On the use of second-order derivatives and metamodel-based Monte-Carlo for uncertainty estimation in aerodynamics

Massimiliano Martinelli, Régis Duvigneau

On the use of second-order derivatives and metamodel-based Monte-Carlo for uncertainty estimation in aerodynamics

M. Martinelli* & R. Duvigneau**

INRIA Sophia-Antipolis Mediterranee
2004 route des lucioles, 06902 Sophia-Antipolis, France

Abstract

This article adresses the delicate issue of estimating physical uncertainties in aerodynamics. Usually, flow simulations are performed in a fully deterministic approach, although in real life operational uncertainty arises due to unpredictable factors that alter the flow conditions. In this article, we present and compare two methods to account for uncertainty in aerodynamic simulation. Firstly, automatic differentiation tools are used to estimate first- and second-order derivatives of aerodynamic coefficients with respect to uncertain variables, yielding an estimate of expectation and variance values (Method of Moments). Secondly, metamodelling techniques (radial basis functions, kriging) are employed in conjunction with Monte-Carlo simulations to derive statistical information. These methods are demonstrated for 3D Eulerian flows around the wing of a business aircraft at different regimes subject to uncertain Mach number and angle of attack.

Key words: Uncertainty, automatic differentiation, metamodels

*Tropics Project-Team
**Opale Project-Team

1. Introduction

Thanks to the high sophistication reached by Computational Fluid Dynamics (CFD) codes combined with the growth of computational facilities, research in the field of aerodynamic analysis and design has experienced a large development in the last years, allowing to deal with more and more complex problems. However, high fidelity models are usually used in deterministic approaches, which assume a perfect knowledge of the environmental and operational parameters. In real life, uncertainty can arise in many aspects of the entire design-production-operation process: from the assumptions done in the mathematical model describing the underlying physical phenomena, to the manufacturing tolerances, and to the operational parameters and conditions that could be affected by unpredictable factors (see [14] for a systematic treatment).

In particular, in aerodynamics there could be some operational uncertainties due to unpredictable factors that alter the flow conditions:

- the angle of attack may vary during the flight due to atmospheric conditions;
- instrumentation errors, or changes in flight profile compared to the scenario, or atmospheric conditions could result in uncertainty on Mach and Reynolds numbers.

It should be noted that geometric uncertainty, in addition to manifacturing tolerances, can arise during the working cycle also:

- temporary geometric variations, due to factors as icing or deformation under the weightloads;
- transient or permanent geometric variations due to the motion regime (e.g. blades in compressors);
- permanent geometric variations due to degradation (erosion, etc).

These uncertainties modify the aerodynamic coefficients and in some cases significantly degrade the performance of a system that has been optimized for some precise operational conditions. Therefore it would be beneficial to identify the most important uncertainties, quantify them and compute their impact on the performance. Then, statistical information, such as expectation and variance of an objective function, can be used in the design phase: this kind of approach is commonly known as *robust design* (see for instance [1] for a survey).

The practical difficulty with uncertainty quantification is related to the fact that the expected value μ_j and the variance σ_j^2 of the considered objective functional j result from an integration in the multi-dimensional space of the uncertain variables $\alpha \in \mathbb{R}^n$, i.e.

$$\mu_j = E\big[J(\alpha)\big] = \int j(\alpha) f(\alpha)\, d\alpha \tag{1}$$

$$\sigma_j^2 = E\big[(j(\alpha) - \mu_j)^2\big] = \int j(\alpha)^2 f(\alpha)\, d\alpha - \mu_j^2 \tag{2}$$

where $f(\alpha)$ is the Probability Density Function (PDF) for the uncertain variables.

Since the nonlinear equations governing the flow model are computationally expensive to solve (and thus the evaluation of the objective function j),

the use of a brute–force numerical integration method or Monte-Carlo analysis would become prohibitively expensive. Therefore, we need to introduce some new techniques that allow to obtain an approximate expectation value and variance of the functional. These techniques can be grouped in three main categories:

- Monte-Carlo estimates using surrogates ;
- Method of Moments (or Taylor series approximations);
- Polynomial Chaos Expansion.

The first approach consists in replacing the expensive simulation procedure by a cheap approximation to carry out the Monte-Carlo analysis. Usually, surrogate models are employed, such as response surface techniques or meta-models [7]. For the second approach, the objective functional is replaced by its Taylor series expansion from nominal conditions, yielding an analytical integration of statistical quantities [26, 16]. The latter approach is essentially different since it relies on a stochastic simulation framework. For Polynomial Chaos decomposition methods, the dimension of the problem is increased to account for uncertainty. More precisely, flow variables are expansed using a set of orthogonal basis functions, whose coefficients are used to characterize and quantify the uncertainty. By applying a Galerkin procedure, governing equations for each individual mode are derived. This approach has been applied for uncertainty quantification in several engineering fields. It is well suited to problems governed by PDEs and yields an accurate estimate of uncertainty. In particular, applications to fluid dynamics can be found in [27, 9](incompressible Navier-Stokes eq.) or [10] (compressible Euler eq.).

Nevertheless, this approach requires the development of a new simulation code accounting for the new dimensions. A non-intrusive formulation can be used, yielding however a much more expensive procedure.

Therefore, this work includes the comparison of the two former approaches: Monte-Carlo integration using metamodels and Method of Moments, for the evaluation of first and second-order statistical moments (i.e. mean and variance) for the drag coefficient of a transonic wing in a steady 3D Eulerian flow, where the Mach number and the angle of attack are subject to uncertainty.

2. Method of Moments

2.1. Principles

The idea behind the Method of Moments [16] (MoM) is based on the Taylor series expansion of the original nonlinear functional $j(\alpha)$ around the *mean value* of the uncertain variables. Then, the mean and variance of the output are computed by using the *moments* of the distribution for the input variables. In this way, we replace the (possibly) expensive direct numerical evaluation of the integrals (1)-(2) by the evaluation of the moments, that can often be computed by analytical integration.

Let us consider that the uncertain variables $\alpha = (\alpha_1, \ldots, \alpha_n)$ (either operational conditions or geometrical parameters) can be decomposed as: $\alpha = \mu_\alpha + \delta\alpha$ of a fully deterministic quantity $\mu_\alpha = E\big[\alpha\big]$ (the mean value of α) with a stochastic perturbation $\delta\alpha$. Then, the Taylor expansion of the functional $j(\alpha)$ around μ_α reads:

$$j(\alpha) = j(\mu_\alpha) + \sum_i G_i \delta\alpha_i + \frac{1}{2}\sum_{i,k} H_{i,k}\delta\alpha_i\delta\alpha_k + O(||\delta\alpha||^3), \qquad (3)$$

where $\delta\alpha_i = \alpha_i - \mu_{\alpha_i}$, $G_i = \frac{dj}{d\alpha_i}\big|_{\mu_\alpha}$ and $H_{i,k} = \frac{d^2 j}{d\alpha_i d\alpha_k}\big|_{\mu_\alpha}$.

