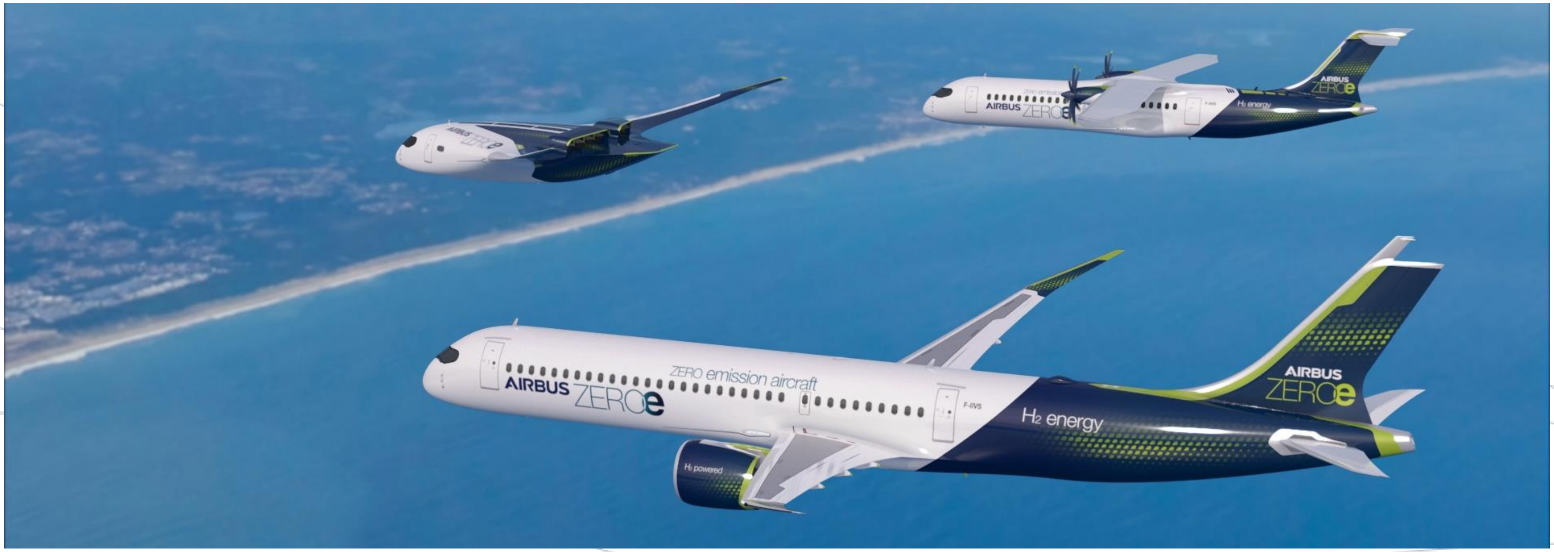
Figure 7.07 Notional depiction of the benefits of MDO.

Source: Cramer, 2010

3.5. Examples

Example 7.2: Aircraft design with minimum environmental impact.

- The environmental impact of aircraft has been a popular topic in the last few years. This example shows the trade-offs between cost, noise, and greenhouse gas emissions, by solving a number of multi-objective problems.



Source: Airbus, 2020

Figure 7.08 Airbus zero-emission concepts, BWB, turboprop and turbofan.

3. Complex Engineering Problems

3.5. Examples

Example 7.2: Aircraft design with minimum environmental impact (continued).

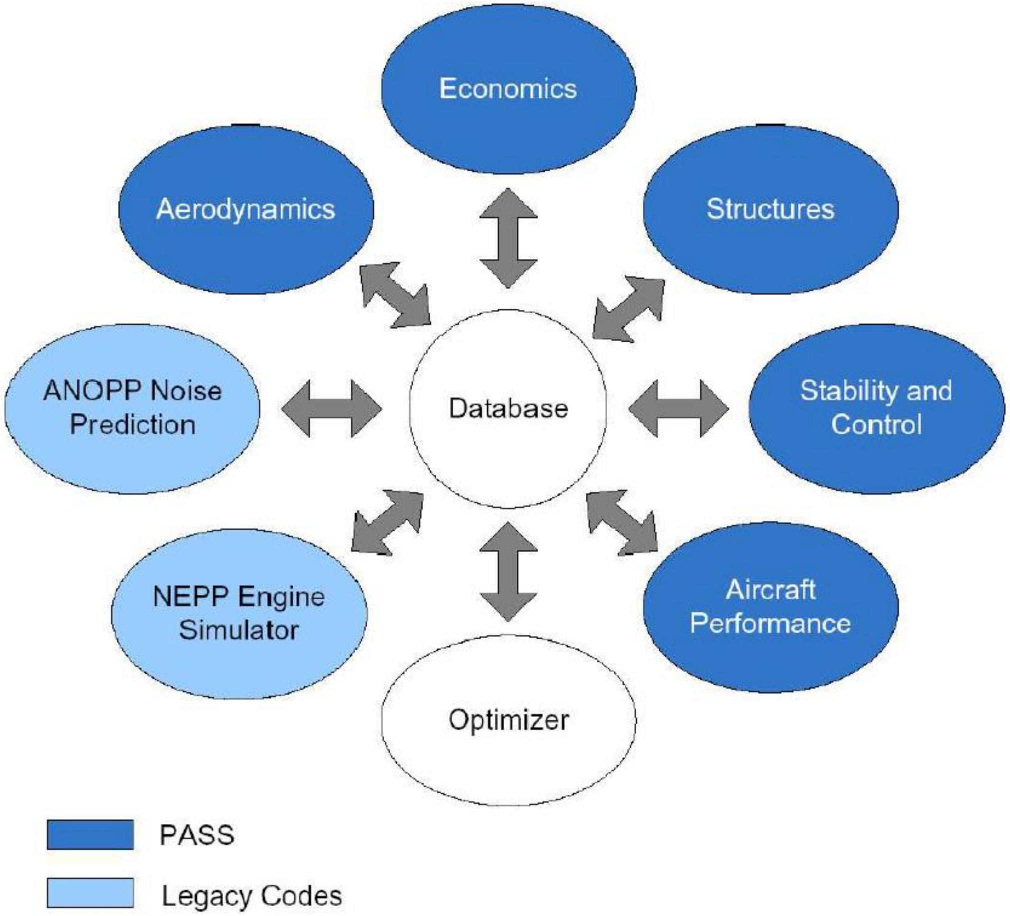


Figure 7.09 MDO tools.

Source: Antoine & Kroo, 2004

MDO in Aeronautical Engineering / MDO em Engenharia Aeronáutica

3.5. Examples

Example 7.2: Aircraft design with minimum environmental impact (continued).

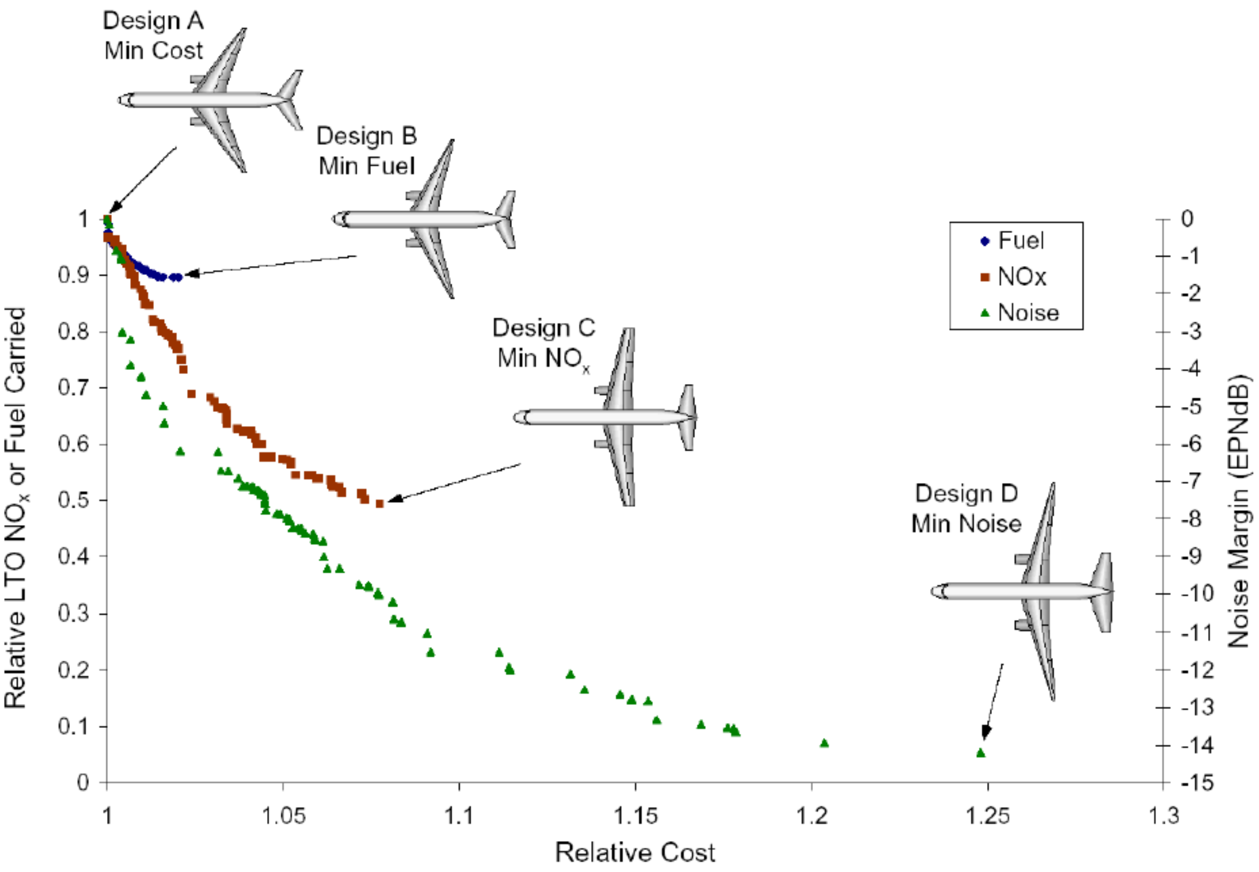
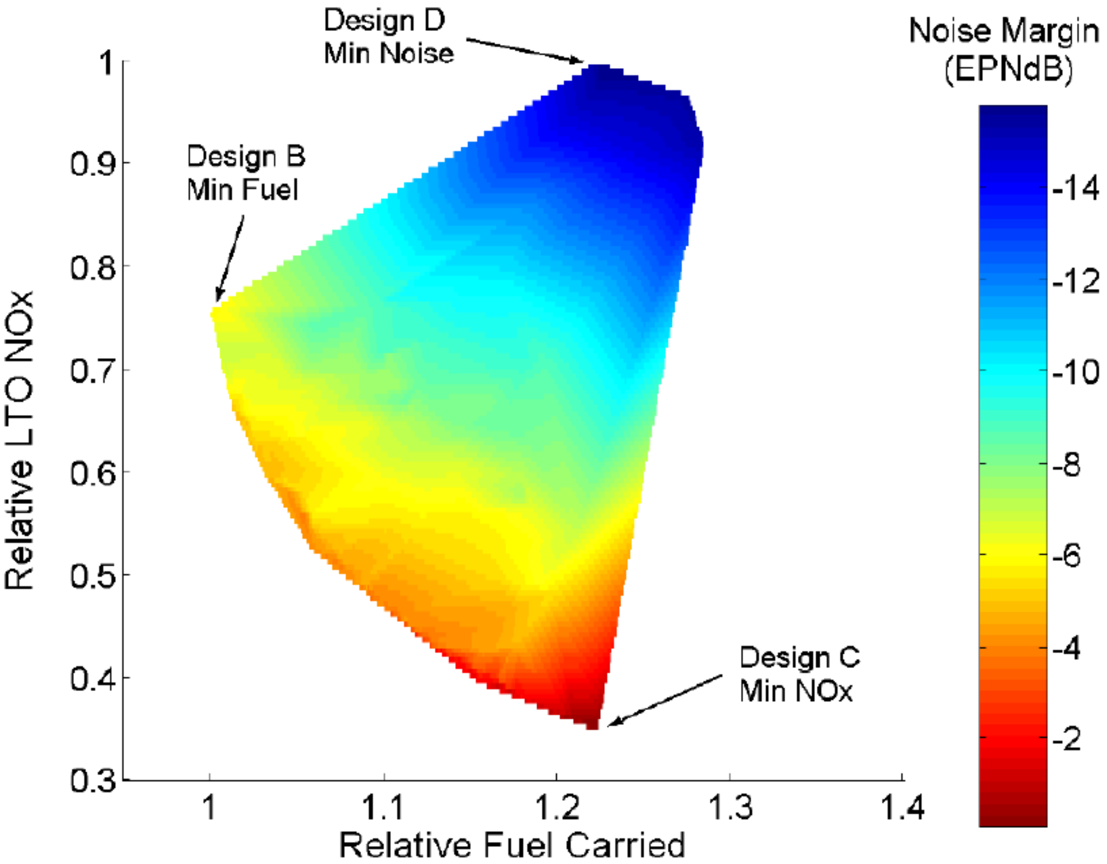


Figure 7.10 Pareto fronts of fuel carried, emissions and noise vs. operating cost.

Source: Antoine & Kroo, 2004

3.5. Examples

Example 7.2: Aircraft design with minimum environmental impact (continued).



Source: Antoine & Kroo, 2004

Figure 7.11 Pareto surface of emissions vs. fuel carried vs. noise.

3.5. Examples

Example 7.3: Aerodynamic design of a Natural Laminar Flow Business Jet.

- Cruise Mach: *M*=1,5; Cruise Altitude: 15,240 m; Payload: 6-8 passengers; Range: 9,260 km; Weight: 445,215 N; *L*/*D*=9-10; Wings designed with low sweep for natural laminar flow.

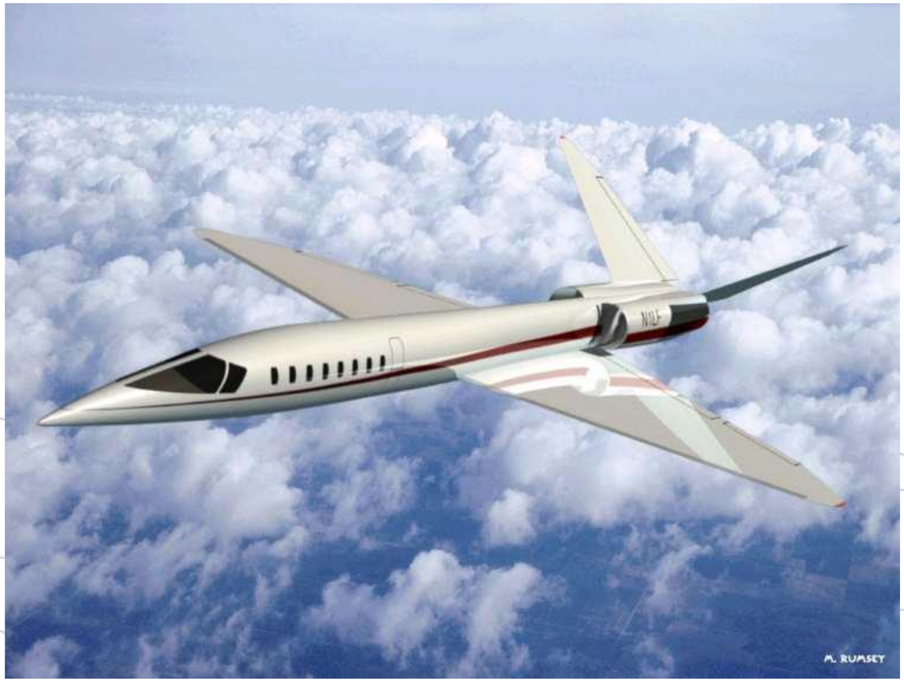


Figure 7.12 ASSET Corporation natural laminar wing flow business jet concept.

Source: Sturdza & Kroo, 2002

3.5. Examples

Example 7.3: Aerodynamic design of a Natural Laminar Flow Business Jet (continued).

- A CFD Euler code was combined with a boundary-layer solver to compute the flow on a wing-body.
- The fuselage spoils the laminar flow that can normally be maintained on a thin, low sweep wing in supersonic flow
- The goal was to reshape the fuselage at the wing-body junction to maximize the extent of laminar flow on the wing.
- Three design variables were used initially, with quadratic response surfaces and a trust region update algorithm.

3.5. Examples

Example 7.3: Aerodynamic design of a Natural Laminar Flow Business Jet (continued).

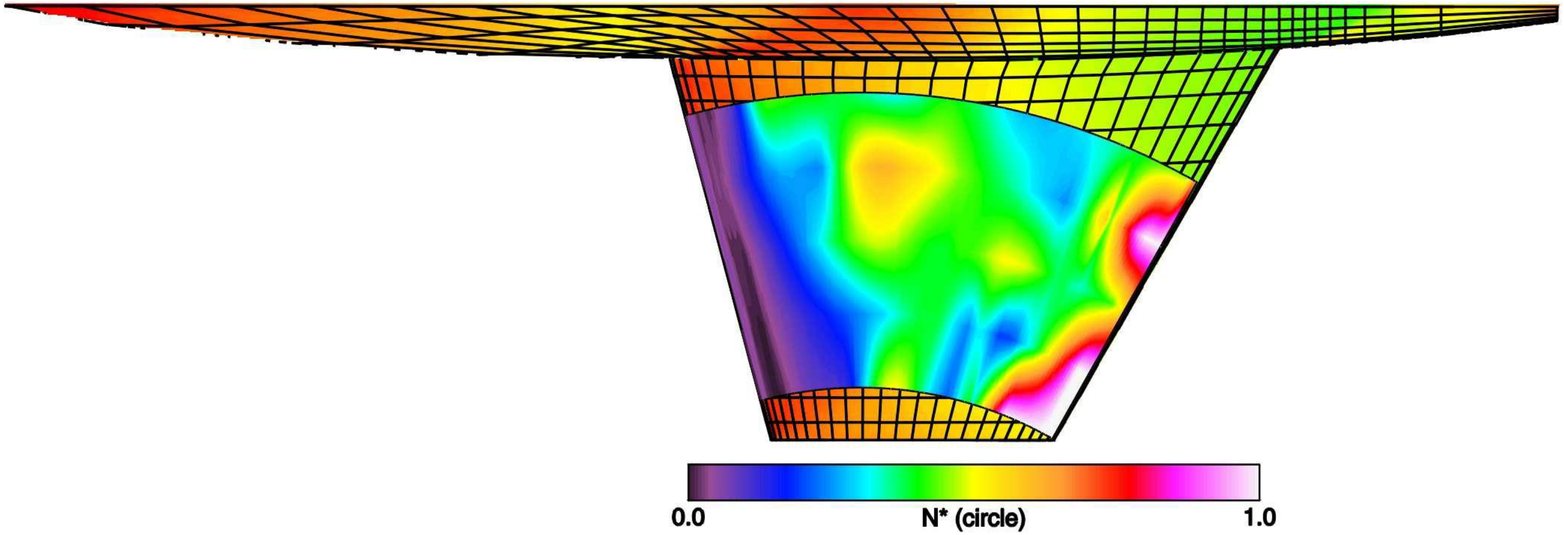


Figure 7.13 The boundary-layer solution appears superimposed on the inviscid Euler pressures on the surface grid.

Source: Sturdza & Kroo, 2002

3.5. Examples

Example 7.3: Aerodynamic design of a Natural Laminar Flow Business Jet (continued).

- Baseline design: a Sears-Haack body with wing results in early transition (the white areas in the boundary-layer solution).
- N* is the measure of laminar instability, with 1.0 (white) being the prediction of transition.
- The flow is then turbulent from the first occurrence of N*=1 to the trailing edge irrespective of further values of N*.

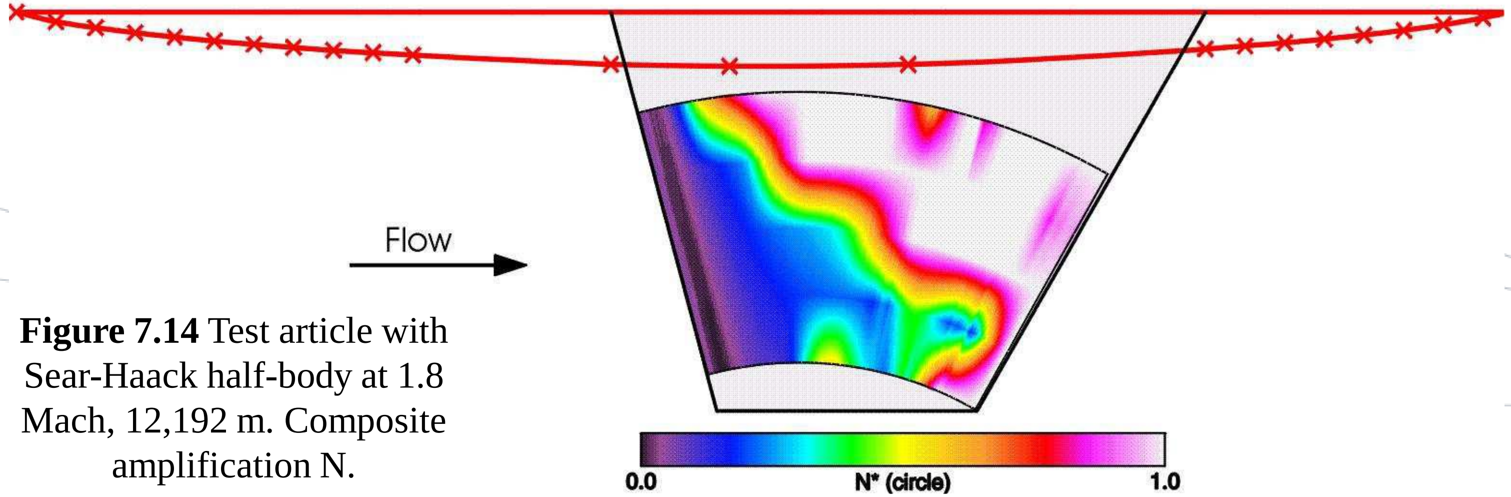


Figure 7.14 Test article with Sear-Haack half-body at 1.8 Mach, 12,192 m. Composite amplification N.

Source: Sturdza & Kroo, 2002

3.5. Examples

Example 7.3: Aerodynamic design of a Natural Laminar Flow Business Jet (continued).

- With only 3 design variables (the crosses on the fuselage outline that sit on the wing) and two iterations (not even near a converged optimization) the improvement is dramatic.

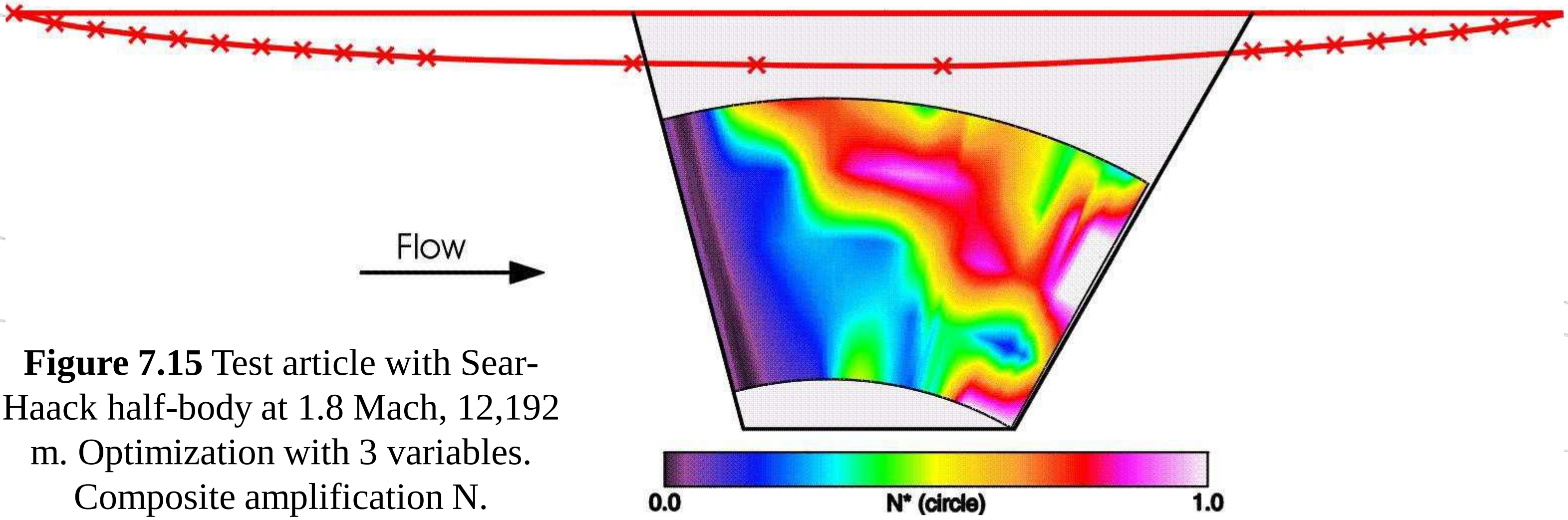


Figure 7.15 Test article with Sear-Haack half-body at 1.8 Mach, 12,192 m. Optimization with 3 variables. Composite amplification N.

Source: Sturdza & Kroo, 2002

3.5. Examples

Example 7.3: Aerodynamic design of a Natural Laminar Flow Business Jet (continued).

- With 5 variables, the boundary layer is much farther from transition to turbulent flow as can be seen by comparing the green and yellow colours on this wing with the red and violet colours in Fig. 7.15.
- Also notice how subtle the reshaping of the fuselage is.

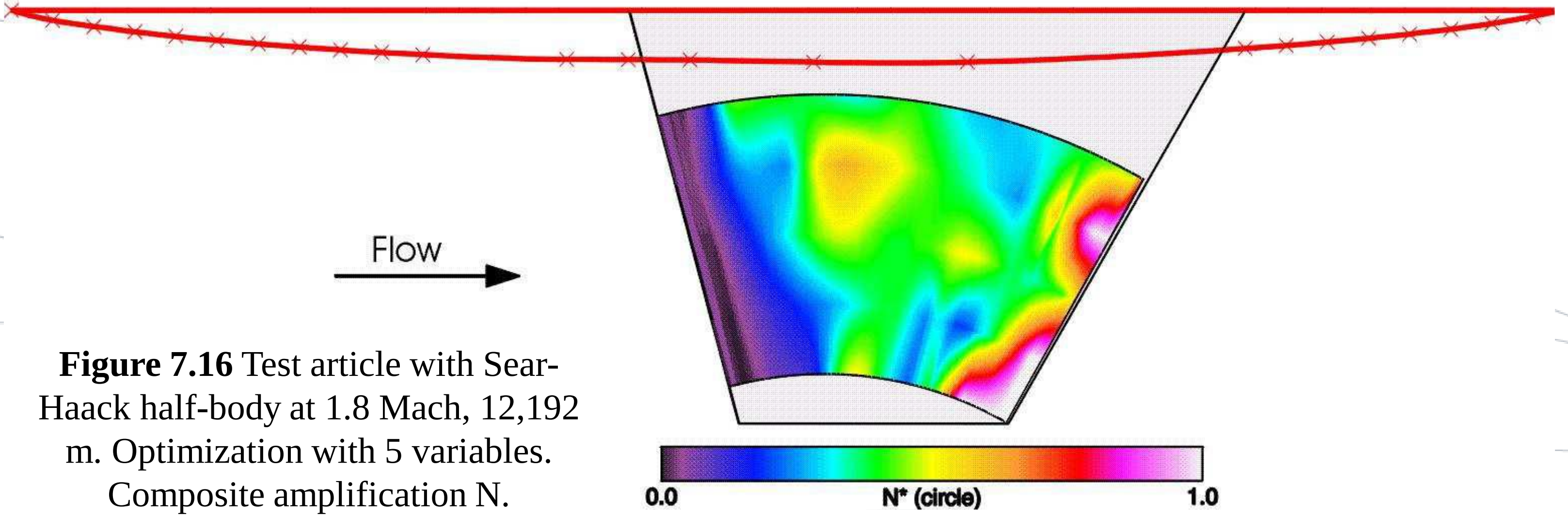


Figure 7.16 Test article with Sear-Haack half-body at 1.8 Mach, 12,192 m. Optimization with 5 variables. Composite amplification N.

Source: Sturdza & Kroo, 2002

3.5. Examples

MDO in Aeronautical Engineering / MDO em Engenharia Aeronáutica

Example 7.3: Aerodynamic design of a Natural Laminar Flow Business Jet (continued).

- With 5 design variables, and a few more trust-region update cycles, a better solution is found.

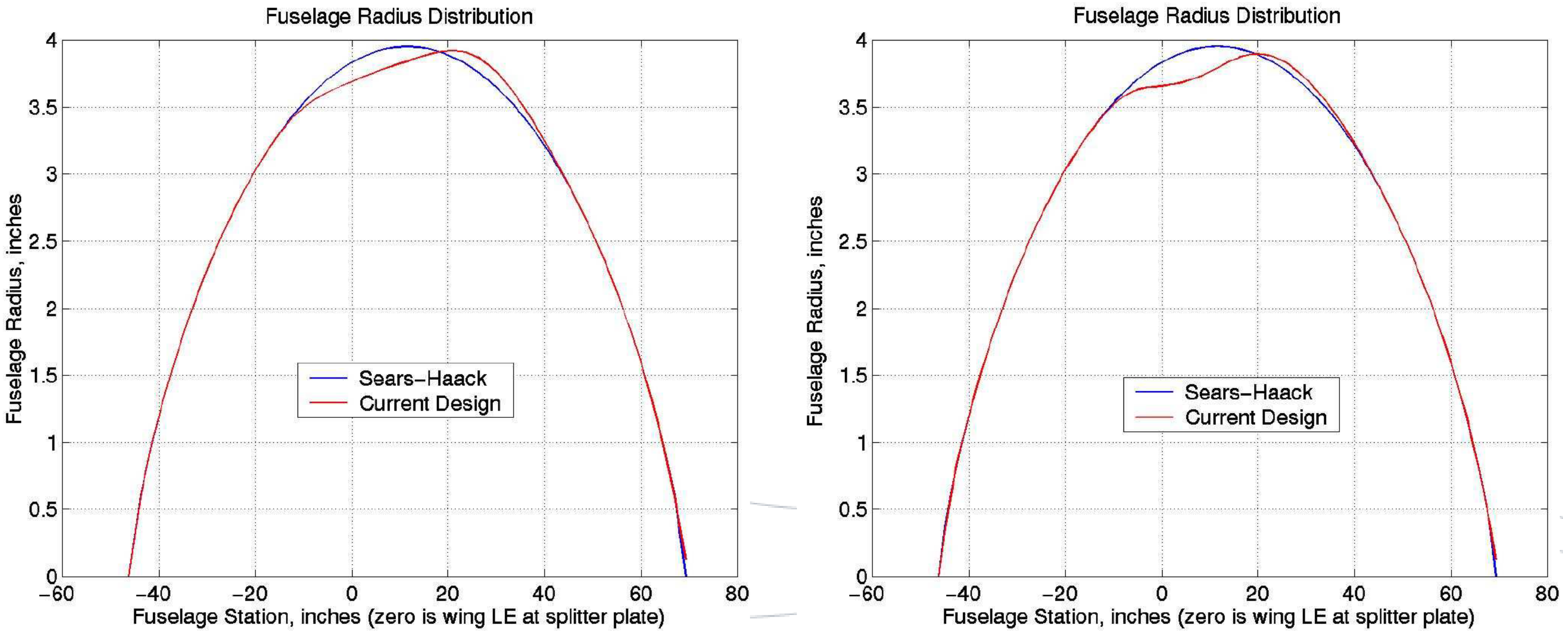


Figure 7.17 From the nose at left, to the tail at right, this is the radius of the original (blue) and re-designed (red) with 3 variables (left) and 5 variables (right).

Source: Sturdza & Kroo, 2002

3.5. Examples

Example 7.4: Aero-Structural Design of a Supersonic Business Jet.

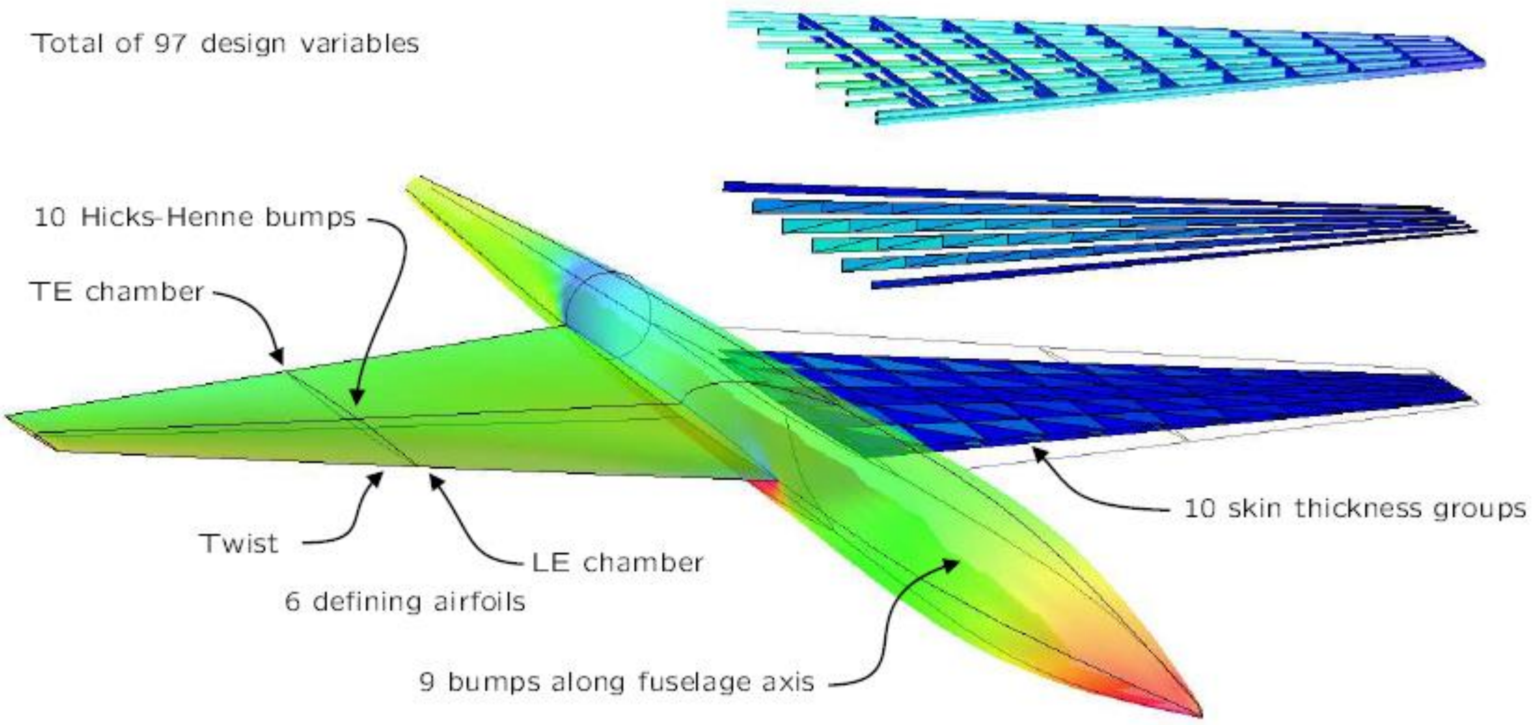


Figure 7.18 Baseline configuration and design variables.

