Structure-Preserving Differentiation of Functional Networks in Design Optimization and Optimal Control

Tobias Weigl and Rainer Callies

Centre for Mathematical Sciences M2, Technische Universität München
Boltzmannstraße 3, 85748 Garching, Germany
e-mail: weigl@ma.tum.de and callies@ma.tum.de

Abstract

Model-based design of complete technical processes or systems is increasingly more often realized by modular development approaches like MATLAB and SIMULINK. They enable engineering teams from various disciplines easily to cooperate. Subsystems are encapsulated into blocks of information, total models are formed by nesting and linking single modules or blocks. A profound knowledge of the details of the respective subsystem is no longer necessary for system design and integration, a new level of abstraction is introduced. For multidisciplinary design optimization and optimal system control – both in a mathematically strict sense – the knowledge of the model system alone is not sufficient. The need of sensitivity information makes it necessary to compute first and second order derivatives of the model equations, optimal control of those systems subject to multiple constraints requires the calculation of even higher order derivatives. Analytical differentiation of complex models is extremely time consuming and error-prone. Numerical differentiation suffers from inaccuracy and high computational costs. Algorithmic or automatic differentiation techniques destroy the original structure of the model and thus show a reduced efficiency in case of higher order derivatives. In this paper, a modified approach is presented combining analytical and automatic differentiation while preserving the modular structure of the model. The analytical derivatives of a single (sub-)module are typically available with a tolerable effort. The information of a single module is forwarded via local transfer matrices, which encode the linking structure between two modules. It is shown that the model’s global topology can be computed from this local information: One can determine those components of the model input, which affect a particular output component, if only the local dependencies are known. The proposed algorithm works especially well for hierarchical models with a nested module structure. Only the derivatives of the module lowest level have to be available. The sensitivity information is automatically propagated through the different levels. Two algorithmic implementations of this approach have been realized: An elegant recursion for each (sub)module and a more efficient forward propagation of efficiently calculated intermediate values. As an industrial application, contributions to aerobatic airplanes in the context of the Red Bull AirRaces are presented. The full 6 DoF airplane model is split into its dynamic layers: position, translation, attitude and rotation. The layers are modularly formulated as sets of blocks, each representing a distinct effect (e.g. deflection of a rudder) or quantity (e.g. aerodynamic force). This results in a hierarchical model structure. Derivatives of orders up to three are calculated for optimization and optimal control purposes. Keywords: modular model, optimal control, structure preserving derivatives, automatic differentiation.

1. Introduction

One recent trend in engineering optimization is the widespread use of modelling tools like MATLAB and SIMULINK. They allow the design of complete technical processes or systems on a higher level of abstraction. Engineering teams from various disciplines can deliver their subsystems encapsulated into blocks of information. A detailed knowledge of the respective subsystem is no longer necessary for the non-specialist user: the respective block subsystem can be regarded as a black box with some inputs and outputs having a fixed position and linking structure within the model network. The full model is formed by nesting and linking the single block subsystems.

From an engineering point of view, the modular approach guarantees clearness, manageability and flexibility. It allows to change parts of a complete model without touching the structure of the rest even at runtime of simulation tasks. At the same time, theses model properties are the prerequisites for an efficient numerical implementation of elaborate mathematical multi-model approaches.

For optimization or optimal control purposes, the knowledge of the mathematical model alone is not sufficient. To improve a system, its sensitivity with respect to parameters and controls has to be
known. At the same time, multiple system constraints have to be strictly observed throughout the entire optimization process. This makes it necessary to compute first, second and even higher order derivates of the model equations. For the use in highly efficient numerical optimization algorithms, an efficient access to highly accurate derivative information is crucial.

Whenever the model equations are available explicitly, also the partial derivatives with respect to the state and the design variables can be computed explicitly. For small systems, this computation can be done by hand. The analytical differentiation of complex models is extremely time consuming and error-prone. An alternative e.g. in optimal control is the numerical differentiation, but it suffers from inaccuracy and high computational costs [10, 7]. For larger systems, formula manipulation programs such as MAPLE [12] or alternatively Automatic Differentiation procedures [3, 4, 5, 8, 9] reduce the non-computational effort. Basically, these techniques break down every function to elementary mathematical operations and apply the chain rule for the derivative of the composite of two functions to the sequence of those operations. By these approaches the original structure of the model is destroyed, i.e. not mapped to the calculation of the derivatives. The effort is significantly increasing with system complexity and size [2] and especially in case of higher derivatives [1].