When we compute the expectation value of (3), the term containinig the first order derivative disappears and the mean value of the functional $j(\alpha)$ is approximated by:

$$\mu_j \simeq j(\mu_\alpha) + \frac{1}{2}\sum_{i,k} C_{i,k} H_{i,k} \tag{4}$$

where $C_{i,k}$ is the (i,k)-element of the covariance matrix.

In the same way we can write a second-order approximation for the variance defined in (2):

$$\sigma_j^2 \simeq \sum_{i,k} C_{i,k} G_i G_k + \frac{1}{4}\sum_{i,k,l,m} \big(C_{i,l}C_{k,m} + C_{i,m}C_{k,l}\big) H_{i,k} H_{l,m}. \tag{5}$$

A more general formulation can be found in [1].

It is important to note that the equations for the mean (4) and the variance (5) require the gradient and the Hessian of the functional $j(\alpha)$, both evaluated at μ_α: for this reason the method above is commonly known as *second-order Method of Moments*.

Another important point concerns the accuracy of those approximations for the variance estimate: the use of all the information about the available derivatives (for a given derivative order) does not always result in an increased accuracy. Let us consider for simplicity the unidimensional case: the Taylor series expansion of the functional reads:

$$j(\alpha) = j(\mu_\alpha) + \sum_{k=1}^{\infty} \frac{j^{(k)}(\mu_\alpha)}{k!}\delta\alpha^k \qquad \text{with} \quad \delta\alpha = \alpha - \mu_\alpha$$

and its expectation is

$$\mu_j = E\big[j(\alpha)\big] = j(\mu_\alpha) + \sum_{k=2}^{\infty} \frac{j^{(k)}(\mu_\alpha)}{k!} m_k \qquad \text{with} \quad m_k = \int \delta\alpha^k f(\alpha)\, d\alpha.$$

With some algebra, we obtain the following expression for the variance:

$$\begin{aligned}\sigma_j^2 &= E\big[j^2\big] - \mu_j^2 \\ &= E\left[\left(j(\mu_\alpha) + \sum_{k=1}^{\infty} \frac{j^{(k)}(\mu_\alpha)}{k!}\delta\alpha^k\right)^2\right] - \left(j(\mu_\alpha) + \sum_{k=2}^{\infty} \frac{j^{(k)}(\mu_\alpha)}{k!}m_k\right)^2 \\ &= \sum_{i,k=1}^{\infty} \frac{j^{(i)}(\mu_\alpha)}{i!}\frac{j^{(k)}(\mu_\alpha)}{k!}(m_{i+k} - m_i m_k).\end{aligned}$$

If we consider Gaussian distributions for the uncertain variables, then the previous expression yields:

$$\sigma_j^2 = j'(\mu_\alpha)^2\sigma_\alpha^2 + \left(\frac{j''(\mu_\alpha)^2}{2} + j'(\mu_\alpha)j'''(\mu_\alpha)\right)\sigma_\alpha^4 + O(\sigma_\alpha^6). \tag{6}$$

Therefore, if we have only the derivatives up to order two, the error on the output variance is $O(\sigma_\alpha^4)$ and not $O(\sigma_\alpha^6)$. In particular, if $j'(\mu_\alpha)j'''(\mu_\alpha) \approx -\frac{j''(\mu_\alpha)^2}{2}$, we may have a better approximation of the variance using only the first-order derivative instead of first- and second-order.

Besides the difficulties pointed out in the previous comments, the most challenging task to apply the Method of Moments is the evaluation of the gradient and the Hessian of a functional constrained by a nonlinear equation (typically a set of PDEs). In the context of a discretize-than-differentiate approach, the most accurate and efficient method to obtain such derivatives (without the tedious and error-prone hand-coding differentiation) is given by using Automatic Differentiation.

2.2. Gradient and Hessian evaluation of constrained functionals

The computation of the Hessian matrix of a constrained functional could be performed using tools for Automatic Differentiation (TAPENADE [6],

OpenAD [23], TAF [4], ADOL-C [5]). Several strategies are possible [20, 21] relying on the two first-order differentiation modes usually implemented by the AD packages: tangent (or forward) mode and reverse (or backward) mode. These strategies for second-order derivatives (Tangent-on-Tangent and Tangent-on-Reverse/Reverse-on-Tangent) are theoretically equivalent but their computational cost depends on the number n of uncertain variables α_i for the Hessian matrix [20, 3]. In [12] it is shown that the CPU-time cost is quadratic in n for Tangent-on-Tangent (ToT) and linear for Tangent-on-Reverse (ToR) approach, but ToT is cheaper than ToR for a small number of uncertain variables. Since we are considering only two uncertain variables (Mach number and angle of attack), we use the ToT approach, that is described below.

Computation of first-order derivatives. Let j be a constrained functional expressed as:

$$j\colon \alpha \mapsto j(\alpha) = J(\alpha, W) \in \mathbb{R},$$

where $\alpha \in \mathbb{R}^n$ are parameters subject to random fluctuations and $W = W(\alpha) \in \mathbb{R}^N$ is the solution of the (nonlinear) state equation:

$$\Psi(\alpha, W) = 0.$$

Using the chain rule, the gradient of the functional with respect to each component of α is given by:

$$\frac{dj}{d\alpha_i} = \frac{\partial J}{\partial \alpha_i} + \frac{\partial J}{\partial W}\frac{dW}{d\alpha_i}. \tag{7}$$

The differentiation of the state equation reads:

$$\frac{\partial \Psi}{\partial \alpha_i} + \frac{\partial \Psi}{\partial W}\frac{dW}{d\alpha_i} = 0. \tag{8}$$

This equation yields the computation of the flow sensitivities $\theta_i = \frac{dW}{d\alpha_i}$ by solving the following linear system:

$$\frac{\partial \Psi}{\partial W}\theta_i = -\frac{\partial \Psi}{\partial \alpha_i}. \tag{9}$$

The first-order derivatives of j with respect to uncertain parameters α can be obtained by solving equation (9) to obtain the flow sensitivities first, and then by using (7). However, using such a method, we should solve one linear system for each uncertain parameter α_i. It is more efficient to adopt a so-called adjoint approach by combining equations (7) and (8). Then, we can easily obtain the gradient of j by solving the *adjoint system* first:

$$\left(\frac{\partial \Psi}{\partial W}\right)^T \Pi = \left(\frac{\partial J}{\partial W}\right)^T, \tag{10}$$

where Π are the adjoint variables, and then by computing:

$$\left(\frac{dj}{d\alpha}\right)^T = \left(\frac{\partial J}{\partial \alpha}\right)^T - \left(\frac{\partial \Psi}{\partial \alpha}\right)^T \Pi. \tag{11}$$

Using the adjoint approach, only one linear system must be solved, whatever the number of uncertain variables.