Source: Martins, Alonso & Reuther, 2002

3.5. Examples

Example 7.4: Aero-Structural Design of a Supersonic Business Jet (continued).

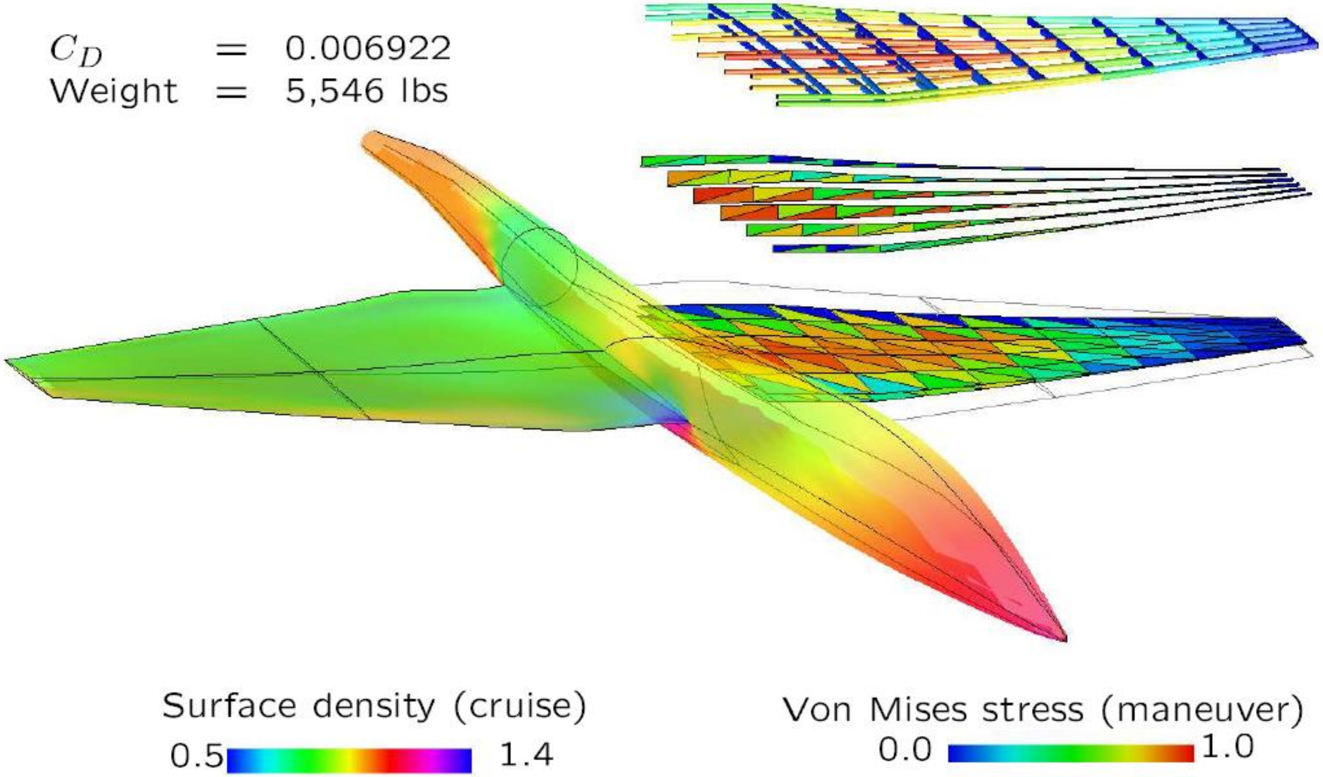


Figure 7.19 Optimized design.

Source: Martins, Alonso & Reuther, 2002

4. Coupled Models

- A model is a set of equations that we solve to predict the state of the engineering system and compute the objective and constraint function values.
- More generally, we can have a coupled model, which consists of multiple models (or components) that depend on each other's state variables.
- The same steps for formulating a design optimization problem (Chapter 1) apply in the formulation of MDO problems.
- The main difference in MDO problems is that the objective and constraints are computed by the coupled model.
- Once such a model is in place, the design optimization problem statement (Eq. 1.04) applies, with no changes needed.
- A generic example of a coupled model with three components is illustrated in Fig. 7.20.

4. Coupled Models

- Here, the states of each component affect all other components.
- However, it is common for a component to depend only on a subset of the other system components.
- Furthermore, we might distinguish variables between internal state variables and coupling.

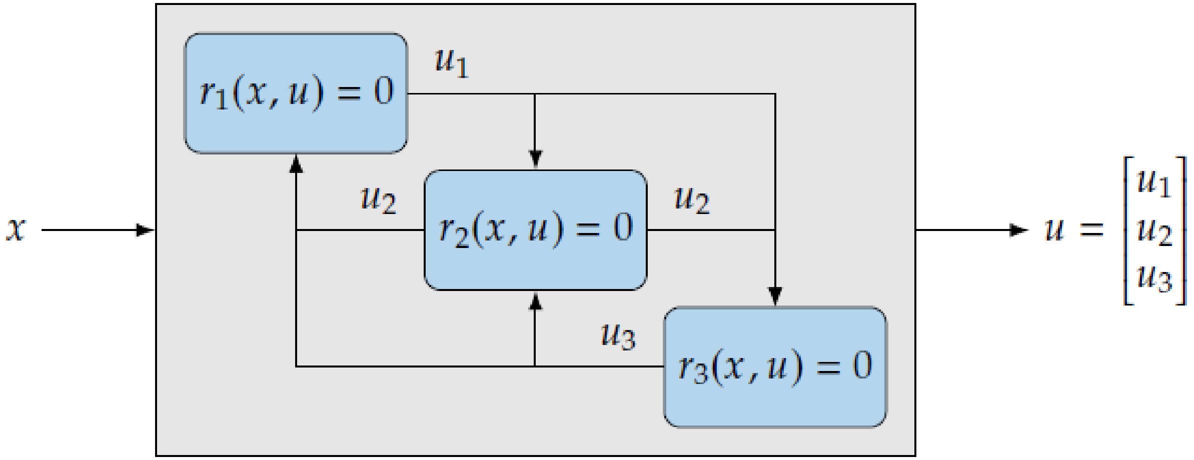


Figure 7.20 Coupled model composed of three numerical models.

Source: Martins et Ning, 2021

4. Coupled Models

- Mathematically, a coupled model is no more than a larger set of equations to be solved, where all the governing equation residuals (*r*), the corresponding state variables (*u*), and all the design variables (*x*) are concatenated into single vectors.
- Then, we can still just write the whole multidisciplinary model as $r(x,u)=0$.
- However, it is often necessary or advantageous to partition the system into smaller components for three main reasons.
- First, specialized solvers are often already in place for a given set of governing equations, which may be more efficient at solving their set of equations than a general-purpose solver.
- In addition, some of these solvers might be black boxes that do not provide an interface for using alternative solvers.

4. Coupled Models

- Second, there is an incentive for building the multidisciplinary system in a modular way.
- For example, a component might be useful on its own and should therefore be usable outside the multidisciplinary system.
- A modular approach also facilitates the extension of the multidisciplinary system and makes it easy to replace the model of a given discipline with an alternative one.
- Finally, the overall system of equations may be more efficiently solved if it is partitioned in a way that exploits the system structure.
- These reasons motivate an implementation of coupled models that is flexible enough to handle a mixture of different types of models and solvers for each component.

4. Coupled Models

- We start the remainder of this section by defining components in more detail (Section 7.4.1).
- We explain how the coupling variables relate to the state variables (Section 7.4.2) and coupled system formulation (Section 7.4.3).
- Then, we discuss the coupled system structure (Section 7.4.4).
- Finally, we explain methods for solving coupled systems (Section 7.4.5), including a hierarchical approach that can handle a mixture of models and solvers (Section 7.4.6).

4.1. Components

- All models can ultimately be written as a system of residuals, $r(x,u)=0$.
- When the system is large or includes sub-models, it might be natural to partition the system into components.
- We prefer to use the more general term components instead of disciplines to refer to the sub-models resulting from the partitioning because the partitioning of the overall model is not necessarily by discipline (e.g., aerodynamics, structures).
- A system model might also be partitioned by physical system components (e.g., wing, fuselage, or an aircraft in a fleet) or by different conditions applied to the same model (e.g., aerodynamic simulations at different flight conditions).
- The partitioning can also be performed within a given discipline for the same reasons cited previously.

4.1. Components

- In theory, the system model equations in $r(x,u)=0$ can be partitioned in any way, but only some partitions are advantageous or make sense.
- We denote a partitioning into n components as

$$r(u) = 0 \equiv \begin{cases} r_1(u_1, \dots, u_i, \dots, u_n) = 0 \\ \vdots \\ r_i(u_1, \dots, u_i, \dots, u_n) = 0 \\ \vdots \\ r_n(u_1, \dots, u_i, \dots, u_n) = 0 \end{cases} \qquad (7.01)$$

where each r_i and u_i are vectors corresponding to the residuals and states of component i.

- Here, we assume that each component can drive its residuals to zero by varying only its states, although this is not guaranteed in general.

4.1. Components

- We have omitted the dependency on x in Eq. 7.01 because, for now, we just want to find the state variables that solve the governing equations for a fixed design.
- Components can be either implicit or explicit.
- To solve an implicit component i, we need an algorithm for driving the equation residuals, $r_i(u_1, \dots, u_i, \dots, u_n)$, to zero by varying the states u_i while the other states $(u_j \; for \; all \; j \neq i)$ remain fixed.
- This algorithm could involve a matrix factorization for a linear system or a Newton solver for a nonlinear system.
- An explicit component is much easier to solve because that components' states are explicit functions of other components' states.

4.1. Components

- The states of an explicit component can be computed without factorization or iteration.
- Suppose that the states of a component *i* are given by the explicit function $u_i = f\left(u_j\right) \ \ for\ all\ j \neq i.$
- As previously explained, we can convert an explicit equation to the residual form by moving the function on the right-hand side to the left-hand side.
- Then, we obtain the set of residuals,

$$r_i(u_1, \dots, u_n) = u_i - f\left(u_j\right) for\ all\ j \neq i \qquad (7.02)$$

- Therefore, there is no loss of generality when using the residual notation in Eq. 7.01.

4.1. Components

- Most disciplines involve a mix of implicit and explicit components because the state variables are implicitly defined, whereas the objective function and constraints are usually explicit functions of the state variables.
- In addition, a discipline usually includes functions that convert inputs and outputs, as discussed in Section 7.3.
- As we will see in Section 7.4.6, the partitioning of a model can be hierarchical, where components are gathered in multiple groups.
- These groups can be nested to form a hierarchy with multiple levels.
- Again, this might be motivated by efficiency, modularity, or both.

4.1. Components

Example 7.5: Residuals of the coupled aerostructural problem.

- Let us formulate models for the aerostructural problem described in Ex. 7.1. A possible model for the aerodynamics is a vortex-lattice model given by the linear system

$$A\Gamma = v$$

where A is the matrix of aerodynamic influence coefficients, and v is a vector of boundary conditions, both of which depend on the wing shape. The state Γ is a vector that represents the circulation (vortex strength) at each spanwise position on the wing, as shown on the left-hand side of Fig. 7.21.

- The lift and drag scalars can be computed explicitly for a given Γ, so we write these dependencies as $L=L(\Gamma)$ and $D=D(\Gamma)$, omitting the detailed explicit expressions for conciseness.

4.1. Components

Example 7.5: Residuals of the coupled aerostructural problem (continued).

- A possible model for the structures is a cantilevered beam modelled with Euler–Bernoulli elements,

$$Kd = q$$

where K is the stiffness matrix, which depends on the beam shape and sizing.

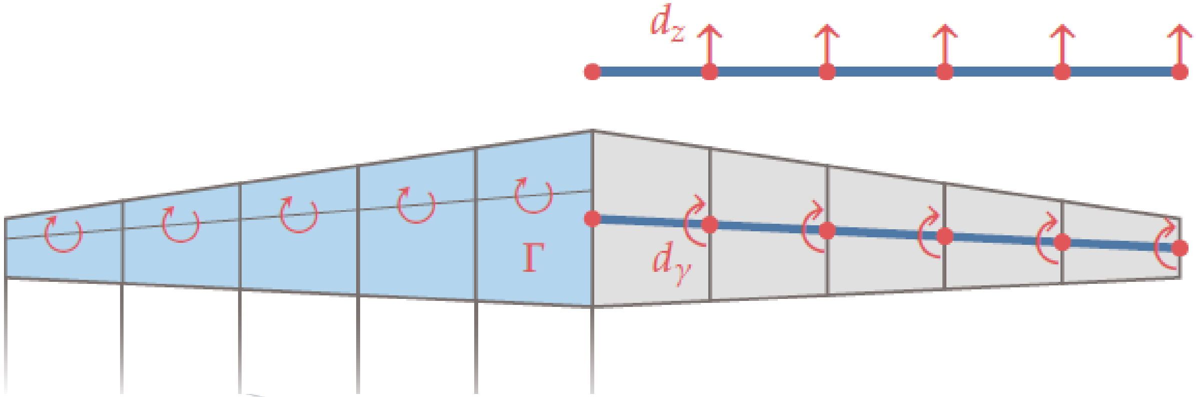


Figure 7.21 Aerostructural wing model showing the aerodynamic state variables (circulations Γ) on the left and structural state variables (displacements d_z and rotations d_γ) on the right.

Source: Martins et Ning, 2021

4.1. Components

Example 7.5: Residuals of the coupled aerostructural problem (continued).

- The right-hand-side vector represents the applied forces at the spanwise position on the beam.
- The states d are the displacements and rotations at each node, as shown on the right-hand side of Fig. 7.21.
- The weight does not depend on the states, and it is an explicit function of the beam sizing and shape, so it does not involve the structural model.
- The stresses are an explicit function of the displacements, so we can write $\sigma = \sigma(d)$, where σ is a vector whose size is the number of elements.
- When we couple these two models, A and v depend on the wing displacements d, and f depends on Γ.

4.1. Components

Example 7.5: Residuals of the coupled aerostructural problem (continued).

- We can write all the implicit and explicit equations as residuals:

$$r_1 = A(d)\Gamma - v(d)$$
$$r_2 = Kd - q(\Gamma)$$

- The states of this system are as follows:

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \equiv \begin{bmatrix} \Gamma \\ d \end{bmatrix}$$

- This coupled system is illustrated in Fig. 7.22.

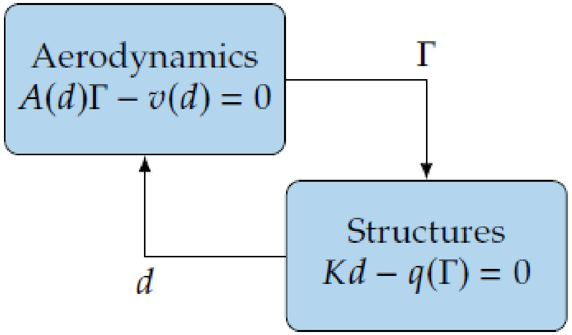


Figure 7.22 The aerostructural model couples aerodynamics and structures through a displacement and force transfer.

Source: Martins et Ning, 2021

4.2. Models and coupling variables

- In MDO, the coupling variables are variables that need to be passed from the model of one discipline to the others because of interdependencies in the system.
- Thus, the coupling variables are the inputs and outputs of each model.
- Sometimes, the coupling variables are just the state variables of one model (or a subset of these) that get passed to another model, but often we need to convert between the coupling variables and other variables within the model.
- We represent the coupling variables by a vector $\hat{u}_i$, where the subscript i denotes the model that computes these variables.
- In other words, $\hat{u}_i$ contains the outputs of model i.

4.2. Models and coupling variables

- A model *i* can take any coupling variable vector $\hat{u}_{j\neq i}$ as one of its inputs, where the subscript indicates that *j* can be the output from any model except its own.
- Figure 7.23 shows the inputs and outputs for a model.

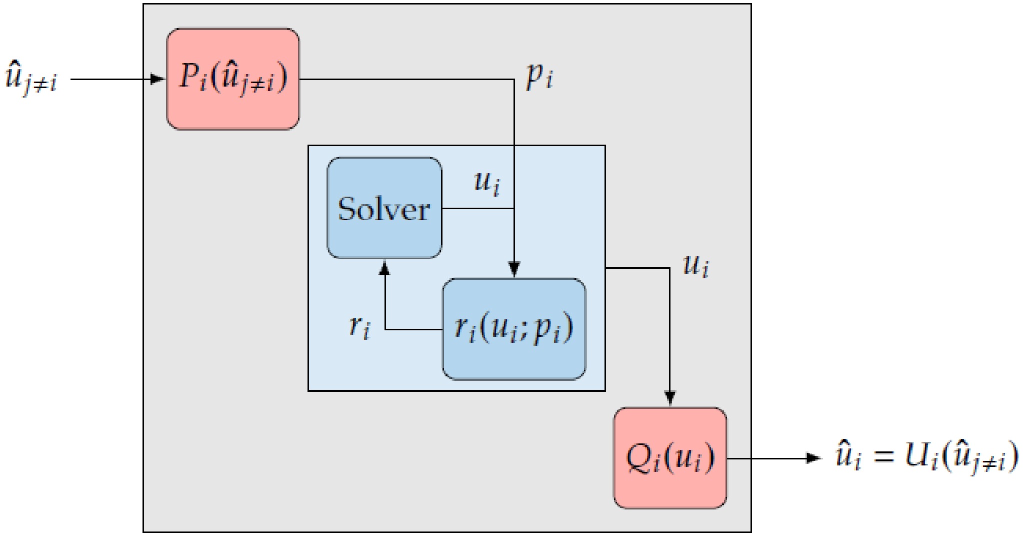


Figure 7.23 In the general case, a model may require conversions of inputs and outputs distinct from the states that the solver computes.

Source: Martins et Ning, 2021

4.2. Models and coupling variables

- The model solves for the set of its state variables, u_i.
- The residuals in the solver depend on the input variables coming from other models.
- In general, this is not a direct dependency, so the model may require explicit function (P_i) that converts the inputs ($\hat{u}_{j\neq i}$) to the required parameters p_i.
- These parameters remain fixed when the model solves its implicit equations for u_i.
- After the model solves for its state variables (u_i), there may be another explicit function (Q_i) that converts these states to output variables ($\hat{u}_i$) for the other models.
- The function (Q_i) typically reduces the number of output variables relative to the number of internal states, sometimes by orders of magnitude.

4.2. Models and coupling variables

- The model shown in Fig. 7.23 can be viewed as an implicit function that computes its outputs as a function of all the inputs, so we can write $\hat{u}_i = U_i(\hat{u}_{j\neq i})$.
- The model contains three components: two explicit and one implicit.
- We can convert the explicit components to residual equations using Eq. 7.02 and express the model as three sets of residuals as shown in Fig. 7.24.
- The result is a group of three components that we can represent as $r(u) = 0$.
- This conversion and grouping hint at a powerful concept that we will use later, which is hierarchy, where components can be grouped using multiple levels.

4.2. Models and coupling variables

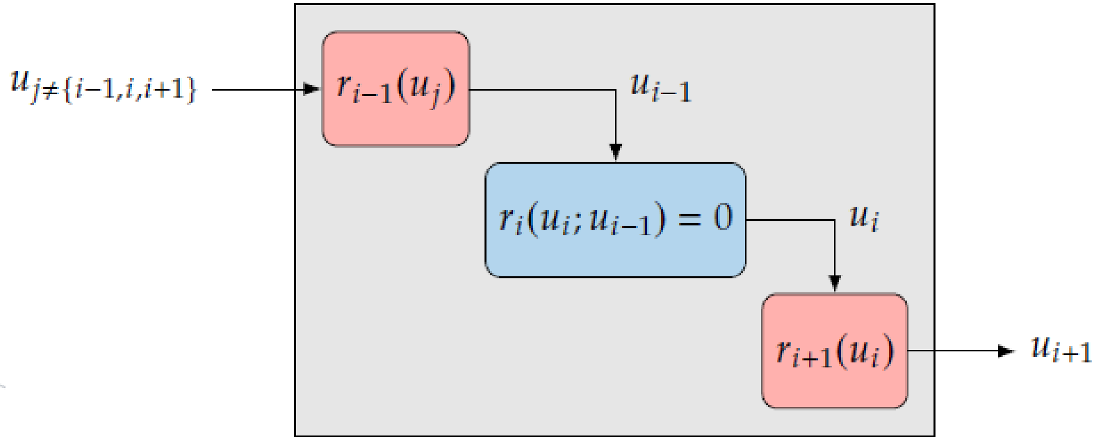


Figure 7.24 The conversion of inputs and outputs can be represented as explicit components with corresponding state variables. Using this form, any model can be entirely expressed $r(u)$=0. The inputs could be any subset of u except for the ones handled in the component ($u_{i\text{-}1}$, u_i, and u_{i+1}).

Source: Martins et Ning, 2021

4.3. Residuals and functional forms

- The system-level representation of a coupled system is determined by the variables that are “seen” and controlled at this level.
- Representing all models and variable conversions as $r(u)=0$ leads to the residual form of the coupled system, already written in Eq. 7.01, where n is the number of components.
- In this case, the system level has direct access and control over all the variables.
- This residual form is desirable because, as we will see later in this chapter, it enables us to formulate efficient ways to solve coupled systems and compute their derivatives.
- The functional form is an alternate system-level representation of the coupled system that considers only the coupling variables and expresses them as implicit functions of the others.

4.3. Residuals and functional forms

- We can write this form as

$$\hat{u} = U(\hat{u}) \Leftrightarrow \begin{cases} \hat{u}_1 = U_1(\hat{u}_2, \dots, \hat{u}_m) \\ \vdots \\ \hat{u}_i = U_i(\hat{u}_1, \dots, \hat{u}_{i-1}, \hat{u}_{i+1}, \dots, \hat{u}_m) \\ \vdots \\ \hat{u}_m = U_i(\hat{u}_1, \dots, \hat{u}_m - 1) \end{cases} \qquad (7.03)$$

where *m* is the number of models and *m*≤*n*.

- If a model is a black box and we have no access to the residuals and the conversion functions, this is the only form we can use.
- In this case, the system-level solver only iterates the coupling variables $\hat{u}$ and relies on each model *i* to solve or compute its outputs $\hat{u}_i$.

4.3. Residuals and functional forms

- These two forms are shown in Fig. 7.25 for a generic example with three models (or disciplines).
- The left of this figure shows the residual form, where each model is represented as residuals and states, as in Fig. 7.24.
- This leads to a system with nine sets of residuals and corresponding state variables.
- The number of state variables in each of these sets is not specified but could be any number.
- The functional form of these three models is shown on the right of Fig. 7.25.
- In the case where the model is a black box, the residuals and conversion functions shown in Fig. 7.23 are hidden, and the system level can only access the coupling variables.

4.3. Residuals and functional forms

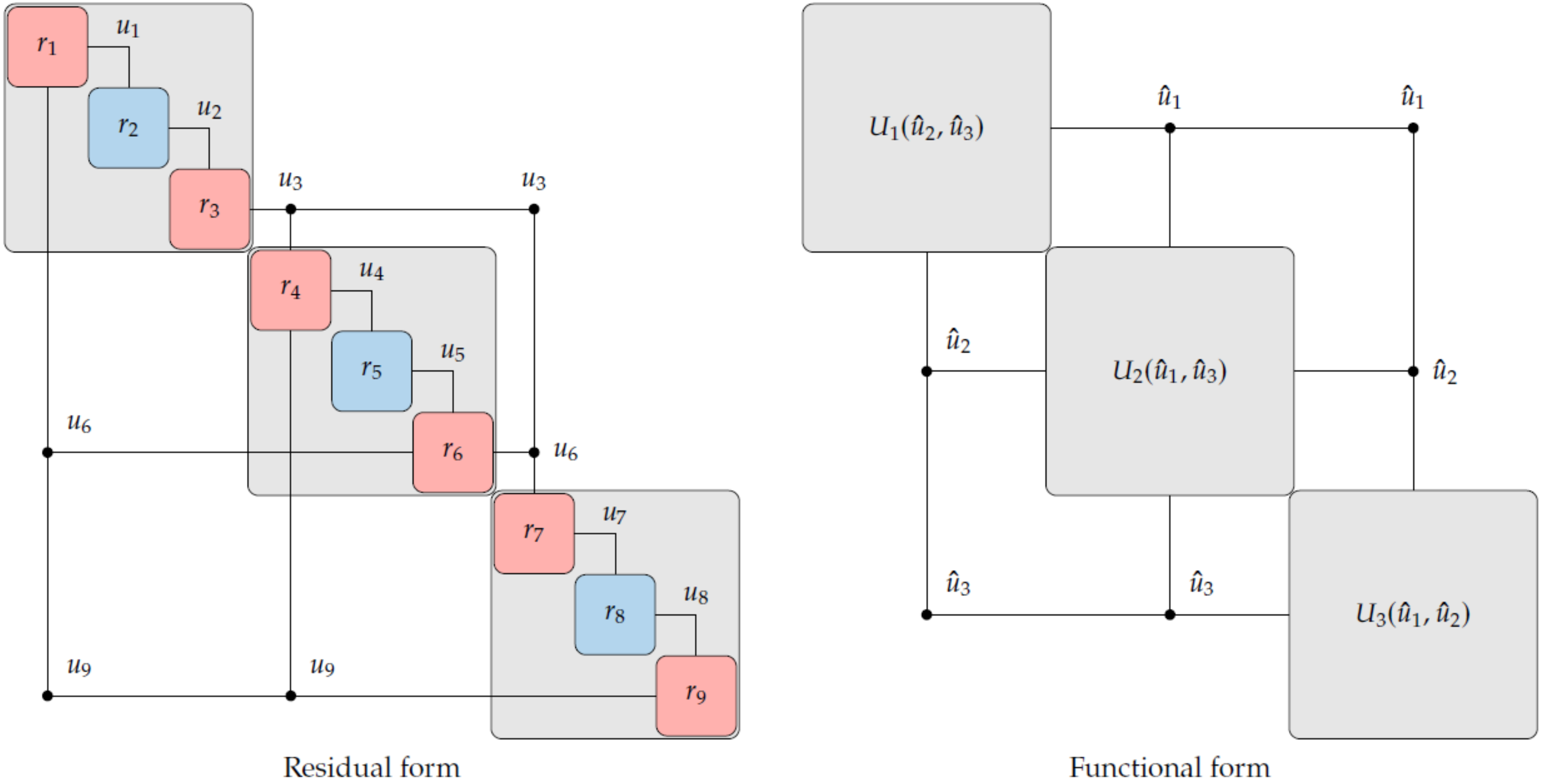


Figure 7.25 Two system-level views of coupled system with three solvers. In the residual form, all components and their states are exposed (left); in the functional (black-box) form, only inputs and outputs for each solver are visible (right), where $\hat{u}_1 \equiv u_3$, $\hat{u}_2 \equiv u_6$, and $\hat{u}_3 \equiv u_9$.

Source: Martins et Ning, 2021

4.3. Residuals and functional forms

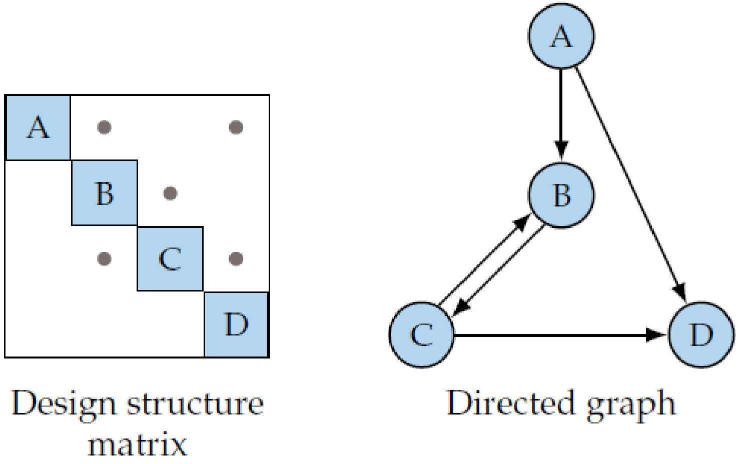
- In this case, we consider each black-box model to be a component.
- In an even more general case, these two views can be mixed in a coupled system.
- The models in residual form expose residuals and states, in which case, the model potentially has multiple components at the system level.
- The models in functional form only expose inputs and outputs; in that case, the model is just a single component.

4.4. Coupled system structure

- To show how multidisciplinary systems are coupled, we use a design structure matrix (DSM), which is sometimes referred to as a dependency structure matrix or an N^2 matrix.
- An example of the DSM for a hypothetical system is shown on the left in Fig. 7.26.
- In this matrix, the diagonal elements represent the components, and the off-diagonal entries denote coupling variables.
- A given coupling variable is computed by the component in its row and is passed to the component in its column.
- As shown in the DSM in Fig. 7.26, there are generally off-diagonal entries both above and below the diagonal, where the entries above feed forward, whereas entries below feed backward.

4.4. Coupled system structure

- The mathematical representation of these dependencies is given by a graph (Fig. 7.26, right), where the graph nodes are the components, and the edges represent the information dependency.
- This graph is a directed graph because, in general, there are three possibilities for coupling two components: single coupling one way, single coupling the other way, and two-way coupling.



Design structure matrix

Directed graph

Figure 7.26 Different ways to represent the dependencies of a hypothetical coupled system.

Source: Martins et Ning, 2021

4.4. Coupled system structure

- A directed graph is cyclic when there are edges that form a closed loop (i.e., a cycle).
- The graph on the right of Fig. 7.26 has a single cycle between components B and C.
- When there are no closed loops, the graph is acyclic.
- In this case, the whole system can be solved by solving each component in turn without iterating.
- The DSM can be viewed as a matrix where the blank entries are zeros.
- For real-world systems, this is often a sparse matrix.
- This means that in the corresponding DSM, each component depends only on a subset of all the other components.
- We can take advantage of the structure of this sparsity in the solution of coupled systems.

4.4. Coupled system structure

- The components in the DSM can be reordered without changing the solution of the system.
- This is analogous to reordering sparse matrices to make linear systems easier to solve.
- In one extreme case, reordering could achieve a DSM with no entries below the diagonal.
- In that case, we would have only feed-forward connections, which means all dependencies could be resolved in one forward pass (as we will see in Ex. 7.06).
- This is analogous to having a linear system where the matrix is lower triangular, in which case the linear solution can be obtained with forward substitution.
- The sparsity of the DSM can be exploited using ideas from sparse linear algebra.

4.4. Coupled system structure

- For example, reducing the bandwidth of the matrix (i.e., moving nonzero elements closer to the diagonal) can also be helpful.
- This can be achieved using algorithms such as Cuthill–McKee, reverse Cuthill–McKee (RCM), and approximate minimum degree (AMD) ordering.
- We now introduce an extended version of the DSM, called XDSM, which we use later in this chapter to show the process in addition to the data dependencies.
- Figure 7.27 shows the XDSM for the same four-component system.
- When showing only the data dependencies, the only difference relative to DSM is that the coupling variables are labelled explicitly, and the data paths are drawn.

4. Coupled Models

4.4. Coupled system structure

- In the next section, we add the process to the XDSM.

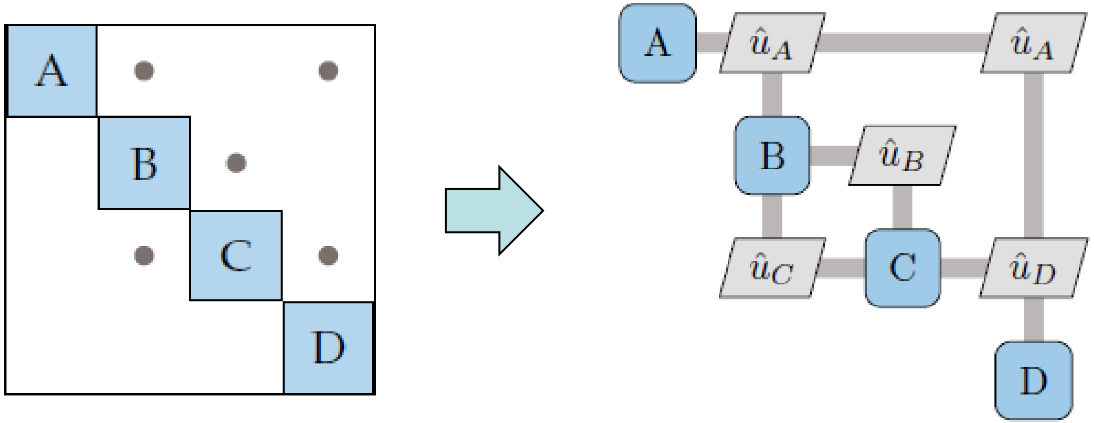


Figure 7.27 XDSM showing data dependencies for the four-component coupled system of Fig. 7.26.

Source: Martins et Ning, 2021

4.5. Solving coupled numerical models

- The solution of coupled systems, also known as multidisciplinary analysis (MDA) usually involves multiple levels of solvers.
- When using the residual form described in Section 7.4.3, any solver (such as a Newton solver) can be used to solve for the state of all components (the entire vector u) simultaneously to satisfy $r(u)=0$ for the coupled system (Eq. 7.01).
- This is a monolithic solution approach.
- When using the functional form, we do not have access to the internal states of each model and must rely on the model's solvers to compute the coupling variables.