In this paper, the modular approach is extended to the evaluation of derivatives of different orders. Based on the same network of linked blocks which is used for model evaluation, model derivatives are calculated by solely evaluating the sensitivities of each single block. Block sensitivities are defined as the partial derivatives of the output channels of a block with respect to its input channels. For the basic building blocks analytical derivatives are typically available at a tolerable effort. This information is then accumulated throughout the total network. Information of each block module is forwarded via so-called local transfer matrices, which encode the linking structure between two modules. Thus the method is perfectly integrable into an existing modular model. No approximation steps are necessary (in contrast to e.g. the finite difference approach), the derivatives are calculated with extreme accuracy.

The algorithm is especially well suited for hierarchical models with a nested module structure, i.e. a module may contain submodules, which again contain submodules, and so on. Only the derivatives of the lowest modelling level have to be available. The sensitivity information is automatically propagated from the lowermost to the uppermost level to yield the desired derivative of the full model.

2. Graphs and Transfer Matrices

This chapter introduces the concept of transfer matrices for especially structured graphs containing blocks as sub-graphs. These particular structures cover all cases arising in modular modelling. The results are hierarchical graphs (or networks). Local and global linking information is carried by transfer matrices of explicit and implicit type, respectively.

Theorem 3 allows the elegant calculation of the implicit matrices based on the explicit ones throughout different levels of the network hierarchy by nesting two different types of recursions.

Let $B = (V_B, E_B)$ be a directed graph with the set of vertices $V_B$ and the set of edges $E_B \subseteq V_B \times V_B$, respectively. Let furthermore $I_B \subseteq O_B \subseteq V_B$ satisfy $V_B = I_B \cup O_B$ and $E_B \subseteq I_B \times O_B$. These sets are called the input- and the output-vertices of $B$ and it is assumed that in every vertex of $I_B$ at least one edge starts and in every vertex of $O_B$ at least one edge ends, i.e.

$$\forall v \in V_B \exists e \in E_B : v \text{ lies on } e.$$  

Such a graph is called a block and will later be used to represent functions only by means of their input and output channels, corresponding to the sets $I_B$ and $O_B$. The dependencies between the functions are described by the edges.

Concatenation of functions is done by linking the outputs of one block with the inputs of another block resulting in a whole level of linked blocks. To form a function in a mathematical correct way, the directed graph $S = (V_S, E_S)$ corresponding to a set $B = \{B_i | i = 1, \ldots, m\}$ of $m$ linked blocks is assumed to have the following properties:

1. $V_S = \bigcup_{i=1}^{m} I_{B_i} \cup O_{B_i}$
2. $\forall (e_1, e_2) \in E_S \subseteq V_S \times V_S$ holds
   (a) $e_1 \in I_{B_i} \Rightarrow e_2 \in O_{B_i}$
   (b) $e_1 \in O_{B_i} \Rightarrow e_2 \in I_{B_j}$ for $i \neq j$
3. \[
n_{\text{input}}(v) := |\{ w \in \bigcup_{i=1,\ldots,m} O_{B_i} : (v, w) \in E_S \}| \leq 1 \quad \forall v \in \bigcup_{i=1,\ldots,m} I_{B_i} \tag{1}
\]

4. For all blocks \( B \in \mathcal{B} \) holds
\[
|\{(v, w) \in E_S : w \in I_B\}| \leq 1 \quad \forall v \in \bigcup_{i=1,\ldots,m} O_{B_i} \tag{2}
\]

5. \( S \) contains no closed loop.

With the sets
\[
I_S := \left\{ v \in \bigcup_{i=1,\ldots,m} I_{B_i} \mid n_{\text{input}}(v) = 0 \right\},
\]
\[
O_S := \left\{ v \in \bigcup_{i=1,\ldots,m} O_{B_i} \mid \not\exists w \in V_S : (v, w) \in E_S \right\},
\]
\[
X := \{i_1, \ldots, i_{|I_S|}\},
\]
\[
Y := \{o_1, \ldots, o_{|O_S|}\},
\]
the triple \((X, B, Y)\) is defined as the directed graph which results from merging \(X, Y\) and \(V_S\) and adding appropriate edges between \(X\) and \(I_S\) and between \(O_S\) and \(Y\). \(i_j\) denotes the input channel \(j\), \(o_k\) the output channel \(k\). The result is a network with input \(^*\) \(X\), output \(Y\) and a set of linked boxes (see Fig. 1(a)).