Computation of second-order derivatives. Starting from the first-order derivative (7), we perform another differentiation with respect to the k-th component of α, which reads:

$$\frac{d^2 j}{d\alpha_i d\alpha_k} = D^2_{i,k} J + \frac{\partial J}{\partial W}\frac{d^2 W}{d\alpha_i d\alpha_k}, \tag{12}$$

where we have introduced the differential operator $D^2_{i,k}$ acting on a function $F\colon (\alpha, W) \mapsto F(\alpha, W)$:

$$D^2_{i,k} F = \frac{\partial^2 F}{\partial \alpha_i \partial \alpha_k} + \frac{\partial^2 F}{\partial W \partial \alpha_i}\frac{dW}{d\alpha_k} + \frac{\partial^2 F}{\partial W \partial \alpha_k}\frac{dW}{d\alpha_i} + \frac{\partial}{\partial W}\left(\frac{\partial F}{\partial W}\frac{dW}{d\alpha_i}\right)\frac{dW}{d\alpha_k}. \tag{13}$$

Then, if we differentiate the equation (8) w.r.t. α_k and perform some algebric manipulation, we get:

$$\frac{d^2W}{d\alpha_i d\alpha_k} = -\left(\frac{\partial \Psi}{\partial W}\right)^{-1} D^2_{i,k}\Psi \ . \tag{14}$$

If we substitute the equation (14) in the equation (12), we finally obtain:

$$\begin{aligned}\frac{d^2j}{d\alpha_i d\alpha_k} &= D^2_{i,k}J - \frac{\partial J}{\partial W}\left(\frac{\partial \Psi}{\partial W}\right)^{-1} D^2_{i,k}\Psi \\ &= D^2_{i,k}J - \Pi^T\, D^2_{i,k}\Psi \ ,\end{aligned} \tag{15}$$

where Π is the solution of the adjoint system (10). This approach was firstly proposed in [20] and is usually known as *Tangent on Tangent* (ToT) approach or *Forward-on-Forward* ([3]) because we can compute the required second-order derivatives using AD with two successive *tangent-mode differentiations*.

Finally, the ToT algorithm for computing the gradient and the Hessian matrix of a functional $j(\alpha) = J(\alpha, W)$ constrained by $\Psi(\alpha, W) = 0$ writes as:

Solve for Π **in** $\left(\frac{\partial \Psi}{\partial W}\right)^T \Pi = \left(\frac{\partial J}{\partial W}\right)^T$
For each $i \in 1..n$
  Solve for θ_i **in** $\left(\frac{\partial \Psi}{\partial W}\right)\theta_i = -\left(\frac{\partial \Psi}{\partial \alpha_i}\right)$ **and store it**
End For
For each $i \in 1..n$
  For each $k \in 1..i$
    Compute $\frac{d^2j}{d\alpha_i d\alpha_k} = D^2_{i,k}J - \Pi^T(D^2_{i,k}\Psi)$
  End For
End For

In order to obtain the terms that appear in the algorithm above and containing the first- and second-order derivatives, we need a differentiated

version of the original CFD code. This differentiation, if performed "by hand", is tedious and error-prone. Then, we prefer to compute them using the automatic differentiation (AD) software TAPENADE [6], developed by Tropics Project-Team at INRIA. This software is used to generate automatically a source code that computes the derivatives of an original Fortran or C code. Implementation details are given in appendix.

3. Metamodel-based Monte-Carlo

3.1. Metamodeling techniques

The key idea of the metamodel-based approach is to approximate the functional $j(\alpha) = J(\alpha, W)$ with a function that depends only on the uncertain variables (in our case α) and whose evaluations are inexpensive. Metamodels are constructed according to available data that are stored in a database. It consists in using these data (e.g. drag computed for some parameter sets) to predict the fonctional for new parameters. Then, a Monte-Carlo analysis for uncertainty estimation is straightforward.

Metamodels mostly used for data fitting are:

- polynomial fitting (least-squares approximation) ;
- artificial neural networks (multi-layer perceptrons) [17] ;
- radial basis functions [15] ;
- Kriging methods (Gaussian process models) [19] .

The last three options are well suited to highly non-linear behaviors, such as those encountered in aerodynamics. In appendix, we describe the use of Radial Basis Functions (RBFs) and kriging methods.

The database is composed of a set of functional values corresponding to different uncertain parameter values. In practice, it is constructed by performing some independent simulations and storing the results. The choice of the database points is critical for the accuracy of the metamodel. If the distribution of the known functional values does not explore the parameter space uniformly, the metamodel predictions will be of poor accuracy, since the database building is done without any *a priori* knowledge about the functional behavior. Several methods can be found in the literature that describe how to choose the database points [8]. In the present study, the database points are generated using a Latin Hypercube Sampling [8] (LHS) of size N. Latin hypercube samples have the property that any projection along one parameter direction yields a uniform distribution. We also add systematically in the database the points that correspond to the corners of the parameters variation domain.

3.2. Uncertainty estimation using Monte-Carlo simulation

Using the metamodels described in appendix, one can construct an inexpensive model $\hat{j}(\alpha)$ of the functional with respect to the uncertain parameters, on the basis of a few CFD evaluations that correspond to different parameter values. Then, the mean μ_J and the variance σ_J^2 can be easily estimated using Monte-Carlo simulations on the basis of the metamodel approximation:

$$\mu_J \approx \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \hat{j}(\alpha^i) \tag{16}$$

$$\sigma_J^2 \approx \frac{1}{N_{MC}-1} \sum_{i=1}^{N_{MC}} (\hat{j}(\alpha^i) - \mu_J)^2 \ , \tag{17}$$

where N_{MC} is the size of the sample considered for the Monte-Carlo simulation. The sample used for the analyses $(\alpha^i)_{i=1,\dots,N_{MC}}$ should be generated in accordance with the PDF for the uncertain parameters. One should underline that samples of very large size (e.g. larger than 100,000) must be used in order to have an accurate estimation of the variance. Therefore, one should select a random number generator with care, in order to avoid some periodicity effects. In the present study, the samples are generated using the Mersenne Twister algorithm [13].