4.5. Solving coupled numerical models

- The model solver is responsible for computing its output variables for a given set of coupling variables from other models, that is,

$$\hat{u}_i = U_i(\hat{u}_{j\neq i}) \qquad (7.04)$$

- In some cases, we have access to the model's internal states, but we may want to use a dedicated solver for that model anyway.
- Because each model, in general, depends on the outputs of all other models, we have a coupled dependency that requires a solver to resolve.
- This means that the functional form requires two levels: one for the model solvers and another for the system-level solver.
- At the system level, we only deal with the coupling variables ($\hat{u}$), and the internal states (u) are hidden.

4.5. Solving coupled numerical models

- The rest of this section presents several system-level solvers.
- We will refer to each model as a component even though it is a group of components in general.

4.5.1. Nonlinear block Jacobi

- The most straightforward way to solve coupled numerical models (systems of components) is through a fixed-point iteration.
- Here, instead of updating one state at a time, we update a vector of coupling variables at each iteration corresponding to a subset of the coupling variables in the overall coupled system.
- Obtaining this vector of coupling variables generally involves the solution of a nonlinear system.
- Therefore, these are called nonlinear block variants of the linear fixed-point iteration methods.
- The nonlinear block Jacobi method requires an initial guess for all coupling variables to start with and calls for the solution of all components given those guesses.

4. Coupled Models

4.5. Solving coupled numerical models

4.5.1. Nonlinear block Jacobi

- Once all components have been solved, the coupling variables are updated based on the new values computed by the components, and all components are solved again.
- This iterative process continues until the coupling variables do not change in subsequent iterations.
- Because each component takes the coupling variable values from the previous iteration, which have already been computed, all components can be solved in parallel without communication.
- This algorithm is formalized in Alg. 7.1.
- When applied to a system of components, we call it the block Jacobi method, where block refers to each component.

4.5.1. Nonlinear block Jacobi

- The nonlinear block Jacobi method is also illustrated using an XDSM in Fig. 7.28 for three components.
- The only input is the initial guess for the coupling variables, $\hat{u}^{(0)}$.
- The MDA block (step 0) is responsible for iterating the system-level analysis loop and for checking if the system has converged.
- The process line is shown as a thin black line to distinguish it from the data dependency connections (thick grey lines) and follows the sequence of numbered steps.
- The analyses for each component are all numbered the same (step 1) because they can be done in parallel.
- Each component returns the coupling variables it computes to the MDA iterator, closing the loop between step 2 and step 1 (denoted as “$2 \to 1$”).

4.5.1. Nonlinear block Jacobi

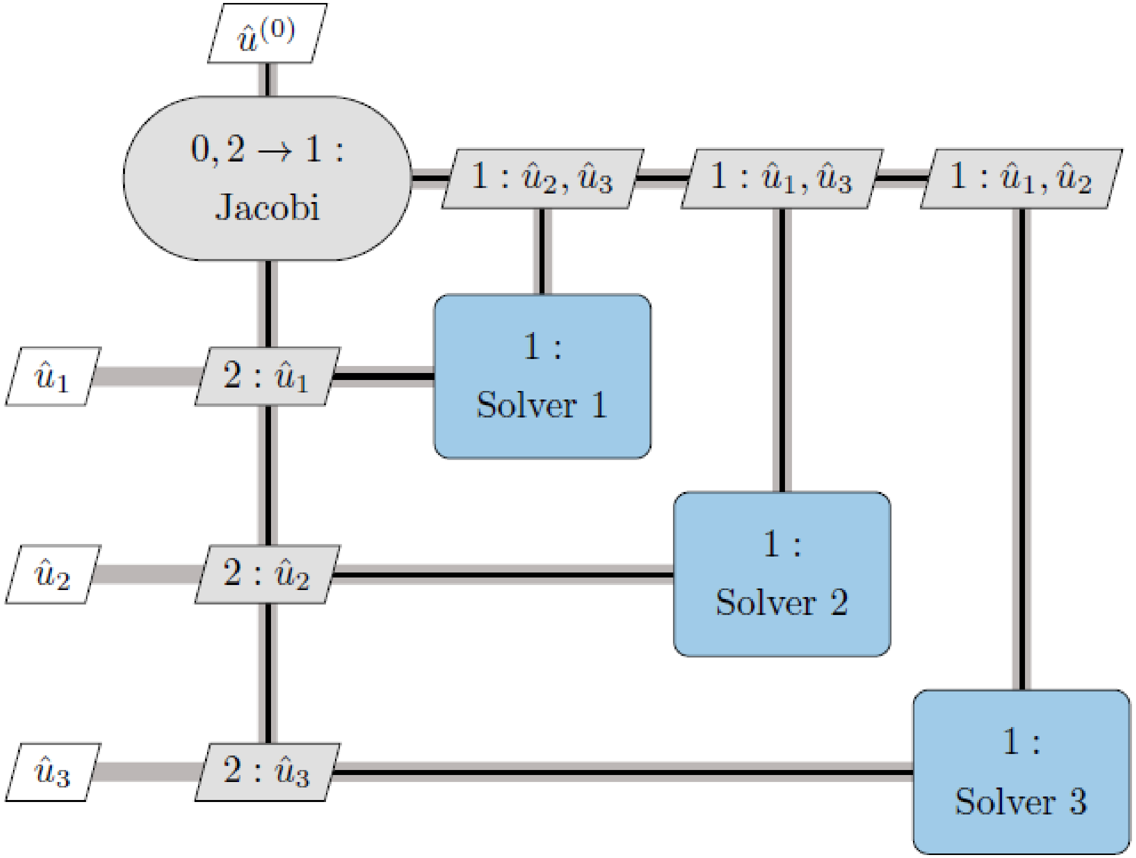
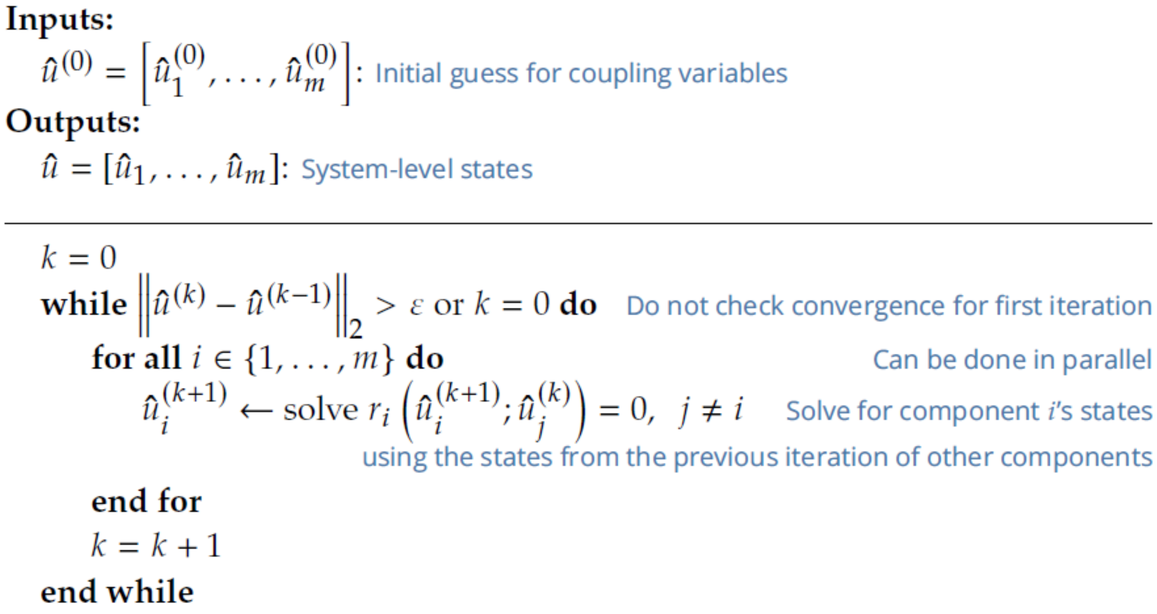


Figure 7.28 Nonlinear block Jacobi solver for a three-component coupled system.

Source: Martins et Ning, 2021

4.5.1. Nonlinear block Jacobi

Algorithm 7.1: Nonlinear block Jacobi algorithm



Inputs:
$\hat{u}^{(0)} = \left[\hat{u}_1^{(0)}, \ldots, \hat{u}_m^{(0)}\right]$: Initial guess for coupling variables

Outputs:
$\hat{u} = [\hat{u}_1, \ldots, \hat{u}_m]$: System-level states

```
$k = 0$
while $\left\|\hat{u}^{(k)} - \hat{u}^{(k-1)}\right\|_2 > \varepsilon$ or $k = 0$ do    Do not check convergence for first iteration
    for all $i \in \{1, \ldots, m\}$ do                                      Can be done in parallel
        $\hat{u}_i^{(k+1)} \leftarrow$ solve $r_i\left(\hat{u}_i^{(k+1)}; \hat{u}_j^{(k)}\right) = 0, \quad j \neq i$    Solve for component $i$'s states
                                   using the states from the previous iteration of other components
    end for
    $k = k + 1$
end while
```

4.5.2. Nonlinear block Gauss-Seidel

- The nonlinear block Gauss–Seidel algorithm is similar to its Jacobi counterpart.
- The only difference is that when solving each component, we use the latest coupling variables available instead of just using the coupling variables from the previous iteration.
- We cycle through each component i=1,…,m in order.
- When computing $\hat{u}_i$ by solving component i, we use the latest available states from the other components.
- Figure 7.29 illustrates this process.
- Both Gauss–Seidel and Jacobi converge linearly, but Gauss–Seidel tends to converge more quickly because each equation uses the latest information available.
- However, unlike Jacobi, the components cannot be solved in parallel.

4.5.2. Nonlinear block Gauss-Seidel

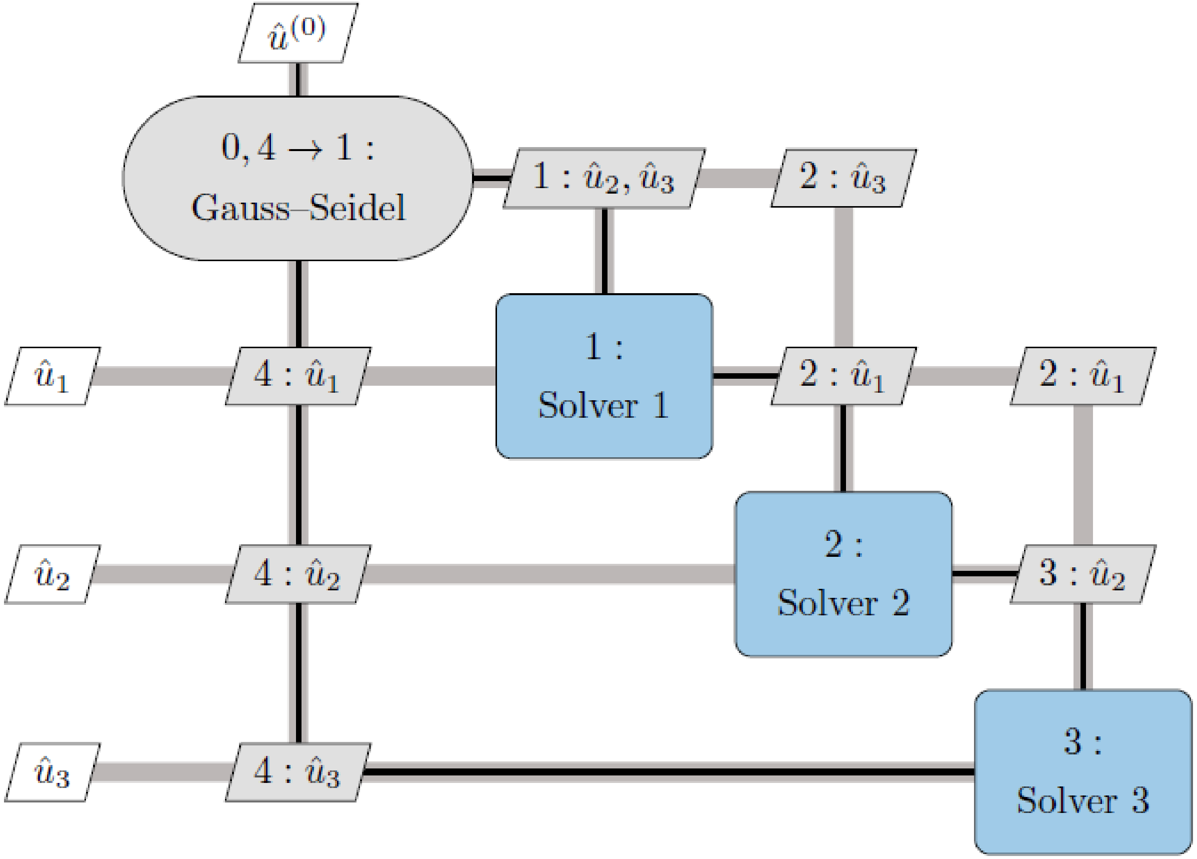


Figure 7.29 Nonlinear block Gauss–Seidel solver for a three-discipline coupled system.

Source: Martins et Ning, 2021

4.5.2. Nonlinear block Gauss-Seidel

- The convergence of nonlinear block Gauss–Seidel can be improved by using a relaxation.
- Suppose that $\hat{u}_{temp}$ is the state of component *i* resulting from the solving of that component given the states of all other components, as we would normally do for each block in the Gauss–Seidel or Jacobi method.
- If we used this, the step would be

$$\Delta\hat{u}_i^{(k)} = \hat{u}_{temp} - \hat{u}_i^{(k)} \qquad (7.05)$$

- Instead of using that step, relaxation updates the variables as

$$\hat{u}_i^{(k)} = \hat{u}_{temp} + \theta^{(k)}\Delta\hat{u}_i^{(k)} \qquad (7.06)$$

where $\theta^{(k)}$ is the relaxation factor, and $\Delta\hat{u}_i^{(k)}$ is the previous update for component *i*.

4.5.2. Nonlinear block Gauss-Seidel

- The relaxation factor, θ, could be a fixed value, which would normally be less than 1 to dampen oscillations and avoid divergence.
- Aitken’s method improves on the fixed relaxation approach by adapting the θ.
- The relaxation factor at each iteration changes based on the last two updates according to

$$\theta^{(k)} = \theta^{(k-1)} \left(1 - \frac{\left(\Delta\hat{u}^{(k)} - \Delta\hat{u}^{(k-1)}\right)^T \Delta\hat{u}^{(k)}}{\|\Delta\hat{u}^{(k)} - \Delta\hat{u}^{(k-1)}\|^2} \right) \qquad (7.07)$$

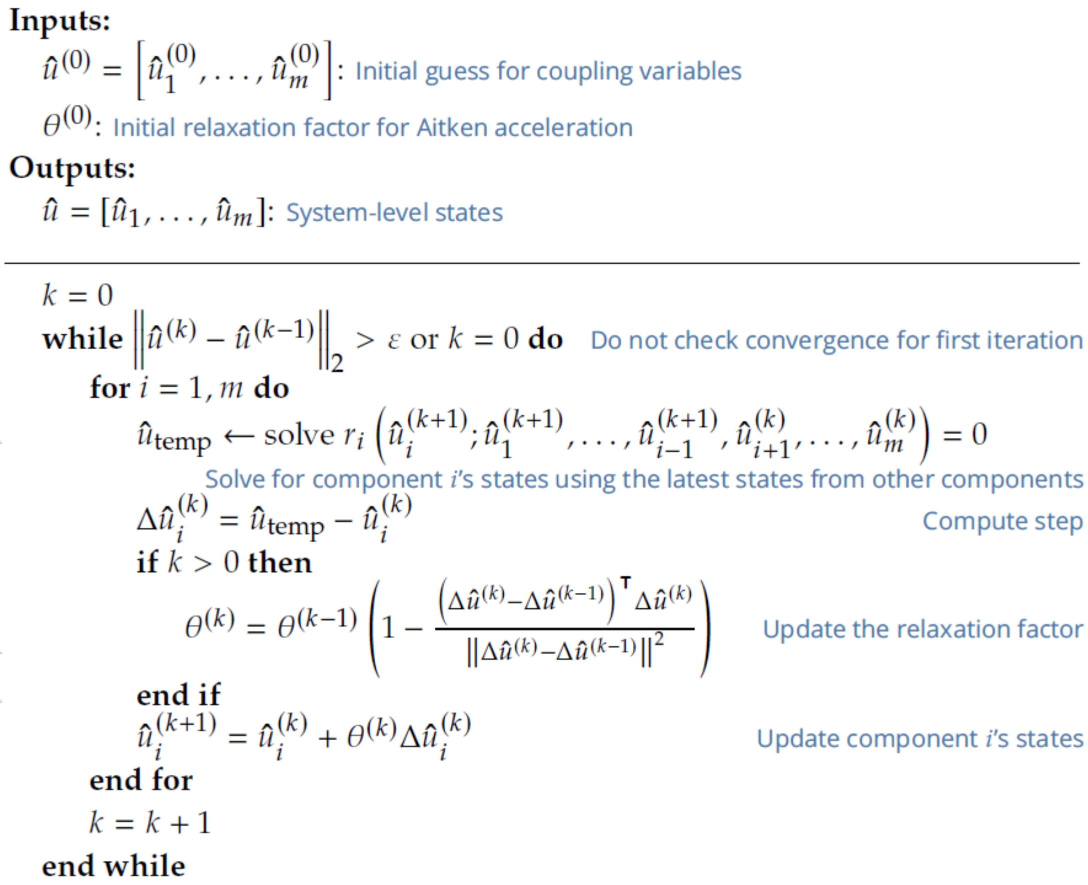
- Aitken’s method usually accelerates convergence and has been shown to work well for nonlinear block Gauss–Seidel with multidisciplinary systems.

4.5.2. Nonlinear block Gauss-Seidel

- It is advisable to override the value of the relaxation factor given by Eq. 7.07 to keep it between 0.25 and 2.
- The steps for the full Gauss–Seidel algorithm with Aitken acceleration are listed in Alg. 7.2.
- The order in which the components are solved makes a significant difference in the efficiency of the Gauss–Seidel method.
- In the best possible scenario, the components can be reordered such that there are no entries in the lower diagonal of the DSM, which means that each component depends only on previously solved components, and there are therefore no feedback dependencies (see Ex. 7.6).
- In this case, the block Gauss–Seidel method would converge to the solution in one forward sweep.

4.5.2. Nonlinear block Gauss-Seidel

Algorithm 7.2: Nonlinear block Gauss–Seidel algorithm with Aitken acceleration



```
Inputs:
    $\hat{u}^{(0)} = \left[\hat{u}_1^{(0)}, \ldots, \hat{u}_m^{(0)}\right]$: Initial guess for coupling variables
    $\theta^{(0)}$: Initial relaxation factor for Aitken acceleration
Outputs:
    $\hat{u} = [\hat{u}_1, \ldots, \hat{u}_m]$: System-level states

    $k = 0$
    while $\left\|\hat{u}^{(k)} - \hat{u}^{(k-1)}\right\|_2 > \varepsilon$ or $k = 0$ do    Do not check convergence for first iteration
        for $i = 1, m$ do
            $\hat{u}_{\text{temp}} \leftarrow \text{solve } r_i\left(\hat{u}_i^{(k+1)}; \hat{u}_1^{(k+1)}, \ldots, \hat{u}_{i-1}^{(k+1)}, \hat{u}_{i+1}^{(k)}, \ldots, \hat{u}_m^{(k)}\right) = 0$
                Solve for component i's states using the latest states from other components
            $\Delta\hat{u}_i^{(k)} = \hat{u}_{\text{temp}} - \hat{u}_i^{(k)}$    Compute step
            if $k > 0$ then
                $\theta^{(k)} = \theta^{(k-1)}\left(1 - \frac{\left(\Delta\hat{u}^{(k)} - \Delta\hat{u}^{(k-1)}\right)^\intercal \Delta\hat{u}^{(k)}}{\left\|\Delta\hat{u}^{(k)} - \Delta\hat{u}^{(k-1)}\right\|^2}\right)$    Update the relaxation factor
            end if
            $\hat{u}_i^{(k+1)} = \hat{u}_i^{(k)} + \theta^{(k)}\Delta\hat{u}_i^{(k)}$    Update component i's states
        end for
        $k = k + 1$
    end while
```

4.5.2. Nonlinear block Gauss-Seidel

- In the more general case, even though we might not eliminate the lower diagonal entries completely, minimizing these entries by reordering results in better convergence.
- This reordering can also mean the difference between convergence and nonconvergence.

4.5.2. Nonlinear block Gauss-Seidel

Example 7.6: Making Gauss–Seidel converge in one pass by reordering components.

- Consider the coupled system of six components with the dependencies shown on the left in Fig. 7.30.
- This system includes both feed-forward and feedback dependencies and would normally require an iterative solver.
- In this case, however, we can reorder the components as shown on the right in Fig. 7.30 to eliminate the feedback loops.
- Then, we only need to solve the sequence of components E → C → A → D → F → B once to get a converged coupled solution.

4.5.2. Nonlinear block Gauss-Seidel

Example 7.6: Making Gauss–Seidel converge in one pass by reordering components (continued).

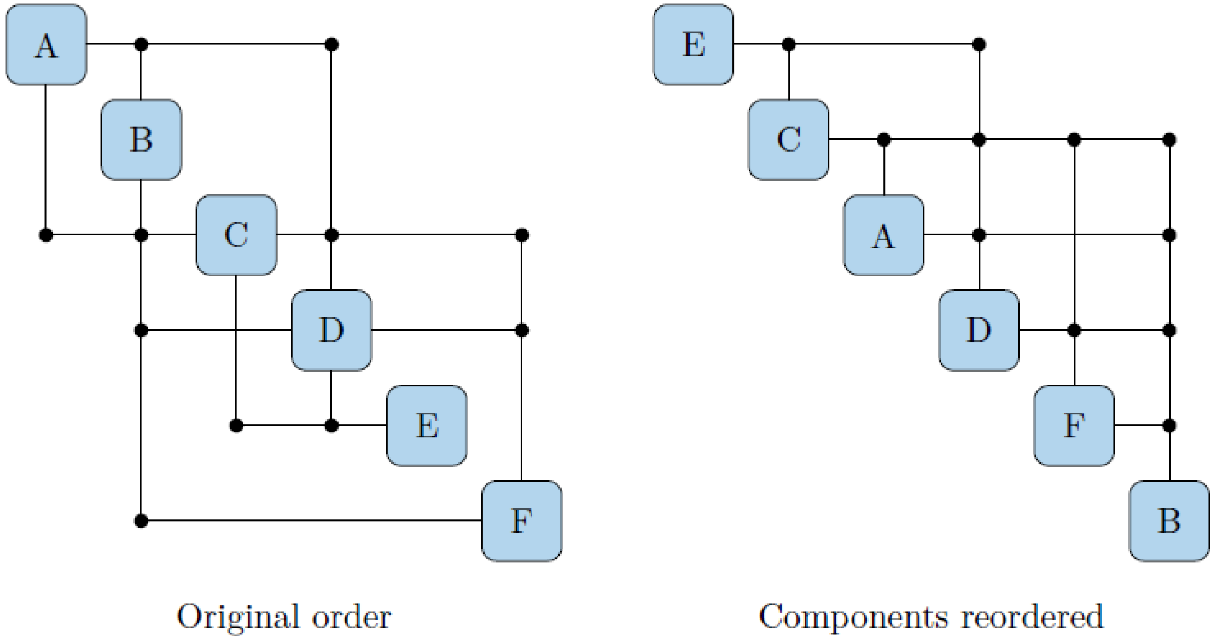


Figure 7.30 The solution of the components of the system shown on the left can be reordered to get the equivalent system shown on the right. This new system has no feedback loops and can therefore be solved in one pass of a Gauss–Seidel solver.

Source: Martins et Ning, 2021

4.5.3. Newton’s method

- As mentioned previously, Newton’s method can be applied to the residual form illustrated in Fig. 7.25 and expressed in Eq. 7.01.
- Recall that in this form, we have *n* components and the coupling variables are part of the state variables.
- Concatenating the residuals and state variables for all components and applying Newton’s method yields the coupled block Newton system,

$$\frac{\partial r}{\partial u}\Delta u = -r \Leftrightarrow \begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \frac{\partial r_1}{\partial u_2} & \cdots & \frac{\partial r_1}{\partial u_n} \\ \frac{\partial r_2}{\partial u_1} & \frac{\partial r_2}{\partial u_2} & \cdots & \frac{\partial r_2}{\partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_n}{\partial u_1} & \frac{\partial r_n}{\partial u_2} & \cdots & \frac{\partial r_n}{\partial u_n} \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ \vdots \\ \Delta u_n \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix} \qquad (7.08)$$

4.5.3. Newton’s method

- We can solve this linear system to compute the Newton step in for all components’ state variables *u* simultaneously, and then iterate to satisfy *r*(*u*)=0 for the complete system.
- This is the monolithic Newton approach illustrated on the left panel of Fig. 7.31.
- As with any Newton method, a globalization strategy (such as a line search) is required to increase the likelihood of successful convergence when starting far from the solution.
- Even with such a strategy, Newton’s method does not necessarily converge robustly.
- A variation on this monolithic Newton approach uses two-level solver hierarchy, as illustrated on the middle panel of Fig. 7.31.

4.5.3. Newton's method

- The system-level solver is the same as in the monolithic approach, but each component is solved first using the latest states.

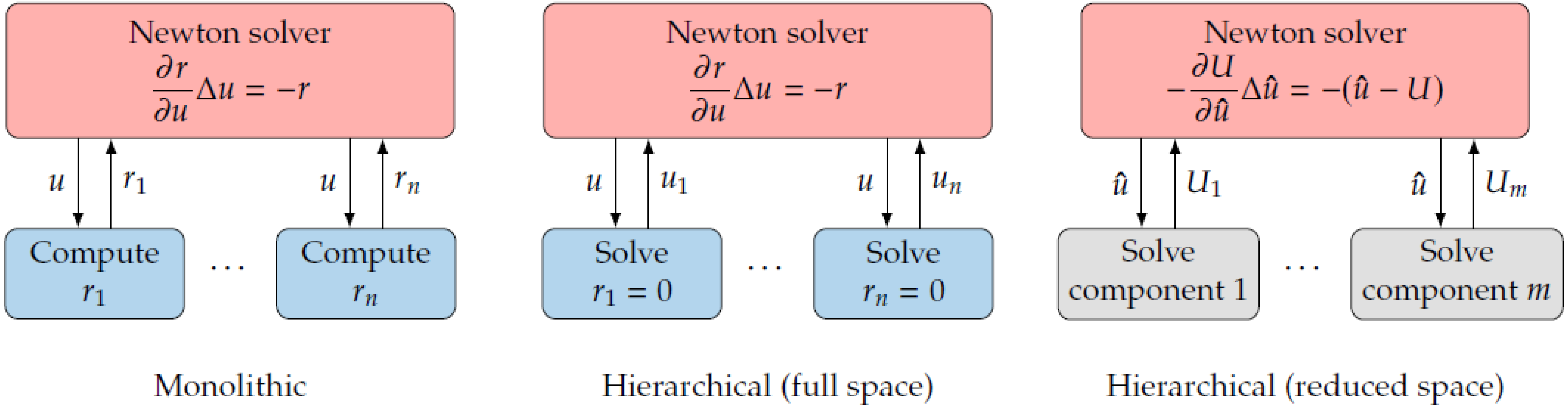

Newton solver
$\frac{\partial r}{\partial u}\Delta u = -r$
u
r_1
u
r_n
Compute r_1
…
Compute r_n
Monolithic
Newton solver
$\frac{\partial r}{\partial u}\Delta u = -r$
u
u_1
u
u_n
Solve $r_1 = 0$
…
Solve $r_n = 0$
Hierarchical (full space)
Newton solver
$-\frac{\partial U}{\partial \hat{u}}\Delta \hat{u} = -(\hat{u} - U)$
$\hat{u}$
U_1
$\hat{u}$
U_m
Solve component 1
…
Solve component m
Hierarchical (reduced space)

Figure 7.31 There are three options for solving a coupled system with Newton's method. The monolithic approach (left) solves for all state variables simultaneously. The block approach (middle) solves the same system as the monolithic approach, but solves each component for its states at each iteration. The black box approach (right) applies Newton's method to the coupling variables.

Source: Martins et Ning, 2021

4.5.3. Newton's method

- The Newton step for each component *i* is given by

$$\frac{\partial r_i}{\partial r_i}\Delta u_i = -r_i\left(u_i; u_{j\neq i}\right) \qquad (7.09)$$

where u_j represents the states from other components (i.e., $j \neq i$), which are fixed at this level.

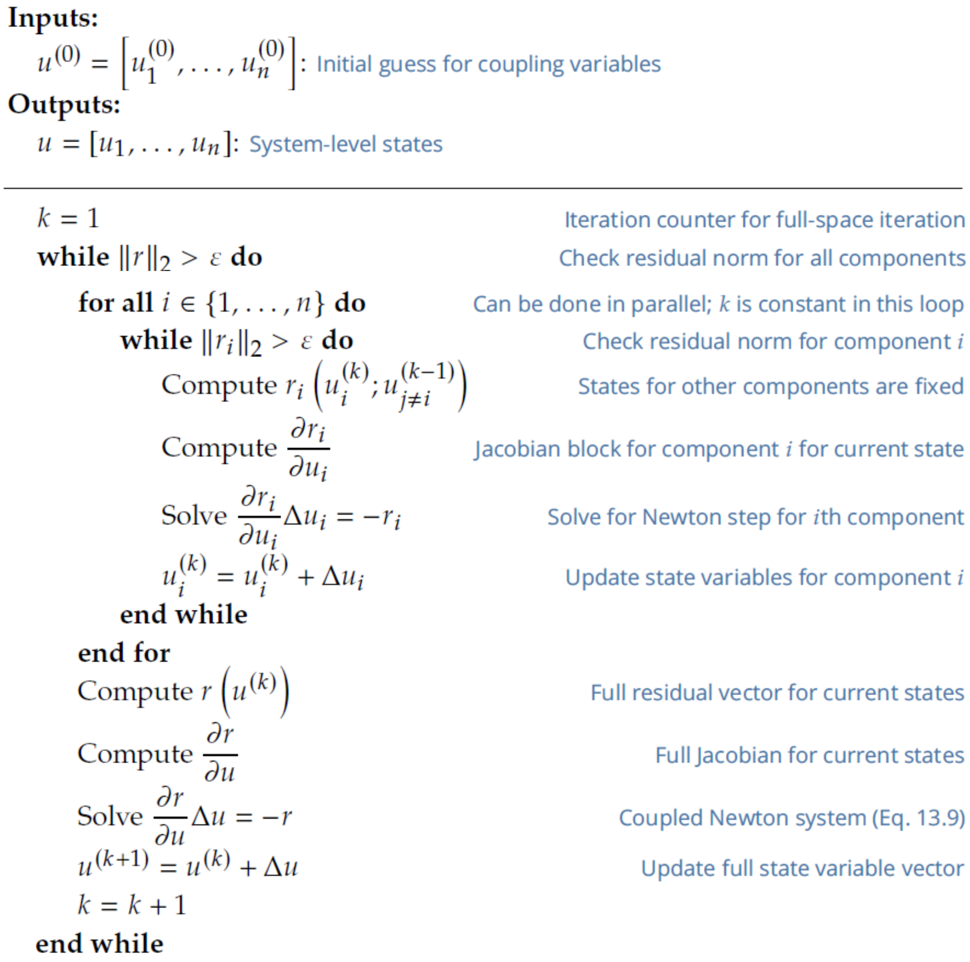
- Each component is solved before taking a step in the entire state vector (Eq. 7.08).
- The procedure is given in Alg. 7.3.
- We call this the full-space hierarchical Newton approach because the system-level solver iterates the entire state vector.
- Solving each component before taking each step in the full space Newton iteration acts as a preconditioner.

4. Coupled Models

4.5. Solving coupled numerical models

4.5.3. Newton's method

Algorithm 7.3: Full-space hierarchical Newton



Inputs:

$u^{(0)} = \left[u_1^{(0)}, \ldots, u_n^{(0)}\right]$: Initial guess for coupling variables

Outputs:

$u = [u_1, \ldots, u_n]$: System-level states

```
$k = 1$                                                  Iteration counter for full-space iteration
while $\|r\|_2 > \varepsilon$ do                          Check residual norm for all components
    for all $i \in \{1, \ldots, n\}$ do                   Can be done in parallel; $k$ is constant in this loop
        while $\|r_i\|_2 > \varepsilon$ do                Check residual norm for component $i$
            Compute $r_i\left(u_i^{(k)}; u_{j \neq i}^{(k-1)}\right)$    States for other components are fixed
            Compute $\frac{\partial r_i}{\partial u_i}$   Jacobian block for component $i$ for current state
            Solve $\frac{\partial r_i}{\partial u_i} \Delta u_i = -r_i$    Solve for Newton step for $i$th component
            $u_i^{(k)} = u_i^{(k)} + \Delta u_i$          Update state variables for component $i$
        end while
    end for
    Compute $r\left(u^{(k)}\right)$                       Full residual vector for current states
    Compute $\frac{\partial r}{\partial u}$               Full Jacobian for current states
    Solve $\frac{\partial r}{\partial u} \Delta u = -r$   Coupled Newton system (Eq. 13.9)
    $u^{(k+1)} = u^{(k)} + \Delta u$                      Update full state variable vector
    $k = k + 1$
end while
```

4.5.3. Newton’s method

- In general, the monolithic approach is more efficient, and the hierarchical approach is more robust, but these characteristics are case-dependent.
- Newton’s method can also be applied to the functional form illustrated in Fig. 7.32 to solve only for the coupling variables.
- We call this the reduced-space hierarchical Newton approach because the system-level solver iterates only in the space of the coupling variables, which is smaller than the full space of the state variables.
- Using this approach, the solver for each component can be considered to be a black box, as we did for the nonlinear block Jacobi and Gauss–Seidel solvers.
- This approach is illustrated on the right panel of Fig. 7.31.