![Network layer as a graph.](a)

By construction, there is a path from every input node to the set of output nodes and every output node is the terminal point of a path starting in the set of the input nodes. These properties allow to transform \((X, B, Y)\) into a block \(B = (V, E)\) in two steps:

1. \( V := X \cup Y \) and \( E := \emptyset \)
2. Add \((X_i, Y_j)\) to \(E\), if there is a path between them \((1 \leq i \leq |X|, 1 \leq j \leq |Y|)\).

\*From a more graph-theoretic point of view, we could call \(X\) the sources, \(Y\) the sinks and \(B\) a set of subgraphs of the constructed graph.

![Sub-layer with nested blocks in block B2](b)

Figure 1: Example of a network layer with two blocks in graph and in block representation.
$B$ is called the top-block and the blocks in the associated sub-layer are called sub-blocks. Thus one gets $I_B = X$ and $O_B = Y$; in this way a whole hierarchy of nested blocks and layers can be constructed recursively (see Fig. 1(b)).

The aim is to deal with a block as some kind of black box, which gets some input, produces some output and sends this output into the surrounding network along the corresponding edges. Although the complexity of the whole network might be considerable, every single block is typically linked only to a few other blocks by an explicit edge. This leads to a sparse structure of the global network. From a local or block-oriented point of view, dimensions and complexity are rather small, as one just has to deal with the connected blocks.

The linking structure is assumed to be known for every level of the network hierarchy. The aim is to generate the global information of the whole network from the given local information. Based on the concept of a graph’s adjacency matrix and inspired by the idea that the input in some sense flows [11] through the network, local transfer matrices are defined, which could be interpreted as a local version of an adjacency matrix.

**Definition 1 (Transfer matrices).** Let $(X, B, Y)$ be a network layer. For two blocks $A \neq B \in B$ we define the explicit transfer matrix $\langle A \rightarrow B \rangle \in \mathbb{R}^{|O_A| \times |I_B|}$ and the implicit transfer matrix $\langle A \Rightarrow B \rangle \in \mathbb{R}^{|O_A| \times |I_B|}$ by

$$\langle A \rightarrow B \rangle_{i,j} := \begin{cases} 1 & \text{there is an edge from } O_A,i \text{ to } I_B,j \\ 0 & \text{otherwise.} \end{cases}$$

$$\langle A \Rightarrow B \rangle_{i,j} := \begin{cases} \text{n} & \text{there are \text{n} paths from } O_A,i \text{ to } I_B,j \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

For a single block $B \in B$ we define the block transfer matrix $\langle B \rightarrow B \rangle \in \mathbb{R}^{|I_B| \times |O_B|}$ by

$$\langle B \rightarrow B \rangle_{i,j} := \begin{cases} 1 & \text{there is an edge from } I_B,i \text{ to } O_B,j \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The transfer matrices $\langle X \rightarrow B \rangle, \langle X \Rightarrow B \rangle \in \mathbb{R}^{|X| \times |I_B|}$ and $\langle B \rightarrow Y \rangle, \langle B \Rightarrow Y \rangle \in \mathbb{R}^{|O_B| \times |Y|}$ as well as $\langle X \Rightarrow Y \rangle \in \mathbb{R}^{|X| \times |Y|}$ are defined analogously.

**Lemma 2.** For two blocks $A \neq B$ we have

$$\sum_{i=1}^{|O_A|} \langle A \rightarrow B \rangle_{i,j} \leq 1 \quad \forall 1 \leq j \leq |I_B|$$

$$\sum_{j=1}^{|I_B|} \langle A \rightarrow B \rangle_{i,j} \leq 1 \quad \forall 1 \leq i \leq |O_A|.$$  

**Proof** It follows directly from Eq. (1), (2) and the subsequent constructions. \hfill \Box

By assumption, only the local linkage-structure of the blocks is known, i.e. the explicit transfer matrices. The global structural information of one network layer, however, is contained in the implicit matrices. These can be calculated in an elegant recursive way.