4. Application to aerodynamics

4.1. Flow solver description

Modeling. This study is restricted to three-dimensional inviscid compressible flows governed by the Euler equations. Then, the state equations can be written in the conservative form:

$$\frac{\partial W}{\partial t} + \frac{\partial F_1(W)}{\partial x} + \frac{\partial F_2(W)}{\partial y} + \frac{\partial F_3(W)}{\partial z} = 0, \tag{18}$$

where W are the conservative flow variables $(\rho, \rho u, \rho v, \rho w, E)$, with ρ the density, (u, v, w) the velocity components and E the total energy per unit of volume. $\overrightarrow{F} = (F_1(W), F_2(W), F_3(W))$ is the vector of the convective fluxes, whose components are given by:

$$F_1(W) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E+p) \end{pmatrix} \quad F_2(W) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v(E+p) \end{pmatrix} \quad F_3(W) = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(E+p) \end{pmatrix}. \tag{19}$$

The pressure p is obtained from the perfect gas state equation.

Spatial discretization. Provided that the flow domain Ω is discretized by a tetrahedrization $\mathcal{T}_h$, a discretization of equation (18) at the mesh node s_i is obtained by integrating (18) over the volume C_i, that is built around the node s_i by joining barycenters of the tetrahedra and triangles containing s_i and midpoints of the edges adjacent to s_i:

$$Vol_i \frac{\partial W_i}{\partial t} + \sum_{j \in N(i)} \Phi(W_i, W_j, \overrightarrow{\sigma}_{ij}) = 0, \tag{20}$$

where W_i represents the cell averaged state and Vol_i the volume of the cell C_i. $N(i)$ is the set of the neighboring nodes. $\Phi(W_i, W_j, \overrightarrow{\sigma}_{ij})$ is an approximation of the integral of the fluxes (19) over the boundary ∂C_{ij} between C_i and C_j, which depends on W_i, W_j and $\overrightarrow{\sigma}_{ij}$ the integral of a unit normal vector over ∂C_{ij}. These numerical fluxes are evaluated using upwinding, according to the approximate Riemann solver of Roe [22].

A high order scheme is obtained by interpolating linearly the physical variables from s_i to the midpoint of $[s_i s_j]$ to evaluate the fluxes. Nodal gradients are obtained from a weighting average of the P1 Galerkin gradients computed on each tetrahedron containing s_i. In order to avoid spurious oscillations of the solution in the vicinity of the shock, a slope limitation procedure is introduced using a modified version of the Van-Leer limiter [25]:

$$L(a,b) = \begin{cases} \dfrac{a^2 b + a b^2}{a^2 + b^2} & \text{if } ab > 0 \\ 0 & \text{if } ab \leq 0 \end{cases}. \tag{21}$$

Time integration. A first order implicit backward scheme is employed for the pseudo-time integration of (20) to the steady state. The linearization of the

numerical fluxes provides the following integration scheme :

$$\left(\frac{Vol_i}{\Delta t} + J_i^n\right)\delta W_i = -\sum_{j\in N(i)} \Phi(W_i^n, W_j^n, \overrightarrow{\sigma}_{ij}). \tag{22}$$

with $\delta W_i = W_i^{n+1} - W_i^n$ and J_i^n the Jacobian matrix of the first order numerical fluxes.

4.2. Testcase description

The testcases considered here correspond to the flows around the wing of a business aircraft for three regimes : a subsonic, a supersonic and a transonic regime. The nominal operating conditions are defined by the free-stream Mach number $M_\infty = \{0.65, 0.95, 0.83\}$ and the incidence $\alpha = 2°$. The wing section is supposed to correspond to the NACA 0012 airfoil. The wing shape and the mesh in the symmetry plane are depicted in (1). The mesh employed counts 31124 nodes.

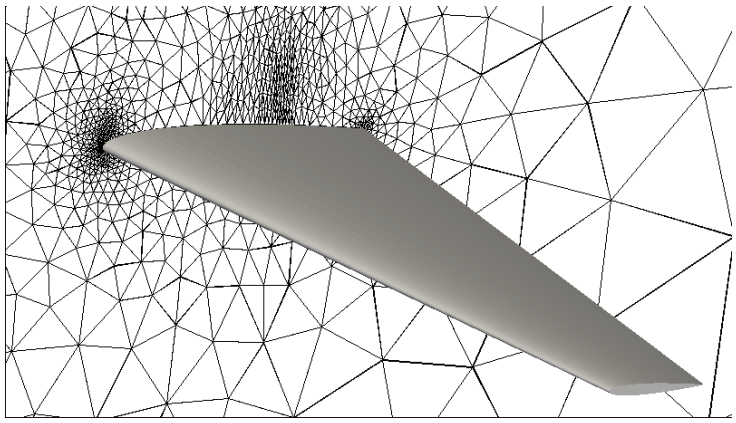

Figure 1: Wing shape and mesh in the symmetry plane.

We suppose that the free-stream Mach number and the angle of attack are subject to random fluctuations. For the sake of simplicity, we assume that their PDFs are Gaussian and uncorrelated. They are characterized by:

	Mach	Incidence (deg.)
Mean	$\{0.65, 0.95, 0.83\}$	2
Standard deviation	0.01666	0.1666

We aim at using the methods presented above to estimate the statistics on the drag.

4.3. Testcase 1: subsonic flow ($M_\infty = 0.65$)

4.3.1. Reference results

We compute first some reference results, obtained by performing 21×21 CFD analyses as seen in figure (2). Reference statistical values are computed

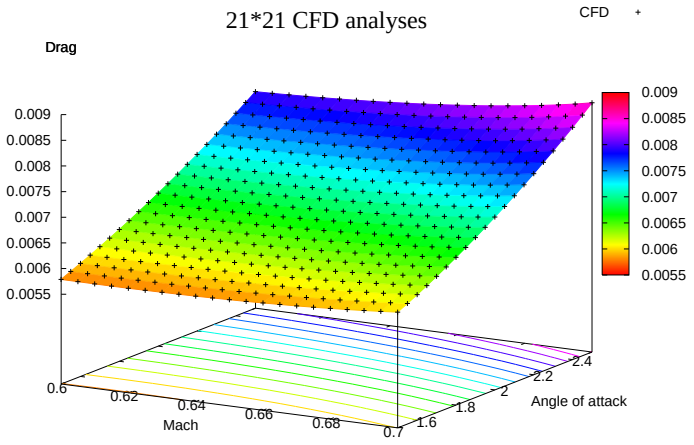

Figure 2: Drag for 21×21 CFD analyses around the nominal conditions $M_\infty = 0.65$, $\alpha = 2°$.

by constructing a fine metamodel based on these 21×21 points and performing a Monte-Carlo analysis. The following reference values for the mean and variance of the drag coefficient are obtained:

Reference expectation	$\mu_j = 6.857\,10^{-3}$
Reference variance	$\sigma_j^2 = 1.553\,10^{-7}$

We have verified that these values do not depend on the choice of the metamodel (either RBFs or Kriging), and are not modified if extra points are added or if larger samples are used for the Monte-Carlo simulation.