4.5.3. Newton's method

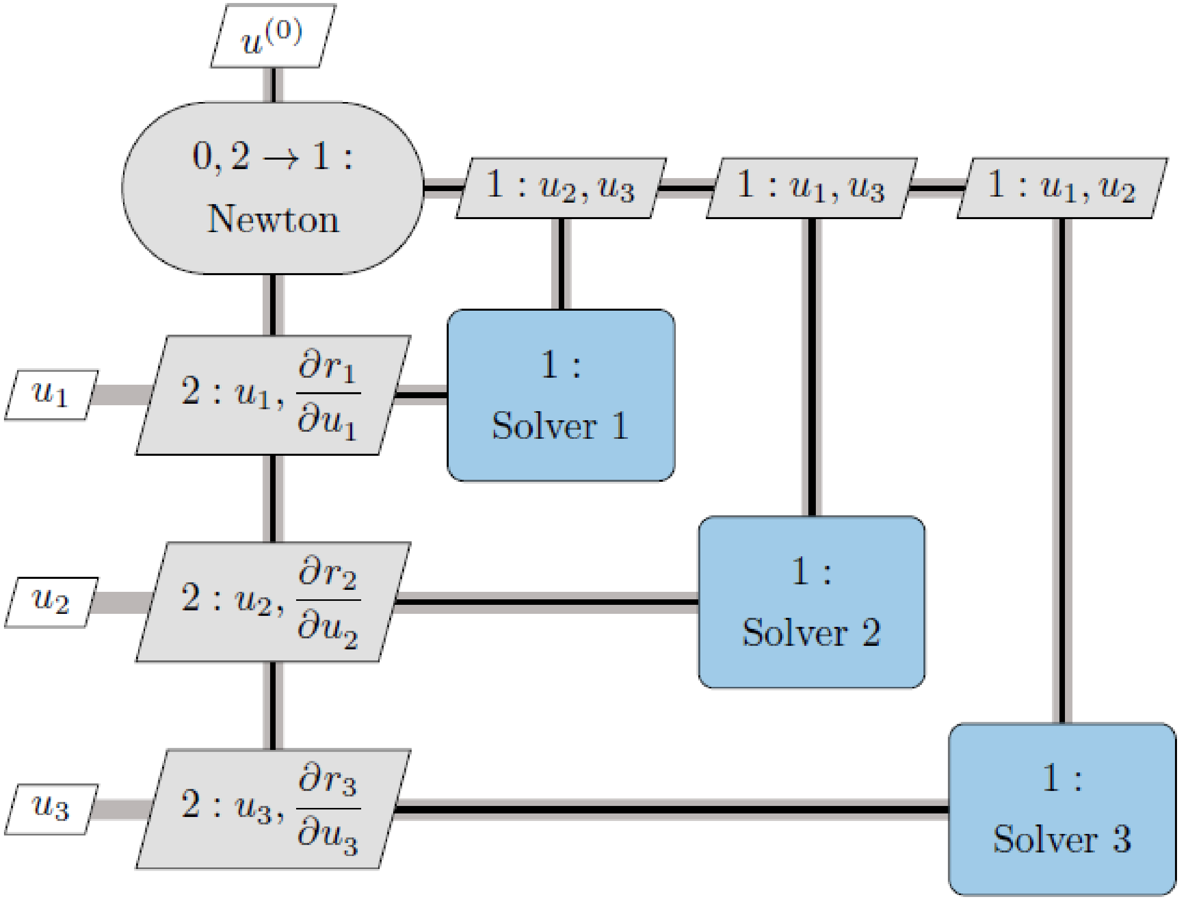

$u^{(0)}$
$0, 2 \to 1 :$ Newton
$1 : u_2, u_3$
$1 : u_1, u_3$
$1 : u_1, u_2$
u_1
$2 : u_1, \frac{\partial r_1}{\partial u_1}$
$1 :$ Solver 1
u_2
$2 : u_2, \frac{\partial r_2}{\partial u_2}$
$1 :$ Solver 2
u_3
$2 : u_3, \frac{\partial r_3}{\partial u_3}$
$1 :$ Solver 3

Figure 7.32 Full-space Newton solver for a three-component coupled system.

Source: Martins et Ning, 2021

4.5.3. Newton's method

- To apply Newton's method in this case, we express the functional form (Eq. 7.04) as residuals by using the same technique we used to convert an explicit function to the residual form (Eq. 7.02).
- This yields

$$\hat{r}_i(\hat{u}) = \hat{u}_i - U_i\left(u_{j\neq i}\right) \tag{7.10}$$

where $\hat{u}_i$ represents the guesses for the coupling variables, and U_i represents the actual computed values.

4.5.3. Newton's method

- For a system of nonlinear residual equations, the Newton step in the coupling variables, $\Delta\hat{u} = \hat{u}^{(k+1)} - \hat{u}^{(k)}$, can be found by solving the linear system

$$\left.\frac{\partial \hat{r}}{\partial \hat{u}}\right|_{\hat{u}=\hat{u}^{(k)}} \Delta\hat{u} = -\hat{r}\left(\hat{u}^{(k)}\right) \qquad (7.11)$$

where we need the partial derivatives of all the residuals with respect to the coupling variables to form the Jacobian matrix $\partial\hat{r}/\partial\hat{u}$.

- The Jacobian can be found by differentiating Eq. 7.10 with respect to the coupling variables.

4.5.3. Newton's method

- Then, expanding the concatenated residuals and coupling variable vectors yields

$$\begin{bmatrix} I & -\frac{\partial U_1}{\partial \hat{u}_2} & \cdots & -\frac{\partial U_1}{\partial \hat{u}_m} \\ -\frac{\partial U_2}{\partial \hat{u}_1} & I & \cdots & -\frac{\partial U_2}{\partial \hat{u}_m} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{\partial U_m}{\partial \hat{u}_1} & -\frac{\partial U_m}{\partial \hat{u}_2} & \cdots & I \end{bmatrix} \begin{bmatrix} \Delta\hat{u}_1 \\ \Delta\hat{u}_2 \\ \vdots \\ \Delta\hat{u}_m \end{bmatrix} = - \begin{bmatrix} \hat{u}_1 - U_1(\hat{u}_2, \ldots, \hat{u}_m) \\ \hat{u}_2 - U_2(\hat{u}_1, \hat{u}_3, \ldots, \hat{u}_m) \\ \vdots \\ \hat{u}_m - U_m(\hat{u}_1, \ldots, \hat{u}_{m-1}) \end{bmatrix} \quad (7.12)$$

- The residuals in the right-hand side of this equation are evaluated at the current iteration.
- The derivatives in the block Jacobian matrix are also computed at the current iteration.

4.5.3. Newton’s method

- Each row *i* represents the derivatives of the (potentially implicit) function that computes the outputs of component *i* with respect to all the inputs of that component.
- The Jacobian matrix in Eq. 7.12 has the same structure as the DSM (but transposed) and is often sparse.
- The derivatives are partial derivatives in the sense that they do not take into account the coupled system.
- However, they must take into account the respective model and can be computed using implicit analytic methods when the model is implicit.
- This Newton solver is shown in Fig. 7.33 and detailed in Alg. 7.4.
- Each component corresponds to a set of rows in the block Newton system (Eq. 7.12).

4.5.3. Newton's method

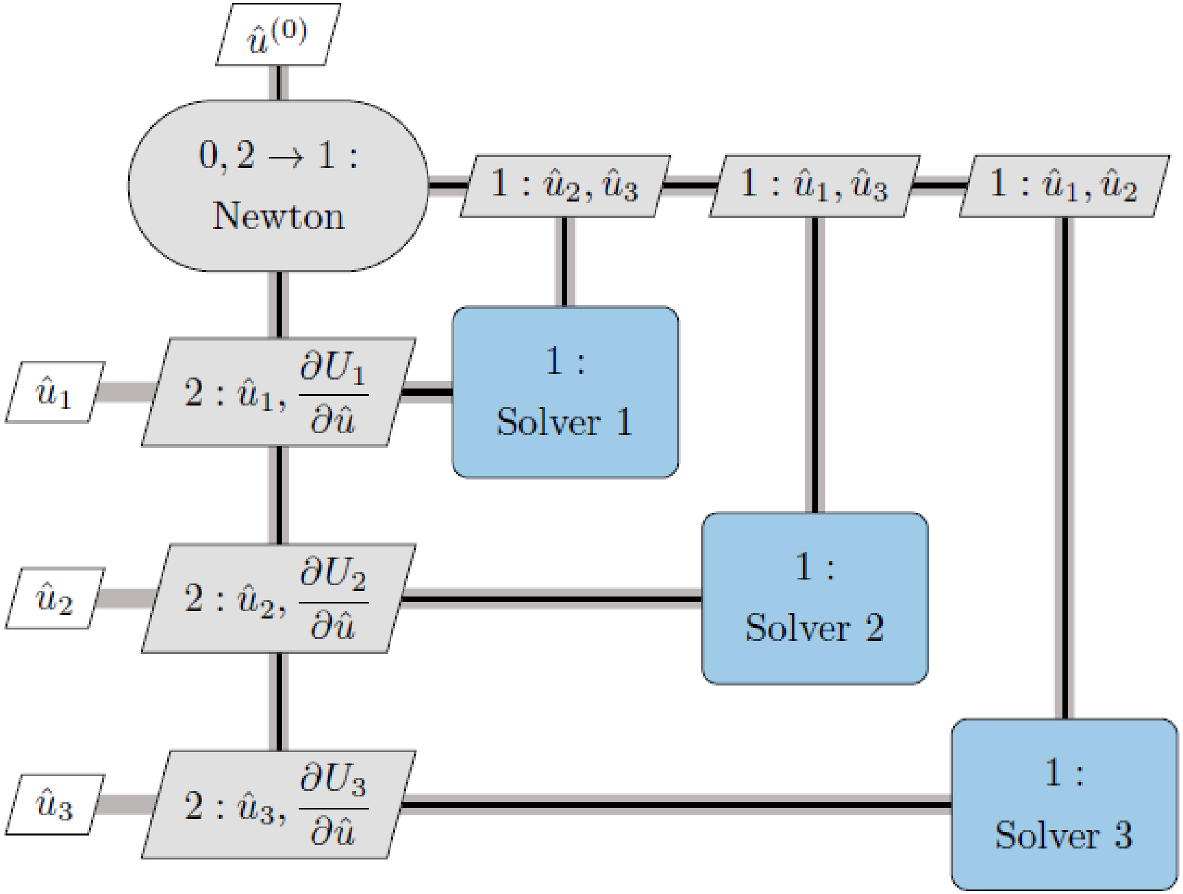

$\hat{u}^{(0)}$
$0, 2 \to 1:$ Newton
$1: \hat{u}_2, \hat{u}_3$
$1: \hat{u}_1, \hat{u}_3$
$1: \hat{u}_1, \hat{u}_2$
$\hat{u}_1$
$2: \hat{u}_1, \frac{\partial U_1}{\partial \hat{u}}$
$1:$ Solver 1
$\hat{u}_2$
$2: \hat{u}_2, \frac{\partial U_2}{\partial \hat{u}}$
$1:$ Solver 2
$\hat{u}_3$
$2: \hat{u}_3, \frac{\partial U_3}{\partial \hat{u}}$
$1:$ Solver 3

Figure 7.33 Reduced-space Newton solver for a three-component coupled system..

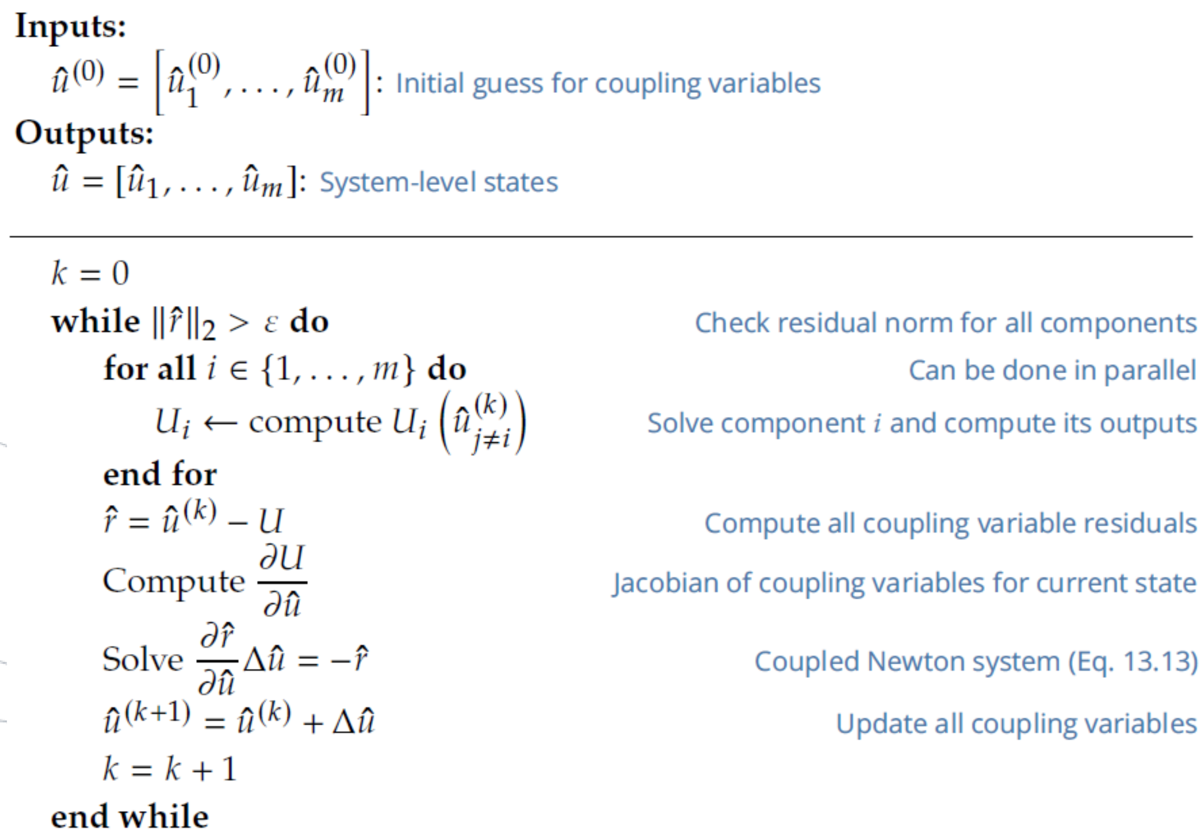
Source: Martins et Ning, 2021

4.5.3. Newton’s method

- To compute each set of rows, the corresponding component must be solved, and the derivatives of its outputs with respect to its inputs must be computed as well.
- Each set can be computed in parallel, but once the system is assembled, a step in the coupling variables is computed by solving the full system (Eq. 7.12).
- These coupled Newton methods have similar advantages and disadvantages to the plain Newton method.
- The main advantage is that it converges quadratically once it is close enough to the solution (if the problem is well-conditioned).
- The main disadvantage is that it might not converge at all, depending on the initial guess.

4. Coupled Models
4.5. Solving coupled numerical models
4.5.3. Newton’s method

Algorithm 7.4: Reduced-space hierarchical Newton



Inputs:
$\hat{u}^{(0)} = \left[\hat{u}_1^{(0)}, \ldots, \hat{u}_m^{(0)}\right]$: Initial guess for coupling variables

Outputs:
$\hat{u} = [\hat{u}_1, \ldots, \hat{u}_m]$: System-level states

```
$k = 0$
while $\|\hat{r}\|_2 > \varepsilon$ do                              Check residual norm for all components
    for all $i \in \{1, \ldots, m\}$ do                          Can be done in parallel
        $U_i \leftarrow$ compute $U_i\left(\hat{u}_{j \neq i}^{(k)}\right)$           Solve component $i$ and compute its outputs
    end for
    $\hat{r} = \hat{u}^{(k)} - U$                                 Compute all coupling variable residuals
    Compute $\frac{\partial U}{\partial \hat{u}}$                    Jacobian of coupling variables for current state
    Solve $\frac{\partial \hat{r}}{\partial \hat{u}} \Delta \hat{u} = -\hat{r}$                   Coupled Newton system (Eq. 13.13)
    $\hat{u}^{(k+1)} = \hat{u}^{(k)} + \Delta \hat{u}$                       Update all coupling variables
    $k = k + 1$
end while
```

4. Coupled Models

4.5. Solving coupled numerical models

4.5.3. Newton's method

- One disadvantage specific to the coupled Newton methods is that it requires formulating and solving the coupled linear system (Eq. 7.12) at each iteration.
- If the Jacobian $\partial r/\partial u$ is not readily available, Broyden's method can compute an approximate of the Jacobian inverse ($\tilde{J}^{-1}$) by starting with a guess (say,$\tilde{J}_0^{-1} = 0$) and then using the update

$$\tilde{J}^{-1}{}^{(k+1)} = \tilde{J}^{-1}{}^{(k)} + \frac{\left(\Delta u^{(k)} - \tilde{J}^{-1}{}^{(k)} \Delta r^{(k)}\right) \Delta u^{(k)}{}^{T}}{\Delta r^{(k)}{}^{T} \Delta r^{(k)}} \qquad (7.13)$$

where $\Delta u^{(k)}$ is the last step in the states, and $\Delta r^{(k)}$ is the difference between the two latest residual vectors.

4. Coupled Models

4.5. Solving coupled numerical models

4.5.3. Newton's method

- Because the inverse is provided explicitly, we can find the update by performing the multiplication

$$\Delta u^{(k)} = \tilde{J}^{-1} r^{(k)} \tag{7.14}$$

4.5. Solving coupled numerical models

Example 7.7: Aerostructural solver comparison.

- We now apply the coupled solution methods presented in this section to the implicit parts of the aerostructural model, which are the two first residuals from Ex. 7.5,

$$r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} A(d)\Gamma - v(d) \\ Kd - q(\Gamma) \end{bmatrix}$$

and the variables are the circulations and displacements,

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \equiv \begin{bmatrix} \Gamma \\ d \end{bmatrix}$$

- In this case, the linear systems defined by r_1 and r_2 are small enough to be solved using a direct method, such as LU factorization.
- Thus, we can solve r_1 for Γ, for a given d, and solve r_2 for d, for a given Γ.

4.5. Solving coupled numerical models

Example 7.7: Aerostructural solver comparison (continued).

- Also, no conversions are involved, so the set of coupling variables is equivalent to the set of state variables ($\hat{u} = u$).

Solving with nonlinear block Jacobi:

- Using the nonlinear block Jacobi method (Alg. 7.1), we start with an initial guess (e.g., Γ=0, d=0) and solve r_1=0 and r_2=0 separately for the new values of Γ and d, respectively.
- Then we use these new values of Γ and d to solve r_1=0 and r_2=0 again, and so on until convergence.

4.5. Solving coupled numerical models

Example 7.7: Aerostructural solver comparison (continued).

Solving with nonlinear block Gauss-Seidel:

- Using the nonlinear block Gauss–Seidel method (Alg. 7.2) is similar, but we need to solve the two components in sequence.
- We can start by solving r_1=0 for Γ with d=0.
- Then we use the Γ obtained from this solution in r_2 and solve for a new d.
- We now have a new d to use in r_1 to solve for a new Γ, and so on.

4.5. Solving coupled numerical models

Example 7.7: Aerostructural solver comparison (continued).

Solving with Newton:

- The Jacobian for the Newton system (Eq. 7.08) is

$$\frac{\partial r}{\partial u} = \begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \frac{\partial r_1}{\partial u_2} \\ \frac{\partial r_2}{\partial u_1} & \frac{\partial r_2}{\partial u_2} \end{bmatrix} = \begin{bmatrix} A & \frac{\partial A}{\partial d}\Gamma - \frac{\partial v}{\partial d} \\ -\frac{\partial q}{\partial \Gamma} & K \end{bmatrix}$$

because

$$r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} A(d)\Gamma - v(d) \\ Kd - q(\Gamma) \end{bmatrix}$$

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \equiv \begin{bmatrix} \Gamma \\ d \end{bmatrix}$$

4.5. Solving coupled numerical models

Example 7.7: Aerostructural solver comparison (continued).

Solving with Newton (continued):

- We already have the block diagonal matrices in this Jacobian from the governing equations, but we need to compute the off-diagonal partial derivative blocks, which can be done analytically or with algorithmic differentiation (AD).
- The monolithic Newton approach does not converge in this case.
- We apply the full-space hierarchical approach (Alg. 7.3), which converges more reliably.
- In this case, the reduced-space approach is not used because there is not distinction between coupling variables and state variables.

4.5. Solving coupled numerical models

Example 7.7: Aerostructural solver comparison (continued).

- The solution is shown in Fig. 7.34, where we plot the variation of lift, vertical displacement, and rotation along the span.
- The vertical displacements are a subset of *d*, and the rotations are a conversion of a subset of *d* representing the rotations of the wing section at each spanwise location.
- The lift is the vertical force at each spanwise location, which is proportional to Γ times the wing chord at that location.

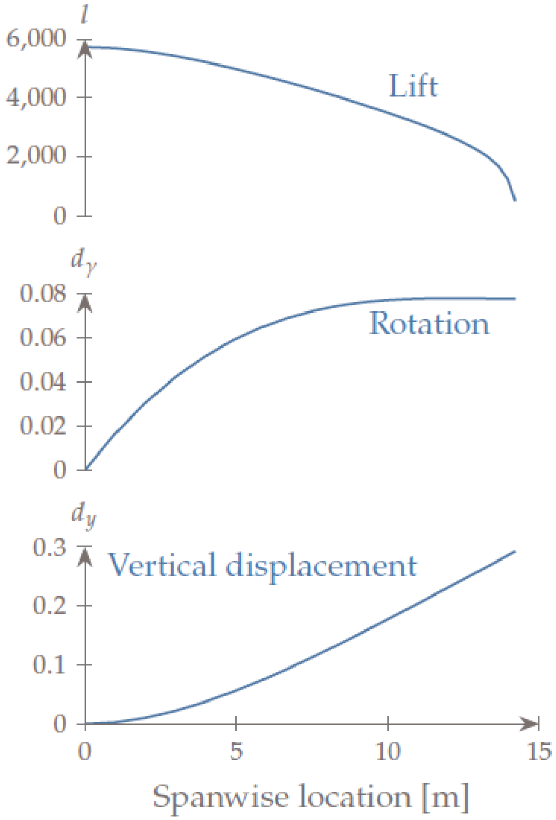


Figure 7.34 Spanwise distribution of the lift, wing rotation (d_γ), and vertical displacement (d_ψ) for the coupled aerostructural solution.

Source: Martins et Ning, 2021

4.5. Solving coupled numerical models

Example 7.7: Aerostructural solver comparison (continued).

- In Fig. 7.35, we compare the convergence of the methods introduced in this section.
- The Jacobi method has the poorest convergence rate and oscillates.
- The Gauss–Seidel method is much better, and it is even better with Aitken cceleration.
- Newton has the highest convergence rate, as expected.
- Broyden performs about as well as Gauss–Seidel in this case.

4.5. Solving coupled numerical models

Example 7.7: Aerostructural solver comparison (continued).

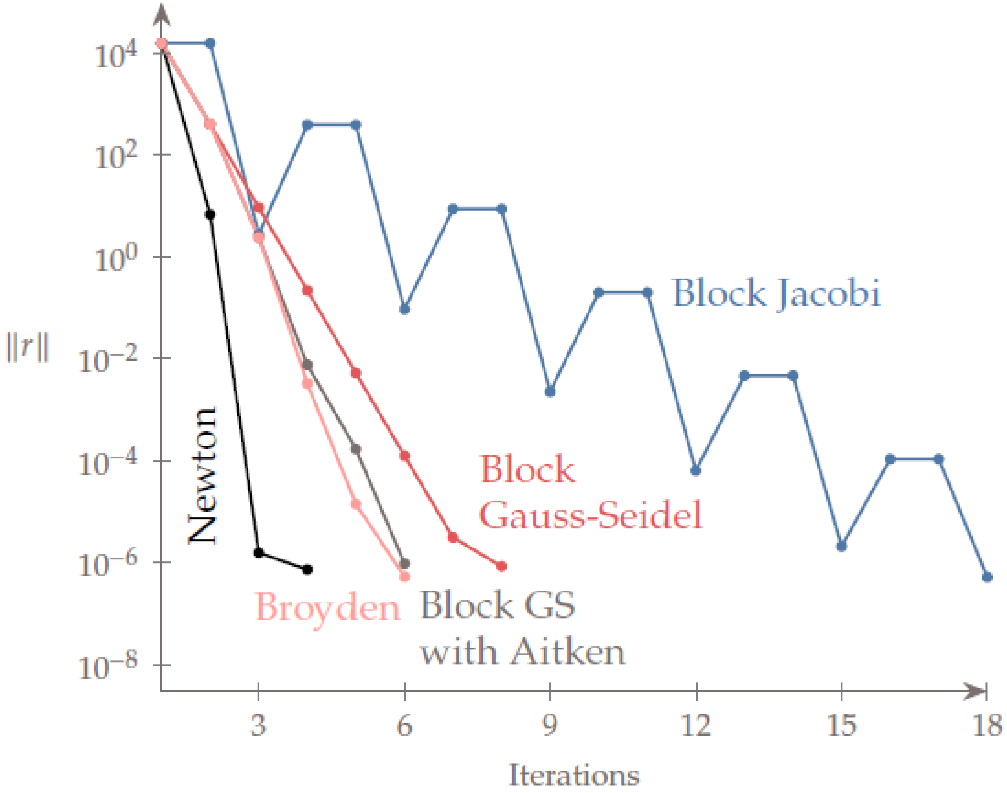


Figure 7.34 Convergence of each solver for aerostructural system.

Source: Martins et Ning, 2021

4.6. Hierarchical solvers for coupled systems

- The coupled solvers we discussed so far already use a two-level hierarchy because they require a solver for each component and a second level that solves the group of components.
- This hierarchy can be extended to three and more levels by making groups of groups.
- Modular analysis and unified derivatives (MAUD) is a mathematical framework developed for this purpose.
- Using MAUD, we can mix residual and functional forms and seamlessly handle implicit and explicit components.□
- The hierarchy of solvers can be represented as a tree data structure, where the nodes are the solvers and the leaves are the components, as shown in Fig. 7.35 for a system of six components and five solvers.

4.6. Hierarchical solvers for coupled systems

- The root node ultimately solves the complete system, and each solver is responsible for a subsystem and thus handles a subset of the variables.

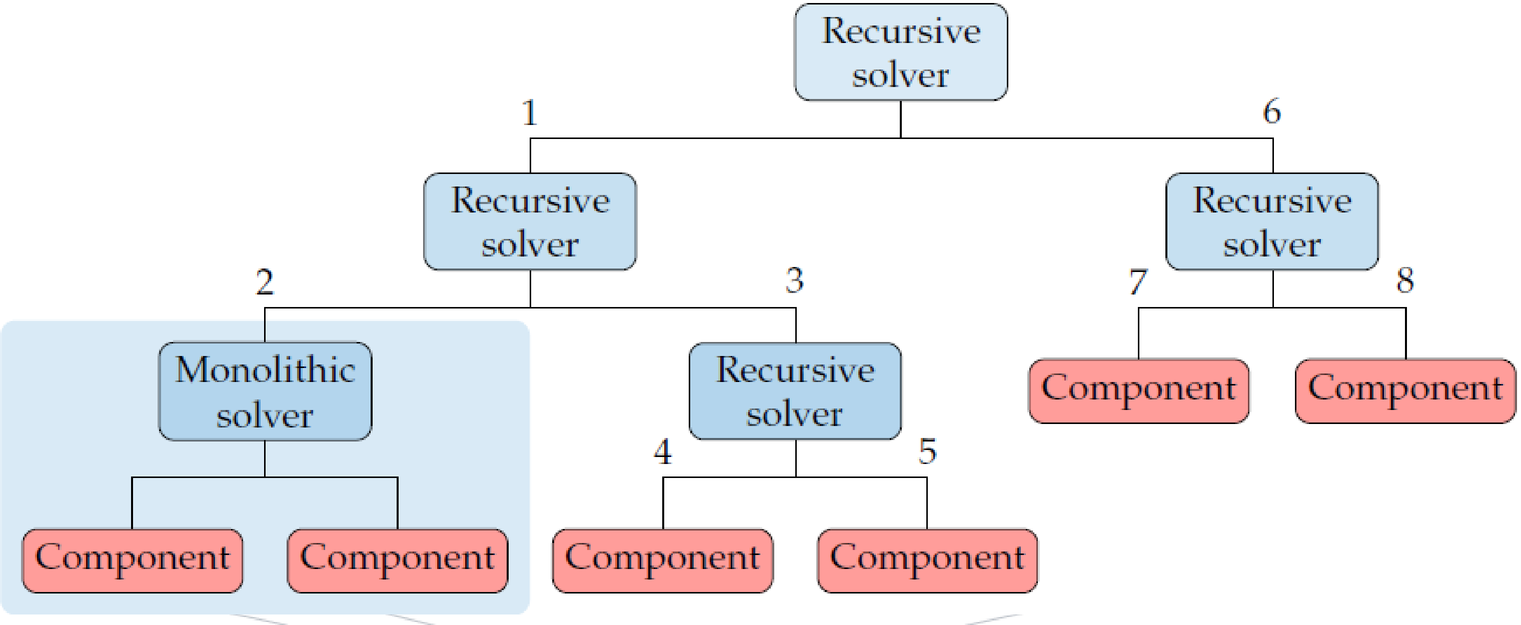


Figure 7.35 A system of components can be organized in a solver hierarchy.

Source: Martins et Ning, 2021

4.6. Hierarchical solvers for coupled systems

- There are two possible types of solvers: monolithic and recursive.
- Monolithic solvers can only have components as children and handle all their variables simultaneously using the residual form.
- Of the methods we introduced in the previous section, only monolithic and full-space Newton (and Broyden) can do this for nonlinear systems.
- Linear systems can be solved in a monolithic fashion using a direct solver or an iterative linear solver, such as a Krylov subspace method.
- Recursive solvers, as the name implies, visit all the child nodes in turn.

4.6. Hierarchical solvers for coupled systems

- If a child node turns out to be another recursive solver, it does the same until a component is reached.
- The block Jacobi and Gauss–Seidel methods can be used as recursive solvers for nonlinear and linear systems.
- The reduced-space Newton and Broyden methods can also be recursive solvers.
- For the hypothetical system shown in Fig. 7.35, the numbers show the order in which each solver and component would be called.
- The hierarchy of solvers should be chosen to exploit the system structure.
- MAUD also facilitates parallel computation when subsystems are uncoupled, which provides further opportunities to exploit the structure of the problem.

4.6. Hierarchical solvers for coupled systems

- Figs. 7.36 and 7.37 show several possibilities.

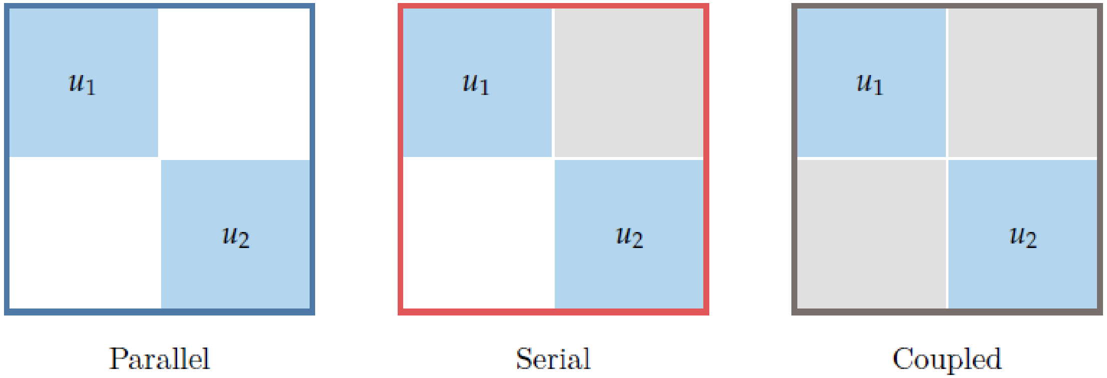


Figure 7.36 There are three main possibilities involving two components.

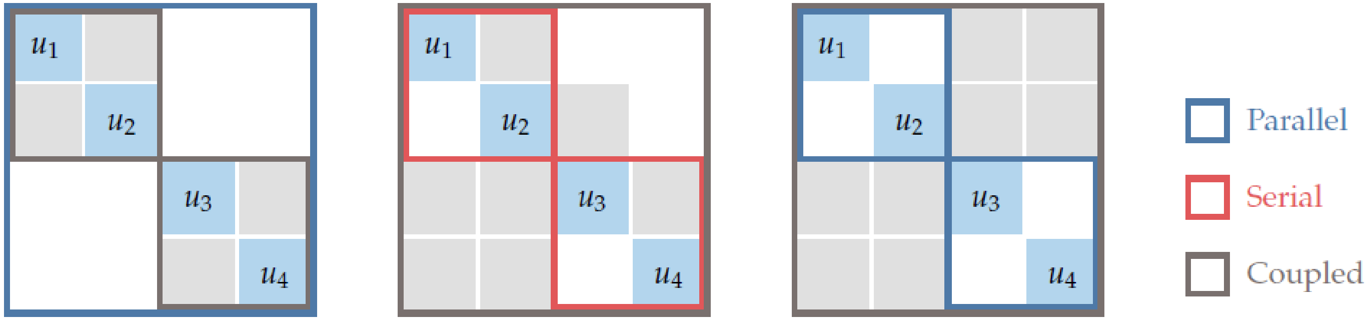


Figure 7.37 Three examples of a system of four components with a two-level solver hierarchy.

Source: Martins et Ning, 2021

4.6. Hierarchical solvers for coupled systems

- The three two-component systems in Fig. 7.36 show three different coupling modes.
- In the first mode, the two components are independent of each other and can therefore be solved in parallel using any solvers appropriate for each of the components.
- In the serial case, component 2 depends on 1, but not the other way around.
- Therefore, we can converge to the coupled solution using one block Gauss–Seidel iteration.
- If the dependency were reversed (feedback but no feed forward), the order of the two components would be switched.
- Finally, the fully coupled case requires an iterative solution using any of the methods from Section 7.4.5.
- MAUD is designed to handle these three coupling modes.