**Theorem 3 (Calculation of implicit transfer matrices).** Let $(X, B, Y)$ be a network layer. For $B_1 \neq B_2 \in B$ the recursion

$$\langle B_1 \Rightarrow B_2 \rangle = \langle B_1 \rightarrow B_2 \rangle + \sum_{K \in B \setminus \{B_1, B_2\}} \langle B_1 \rightarrow K \rangle \langle K \rightarrow K \rangle \langle K \Rightarrow B_2 \rangle \tag{5}$$

holds.

**Proof.** A proof is given in [13], but omitted here because of page length limitation. \hfill \Box
Corollary 4. The implicit transfer matrix \( (X \Rightarrow Y) \) of a network layer \((X, B, Y)\) is given by

\[
(X \Rightarrow Y) = \sum_{K \in B} (X \Rightarrow K) (K \Rightarrow Y) (K \Rightarrow Y)
\]

and the equation

\[
(K \Rightarrow Y) = (K \Rightarrow Y) + \sum_{M \in B \setminus \{K\}} (K \Rightarrow M) (M \Rightarrow M) (M \Rightarrow Y).
\]

Proof. The statement follows directly from Theorem 3 and \((X \Rightarrow Y) = 0\).

Using the framework of the transfer matrices, every path within a layer can be split into a part within a certain block and the remaining part outside this block

\[
\langle O_{B_1} \Rightarrow O_{B_2} \rangle = \langle O_{B_1} \Rightarrow I_{B_2} \rangle \langle B_2 \Rightarrow B_2 \rangle
\]

\[
\langle I_{B_1} \Rightarrow I_{B_2} \rangle = \langle B_1 \Rightarrow B_1 \rangle \langle O_{B_1} \Rightarrow I_{B_2} \rangle.
\]

Transposing the block transfer matrices is equivalent to an inversion of the direction of the edges. The transfer matrices behave in a very intuitive way.

Eqs. (6),(7) determine every implicit transfer matrix by a set of explicit and, again, implicit matrices. This recursion induced by the summation term can be interpreted as running through all blocks of the considered network layer and is called recursion of type one. As only explicit transfer matrices can be evaluated, the recursion terminates if no further implicit matrix appears on the right hand side.

Under the assumption that there exists neither a edge nor a path from \(O_B\) to \(I_B\) for every block \(B\) in a network layer, it can easily be shown that a recursion of type one terminates after not more than \(|B| - 1\) steps. Without closed loops within the graph every block that violates that assumption can be split into two blocks and termination again is guaranteed.

In case of dealing with highly hierarchical networks, every single block \(B\) may contain a whole sub-layer \((X, B, Y)\). Then its internal dependencies between inputs and outputs, i.e. \((B \Rightarrow B)\), are unknown a priori and have to be calculated. This is done by a nested recursion called recursion of type two, which is defined by

\[
(B \Rightarrow B)_{i,j} = \begin{cases} 1 & \text{if } \langle \hat{X} \Rightarrow \hat{Y} \rangle_{i,j} \neq 0, \ 1 \leq i \leq |I_B|, 1 \leq j \leq |O_B|, \\ 0 & \text{otherwise} \end{cases}
\]

This recursion can be interpreted as stepping down one level of the network’s hierarchy and starting the recursion of type one on that sub-layer. With only a finite number of levels, the termination of the complete process of nested recursions is equivalent to the termination of every recursion of type one.

3. Functional networks

This Sect. addresses the construction of blocks and corresponding graphs from given nested and concatenated functions or modules. Thus, the transition from functional networks to the established graph-framework is carried out and the results of Sect. 2 can be used to calculate global dependencies within a possibly highly modular function.

Up to now blocks have been introduced as abstract dependencies between inputs and outputs. Using the transfer matrix framework these dependencies can be encoded in a mathematically strict way. It becomes possible to calculate global dependency information even in presence of several levels of a network hierarchy.

For applications in engineering optimization, the abstract formalism has to be applied to functions or – more generally – to modules containing functional dependencies. These are either explicitly known or available as so-called black boxes. A model is a network of nested and concatenated modules. As a result of the previous section it is sufficient to know the linking-structure between the modules and the internal input/output dependencies of the sub-layers at the lowermost level of the network’s hierarchy. By means of the two variants of the recursion defined above, this information is propagated to the uppermost level, i.e. the model one actually is interested in.