4.3.2. Results with metamodels

We construct RBF and kriging metamodels for databases generated by a latin hypercube sampling of size 8 and 23. Results, in terms of mean and variance estimates are provided in the next table:

Metamodel	Points	Expectation	Relative error	Variance	Relative error
RBF	8	$6.855\,10^{-3}$	0.029%	$1.562\,10^{-7}$	0.579%
RBF	23	$6.858\,10^{-3}$	0.014%	$1.566\,10^{-7}$	0.837%
KRG	8	$6.853\,10^{-3}$	0.058%	$1.551\,10^{-7}$	0.128%
KRG	23	$6.858\,10^{-3}$	0.014%	$1.566\,10^{-7}$	0.837%

One can observe that the variance estimate is slightly better using 8 points that 23 points. This is due to the particular choice of the databases. Figure (3) depicts the drag evolution with respect to the uncertain parameters obtained for a RBF metamodel with 8 points, as well as the error computed at 21×21 points. As seen, the error on the drag is less than 0.5%.

4.3.3. Results with AD

AD is then used to estimate drag statistics. Contrary to the previous case, the flow is only computed at the mean value of the uncertain parameters, as

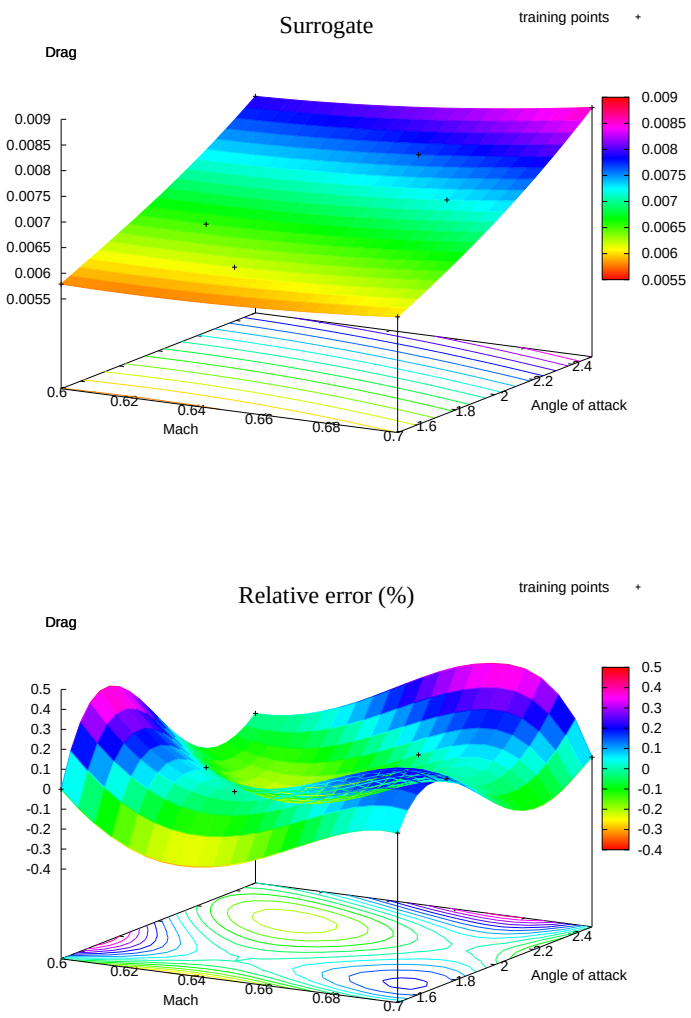

Figure 3: Drag and relative error in % obtained using a RBF metamodel with 8 points (LHS sampling) around the nominal conditions $M_0 = 0.65$, $\alpha = 2°$.

well as the derivatives. The statistics obtained using first- and second-order Taylor series are given in the next table:

	Expectation	Expectation error	Variance	Variance error
First-order	$6.830\,10^{-3}$	0.393%	$1.547\,10^{-7}$	0.386%
Second-order	$6.860\,10^{-3}$	0.043%	$1.558\,10^{-7}$	0.321%

The accuracy of the results is similar to the one obtained with metamodels using 8 points. This is not surprising, since a quadratic model can be obtained using six points at least. Figure (4) shows the drag evolution with respect to the uncertain parameters obtained using AD, as well as the error computed at 21×21 points. One can observe that the error is larger at the corners of the variation domain. However, this is not critical for statistics estimation, since the PDFs of uncertain parameters become smaller and smaller as one moves away from nominal operational conditions.