4.6. Hierarchical solvers for coupled systems

- Figure 7.37 shows three possibilities for a four-component system where two levels of solvers can be used.
- In the first one (on the left), we require a coupled solver for components 1 and 2 and another for components 3 and 4, but no further solving is needed.
- In the second (middle pane of Fig. 7.37), components 1 and 2 as well as components 3 and 4 can be solved serially, but these two groups require a coupled solution.
- For the two levels to converge, the serial and coupled solutions are called repeatedly until the two solvers agree with each other.
- The third possibility (right pane of Fig. 7.37) has two systems that have two independent components, which can each be solved in parallel, but the overall system is coupled.

4.6. Hierarchical solvers for coupled systems

- With MAUD, we can set up any of these sequences of solvers through the solver hierarchy tree, as illustrated in Fig. 7.35.
- To solve the system from Ex. 7.3 using hierarchical solvers, we can use the hierarchy shown in Fig. 7.38.
- We form three groups with three components each.
- Each group includes the input and output conversion components (which are explicit) and one implicit component (which requires its own solver).
- Serial solvers can be used to handle the input and output conversion components.
- A coupled solver is required to solve the entire coupled system, but the coupling between the groups is restricted to the corresponding outputs (components 3, 6, and 9).

4.6. Hierarchical solvers for coupled systems

- Alternatively, we could apply a coupled solver to the functional representation shown on the right of Fig. 7.25.
- This would also use two levels of solvers: a solver within each group and a system-level solver for the coupling of the three groups.
- However, the system-level solver would handle coupling variables rather than the residuals of each component.

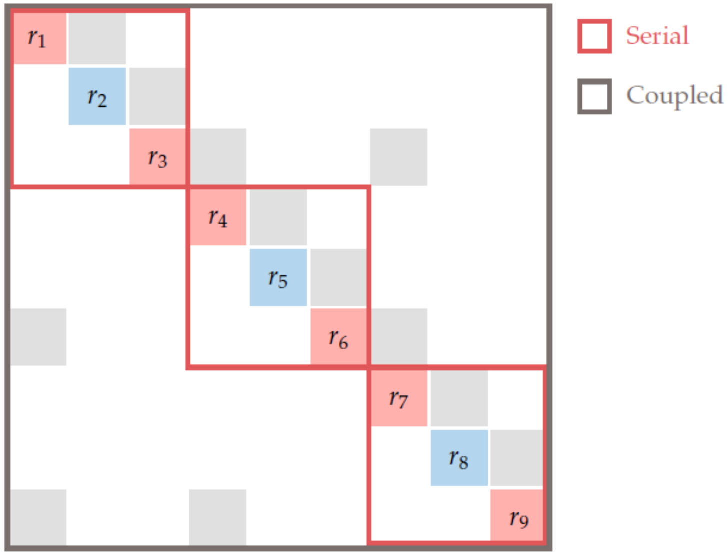


Figure 7.38 For the case of Fig. 7.25, we can use a serial evaluation within each of the three groups and require a coupled solver to handle the coupling between the three groups.

Source: Martins et Ning, 2021

4.6. Hierarchical solvers for coupled systems

- The development of coupled solvers is often done for a specific set of models from scratch, which requires substantial effort.
- OpenMDAO (https://openmdao.org/) is an open-source framework that facilitates such efforts by implementing MAUD.
- All the solvers introduced in this chapter are available in OpenMDAO.
- This framework also makes it easier to compute the derivatives of the coupled system, as we will see in the next section.
- Users can assemble systems of mixed explicit and implicit components.
- For implicit components, they must give OpenMDAO access to the residual computations and the corresponding state variables.

4.6. Hierarchical solvers for coupled systems

- For explicit components, OpenMDAO only needs access to the inputs and the outputs, so it supports black-box models.
- OpenMDAO is usually more efficient when the user provides access to the residuals and state variables instead of treating models as black boxes.
- A hierarchy of multiple solvers can be set up in OpenMDAO, as illustrated in Fig. 7.35.
- OpenMDAO also provides the necessary interfaces for user-defined solvers.
- Finally, OpenMDAO encourages coupling through memory, which is beneficial for numerical precision and computational efficiency.

5. Coupled Derivative Computation

- The gradient-based optimization algorithms from Chapters 3 and 4 require the derivatives of the objective and constraints with respect to the design variables.
- Any of the methods for computing derivatives can be used to compute the derivatives of coupled models, but some modifications are required.
- The main difference is that in MDO, the computation of the functions of interest (objective and constraints) requires the solution of the multidisciplinary model.

5.1. Finite differences

- The finite-difference method can be used with no modification, as long as an MDA is converged well enough for each perturbation in the design variables.
- The cost of computing derivatives with the finite-difference method is proportional to the number of variables.
- The constant of proportionality can increase significantly compared with that of a single discipline because the MDA convergence might be slow (especially if using a block Jacobi or Gauss–Seidel iteration).
- The accuracy of finite-difference derivatives depends directly on the accuracy of the functions of interest.
- When the functions are computed from the solution of a coupled system, their accuracy depends both on the accuracy of each component and the accuracy of the MDA.

5.1. Finite differences

- To address the latter, the MDA should be converged well enough.

5.2. Complex step and AD

- The complex-step method and forward-mode AD can also be used for a coupled system, but some modifications are required.
- The complex-step method requires all components to be able to take complex input and compute the corresponding complex outputs.
- Similarly, AD requires inputs and outputs that include derivative information.
- For a given MDA, if one of these methods is applied to each component and the coupling includes the derivative information, we can compute the derivatives of the coupled system.
- The propagation of the forward mode seed (or the complex step) is illustrated in Figure 7.39 for a system of two components.

5.2. Complex step and AD

- When using AD, manual coupling is required if the components and the coupling are programmed in different languages.
- The complex-step method can be more straightforward to implement than AD for cases where the models are implemented in different languages, and all the languages support complex arithmetic.

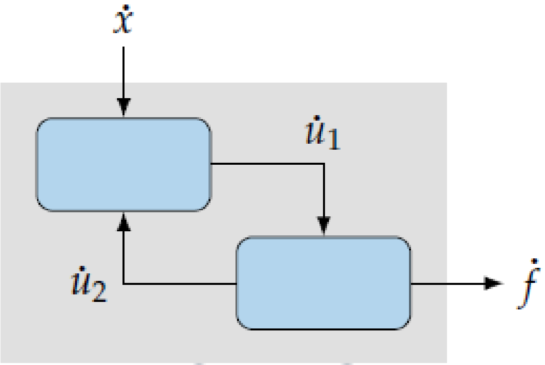


Figure 7.39 Forward mode of AD for a system of two components.

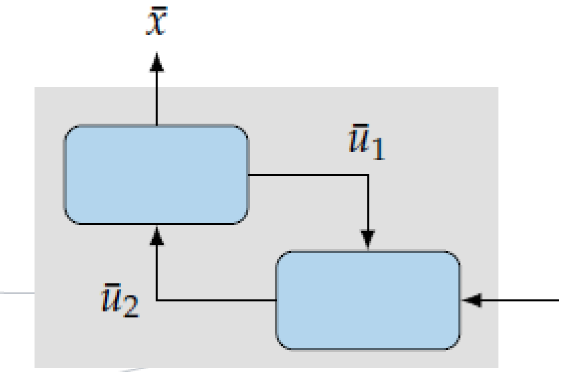


Figure 7.40 Reverse mode of AD for a system of two components.

Source: Martins et Ning, 2021

5.2. Complex step and AD

- Although both of these methods produce accurate derivatives for each component, the accuracy of the derivatives for the coupled system could be compromised by a low level of convergence of the MDA.
- The reverse mode of AD for coupled systems would be more involved: after an initial MDA, we would run a reverse MDA to compute the derivatives, as illustrated in Fig. 7.40.

5.3. Implicit analytic methods

- The implicit analytic methods (both direct and adjoint) can also be extended to compute the derivatives of coupled systems.
- All the equations derived for a single component are valid for coupled systems if we concatenate the residuals and the state variables.
- Furthermore, we can mix explicit and implicit components using concepts introduced in the UDE.
- Finally, when using the MAUD approach, the coupled derivative computation can be done using the same hierarchy of solvers.

5.3.1. Coupled derivatives of residual representation

- In Eq. 7.01, we denoted the coupled system as a series of concatenated residuals, $r_i(u)=0$, and variables u_i corresponding to each component $i=1,\ldots,n$ as

$$r(u) \equiv \begin{bmatrix} r_1(u) \\ \vdots \\ r_n(u) \end{bmatrix}, \qquad u \equiv \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}, \tag{7.15}$$

where the residual for each component, r_i, could depend on all states u.

- To derive the coupled version of the direct and adjoint methods, we apply them to the concatenated vectors.

5.3.1. Coupled derivatives of residual representation

- Thus, the coupled version of the linear system for the direct method is

$$\begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \cdots & \frac{\partial r_n}{\partial u_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_n}{\partial u_1} & \cdots & \frac{\partial r_n}{\partial u_n} \end{bmatrix} \begin{bmatrix} \emptyset_1 \\ \vdots \\ \emptyset_n \end{bmatrix} = \begin{bmatrix} \frac{\partial r_1}{\partial x} \\ \vdots \\ \frac{\partial r_n}{\partial x} \end{bmatrix}, \qquad (7.16)$$

where $\emptyset_i$ represents the derivatives of the states from component i with respect to the design variables.

- Once we have solved for $\emptyset$, we can use the coupled equivalent of the total derivative equation to compute the derivatives:

$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \begin{bmatrix} \frac{\partial f}{\partial u_1} \cdots \frac{\partial f}{\partial u_n} \end{bmatrix} \begin{bmatrix} \emptyset_1 \\ \vdots \\ \emptyset_n \end{bmatrix}. \qquad (7.17)$$

5.3.1. Coupled derivatives of residual representation

- The coupled adjoint equations can be written as

$$\begin{bmatrix} \dfrac{\partial r_1}{\partial u_1}^T & \cdots & \dfrac{\partial r_n}{\partial u_1}^T \\ \vdots & \ddots & \vdots \\ \dfrac{\partial r_n}{\partial u_1}^T & \cdots & \dfrac{\partial r_n}{\partial u_n}^T \end{bmatrix} \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_n \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial u_1}^T \\ \vdots \\ \dfrac{\partial f}{\partial u_n}^T \end{bmatrix}, \qquad (7.18)$$

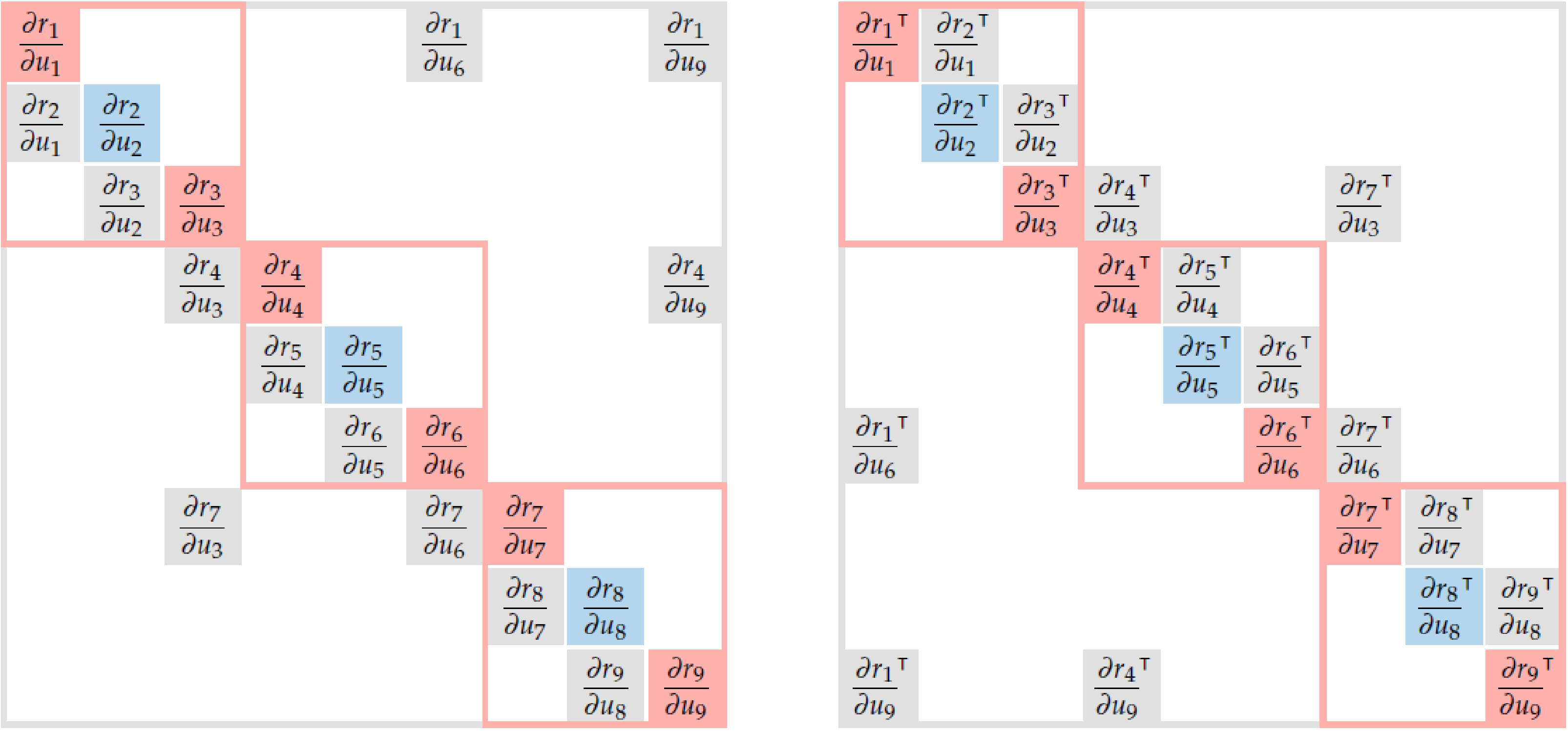
- After solving for the coupled-adjoint vector using the previous equation, we can use the total derivative equation to compute the desired derivatives:

$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \left[\psi_1{}^T \cdots \psi_n{}^T\right] \begin{bmatrix} \dfrac{\partial r_1}{\partial x} \\ \vdots \\ \dfrac{\partial r_n}{\partial x} \end{bmatrix}. \qquad (7.19)$$

5.3.1. Coupled derivatives of residual representation

- Like the adjoint method, the coupled adjoint is a powerful approach for computing gradients with respect to many design variables.
- The required partial derivatives are the derivatives of the residuals or outputs of each component with respect to the state variables or inputs of all other components.
- In practice, the block structure of these partial derivative matrices is sparse, and the matrices themselves are sparse.
- This sparsity can be exploited using graph colouring to drastically reduce the computation effort of computing Jacobians at the system or component level.
- Figure 7.41 shows the structure of the Jacobians in Eq. 7.16 and Eq. 7.18 for the three-group case from Fig. 7.35.

5.3.1. Coupled derivatives of residual representation



$$\begin{bmatrix}
\frac{\partial r_1}{\partial u_1} & & & & & \frac{\partial r_1}{\partial u_6} & & & \frac{\partial r_1}{\partial u_9} \\
\frac{\partial r_2}{\partial u_1} & \frac{\partial r_2}{\partial u_2} & & & & & & & \\
 & \frac{\partial r_3}{\partial u_2} & \frac{\partial r_3}{\partial u_3} & & & & & & \\
 & & \frac{\partial r_4}{\partial u_3} & \frac{\partial r_4}{\partial u_4} & & & & & \frac{\partial r_4}{\partial u_9} \\
 & & & \frac{\partial r_5}{\partial u_4} & \frac{\partial r_5}{\partial u_5} & & & & \\
 & & & & \frac{\partial r_6}{\partial u_5} & \frac{\partial r_6}{\partial u_6} & & & \\
 & & \frac{\partial r_7}{\partial u_3} & & & \frac{\partial r_7}{\partial u_6} & \frac{\partial r_7}{\partial u_7} & & \\
 & & & & & & \frac{\partial r_8}{\partial u_7} & \frac{\partial r_8}{\partial u_8} & \\
 & & & & & & & \frac{\partial r_9}{\partial u_8} & \frac{\partial r_9}{\partial u_9}
\end{bmatrix}$$

Direct Jacobian

$$\begin{bmatrix}
\frac{\partial r_1}{\partial u_1}^{\intercal} & \frac{\partial r_2}{\partial u_1}^{\intercal} & & & & & & & \\
 & \frac{\partial r_2}{\partial u_2}^{\intercal} & \frac{\partial r_3}{\partial u_2}^{\intercal} & & & & & & \\
 & & \frac{\partial r_3}{\partial u_3}^{\intercal} & \frac{\partial r_4}{\partial u_3}^{\intercal} & & & \frac{\partial r_7}{\partial u_3}^{\intercal} & & \\
 & & & \frac{\partial r_4}{\partial u_4}^{\intercal} & \frac{\partial r_5}{\partial u_4}^{\intercal} & & & & \\
 & & & & \frac{\partial r_5}{\partial u_5}^{\intercal} & \frac{\partial r_6}{\partial u_5}^{\intercal} & & & \\
\frac{\partial r_1}{\partial u_6}^{\intercal} & & & & & \frac{\partial r_6}{\partial u_6}^{\intercal} & \frac{\partial r_7}{\partial u_6}^{\intercal} & & \\
 & & & & & & \frac{\partial r_7}{\partial u_7}^{\intercal} & \frac{\partial r_8}{\partial u_7}^{\intercal} & \\
 & & & & & & & \frac{\partial r_8}{\partial u_8}^{\intercal} & \frac{\partial r_9}{\partial u_8}^{\intercal} \\
\frac{\partial r_1}{\partial u_9}^{\intercal} & & & \frac{\partial r_4}{\partial u_9}^{\intercal} & & & & & \frac{\partial r_9}{\partial u_9}^{\intercal}
\end{bmatrix}$$

Adjoint Jacobian

Figure 7.41 Jacobian structure for residual form of the coupled direct (left) and coupled adjoint (right) equations for the three-group system of Fig. 7.25. The structure of the transpose of the Jacobian is the same as that of the DSM.

Source: Martins et Ning, 2021

5.3.1. Coupled derivatives of residual representation

- The sparsity structure of the Jacobian is the transpose of the DSM structure.
- Because the Jacobian Eq. 7.18 is transposed, the Jacobian in the adjoint equation has the same structure as the DSM.
- The structure of the linear system can be exploited in the same way as for the nonlinear system solution using hierarchical solvers: serial solvers within each group and a coupled solver for the three groups.
- The partial derivatives in the coupled Jacobian, the right-hand side of the linear systems (Eqs. 7.16 and 7.18), and the total derivatives equations (Eqs. 7.17 and 7.19) can be easily computed.
- The nature of these derivatives is the same as that for implicit analytic methods.

5.3.1. Coupled derivatives of residual representation

- They do not require the solution of the equation and are typically cheap to compute.
- Ideally, the components would already have analytic derivatives of their outputs with respect to their inputs, which are all the derivatives needed at the system level.
- The partial derivatives can also be computed using the finite difference or complex-step methods.
- Even though these are not efficient for cases with many inputs, it might still be more efficient to compute the partial derivatives with these methods and then solve the coupled derivative equations instead of performing a finite difference of the coupled system, as described in Section 7.4.1.

5.3.1. Coupled derivatives of residual representation

- The reason is that computing the partial derivatives only avoids having to reconverge the coupled system for every input perturbation.
- In addition, the coupled system derivatives should be more accurate when finite differences are used only to compute the partial derivatives.

5.3.2. Coupled derivatives of functional representation

- Variants of the coupled direct and adjoint methods can also be derived for the functional form of the system-level representation (Eq. 7.03), by using the residuals defined for the system-level Newton solver (Eq. 7.10),

$$\hat{r}_i(\hat{u}) = \hat{u}_i - U_i\left(u_{j\neq i}\right) = 0 \quad i = 1, \cdots, m \qquad (7.20)$$

- Recall that driving these residuals to zero relies on a solver for each component to solve for each component's states and another solver to solve for the coupling variables $\hat{u}$.

5.3.2. Coupled derivatives of functional representation

- Using this new residual definition and the coupling variables, we can derive the functional form of the coupled direct method as

$$\begin{bmatrix} I & -\frac{\partial U_1}{\partial \hat{u}_2} & \cdots & -\frac{\partial U_1}{\partial \hat{u}_m} \\ -\frac{\partial U_2}{\partial \hat{u}_1} & I & \cdots & -\frac{\partial U_2}{\partial \hat{u}_m} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{\partial U_m}{\partial \hat{u}_1} & -\frac{\partial U_m}{\partial \hat{u}_2} & \cdots & I \end{bmatrix} \begin{bmatrix} \hat{\phi}_1 \\ \hat{\phi}_2 \\ \vdots \\ \hat{\phi}_m \end{bmatrix} = \begin{bmatrix} \frac{\partial \hat{U}_1}{\partial x} \\ \frac{\partial \hat{U}_2}{\partial x} \\ \vdots \\ \frac{\partial \hat{U}_m}{\partial x} \end{bmatrix} \qquad (7.21)$$

where the Jacobian is identical to the one we derived for the coupled Newton step (Eq. 7.12).

- Here, $\hat{\phi}_i$ represents the derivatives of the coupling variables from component *i* with respect to the design variables.

5.3.2. Coupled derivatives of functional representation

- The solution can then be used in the following equation to compute the total derivatives:

$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \left[\frac{\partial f}{\partial \hat{u}_1} \cdots \frac{\partial f}{\partial \hat{u}_m}\right]\begin{bmatrix}\hat{\phi}_1\\ \vdots\\ \hat{\phi}_m\end{bmatrix}. \tag{7.22}$$

- Similarly, the functional version of the coupled adjoint equations can be derived as

$$\begin{bmatrix} I & -\frac{\partial U_1}{\partial \hat{u}_2}^T & \cdots & -\frac{\partial U_1}{\partial \hat{u}_m}^T \\ -\frac{\partial U_2}{\partial \hat{u}_1}^T & I & \cdots & -\frac{\partial U_2}{\partial \hat{u}_m}^T \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{\partial U_m}{\partial \hat{u}_1}^T & -\frac{\partial U_m}{\partial \hat{u}_2}^T & \cdots & I \end{bmatrix}\begin{bmatrix}\hat{\psi}_1\\ \hat{\psi}_2\\ \vdots\\ \hat{\psi}_m\end{bmatrix} = \begin{bmatrix}\frac{\partial f}{\partial \hat{u}_1}^T\\ \frac{\partial f}{\partial \hat{u}_2}^T\\ \vdots\\ \frac{\partial f}{\partial \hat{u}_m}^T\end{bmatrix} \tag{7.23}$$

5.3.2. Coupled derivatives of functional representation

- After solving for the coupled-adjoint vector using the previous equation, we can use the total derivative equation to compute the desired derivatives:

$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \left[{\psi_1}^T \cdots {\psi_m}^T\right] \begin{bmatrix} \frac{\partial \hat{r}_1}{\partial x} \\ \vdots \\ \frac{\partial \hat{r}_m}{\partial x} \end{bmatrix}. \qquad (7.24)$$

- Because the coupling variables ($\hat{u}$) are usually a reduction of the internal state variables (u), the linear systems in Eqs. 7.21 and 7.23 are usually much smaller than that of the residual counterparts (Eqs. 7.16 and 7.18).

5.3.2. Coupled derivatives of functional representation

- However, unlike the partial derivatives in the residual form, the partial derivatives in the functional form Jacobian need to account for the solution of the corresponding component.
- When viewed at the component level, these derivatives are actually total derivatives of the component.
- When the component is an implicit set of equations, computing these derivatives with finite-differencing would require solving the component's equations for each variable perturbation.
- Alternatively, an implicit analytic method could be applied to the component to compute these derivatives.
- Figure 7.42 shows the Jacobian structure in the functional form of the coupled direct method (Eq. 7.21) for the case of Fig. 7.38.

5.3.2. Coupled derivatives of functional representation

- The dimension of this Jacobian is smaller than that of the residual form.
- Recall from Fig. 7.25 that U_1 corresponds to r_3, U_2 corresponds to r_6, and U_3 corresponds to r_9.
- Thus, the total size of this Jacobian corresponds to the sum of the sizes of components 3, 6, and 9, as opposed to the sum of the sizes of all nine components for the residual form.

$$\begin{bmatrix} I & -\dfrac{\partial U_1}{\partial \hat{u}_2} & -\dfrac{\partial U_1}{\partial \hat{u}_3} \\ -\dfrac{\partial U_2}{\partial \hat{u}_1} & I & -\dfrac{\partial U_2}{\partial \hat{u}_3} \\ -\dfrac{\partial U_3}{\partial \hat{u}_1} & -\dfrac{\partial U_3}{\partial \hat{u}_2} & I \end{bmatrix}$$

Figure 7.42 Jacobian of coupled derivatives for the functional form of Fig. 7.38.

Source: Martins et Ning, 2021

5.3.2. Coupled derivatives of functional representation

- However, as mentioned previously, partial derivatives for the functional form are more expensive to compute because they need to account for an implicit solver in each of the three groups.

5.3.3. UDE for coupled systems

- The coupled direct and adjoint equations derived in this section can be obtained from the unified derivatives equation (UDE) with the appropriate definitions of residuals and variables.
- The components corresponding to each block in these equations can also be implicit or explicit, which provides the flexibility to represent systems of heterogeneous components.
- MAUD implements the linear systems from these coupled direct and adjoint equations using the UDE.
- The overall linear system inherits the hierarchical structure defined for the nonlinear solvers.
- Instead of nonlinear solvers, we use linear solvers, such as a direct solver and Krylov (both monolithic) or block Jacobi or Gauss–Seidel (both recursive).

5.3.3. UDE for coupled systems

- Components can be expressed using residual or functional forms, making it possible to include black-box solvers in a hierarchy.
- Example 7.8 demonstrates the UDE approach to computing derivatives by building on the wing design problem presented in Ex. 7.4.

5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives.

- Let us now consider a wing design optimization problem based on the aerostructural model considered in Ex. 7.1.
- The design variables are as follows:

α: Angle of attack. This controls the amount of lift produced by the airplane.

b: Wingspan. This is a shared variable because it directly affects both the aerodynamic and structural models.

γ: Twist distribution along the wingspan, represented by a vector. This controls the relative lift loading in the spanwise direction, which affects the drag and the load distribution on the structure. It affects the aerodynamic model but not the structural model (because it is idealized as a beam).

5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives (continued).

t: Thickness distribution of beam along the wingspan, represented by a vector. This directly affects the weight and the stiffness. It does not affect the aerodynamic model.

- The objective is to minimize the fuel required for a given range *R*, which can be written as a function of drag, lift, and weight, as follows:

$$f = W\left[exp\left(\frac{RcD}{VL}\right) - 1\right]. \tag{7.25}$$

- The empty weight *W* only depends on *t* and *b*, and the dependence is explicit (it does not require solving the aerodynamic or structural models).

5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives (continued).

- The drag D and lift L depend on all variables once we account for the coupled system of equations.
- The remaining variables are fixed: R is the required range, V is the airplane's cruise speed, and c is the specific fuel consumption of the airplane's engines.
- We also need to constrain the stresses in the structure, σ, which are an explicit function of the displacements.
- To solve this optimization problem using gradient-based optimization, we need the coupled derivatives of f and σ with respect to α, b, γ, and t.
- Computing the derivatives of the aerodynamic and structural models separately is not sufficient.

5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives (continued).

- For example, a perturbation on the twist changes the loads, which then changes the wing displacements, which requires solving the aerodynamic model again.
- Coupled derivatives take this effect into account.
- We show the DSM for the system in Fig. 7.43.
- For brevity, we only discuss the derivatives required to compute the derivative of fuel burn with respect to span, but the other partial derivatives would follow the same rationale.
  - $\partial r / \partial u$ is identical to what we derived when solving the coupled aerostructural system in Ex. 7.7.

5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives (continued).

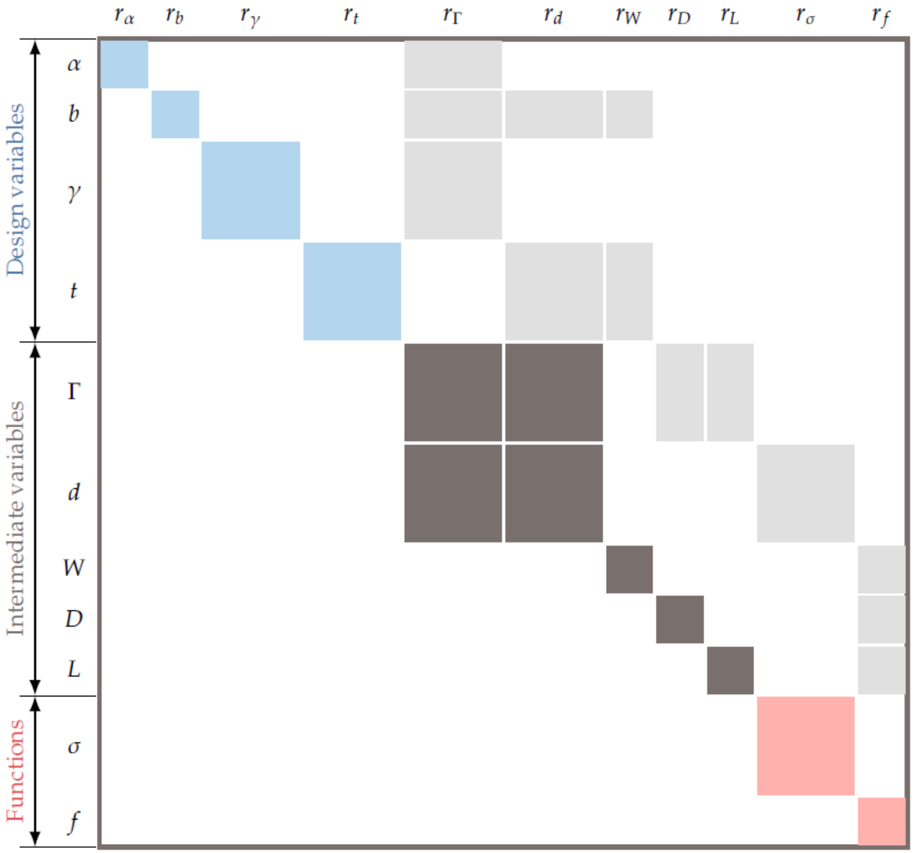


Figure 7.43 The DSM of the aerostructural problem shows the structure of the reverse UDE.

Source: Martins et Ning, 2021

5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives (continued).

- $\partial r/\partial x$ has two components, which we can obtain by differentiating the residuals:

$$\frac{\partial}{\partial b}(A\Gamma - v) = \frac{\partial A}{\partial b}\Gamma - \frac{\partial v}{\partial b}, \qquad \frac{\partial}{\partial b}(Kd - q) = \frac{\partial K}{\partial b}\Gamma.$$

- $\partial f/\partial x = \partial f/\partial b = 0$ because the fuel burn does not depend directly on the span if we just consider Eq. 7.25. However, it does depend on the span through, *W*, *D*, and *L*. This is where the UDE description is more general and clearer than the standard direct and adjoint formulation. By defining the explicit components of the function in the bottom-right corner, the solution of the linear system yields the chain rule

$$\frac{df}{db} = \frac{\partial f}{\partial D}\frac{dD}{db} + \frac{\partial f}{\partial L}\frac{dL}{db} + \frac{\partial f}{\partial W}\frac{dW}{db},$$

5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives (continued).

where the partial derivatives can be obtained by differentiating Eq. 7.25 symbolically, and the total derivatives are part of the coupled linear system solution.

- After computing all the partial derivative terms, we solve either the forward or reverse UDE system.
- For the derivative with respect to span, neither method has an advantage.
- However, for the derivatives of fuel burn with respect to the twist and thickness variables, the reverse mode is much more efficient.
- In this example, df/db=-11.0kg/m, so each additional meter of span reduced the fuel burn by 11 kg.

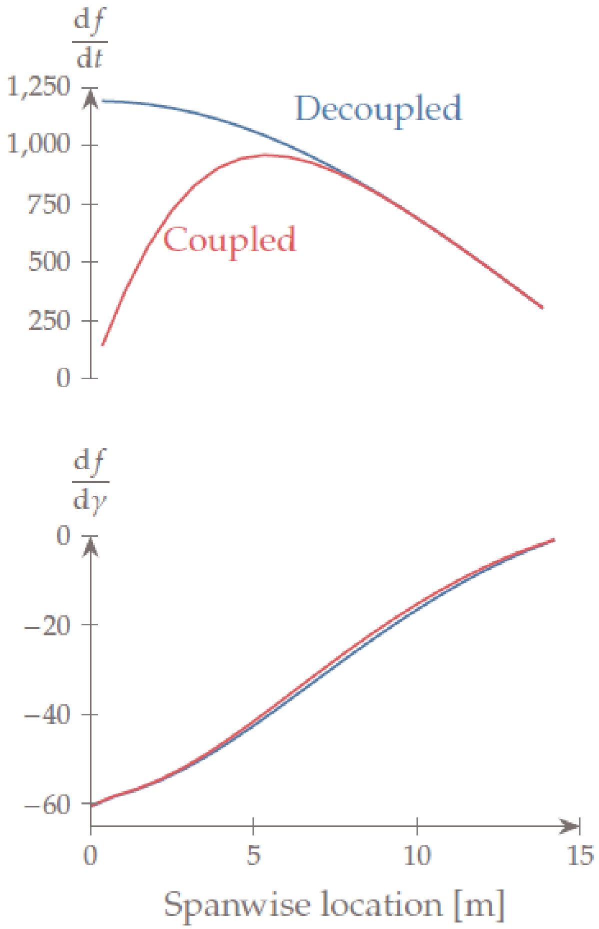
5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives (continued).

- If we compute this same derivative without coupling (by converging the aerostructural model but not considering the off-diagonal terms in the aerostructural Jacobian), we obtain df/db=-17.7kg/m, which is significantly different.
- The derivatives of the fuel burn with respect to the twist distribution and the thickness distribution along the wingspan are plotted in Fig. 7.44, where we can see the difference between coupled and uncoupled derivatives.

5.3.3. UDE for coupled systems

Example 7.8: Aerostructural derivatives (continued).