The transformation of a function into a block can be formalized in the following way:

Let \(f : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_o}\), \(n_x, n_o \in \mathbb{N}\), be a smooth function with input \(x\) and output \(f(x)\). The corresponding block \(B_f = (V_f, E_f)\) is characterized by
• $|I_B| = n_i$ and $|O_B| = n_o$
• $I_B = \{x_1, ..., x_{n_i}\}$ and $O_B = \{f_1, ..., f_{n_o}\}$
• There is an edge between $x_i$ and $f_j$ if the $j^{th}$ output of $f$ depends on its $i^{th}$ input, i.e.

$$(x_i, f_j) \in E_f \iff \exists x_0 \in \mathbb{R}^{n_i} : Df(x_0)_{j,i} = \frac{\partial f_j(x)}{\partial x_i} \Big|_{x=x_0} \neq 0 \iff \langle B_f \rightarrow B_f \rangle_{i,j} = 1$$

Concatenation of functions is achieved by setting an appropriate edge between their inputs and outputs. Let $f, g$ be two functions with corresponding blocks $B_f, B_g$ in the layer $(X, B, Y)$ of the considered network. If the $j^{th}$ output of $g$ is the $i^{th}$ input of $f$, the edge $(O_{B_i, i}, I_{B_j, j})$ has to be added to that layer by setting $\langle B_g \rightarrow B_f \rangle_{i,j} = 1$.

4. First order derivatives

Within the framework established so far, a closed formulation for the first order derivative of the input of a block $B \in \mathcal{B}$ with respect to an input channel of layer $(X, B, Y)$ is possible. For any $K \in \mathcal{B}$ a set of $i^{th}$ unit vectors in various dimensions is defined by

$$e_i \in \mathbb{R}^{|I_B|}, \quad \bar{e}_i \in \mathbb{R}^{|X|}, \quad \bar{e}_i \in \mathbb{R}^{|K|}, \quad \bar{e}_i \in \mathbb{R}^{|O_K|}.$$ 

Let $I_K(x_0)$ be the values of the block input of $K$ resulting from an layer input $x_0$ and $DK$ the Jacobian of $K$.

**Theorem 5** (Derivative of a Block Input). Let $(X, B, Y)$ be a network layer and $B \in \mathcal{B}$. For $1 \leq i \leq |I_B|$, $1 \leq j \leq |X|$ and a layer input $x_0$ one gets

$$\frac{d}{dx_j} I_{B_i}(x_0) = \begin{cases} e_j^T (X \rightarrow B) e_i & \text{, } \langle X \rightarrow B \rangle e_i \neq 0 \\ \sum_{K \in \mathcal{B}} [K \rightarrow B] e_i^T \cdot DK(I_K(x_0)) \cdot A_K(x_0) & \text{, otherwise} \end{cases} \tag{9}$$

with

$$A_K(x_0) := \sum_{k=1,...,|K|} \left\{ [(K \rightarrow K) \langle K \rightarrow B \rangle e_i]^T \cdot e_k \right\} \cdot \frac{d}{dx_j} I_{K,k}(x_0) \cdot \bar{e}_k \tag{10}$$

**Proof.** A proof can be found in [13]. □

**Corollary 6** (Derivative of a Layer). Let $(X, B, Y)$ be a layer and $f$ the corresponding function. For $1 \leq i \leq |Y|$, $1 \leq j \leq |X|$ and a layer input $x_0$ the following equation holds

$$\frac{df_i(x)}{dx_j} \Big|_{x=x_0} = \frac{d}{dx_j} I_{Y,i}(x_0). \tag{11}$$

**Proof** By construction one has $f_i = Y_i$ and $x_j = X_j$ for $i = 1, ..., |Y|$, $j = 1, ..., |X|$, see Sect. 3. The statement directly follows from the observation that the vertices of $Y$ are the input-vertices of $Y$ themselves, i.e. $Y_i = I_{Y,i}$. □

Eq. (9) implies two different types of recursive calculations. Within the current layer each block $K$ is processed; evaluation of $A_K(x_0)$ requires again the evaluation of Eq. (9). Thus a recursion of type one is induced by the summation.

The evaluation of the Jacobians of the blocks leads to a recursion of type two, which steps down to the corresponding sub-layer and starts the recursion of type one. This process stops if the lowermost level in each’s block sub-structure is reached and the derivatives can actually be evaluated without restarting any recursion.
5. Higher order derivatives

For higher order derivatives the notation is getting rather complicated. Therefore, in this paper only the algorithm for derivatives of order two is described in detail. The numerical algorithms have been successfully implemented and tested for derivatives up to order three.