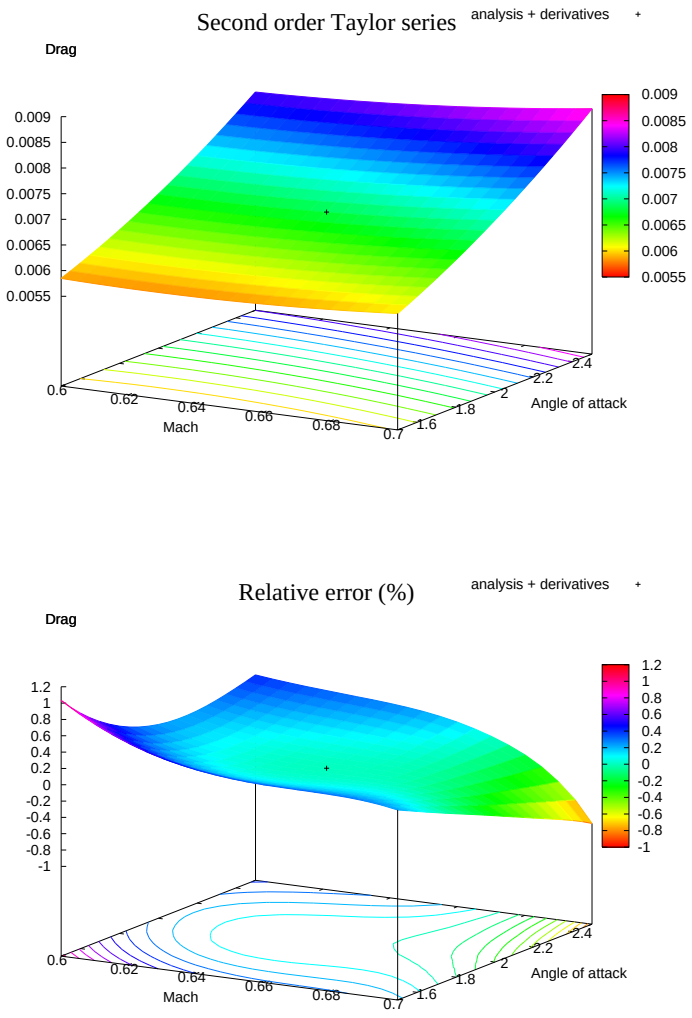

Figure 4: Drag and relative error in % obtained using AD (second-order) around the nominal conditions $M_0 = 0.65$, $\alpha = 2°$.

4.3.4. Comparison of computational performance

Since the two proposed methods are essentially different, it is interesting to compare also their computational performance, in terms of CPU time and memory requirements. The following table details the memory and the CPU-time used by the AD-based approach (using a machine equipped with an Intel Xeon 2.66 GHz):

		Memory in Mb
Flow-solver	Jacobian + Precond. + vectors	86+10+35
First deriv.	ILU(1) Precond. + GMRES + vectors	170+250+120
Second deriv.	ILU(1) Precond. + GMRES + vectors	170+250+240

	CPU time in second
Flow solver	403
Gradient	255
Hessian	278
Total	936

The iterative approach used to solve the linear systems (10) and (9), requires the application of a good preconditionner: for this purpose we use the ILU(1) factorization of the (available) first-order accurate Jacobian $\left(\frac{\partial \Psi}{\partial W}\right)$ for the linear system (9) and the ILU(1) factorization of $\left(\frac{\partial \Psi}{\partial W}\right)^T$ for the linear system (10). As shown above, the CPU-time for the gradient and Hessian evaluation is lower than that for the flow solution (in which a relative reduction of residuals to 10^{-11} was set as convergence condition). This cost is essentially due to the resolution of the linear systems (10) and (9), while the memory

requirement is notably higher.

Concerning the method based on metamodels, the costs are mainly related to the construction of the database. If it is built sequentially, the memory required is the same as that used by the flow solver alone, whereas the CPU-time increases linearly. If it is built using parallel computing, with a number of processors equal to the database size, the CPU-time remains more or less similar to that of a single flow solver run. For instance, we obtain for a database with 8 points:

	CPU-time in second
Sequential database	3250
Parallel database	440

In conclusion, the method based on metamodels is usually more expensive in terms of CPU-time with respect to the MoM, but it is easier to implement and memory requirements are lower.

4.4. Testcase 2: fish-tail transonic flow ($M_\infty = 0.95$)

Following the same approach, we compute first some reference results, obtained by performing 21×21 CFD analyses as seen in figure (5):

Reference expectation	$\mu_j = 9.561\,10^{-2}$
Reference variance	$\sigma_j^2 = 3.982\,10^{-5}$

4.4.1. Results with metamodels

We present here results obtained using a Latin Hypercube Sampling that includes 8, 13 and 23 points. One can observe that results with 8 points are less accurate than that in subsonic regime, due to the more complex drag

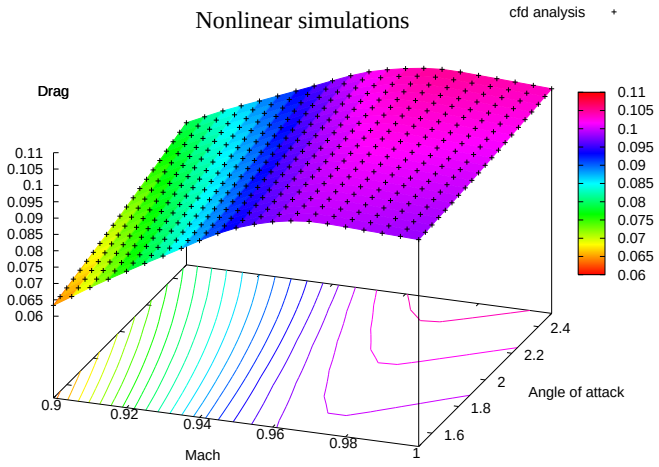

Figure 5: Drag for 21×21 CFD analyses around the nominal conditions $M_\infty = 0.95$, $\alpha = 2°$.

behavior, as can be seen in figure (6). Satisfactory results can be obtained with 13 points. For this regime, Kriging seems to be more accurate than RBFs.

Metamodel	Points	Expectation	Relative error	Variance	Relative error
RBF	8	$9.524\,10^{-2}$	0.386%	$3.575\,10^{-5}$	10.220%
RBF	13	$9.567\,10^{-2}$	0.062%	$3.913\,10^{-5}$	1.732%
RBF	23	$9.566\,10^{-2}$	0.052%	$3.939\,10^{-5}$	1.079%
KRG	8	$9.573\,10^{-2}$	0.125%	$3.619\,10^{-5}$	9.116%
KRG	13	$9.562\,10^{-2}$	0.010%	$3.956\,10^{-5}$	0.652%
KRG	23	$9.563\,10^{-2}$	0.020%	$3.977\,10^{-5}$	0.125%