Source: Martins et Ning, 2021

Figure 7.44 Derivatives of the fuel burn with respect to the spanwise distribution of twist and thickness variables. The coupled derivatives differ from the uncoupled derivatives, especially for the derivatives with respect to structural thicknesses near the wing root.

6. MDO Architectures

- So far in this chapter, we have seen the models, solvers and derivative computation methods applied to coupled systems.
- We now discuss the options to optimize coupled systems, which are given by various MDO architectures.
- An MDO architecture is a combination of the problem formulation and the organizational strategy.
- The MDO architecture defines both how the different models are coupled and how the overall optimization problem is solved.
- There are several terms in the literature used to describe what we mean by “architecture“:
  - “method”, “methodology”, “problem formulation”, “strategy”, “procedure”, and “algorithm”

6. MDO Architectures

- Our preference is for the term “architecture", because the relationship between problem formulation and solution algorithm is not one-to-one.
- For example, replacing a particular disciplinary simulation with a surrogate model or reordering the disciplinary simulations do not affect the problem formulation but strongly affect the solution algorithm.
- Choosing the most appropriate architecture for the problem can significantly reduce the solution time.
- These time savings come from:
  - the methods chosen to solve each discipline
  - the optimization algorithm driving the process
  - the coupling scheme used in the architecture
  - the degree to which operations are carried out in parallel

6. MDO Architectures

- The latter consideration becomes especially important as the design becomes more detailed and the number of variables and/or constraints increases.

6.1. Unified description of MDO architectures

- To represent the various MDO architectures is convenient to use a common strategy to facilitate implementation, comparison and benchmarking the various architectures.

6.1.1 Terminology and mathematical notation

- Design variable:
  - a variable in the MDO problem that is always under the explicit control of an optimizer
  - Local design variable:
    - design variables relevant to a single discipline only —› denoted by x_i for discipline i
  - Shared design variable:
    - design variables used by several disciplines —› denoted by x_0

- Full set of design variables:

$$x = [x_0^T, x_1^T, \dots, x_m^T] \quad . \qquad (7.26)$$

6.1.1 Terminology and mathematical notation

- Think of some examples of local and shared design variables in the context of aerostructural optimization.

- Discipline analysis:
  - a simulation that models the behaviour of one aspect of a multidisciplinary system —› represented by equations $r_i = 0$
- State variables:
  - set of variables determined by solving a discipline analysis —› denoted by u_i

- Think of some examples of aerodynamic discipline analyses. What are the corresponding state variables?

6.1.1 Terminology and mathematical notation

- Coupling variables:
  - variables determined by one discipline and that influence another discipline
  - Response variables:
    - coupling variables from a specific discipline —› denoted by $\hat{u}_i$ for discipline *i*
  - Target variables:
    - values of the response variables that we need to match —› denoted by $\hat{u}_i^t$for discipline *i*
  - Consistency constraints:
    - constraints that ensure the response variables match the values of the target variables —› for discipline *i* we have $h_i^c = \hat{u}_i^t - \hat{u}_i$
- Choose a multidisciplinary problem. Identify the coupling variables.

6.1.2 Architecture diagrams

- An explained before, an Extended Design Structure Matrix (XDSM) is a convenient and compact way to describe the sequence of operations in an MDO architecture.
- The XDSM was developed to simultaneously communicate data dependency and process flow between computational components of the architecture on a single diagram.
- The XDSM is based on the Design Structure Matrix (DSM) and follows its basic rules:
  - architecture components are placed on main diagonal of the “matrix"
  - inputs to a component are placed in the same column
  - outputs to a component are placed in the same row
  - external inputs and outputs may also be defined and are placed on the outer edges of the diagram

6.1.2 Architecture diagrams

- The XDSM is based on the Design Structure Matrix (DSM) and follows its basic rules (continued):
  - thick grey lines are used to show the data flow between components
  - a numbering system is used to show the order in which the components are executed (the algorithm starts at component zero and proceeds in numerical order)
  - consecutive components in the algorithm are connected by a thin black line
  - loops are denoted using the notation $j \rightarrow k \; for \; k < j$ so that the algorithm must return to step k until a looping condition is satisfied before proceeding

6.1. Monolithic MDO architectures

- Monolithic MDO architectures cast the design problem as a single optimization.
- The only difference between the different monolithic architectures is the set of design variables that the optimizer is responsible for, which has repercussions for the set of constraints considered and how the governing equations are solved.

6.1.1 Multidisciplinary Feasible

- The multidisciplinary design feasible (MDF) architecture is the closest to a single-discipline problem because the design variables, objective, and constraints are the same as we would expect for a single-discipline problem.
- The only difference is that the computation of the objective and constraints requires solving a coupled system instead of a single system of governing equations.
- Therefore, all the optimization algorithms covered in the previous chapters can be applied without modification in MDF.
- This approach is also called a reduced-space approach because the optimizer does not handle the space of the state and coupling variables.

6.1.1 Multidisciplinary Feasible

- The resulting optimization problem is as follows:

$$\begin{array}{cc} minimize & f(x;\hat{u}*) \\ by\ varying & x \\ subject\ to & g(x;\hat{u}*) \leq 0 \\ while\ solving & \hat{r}(\hat{u};x) = 0 \\ for & \hat{u} \end{array} \qquad (7.27)$$

where * indicates that the variables after ; remain fixed in the given contest.

- At each optimization iteration, the optimizer has a multidisciplinary feasible point $\hat{u}*$ found through the MDA.
- For a design given by the optimizer (x), the MDA finds the internal component states (u) and the coupling variables ($\hat{u}$).

6.1.1 Multidisciplinary Feasible

- To denote the MDA solution, we use the residuals of the functional form, where the residuals for component *i* are

$$\hat{r}_i(\hat{u}, u_i) = \hat{u}_i - U_i\left(u_i, \hat{u}_{j\neq i}\right) = 0. \qquad (7.28)$$

- Each component is assumed to solve for its state variables u_i internally.
- The MDA finds the coupling variables by solving the coupled system of components $i = 1, \dots, m$ using one of the methods from Section 7.4.5.
- Then, the objective and constraints can be computed based on the current design variables and coupling variables.
- Figure 7.45 shows an XDSM for MDF with three components.

6. MDO Architectures

6.1. Monolithic MDO architectures

6.1.1 Multidisciplinary Feasible

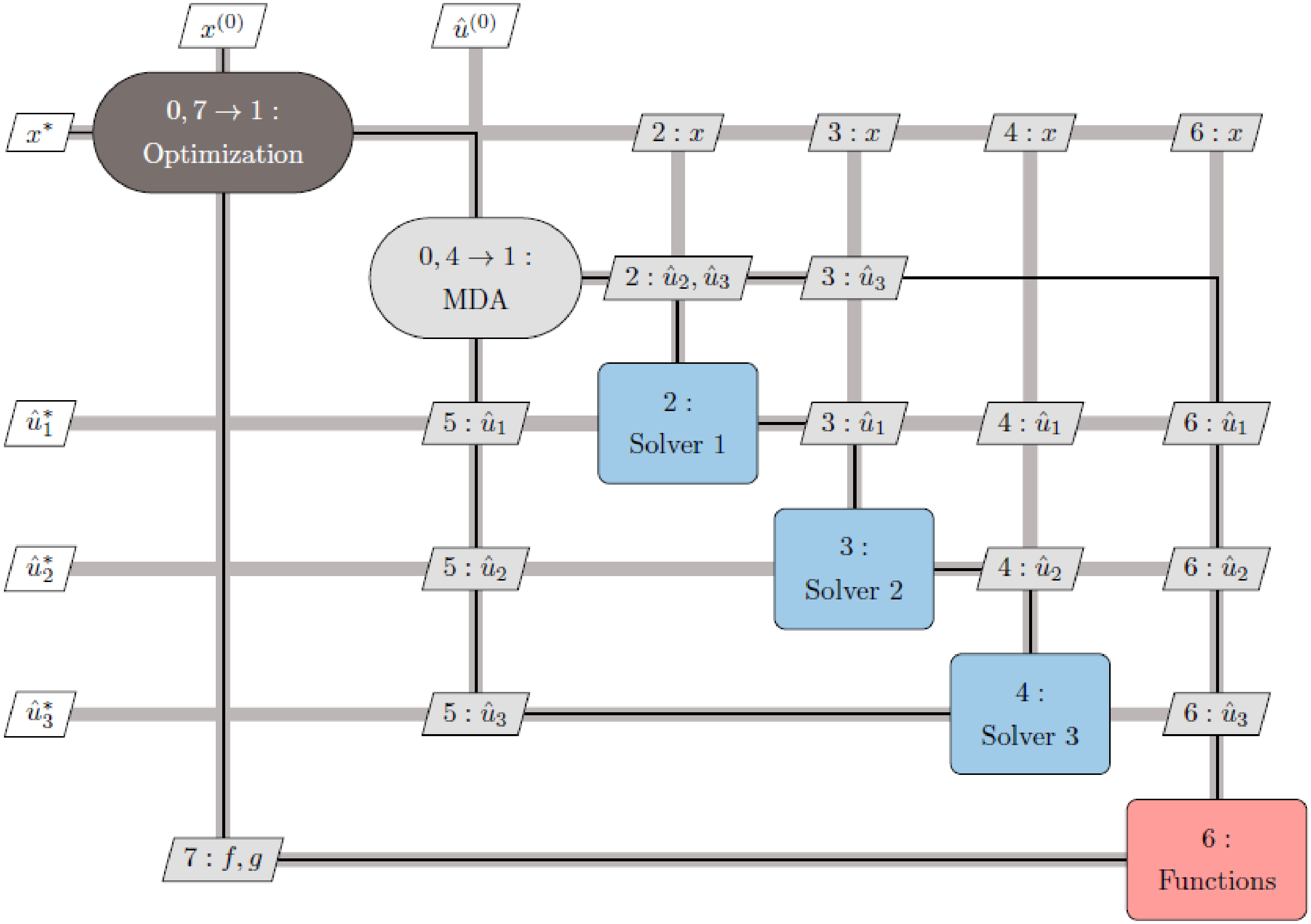


Figure 7.45 The MDF architecture relies on an MDA to solve for the coupling and state variables at each optimization iteration. In this case, the MDA uses the block Gauss–Seidel method.

Source: Martins et Ning, 2021

6.1.1 Multidisciplinary Feasible

- Here we use a nonlinear block Gauss–Seidel method (see Alg. 7.2) to converge the MDA, but any other method from Section 7.4.5 could be used.
- One advantage of MDF is that the system-level states are physically compatible if an optimization stops prematurely.
- This is advantageous in an engineering design context when time is limited, and we are not as concerned with finding an optimal design in the strict mathematical sense as we are with finding an improved design.
- However, it is not guaranteed that the design constraints are satisfied if the optimization is terminated early; that depends on whether the optimization algorithm maintains a feasible design point or not.

6.1.1 Multidisciplinary Feasible

- The main disadvantage of MDF is that it requires an MDA for each optimization iteration, which requires its own algorithm outside of the optimization.
- Implementing an MDA algorithm can be time-consuming if one is not already in place.
- A MAUD-based framework such as OpenMDAO can facilitate this.
- MAUD naturally implements the MDF architecture because it focuses on solving the MDA (Section 7.4.5) and on computing the derivatives corresponding to the MDA (Section 7.5.3).
- When using a gradient-based optimizer, gradient computations are also challenging for MDF because coupled derivatives are required.

6.1.1 Multidisciplinary Feasible

- Finite-difference derivative approximations are easy to implement, but their poor scalability and accuracy are compounded by the MDA, as explained in Section 7.3.
- Ideally, we would use one of the analytic coupled derivative computation methods of Section 7.3, which require a substantial implementation effort.
- Again, OpenMDAO was developed to facilitate coupled derivative computation.

6.1.1 Multidisciplinary Feasible

Example 7.9: Aerostructural optimization using MDF.

- Continuing the wing aerostructural example, we are finally ready to optimize the wing. The MDF formulation is as follows:

$$
\begin{array}{cc}
\textit{minimize} & f \\
\textit{by varying} & \alpha, b, \gamma, t \\
\textit{subject to} & L - W = 0 \\
 & 2.5|\sigma| - \sigma_{yield} \leq 0 \\
\textit{while solving} & A(d)\Gamma - v(d, \alpha) = 0 \\
 & Kd - q(\Gamma) = 0 \\
\textit{for} & \Gamma, d
\end{array}
$$

- The structural stresses are constrained to be less than the yield stress of the material by a safety factor (2.5 in this case).

6.1.1 Multidisciplinary Feasible

Example 7.9: Aerostructural optimization using MDF (continued).

- In Ex. 7.7, we set up the MDA for the aerostructural problem, and in Ex. 7.8, we set up the coupled derivative computations needed to solve this problem using gradient-based optimization.
- Solving this optimization resulted in the wing shown in Fig. 7.46, with the twist and thickness distributions shown in Fig. 7.47.
- The wing twist directly controls the spanwise lift loading.
- The baseline wing had no twist, which resulted in the loading shown in Fig. 7.48.

6.1.1 Multidisciplinary Feasible

Example 7.9: Aerostructural optimization using MDF (continued).

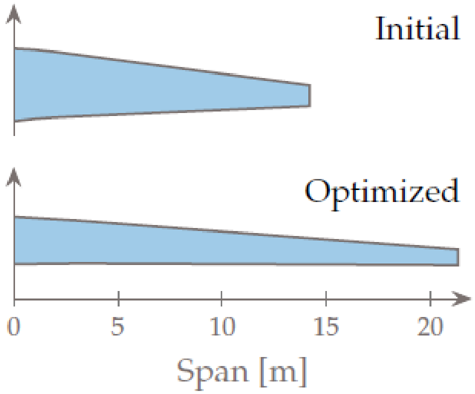


Figure 7.46 The optimization reduces the fuel burn by increasing the span. Although this increases the structural weight, the decrease in drag more than compensates for it.

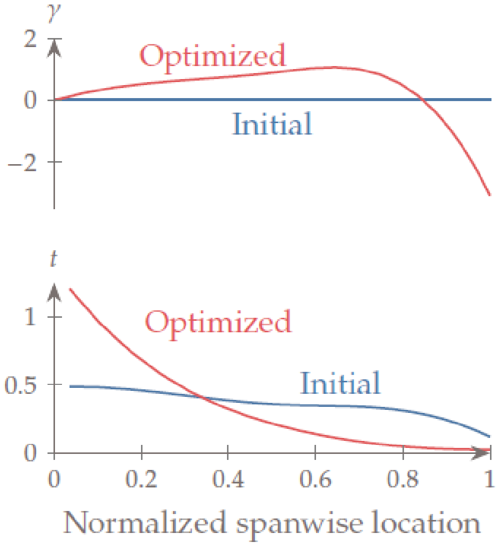


Figure 7.47 Twist and thickness distributions for the baseline and optimized wings.

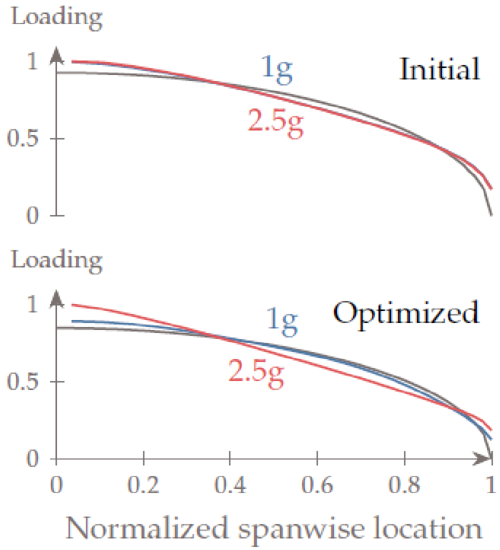


Figure 7.48 Lift loading for the baseline and optimized wings.

Source: Martins et Ning, 2021

6.1.1 Multidisciplinary Feasible

Example 7.9: Aerostructural optimization using MDF (continued).

- The loading distributions for the level flight (1 g) and maneuver conditions (2.5 g) are indistinguishable.
- The optimization increases the twist in the midspan and drastically decreases it toward the tip.
- This twist distribution differentiates the loading at the two different conditions: it makes the loading at level flight closer to the elliptical ideal while shifting the loading at the maneuver condition toward the wing root.
- The thickness distribution also changes significantly, as shown in Fig. 7.47.

6.1.1 Multidisciplinary Feasible

Example 7.9: Aerostructural optimization using MDF (continued).

- The optimization tailors the thickness by adding more thickness in the spar near the root, where the moments are larger, and thins out the wing much more toward the tip, where the loads decrease.
- This more radical thickness distribution is enabled by the tailoring of the spanwise lift loading discussed previously.
- These trades make sense because, at the level flight condition, the optimizer is concerned with minimizing drag, whereas, at the maneuver condition, the optimizer just wants to satisfy the stress constraint for a given total lift.

6.1.1 Multidisciplinary Feasible

Example 7.10: Aerostructural sequential optimization.

- In Section 7.2, we argued that sequential optimization does not, in general, converge to the true optimum for constrained problems.
- We now demonstrate this for a modified version of the wing aerostructural design optimization problem from Ex. 7.9.
- One major modification was to reduce the problem to two design variables to visualize the optimization path: one structural variable corresponding to a constant spar thickness and one twist variable corresponding to the wing tip twist, which controls the slope of a linear twist distribution.

6.1.1 Multidisciplinary Feasible

Example 7.10: Aerostructural sequential optimization (continued).

- To perform sequential optimization for the wing design problem of Ex. 7.3, we could start by optimizing the aerodynamics by solving the following problem:

$$\begin{array}{ll} \textit{minimize} & f \\ \textit{by varying} & \alpha, \gamma \\ \textit{subject to} & L - W = 0 \end{array}$$

- Here, *W*, is constant because the structural thicknesses *t* are fixed, but *L* is a function of the aerodynamic design variables and states.
- We cannot include the span *b* because it is a shared variable, as explained in Section 7.2.

6.1.1 Multidisciplinary Feasible

Example 7.10: Aerostructural sequential optimization (continued).

- Otherwise, this optimization would tend to increase b indefinitely to reduce the lift-induced drag.
- Because f is a function of D and L, and L is constant because $L=W$, we could replace the objective with D.
- Once the aerodynamic optimization has converged, the twist distribution and the forces are fixed, and we then optimize the structure by minimizing weight subject to stress constraints by solving the following problem:

$$\begin{array}{cc} \textit{minimize} & f \\ \textit{by varying} & t \\ \textit{subject to} & 2.5|\sigma| - \sigma_{yield} \leq 0 \end{array}$$

6.1.1 Multidisciplinary Feasible

Example 7.10: Aerostructural sequential optimization (continued).

- Because the drag and lift are constant, the objective could be replaced by *W*.
- Again, we cannot include the span in this problem because it would decrease indefinitely to reduce the weight and internal loads due to bending.
- These two optimizations are repeated until convergence.
- We compare the paths of this optimization and an optimization of the same two variables but performing MDO using the MDF architecture from Ex. 7.9.

6.1.1 Multidisciplinary Feasible

Example 7.10: Aerostructural sequential optimization (continued).

- As shown in Fig. 7.49, sequential optimization only changes one variable at a time, and it converges to a point on the constraint with about 3.5⁰ more twist than the true optimum of the MDO.
- When including more variables, these differences are likely to be even larger.

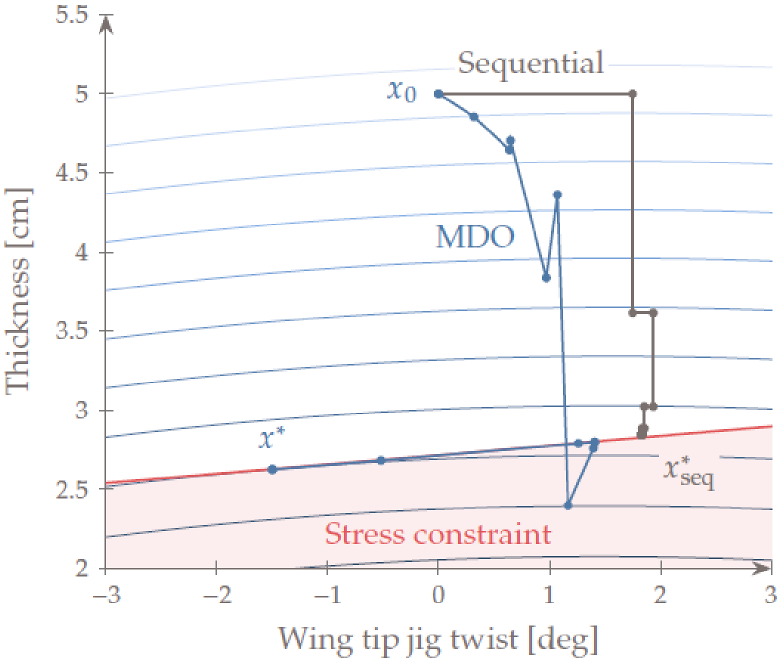


Figure 7.49 Sequential optimization gets stuck at the stress constraint, whereas simultaneous optimization of the aerodynamic and structural variable finds the true multidisciplinary optimum.

Source: Martins et Ning, 2021

6.1.2 Individual Discipline Feasible

- The individual discipline feasible (IDF) architecture adds independent copies of the coupling variables to allow component solvers to run independently and possibly in parallel.
- These copies are known as target variables and are controlled by the optimizer, whereas the actual coupling variables are computed by the corresponding component.
- Target variables are denoted by a superscript *t*, so the coupling variables produced by discipline *i* are denoted as $\hat{u}_i^t$.
- These variables represent the current guesses for the coupling variables that are independent of the corresponding actual coupling variables computed by each component.

6.1.2 Individual Discipline Feasible

- To ensure the eventual consistency between the target coupling variables and the actual coupling variables at the optimum, we define a set of consistency constraints, $h_i^c = \hat{u}_i^t - \hat{u}_i$, which we add to the optimization problem formulation.
- The optimization problem for the IDF architecture is

$$\begin{array}{ccc} \textit{minimize} & f(x;\hat{u}) & \\ \textit{by varying} & x, \hat{u}^t & \\ \textit{subject to} & g(x;\hat{u}) \leq 0 & i = 1,\ldots,m \\ & h_i^c = \hat{u}_i^t - \hat{u}_i = 0 & i = 1,\ldots,m \\ \textit{while solving} & r_i\left(\hat{u}_i; x, \hat{u}_{j\neq i}^t\right) = 0 & \\ \textit{for} & \hat{u} & \end{array} \qquad (7.29)$$

- Each component *i* is solved independently to compute the corresponding output coupling variables $\hat{u}_i$, where the inputs $\hat{u}_{j\neq i}^t$ are given by the optimizer.

6.1.2 Individual Discipline Feasible

- Thus, each component drives its residuals to zero to compute

$$\hat{u}_i = U_i\left(x, \hat{u}^t_{j \neq i}\right). \tag{7.30}$$

- The consistency constraint quantifies the difference between the target coupling variables guessed by the optimizer and the actual coupling variables computed by the components.
- The optimizer iterates the target coupling variables simultaneously with the design variables to find a multidisciplinary feasible point that is also an optimum.
- At each iteration, the objective and constraints are computed using the latest available coupling variables computed by each component.
- Figure 7.50 shows the XDSM for IDF.

6. MDO Architectures

6.1. Monolithic MDO architectures

6.1.2 Individual Discipline Feasible

MDO in Aeronautical Engineering / MDO em Engenharia Aeronáutica

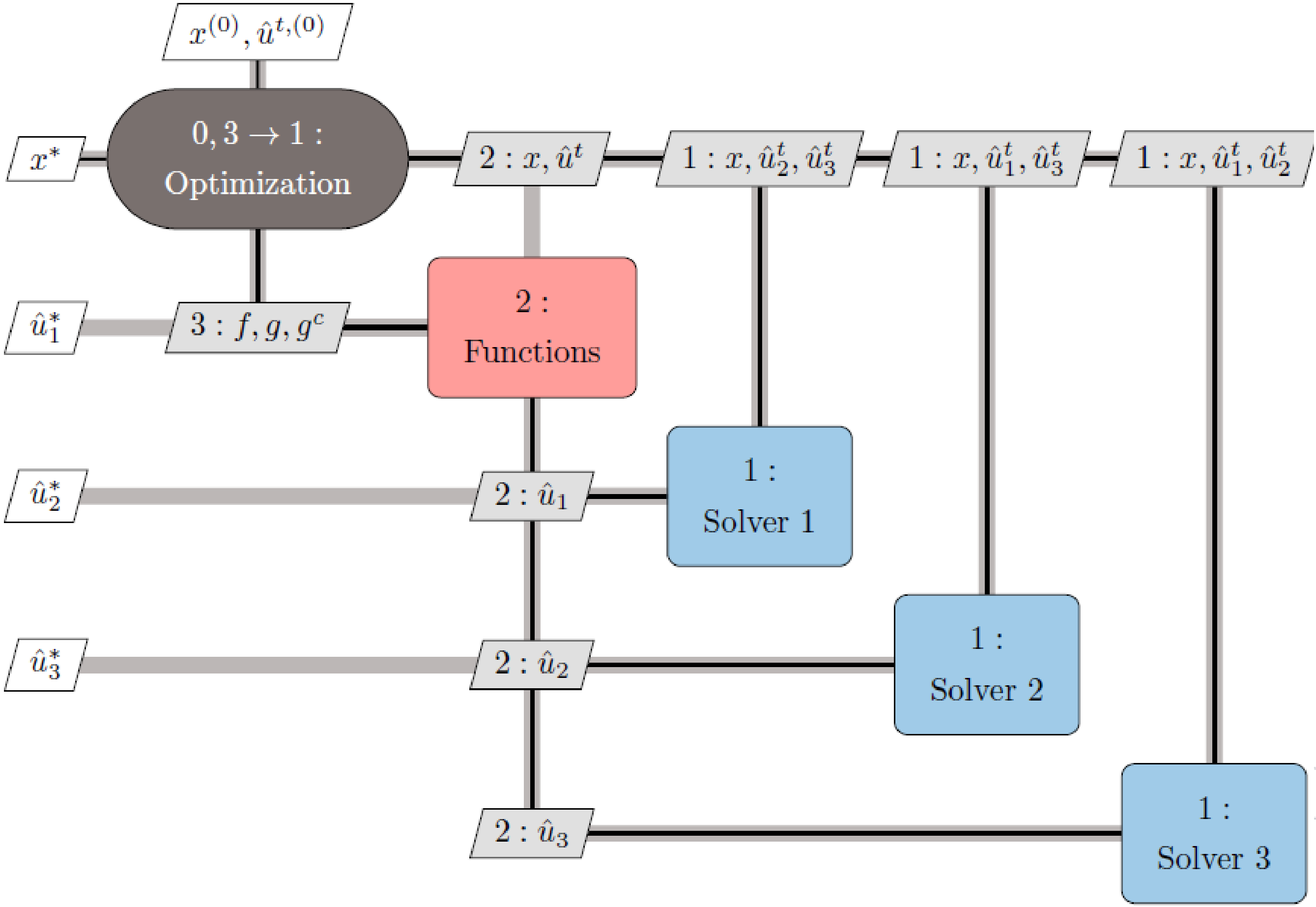


Figure 7.50 The IDF architecture breaks up the MDA by letting the optimizer solve for the coupling variables that satisfy interdisciplinary feasibility.

Source: Martins et Ning, 2021

6.1.2 Individual Discipline Feasible

- One advantage of IDF is that each component can be solved in parallel because they do not depend on each other directly.
- Another advantage of IDF is that if a gradient-based optimization algorithm is used to solve the optimization problem, the optimizer is typically more robust and has a better convergence rate than the fixed-point iteration algorithms of Section 7.4.5.
- The main disadvantage of IDF is that the optimizer must handle more variables and constraints compared with the MDF architecture.
- If the number of coupling variables is large, the size of the resulting optimization problem may be too large to solve efficiently.

6.1.2 Individual Discipline Feasible

- This problem can be mitigated by careful selection of the components or by aggregating the coupling variables to reduce their dimensionality.
- Unlike MDF, IDF does not guarantee a multidisciplinary feasible state if the optimization stops prematurely.
- Multidisciplinary feasibility is only guaranteed at the end of the optimization through the satisfaction of the consistency constraints.

6.1.2 Individual Discipline Feasible

Example 7.11: Aerostructural optimization using IDF.

- For the IDF architecture, we need to make copies of the coupling variables (Γ^t and d^t) and add the corresponding consistency constraints, as highlighted in the following problem statement

$$\begin{array}{cc}
\textit{minimize} & f \\
\textit{by varying} & \alpha, b, \gamma, t, \Gamma^t, d^t \\
\textit{subject to} & L - W = 0 \\
 & 2.5|\sigma| - \sigma_{yield} \leq 0 \\
 & \Gamma^t - \Gamma = 0 \\
 & d^t - d = 0 \\
\textit{while solving} & A(d^t)\Gamma - v(d^t, \alpha) = 0 \\
 & Kd - q(\Gamma^t) = 0 \\
\textit{for} & \Gamma, d
\end{array}$$

6.1.2 Individual Discipline Feasible

Example 7.11: Aerostructural optimization using IDF (continued).

- The aerodynamic and structural models are solved independently.
- The aerodynamic solver finds Γ for the d^t given by the optimizer, and the structural solver finds d for the given Γ^t.
- When using gradient-based optimization, we do not require coupled derivatives, but we do need the derivatives of each model with respect to both state variables.
- The derivatives of the consistency constraints are just a unit matrix when taken with respect to the variable copies and zero otherwise.

6.1.3 Simultaneous Analysis and Design

- Simultaneous analysis and design (SAND) extends the idea of IDF by moving not only the coupling variables to the optimization problem but also all component states.
- The SAND architecture requires exposing all the components in the form of the system-level view previously introduced in Fig. 7.25.
- The residuals of the analysis become constraints that the optimizer is responsible for.
- This means that component solvers are no longer needed, and the optimizer becomes responsible for simultaneously solving the components for their states, the interdisciplinary compatibility for the coupling variables, and the design optimization problem for the design variables.

6.1.3 Simultaneous Analysis and Design

- All that is required from the model is the computation of residuals.
- Because the optimizer is controlling all these variables, SAND is also known as a full-space approach.
- SAND can be stated as follows:

$$\begin{array}{cc} \textit{minimize} & f(x,\hat{u},u) \\ \textit{by varying} & x,\hat{u},u \\ \textit{subject to} & g(x,\hat{u}) \leq 0 \\ & r(x,\hat{u},u) = 0 \end{array} \qquad (7.31)$$

- Here, we use the representation shown in Fig. 7.23, so there are two sets of explicit functions that convert the input coupling variables of the component.
- The SAND architecture is also applicable to single components, in which case there are no coupling variables.

6.1.3 Simultaneous Analysis and Design

- The XDSM for SAND is shown in Fig. 7.51.
- Because it solves for all variables simultaneously, the SAND architecture can be the most efficient way to get to the optimal solution.
- In practice, however, it is unlikely that this is advantageous when efficient component solvers are available.
- The resulting optimization problem is the largest of all MDO architectures and requires an optimizer that scales well with the number of variables.
- Therefore, a gradient-based optimization algorithm is likely required, in which case the derivative computation must also be considered.

6.1.3 Simultaneous Analysis and Design

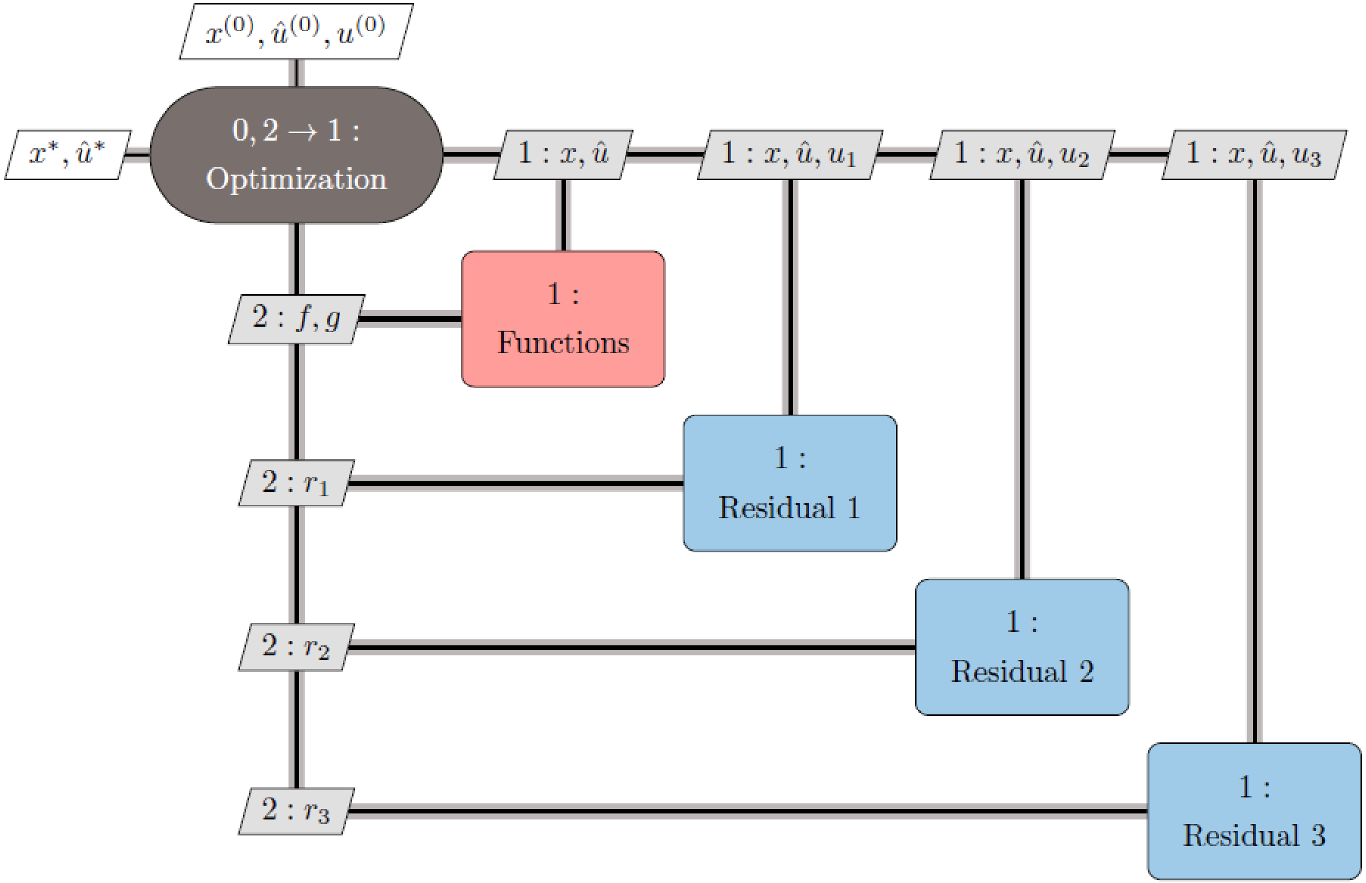


Figure 7.51 The SAND architecture lets the optimizer solve for all variables (design, coupling, and state variables), and component solvers are no longer needed.