**Theorem 7** (Derivatives of order two). Let \((X, B, Y)\) be a layer and \(B \in B\). For \(1 \leq i \leq |I_B|\), \(1 \leq j, m \leq |X|\) and a layer input \(x_0\) one gets

\[
\frac{d^2}{dX_mdX_j} I_{B,i}(x_0) = \begin{cases} 0, & \langle X \to B \rangle e_i \neq 0 \\ \sum_{K \in B} (A_K^m(x_0))^T \cdot \hat{H}K(I_K(x_0)) \cdot A_K^l(x_0) + \\ \hfill + \sum_{K \in B} ((K \to B) e_i)^T \cdot DK(I_K(x_0)) \cdot A_K^{jm}(x_0), & \text{otherwise} \end{cases}
\]

with \((l \in \{j, m\})\)

\[
A_K^l(x_0) := \sum_{k=1,\ldots,|I_K|} \left\{ ([K \to K] (K \to B) e_i)^T \hat{e}_k \right\} \cdot \frac{d}{dX_k} I_{K,k}(x_0) \cdot \hat{e}_k
\]

\[
A_K^{jm}(x_0) := \sum_{k=1,\ldots,|I_K|} \left\{ ([K \to K] (K \to B) e_i)^T \hat{e}_k \right\} \cdot \frac{d^2}{dX_mdX_j} I_{K,k}(x_0) \cdot \hat{e}_k
\]

and the symmetric matrix

\[
\hat{H}K(I_K(x_0)) := D \left\{ ([K \to B] e_i)^T \cdot DK(I_K(x_0)) \right\}.
\]

As usual, \(D\) denotes the differential operator.

**Proof.** Application of the chain rule to Eq. (9).

6. Details of Implementation

The recursive structure of Th. 5 and Cor. 6 allows a very elegant implementation. However, if the number of blocks and levels in the networks increases, a straightforward implementation of the recursive approach can lead to bad performance of the algorithm in case of derivatives of higher order, because many expressions are calculated more than once.

Performance can be improved by an appropriate data storage management in combination with elaborate search strategies. In every step it is checked whether a certain expression has already been calculated or not.

In the present implementation, another strategy has been developed in addition. The direction in which the algorithm runs through the network has been reversed. So far, Cor. 6 induces a movement from the output to the input. Using the results of Sect. 2 it is possible to formulate a similar algorithm running from the input to the output and storing the calculated interim values. This approach is named cascade. By the cascade algorithm it is possible to achieve a significant acceleration of the calculations.

7. Industrial Application
As an industrial application, contributions to the optimization of optimal flight trajectories for aerobatic airplanes in the context of the Red Bull AirRaces are presented.

Modelling of aerobatic planes significantly differs from the modelling approach for aircraft in general. Usually in trajectory optimization the airplane is described by a point mass model. The price to be paid for this simple and stable mathematical model is the reduced attitude and rotational information. These disadvantages are of minor importance in most commercial applications. Attitude information e.g. for a fuel-optimal long-distance flight across an ocean only has a negligible impact on the required objective function. The difference in the characteristic scales describing airplane geometry and the full flight trajectory are many orders of magnitude.

In air race applications the situation changes dramatically and makes it necessary to deal with full six degrees-of-freedom (6 DOF) models:

**Small differences in scales** For the optimization of fast and challenging maneuvers like turning and flying through a gate the relevant geometric scales are of the same order of magnitude as the airplane geometry.

**Course** Air race regulations require a prescribed attitude of the plane to be kept while passing a gate. For this reason the full attitude and rotational information has to be taken into account in the mathematical optimization procedure.

**Agility** An air race is characterized by a high and spectacular degree of agility of the participating airplane fully exploiting the airplane’s flight dynamic. This makes air races attractive to the spectators. At the same time the high sensitivity of those maneuvers is a challenge to numerical optimization and optimal control. Mathematical simulation and optimization can only compete with an excellent pilot in reality, if all degrees of freedom of a real airplane are incorporated into the mathematical description and sensitivity information is available with high accuracy.

**Constraints** On a considerable part of the race flight trajectory, technical or safety constraints are active. These constraints mostly require the exact information of the plane’s position and attitude. For the numerical treatment of realistic constraints, model derivatives of up to order 5 have to be evaluated.