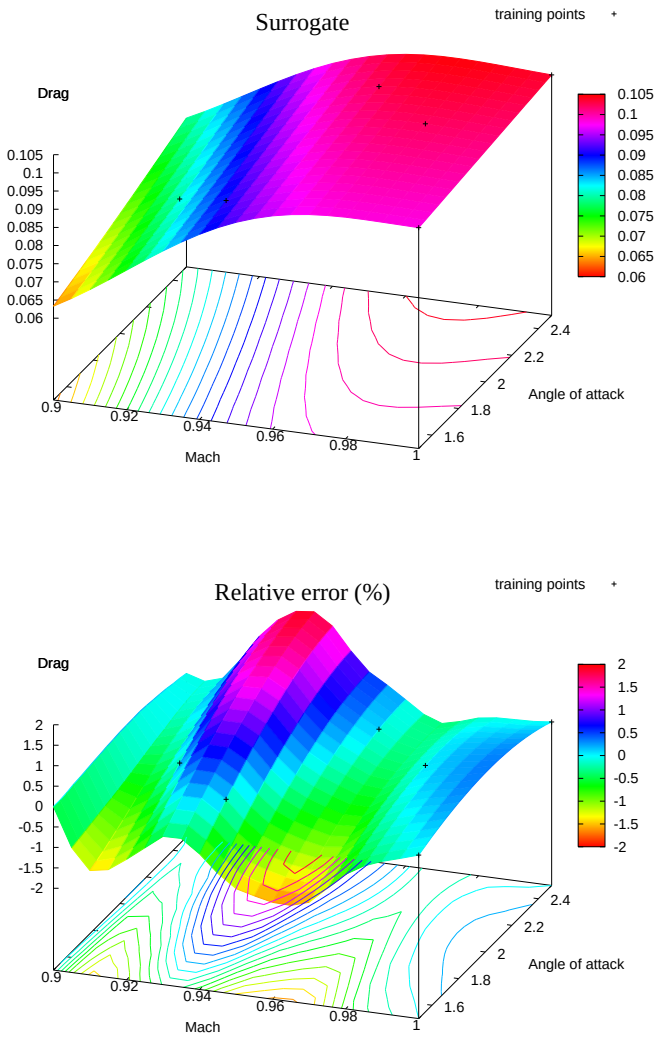

Figure 6: Drag and relative error in % obtained using a RBF metamodel with 8 points (LHS sampling) around the nominal conditions $M_0 = 0.95$, $\alpha = 2°$.

4.4.2. Results with AD

For this regime the second-order Taylor approximation seems to be accurate for the interval $M = [0.92, 0.98]$ while outside it underestimates the

real values of the drag, as can be seen in figure (7). This results in a lower accuracy for the mean and variance.

	Expectation	Relative error	Variance	Relative error
First-order	$9.747\,10^{-2}$	1.945%	$4.038\,10^{-5}$	1.406%
Second-order	$9.545\,10^{-2}$	0.167%	$4.895\,10^{-5}$	22.928%

One can observe that the variance estimate is far better using the first-order approximation than using the second-order one. This can be explained by examining the error in the variance estimate (6), as underlined in Section 2.1. The over-estimate of the variance computed with the second-order Method of Moments is due to the fact that we have neglected in the correct formula (i.e., for the one-dimensional case, (6)) the term involving the third-order derivative. This term can have any sign, while the terms involving first- and second-order derivatives are always positive, resulting in a possible over-estimate for the variance computed with second-order method with respect to the estimate obtained with the first-order method. Therefore, for the general case and without any information about the behaviour of the functional (in terms of signs of derivatives), we advocate the use of the second-order Method of Moments for the estimation of the mean and the first-order Method of Moments for the estimation of the variance.

4.5. Testcase 3: transonic flow ($M = 0.83$)

AD can be faced to another difficulty, arising from non-differentiable programs. For instance, if one considers the previous problem in transonic regime, one observes a noisy gradient computation, although the drag seems to be smooth, as we can see in figure (8).

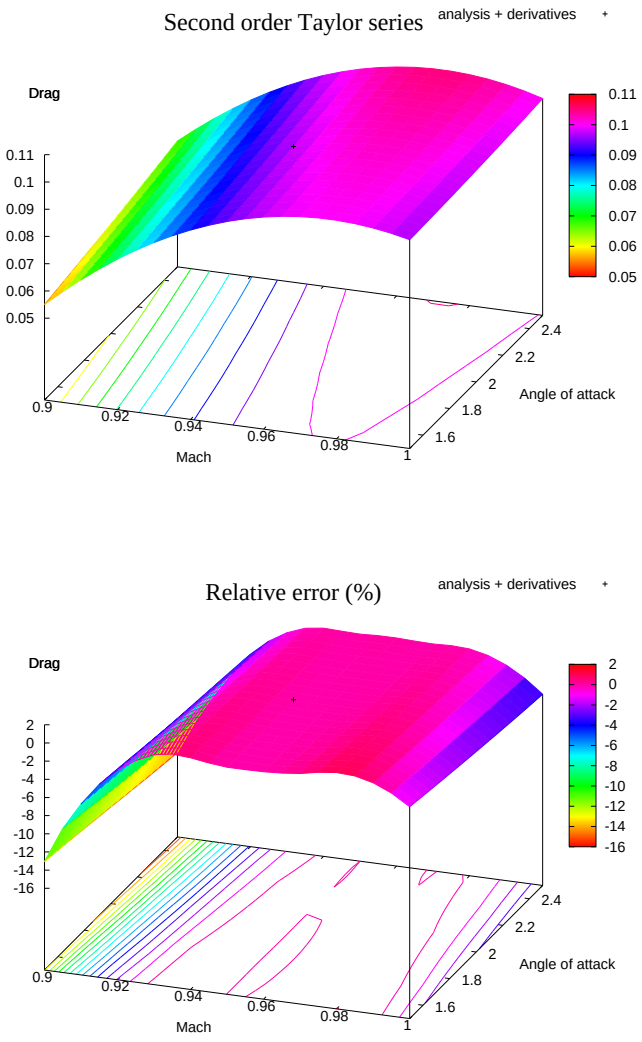

Figure 7: Drag and relative error in % obtained using AD (second-order) around the nominal conditions $M_0 = 0.95$, $\alpha = 2°$.

This behaviour is due to the non-global differentiability of the CFD solver. This may be due to the definition of the Roe's scheme and to the slope limiter

(21), that are not globally differentiable but only piecewise-differentiable: as result the Taylor approximation is acceptable only for tiny intervals. As we can see on the second plot of figure (9), the first-derivative of the drag coefficient with respect to the Mach number is piecewise-continuous and the size of the interval in which the continuity (and therefore the differentiability of the drag with respect to the Mach number) can be assumed is $\simeq 10^{-8}$. It is important to understand that this is not a situation in which AD fails, but a situation in which the approximation of the function by its Taylor expansion is not appropriate. The correctness of the derivatives obtained by AD is validated by comparison with divided differences: we can also see how accurate is the second-order Taylor approximation of the drag coefficient (around $M = 0.83$) in figure (8). Moreover, in figure (9) we can see that the second-order approximation is a nearly perfect description of the first-order derivative in a neighborood of $M = 0.83$: for that regime the drag coefficient can be considered as a piecewise-quadratic function.