Source: Martins et Ning, 2021

6.1.3 Simultaneous Analysis and Design

- Fortunately, SAND does not require derivatives of the coupled system or even total derivatives that account for the component solution; only partial derivatives of residuals are needed.
- SAND is an intrusive approach because it requires access to residuals.
- These might not be available if components are provided as black boxes.
- Rather than computing coupling variables $\hat{u}_i$ and state variables u_i by converging the residuals to zero, each component i just computes the current residuals r_i for the current values of the coupling variables $\hat{u}$ and the component states u_i.

6.1.3 Simultaneous Analysis and Design

Example 7.12: Aerostructural optimization using SAND.

- For the SAND approach, we do away completely with the solvers and let the optimizer find the states. The resulting problem is as follows:

$$\begin{array}{cc} \textit{minimize} & f \\ \textit{by varying} & \alpha, b, \gamma, t, \Gamma, d \\ \textit{subject to} & L - W = 0 \\ & 2.5|\sigma| - \sigma_{yield} \leq 0 \\ & A\Gamma - v = 0 \\ & Kd - q = 0 \end{array}$$

- Instead of being solved separately, the models are now solved by the optimizer.

6.1.3 Simultaneous Analysis and Design

Example 7.12: Aerostructural optimization using SAND (continued).

- When using gradient-based optimization, the required derivatives are just partial derivatives of the residuals (the same partial derivatives we would use for an implicit analytic method).

6.2. Distributed MDO architectures

- The monolithic MDO architectures we have covered so far form and solve a single optimization problem.
- Distributed architectures decompose this single optimization problem into a set of smaller optimization problems, or disciplinary subproblems, which are then coordinated by a system-level subproblem.
- One key requirement for these architectures is that they must be mathematically equivalent to the original monolithic problem to converge to the same solution.
- There are two primary motivations for distributed architectures.
- The first one is the possibility of decomposing the problem to reduce the computational time.

6.2. Distributed MDO architectures

- The second motivation is to mimic the structure of large engineering design teams, where disciplinary groups have the autonomy to design their subsystems so that MDO is more readily adopted in industry.
- Overall, distributed MDO architectures have fallen short on both of these expectations.
- Unless a problem has a special structure, there is no distributed architecture that converges as rapidly as a monolithic one.
- In practice, distributed architectures have not been used much recently.
- There are two main types of distributed architectures:
  - those that enforce multidisciplinary feasibility via an MDA somewhere in the process
  - those that enforce multidisciplinary feasibility in some other way (using constraints or penalties at the system level)

6.2. Distributed MDO architectures

- This is analogous to MDF and IDF, respectively, so we name these types distributed MDF and distributed IDF.
- In MDO problems, it can be helpful to distinguish between design variables that affect only one component directly (called local design variables) and design variables that affect two or more components directly (called shared design variables).
- We denote the vector of design variables local to component *i* by x_i and the shared variables by x_0.
- The full vector of design variables is given by concatenating the shared and local design variables into a single vector $x = [x_0^T, x_1^T, \dots, x_m^T]$, where m is the number of components.
- If a constraint can be computed using a single component and satisfied by varying only the local design variables for that component, it is a local constraint; otherwise, it is nonlocal.

6.2. Distributed MDO architectures

- Similarly, for the design variables, we concatenate the constraints as $g = [g_0^T, g_1^T, \dots, g_m^T]$.
- The same distinction could be applied to the objective function, but we do not usually do this.
- The MDO problem representation we use here is shown in Fig. 7.52 for a general three-component system.
- We use the functional form introduced in Section 7.4.3, where the states in each component are hidden, and we just see its output as a coupling variable at the system level.
- The set of constraints is also split into shared constraints and local ones.
- Local constraints are computed by the corresponding component and depend only on the variables available in that component.

6.2. Distributed MDO architectures

- Shared constraints depend on more than one set of coupling variables.
- These dependencies are also shown in Fig. 7.52.

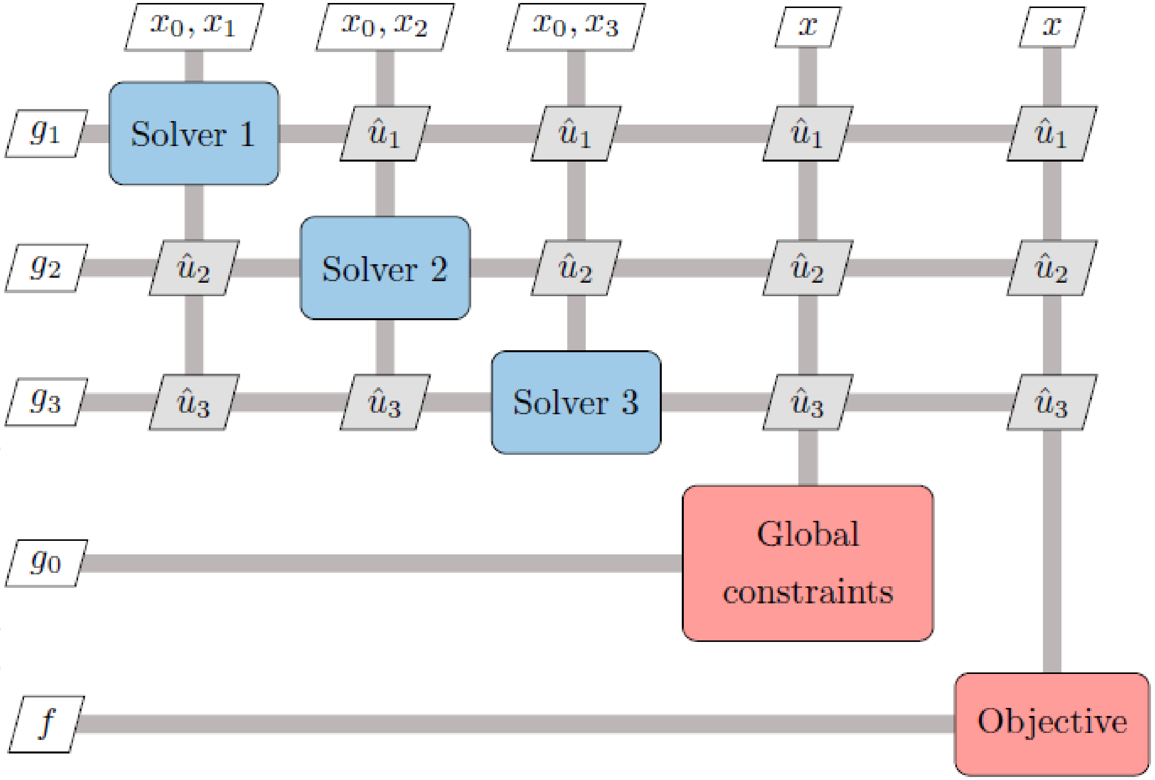


Figure 7.52 MDO problem nomenclature and dependencies.

Source: Martins et Ning, 2021

6.2.1 Collaborative Optimization

- The collaborative optimization (CO) architecture is inspired by how disciplinary teams work to design complex engineered systems.
- This is a distributed IDF architecture, where the disciplinary optimization problems are formulated to be independent of each other by using target values of the coupling and shared design variables.
- These target values are then shared with all disciplines during every iteration of the solution procedure.
- The complete independence of disciplinary subproblems combined with the simplicity of the data-sharing protocol makes this architecture attractive for problems with a small amount of shared data.

6.2.1 Collaborative Optimization

- The system-level subproblem modifies the original problem as follows:
  1. local constraints are removed
  2. target coupling variables, $\hat{u}^t$, are added as design variables
  3. a consistency constraint is added
- This optimization problem can be written as follows:

$$\begin{array}{ll} \textit{minimize} & f(x_0, x_1^t, \dots, x_m^t, \hat{u}^t) \\ \textit{by varying} & x_0, x_1^t, \dots, x_m^t, \hat{u}^t \\ \textit{subject to} & g_0(x_0, x_1^t, \dots, x_m^t, \hat{u}^t) \le 0 \\ & J_i^* = \left\|x_{0i}^t - x_0\right\|_2^2 + \left\|x_i^t - x_i\right\|^2 \\ & +\left\|\hat{u}_i^t - \hat{u}_i\left(x_{0i}^t, x_i, \hat{u}_{j\neq i}^t\right)\right\|^2 \qquad i = 1, \dots, m \end{array} \qquad (7.32)$$

6.2.1 Collaborative Optimization

where x_{0i}^{t} are copies of the shared design variables that are passed to discipline i, and x_i^t are copies of the local design variables passed to the system subproblem.

- The constraint function J_i^* is a measure of the inconsistency between the values requested by the system-level subproblem and the results from the discipline i subproblem.
- The XDSM for CO is shown in Fig. 7.53.
- For each system-level iteration, the disciplinary subproblems do not include the original objective function.
- Instead, the objective of each subproblem is to minimize the inconsistency function.

6. MDO Architectures

6.2. Distributed MDO architectures

6.2.1 Collaborative Optimization

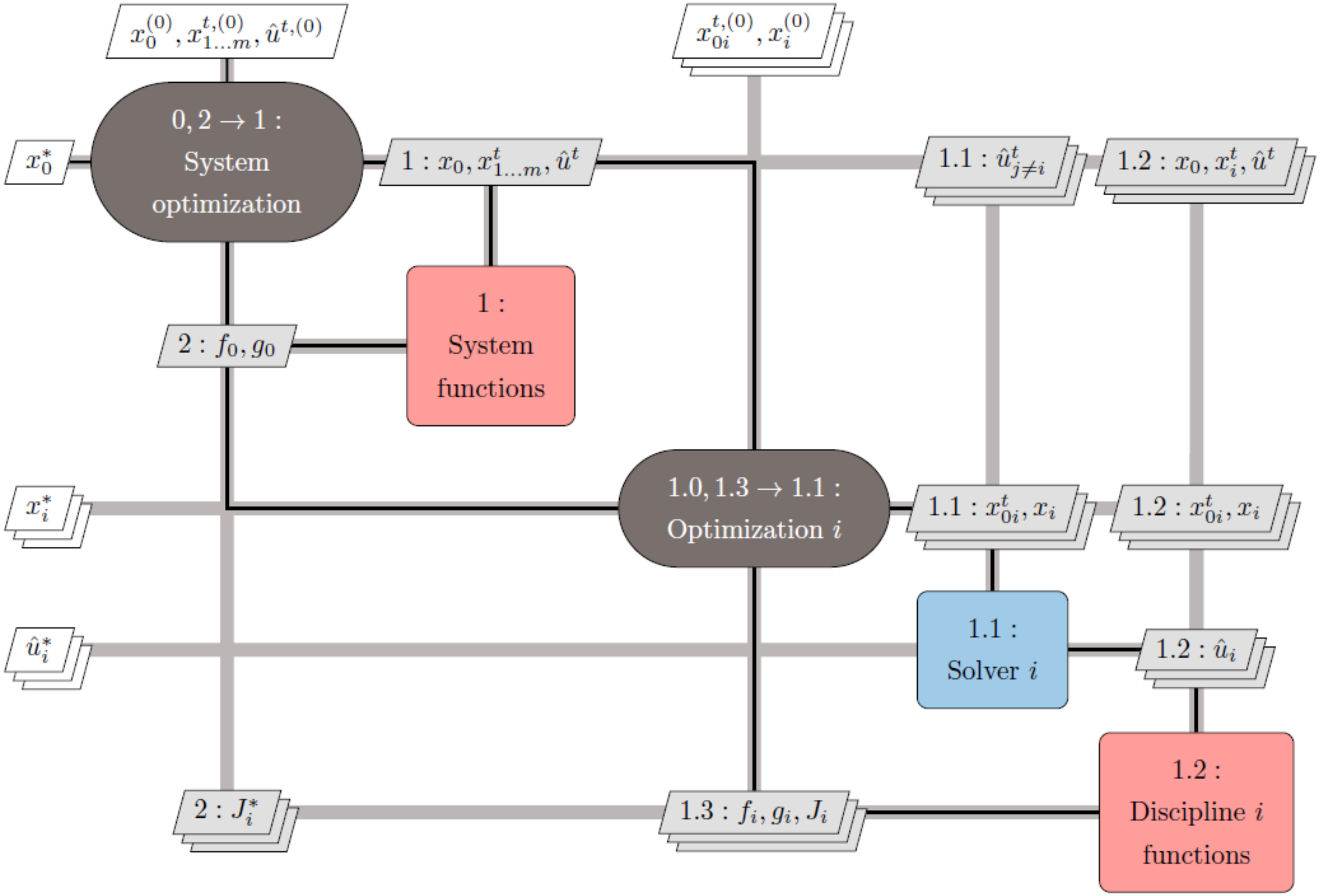


Figure 7.53 Diagram for the CO architecture.

Source: Martins et Ning, 2021

6.2.1 Collaborative Optimization

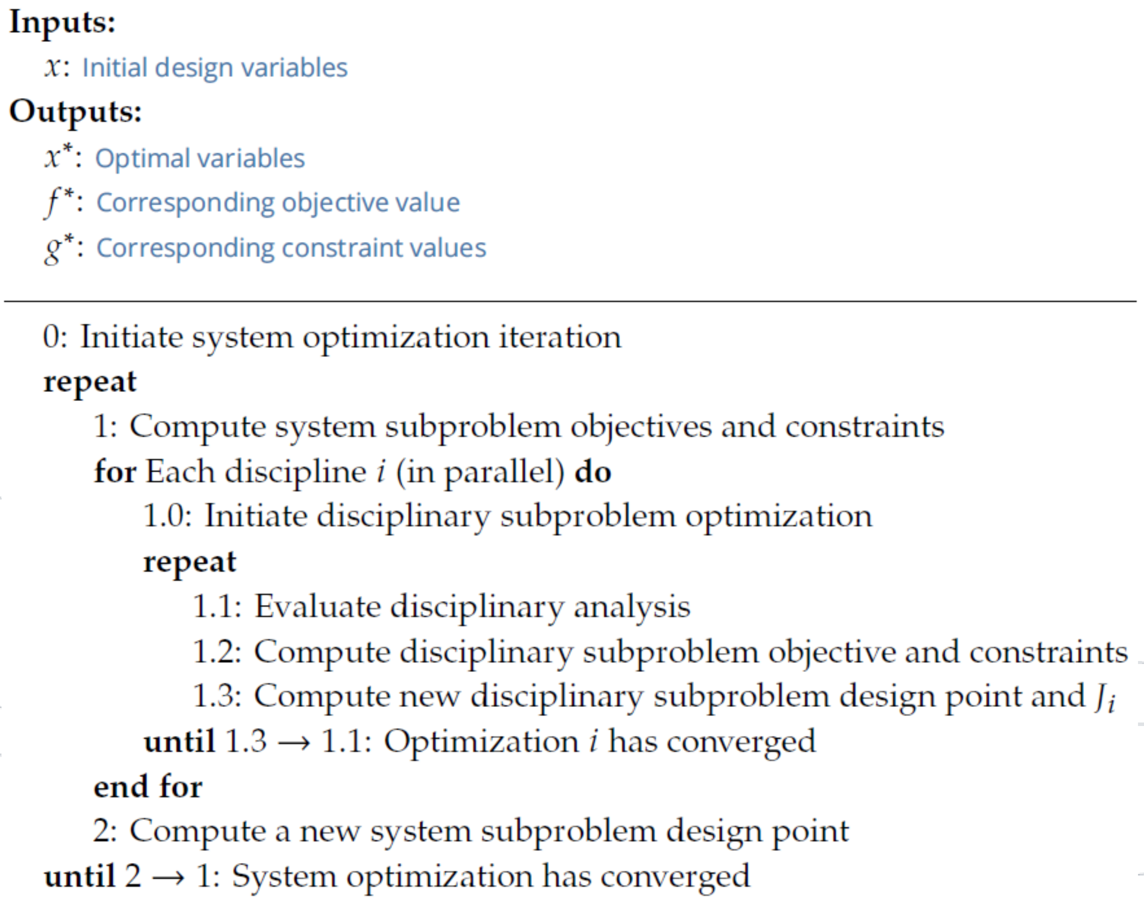
- For each discipline *i*, the subproblem is as follows:

$$\begin{array}{cc} minimize & J_i\left(x_{0i}^t, x_i; \hat{u}_i\right) \\ by\ varying & x_{0i}^t, x_i \\ subject\ to & g_i(x_{0i}^t, x_i; \hat{u}_i) \leq 0 \\ while\ solving & \hat{r}\left(\hat{u}_i; x_{0i}^t, x_i, \hat{u}_{j\neq i}^t\right) = 0 \\ for & \hat{u}_i \end{array} \qquad (7.33)$$

- These subproblems are independent of each other and can be solved in parallel.
- Thus, the system-level subproblem is responsible for minimizing the design objective, whereas the discipline subproblems minimize system inconsistency while satisfying local constraints.
- The CO procedure is detailed in Alg. 7.5.

6.2.1 Collaborative Optimization

Algorithm 7.5: Collaborative optimization



Inputs:

x: Initial design variables

Outputs:

x^*: Optimal variables

f^*: Corresponding objective value

g^*: Corresponding constraint values

```
0: Initiate system optimization iteration
repeat
    1: Compute system subproblem objectives and constraints
    for Each discipline $i$ (in parallel) do
        1.0: Initiate disciplinary subproblem optimization
        repeat
            1.1: Evaluate disciplinary analysis
            1.2: Compute disciplinary subproblem objective and constraints
            1.3: Compute new disciplinary subproblem design point and $J_i$
        until 1.3 → 1.1: Optimization $i$ has converged
    end for
    2: Compute a new system subproblem design point
until 2 → 1: System optimization has converged
```

6.2.1 Collaborative Optimization

- There are two versions of the CO architecture: CO_1 and CO_2.
- The version presented in Section 7.6.2.1 is CO_2.
- Despite the organizational advantage of having entirely separate disciplinary subproblems, CO suffers from numerical ill-conditioning.
- This is because the constraint gradients of the system problem at an optimal solution are all zero vectors, which violates the constraint qualification requirement for the Karush–Kuhn–Tucker (KKT) conditions.
- This slows down convergence when using a gradient-based optimization algorithm or prevents convergence altogether.

6.2.1 Collaborative Optimization

Example 7.13: Aerostructural optimization using CO.

- To apply CO to the wing aerostructural design optimization problem (Ex. 7.3), we need to set up a system-level optimization problem and two discipline-level optimization subproblems.
- The system-level optimization problem is formulated as follows:

$$\begin{array}{ll} minimize & f \\ by\ varying & b^t, \Gamma^t, d^t, W^t \\ subject\ to & J_1^* \leq \varepsilon \\ & J_2^* \leq \varepsilon \end{array}$$

where ε is a specified convergence tolerance.

- The set of variables that are copied as targets includes the shared design variable (b) and the coupling variables (Γ and d).

6.2.1 Collaborative Optimization

Example 7.13: Aerostructural optimization using CO (continued).

- The aerodynamics subproblem is as follows:

$$\begin{array}{cc}
\textit{minimize} & J_1 \equiv \left(1-\frac{b}{b^t}\right)^2 + \sum_{i=1}^{n_\Gamma}\left(1-\frac{\Gamma_i}{\Gamma_i^t}\right)^2 \\
\textit{by varying} & b, \alpha, \gamma \\
\textit{subject to} & L - W^t = 0 \\
\textit{while solving} & A\Gamma - \gamma = 0 \\
\textit{for} & \Gamma
\end{array}$$

- In this problem, the aerodynamic optimization minimizes the discrepancy between the span requested by the system-level optimization (b^t) and the span that aerodynamics is optimizing (b).

6.2.1 Collaborative Optimization

Example 7.13: Aerostructural optimization using CO (continued).

- The same applies to the coupling variables Γ.
- The aerodynamics is fully responsible for optimizing α and γ.
- The structures subproblem is as follows:

$$\begin{array}{cc}
minimize & J_2 \equiv \left(1-\frac{b}{b^t}\right)^2 + \sum_{i=1}^{n_d}\left(1-\frac{d_i}{d_i^t}\right)^2 + \left(1-\frac{W}{W^t}\right)^2 \\
by\ varying & b, t \\
subject\ to & 2.5|\sigma| - \sigma_{yield} \leq 0 \\
while\ solving & Kd - q = 0 \\
for & d
\end{array}$$

6.2.1 Collaborative Optimization

Example 7.13: Aerostructural optimization using CO (continued).

- Here, the structural optimization minimizes the discrepancy between the span wanted by the structures (a decrease) versus what the system level requests (which takes into account the opposite trend from aerodynamics).
- The structural subproblem is fully responsible for satisfying the stress constraints by changing the structural sizing *t*, which are local variables.

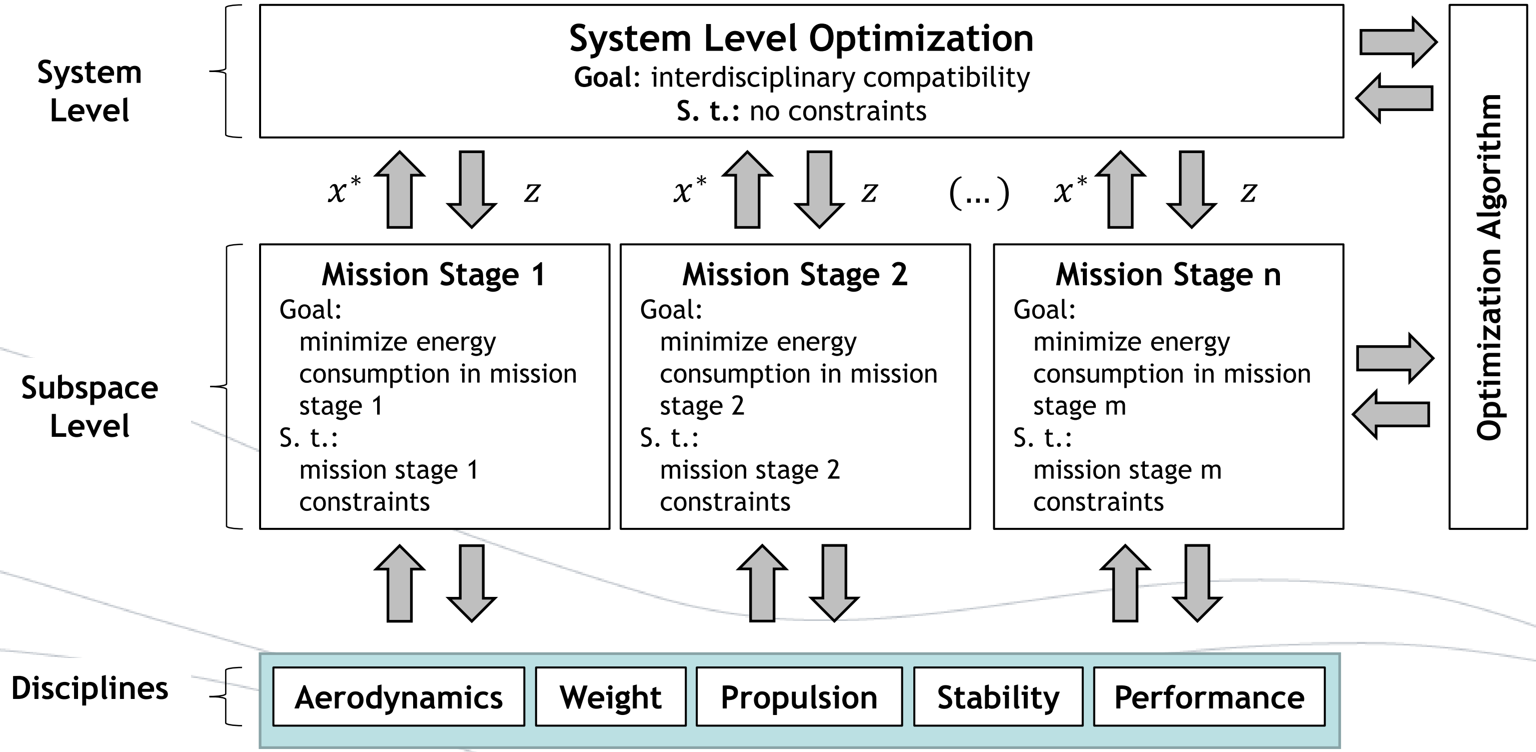
6.2.1 Collaborative Optimization

Example 7.14: Mission-Based optimization for a telescopic wing UAV design using ECO.

- See paper: Albuquerque, P., Gamboa, P., and Silvestre, M., "Mission-Based Multidisciplinary Aircraft Design Optimization Methodology Tailored for Adaptive Technologies", Journal of Aircraft, Vol. 55, No. 2, March 2018, pp. 755-770 (DOI: 10.2514/1.C034403).
- Design a UAV with a telescopic wing considering that the disciplines here are the flight phases (take-off, climb, cruise, loiter, descent).
- MDO architecture used is Enhanced Collaborative Optimization (ECO).
- Figure 7.54 shows the optimization concept.

6.2.1 Collaborative Optimization

Example 7.14: Mission-Based optimization for a telescopic wing UAV design using ECO (continued).


System Level
System Level Optimization
Goal: interdisciplinary compatibility
S. t.: no constraints
x^*
z
x^*
z
(…)
x^*
z
Optimization Algorithm
Subspace Level
Mission Stage 1
Goal:
minimize energy consumption in mission stage 1
S. t.:
mission stage 1 constraints
Mission Stage 2
Goal:
minimize energy consumption in mission stage 2
S. t.:
mission stage 2 constraints
Mission Stage n
Goal:
minimize energy consumption in mission stage m
S. t.:
mission stage m constraints
Disciplines
Aerodynamics
Weight
Propulsion
Stability
Performance

Adapted from: Albuquerque et al, 2018

Figure 7.54 Optimization concept for Ex. 7.14.

6.2.2 Analytic Target Cascading

- Analytical target cascading (ATC) is a distributed IDF architecture that uses penalties in the objective function to minimize the difference between the target variables requested by the system-level optimization and the actual variables computed by each discipline.
- The idea of ATC is similar to the CO architecture in the previous section, except that ATC uses penalties instead of a constraint
- The ATC system-level problem is as follows:

$$\begin{aligned} &\textit{minimize} \quad && f_0(x,\hat{u}^t) + \sum_{i=1}^{m} \Phi_i\left(x_{0i}^t - x_0, \hat{u}_i^t - \hat{u}_i(x_0, x_i, \hat{u}^t)\right) \\ & && +\Phi_0\left(g_0(x,\hat{u}^t)\right) \qquad (7.34) \\ &\textit{by varying} && x_0, \hat{u}^t \end{aligned}$$

6.2.2 Analytic Target Cascading

where Φ_0 is a penalty relaxation of the shared design constraints, and Φ_i is a penalty relaxation of the discipline *i* consistency constraints.

- The *i*th discipline subproblem is as follows:

$$\begin{array}{cc}
\textit{minimize} & f_0\left(x_{0i}^t, x_i; \hat{u}_i, \hat{u}_{j\neq i}^t\right) + f_i\left(x_{0i}^t, x_i; \hat{u}_i\right) \\
 & +\Phi_i\left(\hat{u}_i^t - \hat{u}_i, x_{0i}^t - x_0\ \right) \\
 & +\Phi_0\left(g_0\left(x_{0i}^t, x_i; \hat{u}_i, \hat{u}_{j\neq i}^t\right)\right) \\
\textit{by varying} & x_{0i}^t, x_i \\
\textit{subject to} & g_0\left(x_{0i}^t, x_i; \hat{u}_i\right) \leq 0 \\
\textit{while solving} & r_i\left(\hat{u}_i; x_{0i}^t, x_i, \hat{u}_{j\neq i}^t\right) = 0 \\
\textit{for} & \hat{u}_i
\end{array} \qquad (7.35)$$

6.2.2 Analytic Target Cascading

- Although the most common penalty functions in ATC are quadratic penalty functions, other penalty functions are possible.
- As mentioned in Chapter 5, penalty methods require a good selection of the penalty weight values to converge quickly and accurately enough.
- Figure 7.54 shows the ATC architecture XDSM, where ω denotes the penalty function weights used in the determination of Φ_0 and Φ_i.
- The details of ATC are described in Alg. 7.6.

6. MDO Architectures

6.2. Distributed MDO architectures

6.2.2 Analytic Target Cascading

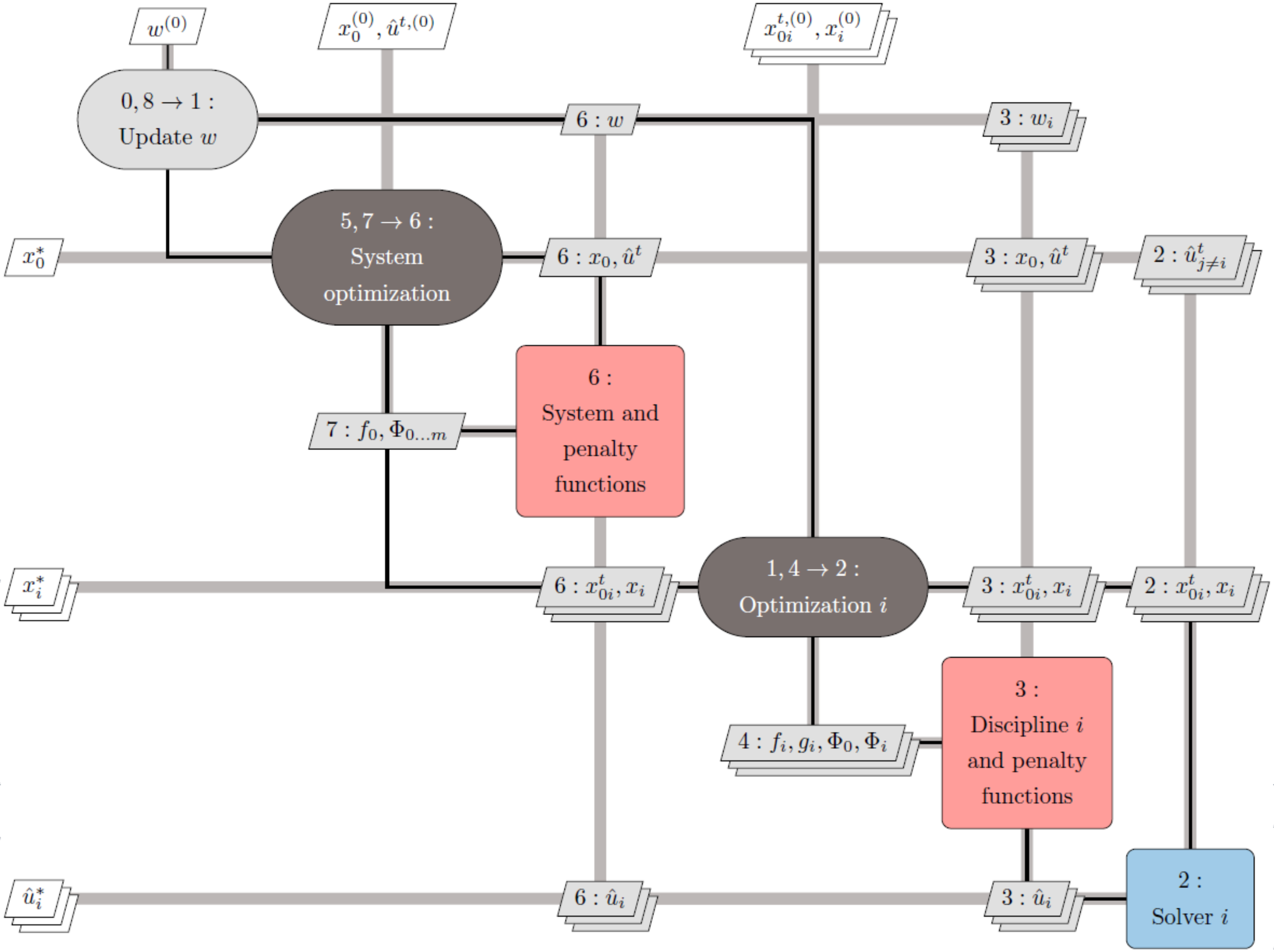
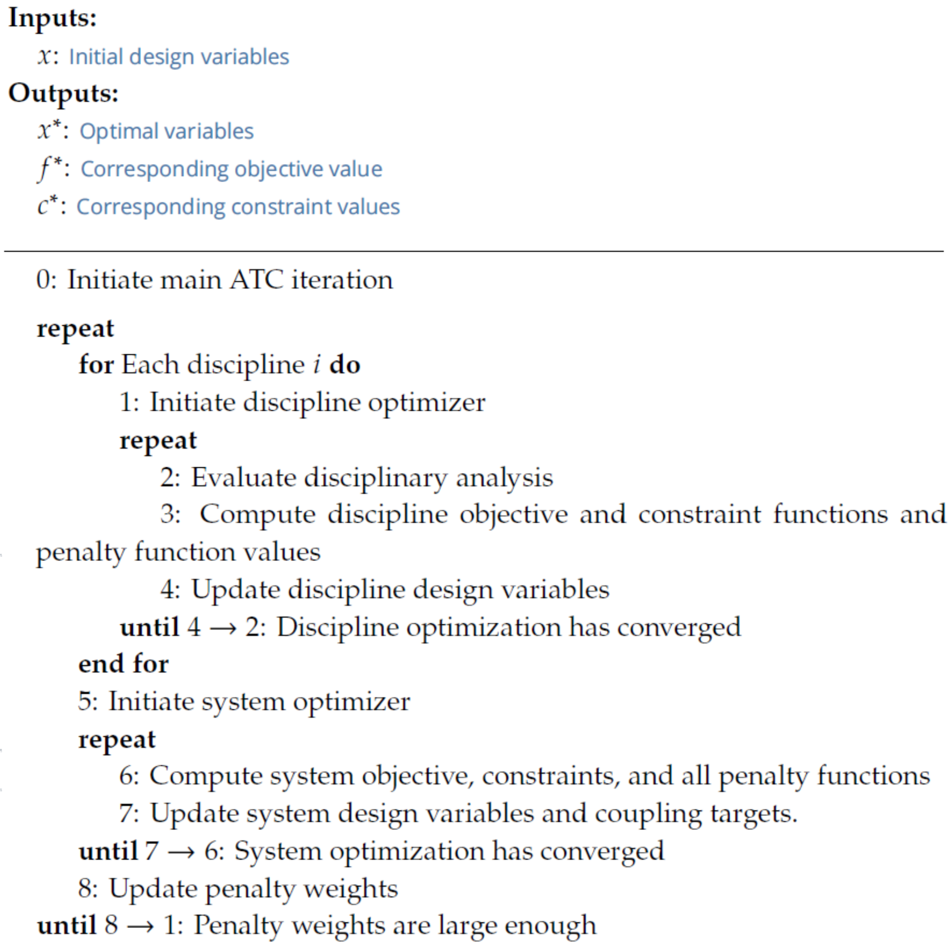

$w^{(0)}$
$x_0^{(0)}, \hat{u}^{t,(0)}$
$x_{0i}^{t,(0)}, x_i^{(0)}$
$0, 8 \to 1$: Update w
$6 : w$
$3 : w_i$
x_0^*
$5, 7 \to 6$: System optimization
$6 : x_0, \hat{u}^t$
$3 : x_0, \hat{u}^t$
$2 : \hat{u}_{j \neq i}^t$
$7 : f_0, \Phi_{0 \dots m}$
6 : System and penalty functions
x_i^*
$6 : x_{0i}^t, x_i$
$1, 4 \to 2$: Optimization i
$3 : x_{0i}^t, x_i$
$2 : x_{0i}^t, x_i$
$4 : f_i, g_i, \Phi_0, \Phi_i$
3 : Discipline i and penalty functions
$\hat{u}_i^*$
$6 : \hat{u}_i$
$3 : \hat{u}_i$
2 : Solver i

Figure 7.54 Diagram for the ATC architecture.