But model complexity is only one part of the problem. The other is based on the numerical solution process: The advanced mathematical theory of optimal control makes use to a great extent of special structures in the model equations and of derivatives of order higher than 1. This is especially true in the presence of constraints. Derivatives necessary during the optimization process should make use of that underlying model structure to reduce the numerical effort.

Therefore, 6 DOF air race optimizations are an excellent application for the new algorithm developed here. The full model of an airplane is taken from [6].

The airplane is modeled as a dynamical system with 12 state variables \( \mathbf{x} \) and 4 control variables \( \mathbf{u} \)

\[
\mathbf{x} = \left( \begin{array}{cccccccc}
  x & y & z & v & \chi & \gamma & \alpha & \beta & \mu & p & q & r \\
  \text{position} & \text{speed} & \text{attitude} & \text{rotation}
\end{array} \right)^T \in \mathbb{R}^{12},
\]

\[
\mathbf{u} = \left( \begin{array}{cccc}
  \xi & \eta & \zeta & \delta \\
  \text{aileron} & \text{elevator} & \text{rudder} & \text{thrust}
\end{array} \right)^T \in \mathbb{R}^4.
\]

This results in a system of 12 ordinary differential equations

\[
\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \quad f : \mathbb{R}^{16} \mapsto \mathbb{R}^{12}.
\]

The \( f \) shows very distinct structure resulting from the underlying physics as well as from a mathematically appropriate rearrangement of the equations. However, the choice of the single blocks, layers and sub-layers is not unique and can to some extent be influenced by an experienced user.

In general, the full 6 DoF airplane model is split into its dynamic layers: position, translation, attitude and rotation. The layers are modularly formulated as sets of blocks, each representing a distinct effect.
(e.g. deflection of a rudder) or quantity (e.g. aerodynamic force). This results in a hierarchical model structure.

In the air race example, 13 blocks within a network hierarchy with 3 levels have been defined. Let \( B_f \) denote the block corresponding to the function \( f \). To find out all implicit dependencies of the function’s output from its inputs, the transfer matrix \( \langle B_f \rightarrow B_f \rangle \) has to be calculated. The algorithm from Sect. 2 yields the following structure

\[
\begin{pmatrix}
\dot{x} & \dot{y} & \dot{z} & \dot{v} & \dot{\chi} & \dot{\alpha} & \dot{\beta} & \dot{\mu} & \dot{p} & \dot{q} & \dot{r}
\end{pmatrix}
\]

with 90 non-vanishing entries out of 192. The fine structure of (16) gives a clear insight into the direct dependence of the components of \( f \) on the respective state and control variables. Execution time of the algorithm on a standard laptop computer is approximately \( 10^{-4} \) s.

For this example, derivatives of different order have been calculated. Tab. 1 and Fig. 2 summarize the results. For a given order \( k \) all derivatives up to this order are simultaneously calculated, i.e. \( f, Df, \ldots, D^{k-1}f, D^k f \). In Tab. 1 the resulting runtimes are compared to a naive calculation via finite differences. The results in milliseconds are logarithmically plotted in Fig. 2.

<table>
<thead>
<tr>
<th>order</th>
<th>evaluation</th>
<th>cascade</th>
<th>finite differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( f )</td>
<td>( 4.4 \cdot 10^{-5} s )</td>
<td>( 4.4 \cdot 10^{-5} s )</td>
</tr>
<tr>
<td>1</td>
<td>( f, Df )</td>
<td>( 1.5 \cdot 10^{-4} s )</td>
<td>( 169.4 \cdot 10^{-4} s )</td>
</tr>
<tr>
<td>2</td>
<td>( f, Df, D^2f )</td>
<td>( 1.2 \cdot 10^{-3} s )</td>
<td>( 232.4 \cdot 10^{-3} s )</td>
</tr>
<tr>
<td>3</td>
<td>( f, Df, D^2f, D^3f )</td>
<td>( 3.3 \cdot 10^{-2} s )</td>
<td>( 195.6 \cdot 10^{-2} s )</td>
</tr>
</tbody>
</table>

This comparison does not take into account the much lower accuracy of the finite difference approach.

**Acknowledgements**

This work has been founded in parts by the Exzellenzprogramm of the Technische Universität München. The generous financial support by the Red Bull AirRace GmbH is gratefully acknowledged.
Figure 2: Time for the evaluation of the function and its derivatives up to order $k$. Cascade algorithm compared to finite difference calculations.

References