Source: Martins et Ning, 2021

6. MDO Architectures
6.2. Distributed MDO architectures
6.2.2 Analytic Target Cascading

Algorithm 7.6: Analytical target cascading



Inputs:
x: Initial design variables
Outputs:
x^*: Optimal variables
f^*: Corresponding objective value
c^*: Corresponding constraint values

```
0: Initiate main ATC iteration
repeat
    for Each discipline $i$ do
        1: Initiate discipline optimizer
        repeat
            2: Evaluate disciplinary analysis
            3: Compute discipline objective and constraint functions and penalty function values
            4: Update discipline design variables
        until 4 → 2: Discipline optimization has converged
    end for
    5: Initiate system optimizer
    repeat
        6: Compute system objective, constraints, and all penalty functions
        7: Update system design variables and coupling targets.
    until 7 → 6: System optimization has converged
    8: Update penalty weights
until 8 → 1: Penalty weights are large enough
```

6.2.3 Bilevel Integrated System Synthesis

- Bilevel integrated system synthesis (BLISS) uses a series of linear approximations to the original design problem, with bounds on the design variable steps to prevent the design point from moving so far away that the approximations are too inaccurate.
- These approximations are constructed at each iteration using coupled derivatives (see Section 7.5).

6.2.3 Bilevel Integrated System Synthesis

- The system-level subproblem is formulated as follows:

$$
\begin{array}{lll}
\textit{minimize} & (f_0^*)_0 + \left(\dfrac{df_0^*}{dx_0}\right)\Delta x_0 & \\
\textit{by varying} & \Delta x_0 & \\
\textit{subject to} & (g_0^*)_0 + \left(\dfrac{dg_0^*}{dx_0}\right)\Delta x_0 \leq 0 & \\
 & (g_i^*)_0 + \left(\dfrac{dg_i^*}{dx_0}\right)\Delta x_0 \leq 0 & i = 1, \dots, m \\
 & \underline{\Delta x_0} \leq \Delta x_0 \leq \overline{\Delta x_0} &
\end{array}
\qquad (7.36)
$$

6.2.3 Bilevel Integrated System Synthesis

- The discipline *i* subproblem is given by the following:

$$\begin{array}{ll} \textit{minimize} & (f_0)_0 + \left(\dfrac{df_0^*}{dx_i}\right)\Delta x_i \\ \textit{by varying} & \Delta x_i \\ \textit{subject to} & \begin{array}{l} (g_0)_0 + \left(\dfrac{dg_0^*}{dx_i}\right)\Delta x_i \leq 0 \\ (g_i)_0 + \left(\dfrac{dg_i^*}{dx_i}\right)\Delta x_i \leq 0 \end{array} \\ & \underline{\Delta x_i} \leq \Delta x_i \leq \overline{\Delta x_i} \end{array} \qquad (7.37)$$

- The extra set of constraints in both system-level and discipline subproblems denotes the design variable bounds.
- To prevent violation of the disciplinary constraints by changes in the shared design variables, post-optimality derivatives are required to solve the system-level subproblem.

6.2.3 Bilevel Integrated System Synthesis

- In this case, the post-optimality derivatives quantify the change in the optimized disciplinary constraints with respect to a change in the system design variables, which can be estimated with the Lagrange multipliers of the active constraints.
- Figure 7.55 shows the XDSM for BLISS, and the corresponding steps are listed in Alg. 7.7.
- Because BLISS uses an MDA, it is a distributed MDF architecture.
- As a result of the linear nature of the optimization problems, repeated interrogation of the objective and constraint functions is not necessary once we have the gradients.
- If the underlying problem is highly nonlinear, the algorithm may converge slowly.

6.2.3 Bilevel Integrated System Synthesis

- The variable bounds may help the convergence if these bounds are properly chosen, such as through a trust-region framework.

6.2.3 Bilevel Integrated System Synthesis

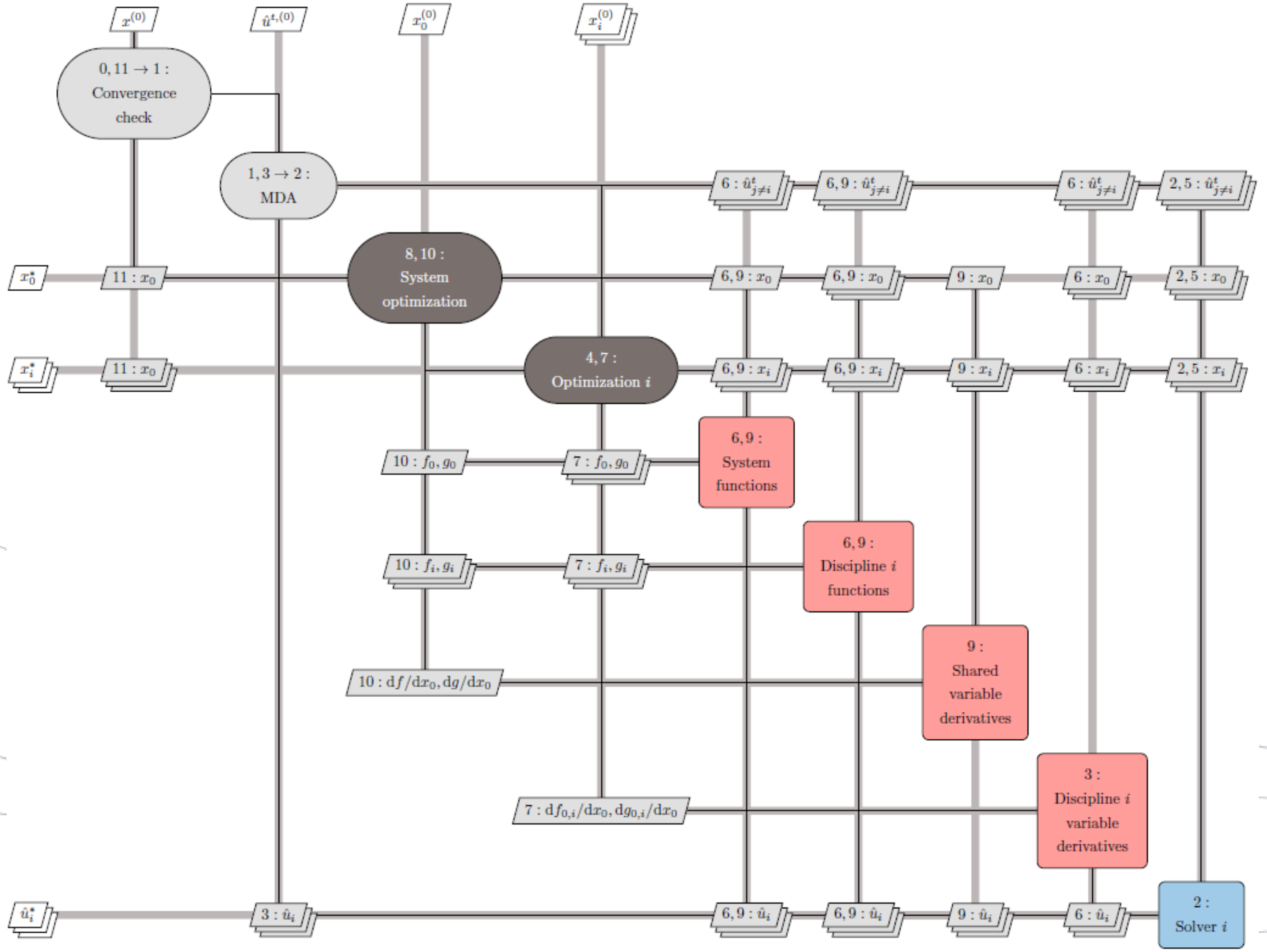
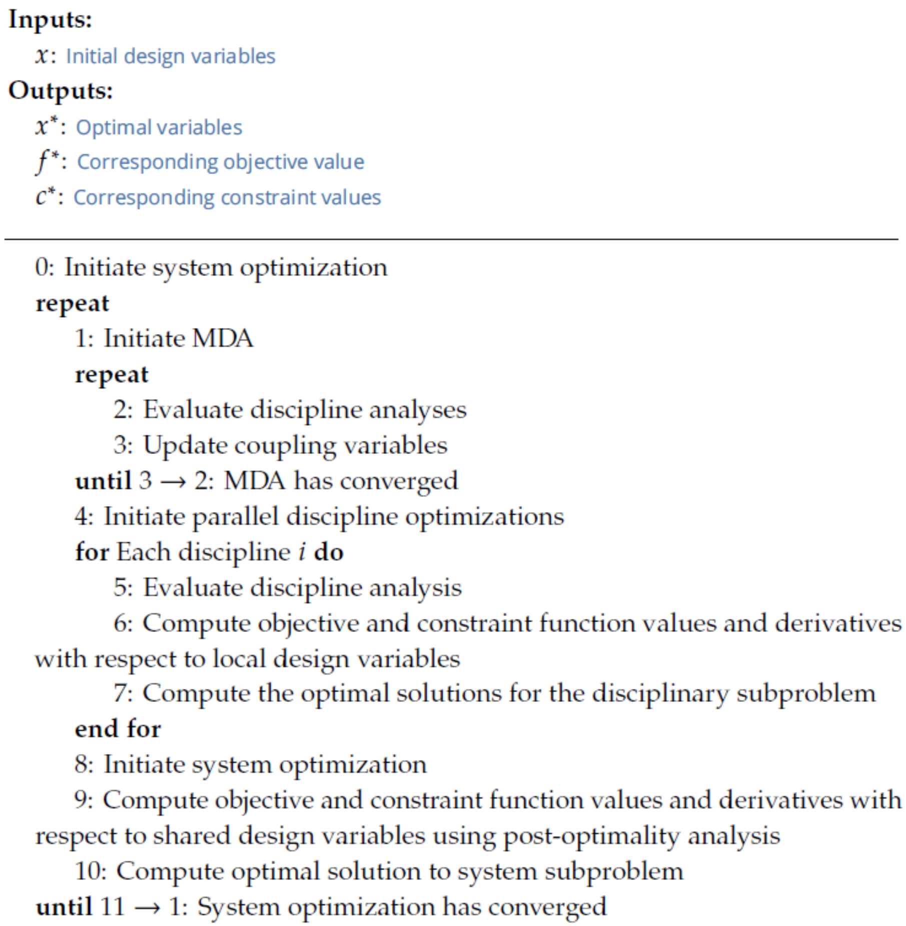


Figure 7.55 Diagram for the BLISS architecture.

Source: Martins et Ning, 2021

6.2.3 Bilevel Integrated System Synthesis

Algorithm 7.7: Bilevel integrated system synthesis



Inputs:
x: Initial design variables
Outputs:
x^*: Optimal variables
f^*: Corresponding objective value
c^*: Corresponding constraint values

```
0: Initiate system optimization
repeat
    1: Initiate MDA
    repeat
        2: Evaluate discipline analyses
        3: Update coupling variables
    until 3 → 2: MDA has converged
    4: Initiate parallel discipline optimizations
    for Each discipline $i$ do
        5: Evaluate discipline analysis
        6: Compute objective and constraint function values and derivatives with respect to local design variables
        7: Compute the optimal solutions for the disciplinary subproblem
    end for
    8: Initiate system optimization
    9: Compute objective and constraint function values and derivatives with respect to shared design variables using post-optimality analysis
    10: Compute optimal solution to system subproblem
until 11 → 1: System optimization has converged
```

6.2.4 Asymmetric Subspace Optimization

- Asymmetric subspace optimization (ASO) is a distributed MDF architecture motivated by cases where there is a large discrepancy between the cost of the disciplinary solvers.
- The cheaper disciplinary analyses are replaced by disciplinary design optimizations inside the overall MDA to reduce the number of more expensive disciplinary analyses.
- The system-level optimization subproblem is as follows:

$$\begin{array}{ccc} \textit{minimize} & f(x;\hat{u}) & \\ \textit{by varying} & x_0, x_k & \\ \textit{subject to} & g_0(x;\hat{u}) \leq 0 & \\ & g_k(x;\hat{u}_k) \leq 0 & \textit{for all } k \\ \textit{while solving} & r_k\left(\hat{u}_k; x_k, \hat{u}_{j\neq i}^{t}\right) = 0 & \\ \textit{for} & \hat{u}_k & \end{array} \qquad (7.38)$$

6.2.4 Asymmetric Subspace Optimization

where subscript *k* denotes disciplinary information that remains outside of the MDA.

- The disciplinary optimization subproblem for discipline *i*, which is resolved inside the MDA, is as follows:

$$\begin{array}{cc} minimize & f(x;\hat{u}) \\ by\ varying & x_i \\ subject\ to & g_i(x_0,x_i;\hat{u}_i) \leq 0 \\ while\ solving & r_i\left(\hat{u}_i;x_i,\hat{u}^t_{j\neq i}\right)=0 \\ for & \hat{u}_i \end{array} \qquad (7.39)$$

- Figure 7.56 shows a three-discipline case where the third discipline is replaced with a design optimization.
- The corresponding sequence of operations in ASO is listed in Alg. 7.8.

6.2.4 Asymmetric Subspace Optimization

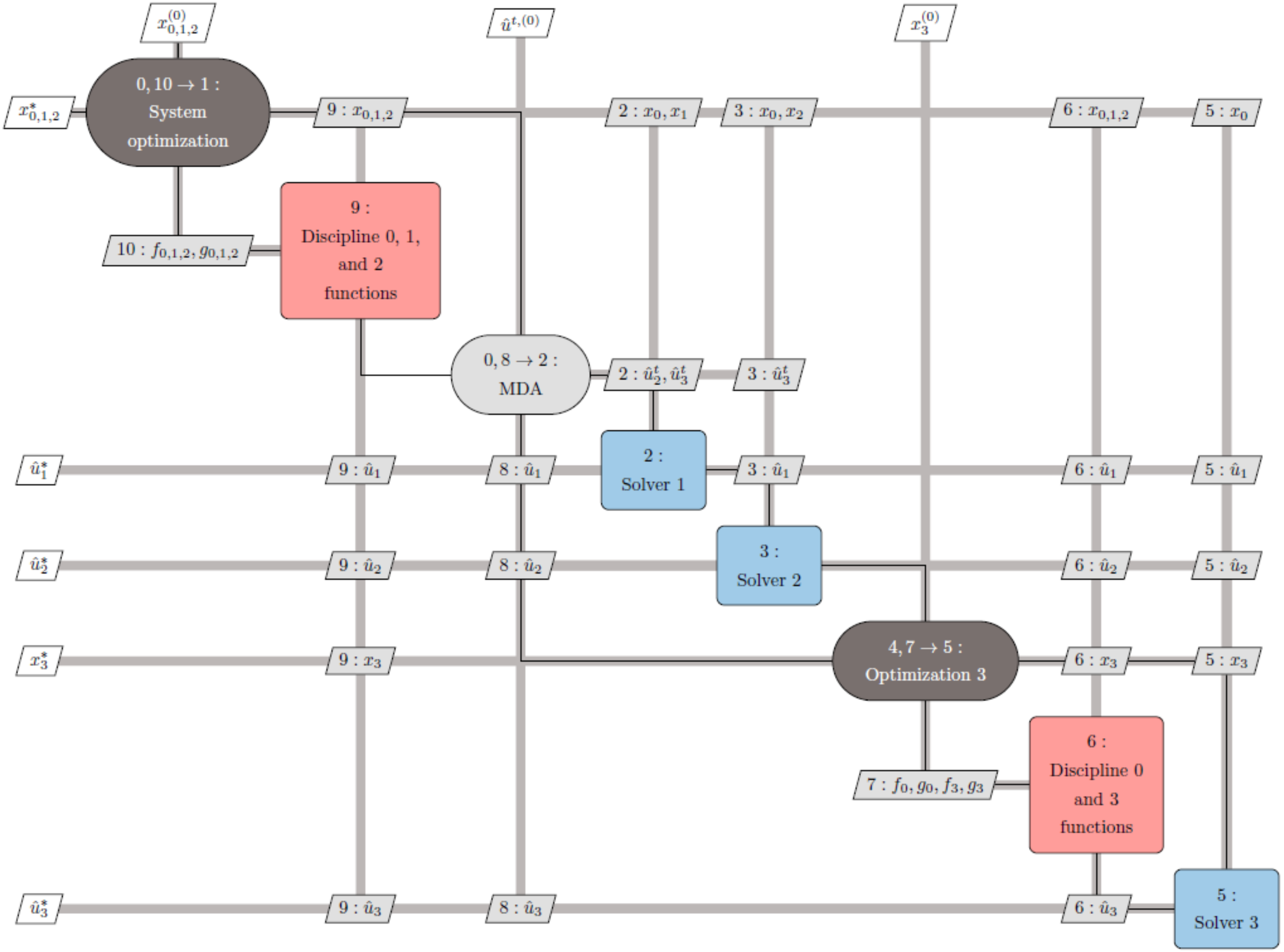
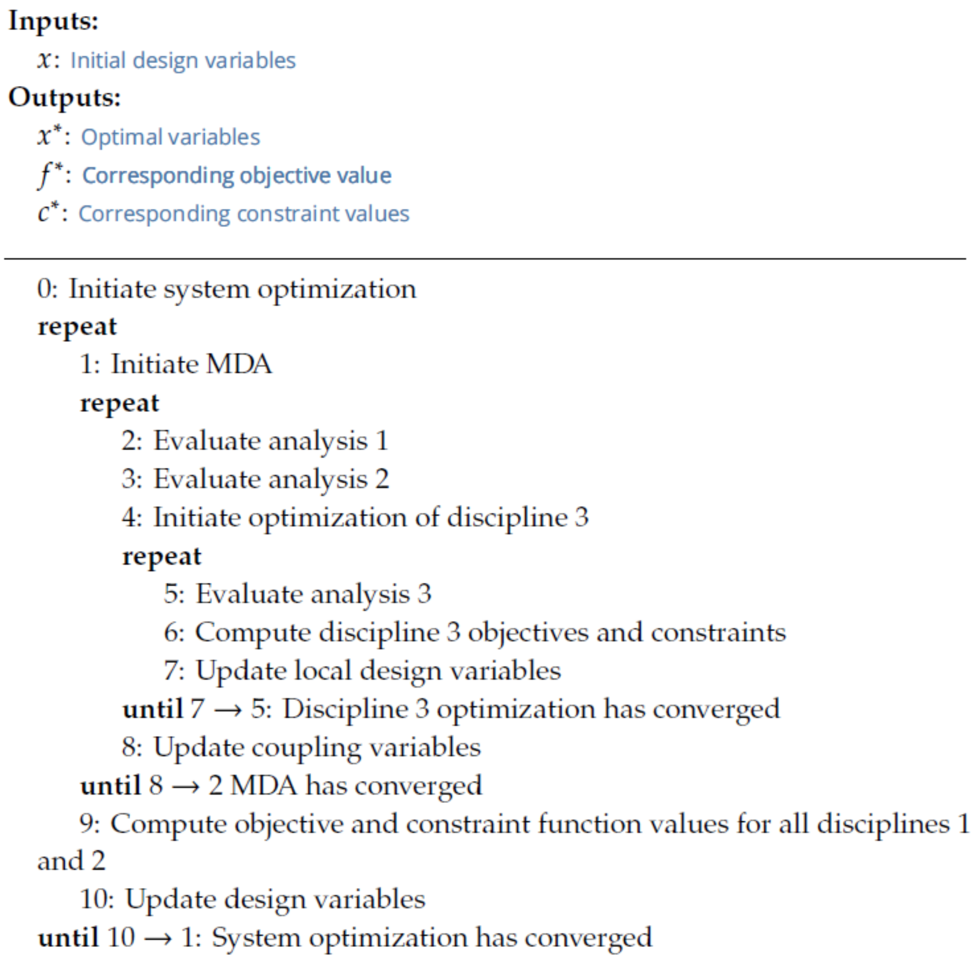


Figure 7.56 Diagram for the ASO architecture.

Source: Martins et Ning, 2021

6.2.4 Asymmetric Subspace Optimization

Algorithm 7.8: Asymmetric Subspace Optimization



Inputs:
x: Initial design variables

Outputs:
x^*: Optimal variables
f^*: Corresponding objective value
c^*: Corresponding constraint values

```
0: Initiate system optimization
repeat
    1: Initiate MDA
    repeat
        2: Evaluate analysis 1
        3: Evaluate analysis 2
        4: Initiate optimization of discipline 3
        repeat
            5: Evaluate analysis 3
            6: Compute discipline 3 objectives and constraints
            7: Update local design variables
        until 7 → 5: Discipline 3 optimization has converged
        8: Update coupling variables
    until 8 → 2 MDA has converged
    9: Compute objective and constraint function values for all disciplines 1 and 2
    10: Update design variables
until 10 → 1: System optimization has converged
```

6.2.4 Asymmetric Subspace Optimization

- To solve the system-level problem with a gradient-based optimizer, we require post-optimality derivatives of objective and constraints with respect to the subproblem inputs.
- For a gradient-based system-level optimizer, the gradients of the objective and constraints must take into account the suboptimization.
- This requires coupled post-optimality derivative computation, which increases computational and implementation time costs compared with a normal coupled derivative computation.
- The total optimization cost is only competitive with MDF if the discrepancy between each disciplinary solver is high enough.

6.2.4 Asymmetric Subspace Optimization

Example 7.15: Aerostructural optimization using ASO.

- Aerostructural optimization is an excellent example of asymmetry in the cost of the models.
- When the aerodynamics is modeled with computational fluid dynamics, it is usually much more expensive than a finite-element structural model.
- Assuming that structure is the less costly, we formulate the system-level optimization problem as follows:

$$\begin{array}{cc}
\textit{minimize} & f \\
\textit{by varying} & b, \gamma \\
\textit{subject to} & L - W^* = 0 \\
\textit{while solving} & A(d^*)\Gamma - \gamma(d^*) = 0 \\
\textit{for} & \Gamma
\end{array}$$

6.2.4 Asymmetric Subspace Optimization

Example 7.15: Aerostructural optimization using ASO (continued).

where W^* and d^* correspond to values obtained from the structural suboptimization,

$$\begin{array}{cc} \textit{minimize} & f \\ \textit{by varying} & t \\ \textit{subject to} & 2.5|\sigma| - \sigma_{yield} \leq 0 \\ \textit{while solving} & Kd - q = 0 \\ \textit{for} & d \end{array}$$

- Similar to the sequential optimization, we could replace f with W in the suboptimization because the other parameters in f are fixed.
- To solve the system-level problem with a gradient-based optimizer, we would need post-optimality derivatives of W^* with respect to b and Γ.

6.2.4 Asymmetric Subspace Optimization

Example 7.15: Aerostructural optimization using ASO (continued).

- To solve the system-level problem with a gradient-based optimizer, we would need post-optimality derivatives of W^* with respect to b and Γ.

6.2.5 Other Distributed Architectures

- There are other distributed MDF architectures in addition to BLISS and ASO: concurrent subspace optimization (CSSO) and MDO of independent subspaces (MDOIS).

- CSSO requires surrogate models for the analyses for all disciplines.
- The system-level optimization subproblem is solved based on the surrogate models and is therefore fast.
- The discipline-level optimization subproblem uses the actual analysis from the corresponding discipline and surrogate models for all other disciplines.
- The solutions for each discipline subproblem are used to update the surrogate models.

6.2.5 Other Distributed Architectures

- MDOIS only applies when no shared variables exist.
- In this case, discipline subproblems are solved independently, assuming fixed coupling variables, and then an MDA is performed to update the coupling.

- There are also other distributed IDF architectures.
- Some of these are similar to CO in that they use a multilevel approach to enforce multidisciplinary feasibility: BLISS-2000 and quasi-separable decomposition (QSD).
- Other architectures enforce multidisciplinary feasibility with penalties, like ATC: inexact penalty decomposition (IPD), exact penalty decomposition (EPD), and enhanced collaborative optimization (ECO).

6.2.5 Other Distributed Architectures

- BLISS-2000 is a variation of BLISS that uses surrogate models to represent the coupling variables for all disciplines.
- Each discipline subproblem minimizes the linearized objective with respect to local variables subject to local constraints.
- The system-level subproblem minimizes the objective with respect to the shared variables and coupling variables while enforcing consistency constraints.
- When using QSD, the objective and constraint functions are assumed to depend only on the shared design variables and coupling variables.
- Each discipline is assigned a “budget” for a local objective, and the discipline problems maximize the margin in their local constraints and the budgeted objective.

6.2.5 Other Distributed Architectures

- The system-level subproblem minimizes the objective and budgets of each discipline while enforcing the shared constraints and a positive margin for each discipline.

- IPD and EPD apply to MDO problems with no shared objectives or constraints.
- They are similar to ATC in that copies of the shared variables are used for every discipline subproblem, and the consistency constraints are relaxed with a penalty function.
- Unlike ATC, however, the more straightforward structure of the discipline subproblems is exploited to compute post-optimality derivatives to guide the system-level optimization subproblem.

6.2.5 Other Distributed Architectures

- Like CO, ECO uses copies of the shared variables.
- The discipline subproblems minimize quadratic approximations of the objective while enforcing local constraints and linear models of the nonlocal constraints.
- The system-level subproblem minimizes the total violation of all consistency constraints with respect to the shared variables.

6.2.6 Summary

- Figure 7.57 summarizes the MDO architecture classification.

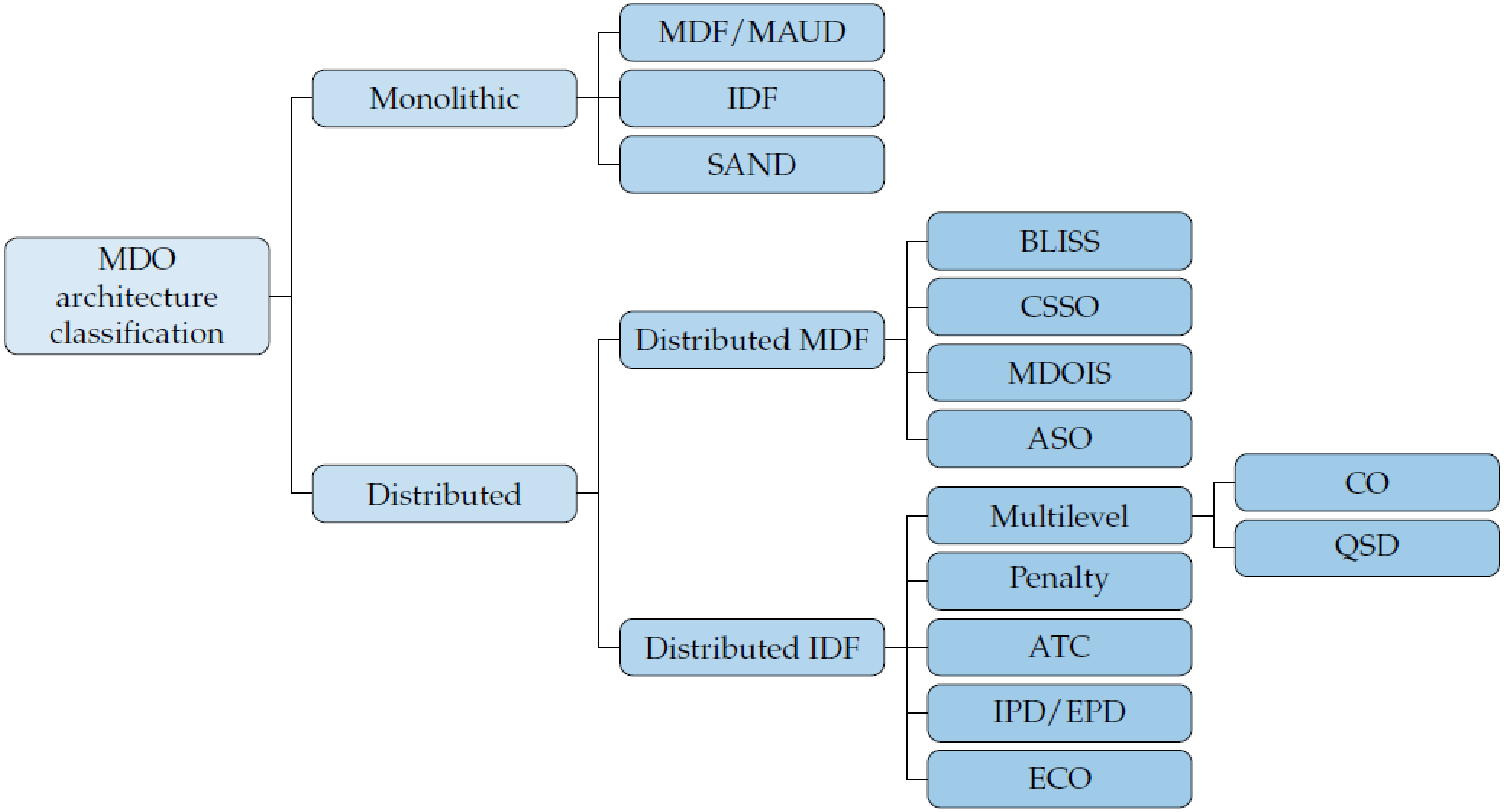


Figure 7.57 Classification of MDO architectures.

Source: Martins et Ning, 2021

7. Solvers’ Fidelity

- Definition of Fidelity in Engineering Solvers:
  - Fidelity refers to the level of detail, accuracy, and realism in the simulation of physical systems.

- Importance of Fidelity:
  - Influences the reliability of results.
  - Affects computational cost and resource usage.

- Different levels of fidelity serve unique purposes in engineering.
- Balancing accuracy and computational cost is crucial.
- Selecting the right solver depends on the project phase and requirements.
- Emerging technologies are expanding the capabilities of engineering solvers.

7.1 Levels of fidelity

- Low-Fidelity Solvers:
  - Simplified models with minimal computational cost.
  - Examples: Empirical correlations, lumped parameter models.
  - Applications: Early-stage design, quick feasibility studies.

- Medium-Fidelity Solvers:
  - More detailed models with moderate computational demand.
  - Examples: Finite difference methods, reduced-order models.
  - Applications: Trade-off studies, prototyping.

- High-Fidelity Solvers:
  - Comprehensive and highly detailed simulations.
  - Examples: Computational Fluid Dynamics (CFD), Finite Element Analysis (FEA).
  - Applications: Final design validation, safety-critical systems.

7.2 Characteristics of low-fidelity solvers

- Advantages:
  - Quick to implement and run.
  - Requires less computational power.
  - Useful for large parameter sweeps.
- Disadvantages:
  - Limited accuracy.
  - May oversimplify complex phenomena.
- Example Applications:
  - Initial sizing of components.
  - Rough performance estimates.

7.3 Characteristics of medium-fidelity solvers

- Advantages:
  - Balanced trade-off between accuracy and computational cost.
  - Provides more insights than low-fidelity models.

- Disadvantages:
  - Requires more detailed input data.
  - Moderate computational demand.

- Example Applications:
  - Iterative design optimization.
  - Sensitivity analyses.

7.4 Characteristics of high-fidelity solvers

- Advantages:
  - High accuracy and detailed results.
  - Captures complex interactions and phenomena.

- Disadvantages:
  - Computationally expensive.
  - Requires significant expertise and time.

- Example Applications:
  - Aerodynamic simulation of aircraft.
  - Stress analysis in critical structures.

7.5 Trade-offs in solver selection

- Factors to Consider:
  - Accuracy vs. computational cost.
  - Time constraints and project scope.
  - Availability of computational resources.
  - Phase of the engineering process.
- Decision Framework:
  - Use low-fidelity solvers for early-stage design.
  - Transition to medium and high-fidelity solvers as the design matures.

7.6 Case study: Aerospace Engineering

- Low-Fidelity Example:
  - Lift and drag estimation using empirical formulas.
- Medium-Fidelity Example:
  - Panel methods for aerodynamic analysis.
- High-Fidelity Example:
  - CFD simulations for turbulent flow analysis.
- Outcome:
  - Combining levels of fidelity ensures efficient and accurate design processes.

7.7 Emerging trends

- Hybrid Fidelity Approaches:
  - Combining low, medium, and high-fidelity models for efficiency.
- Machine Learning Integration:
  - Enhancing solver speed and adaptability.
- Cloud Computing:
  - Enabling access to high-performance computing for high-fidelity solvers.

8. Decision Support