To understand if the problem is caused by the non-differentiability of the Roe's scheme or the slope limiter, we perform some extra experiments. For the first experiment, we replace the Roe scheme with the Van Leer scheme [24] (that is differentiable). This yields no significative change. Then, we solve the flow in the usual way (Roe + slope limiter) and then we switch off the limiter for the gradient and the Hessian computation. The resulting Taylor approximation is shown in figure (10) (dashed line). We note that the first-order derivative is under-estimated. Surprisingly, if we use the first-order derivative computed with the slope limiter active and the second-order derivative with the slope limiter inactive we obtain a pretty good approxima-

tion of the drag coefficient over a large interval of Mach Number (dotted line in figure (10)). From the above experiments, we guess that the piecewise-differentiability of the slope limiter is the cause of the very irregular derivative of the drag coefficient in transonic regime (for which a shock is present on the wing surface).

Finally, one can underline that a Navier-Stokes model instead of the Euler model could help to smooth the behaviour of the first-order derivative, but this hypothesis needs further investigations to be confirmed.

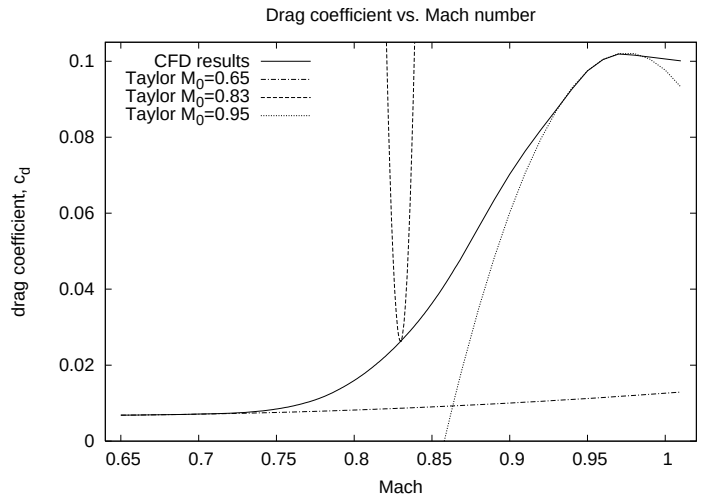

Figure 8: Drag vs. Mach number in transonic regime: CFD simulations and second-order Taylor approximations around $M = 0.65$ (dash-dot), $M = 0.83$ (dash) and $M = 0.95$ (dot) using AD.

5. Conclusion

In this study, we have presented and compared two methods to estimate aerodynamic uncertainty for practical testcases: automatic differentiation

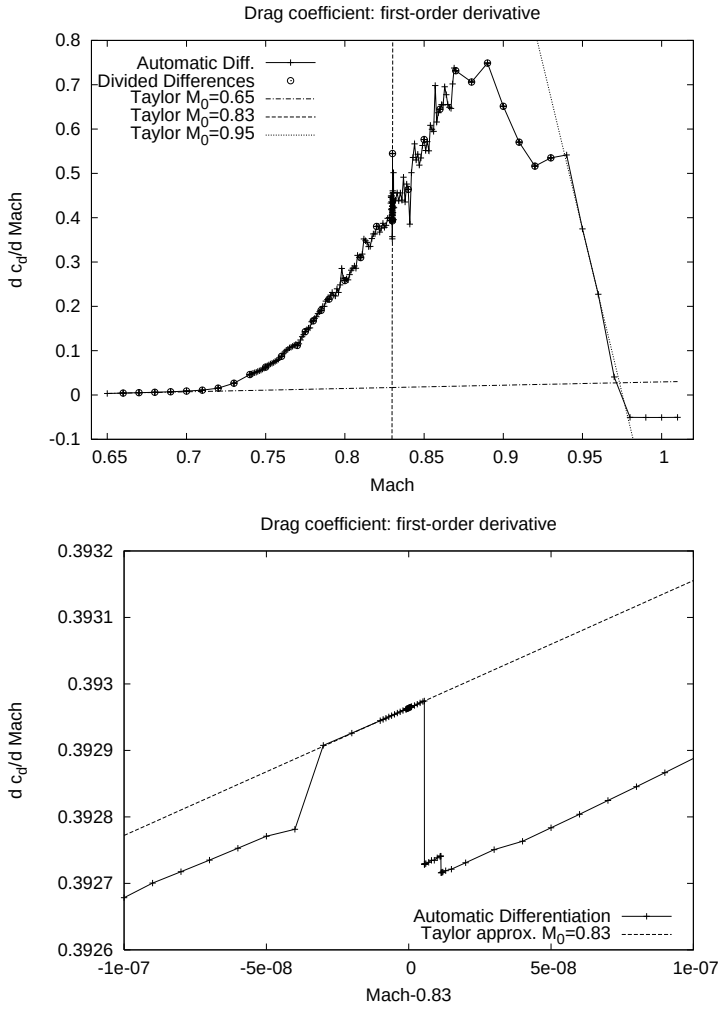

Figure 9: Drag derivatives computed with AD w.r.t. Mach number in transonic regime. In the first plot is shown the fist-order Taylor approximation of the drag derivative for three different Mach numbers. On the second plot, a zoom of the region around $M = 0.83$ is presented. Note the piecewise-linear behaviour of the derivative in the transonic regime. The dotted line is the first-order Taylor approximation of the derivative around $M = 0.83$, obtained using AD.

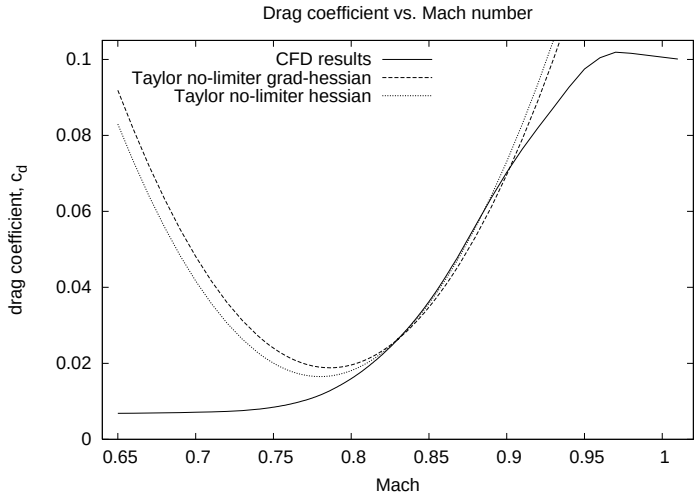

Figure 10: Drag vs. Mach number in transonic regime: second-order approximations using derivatives relative to different slope-limiter strategies. Dashed line: the first and second-order derivatives are computed with the slope limiter inactive in the differentiated code. Dotted line: the first-order derivative is computed with the slope limiter active while the second-order derivative is computed with the slope limiter inactive.

has been applied to a 3D Eulerian flow solver to compute first and second derivatives of the drag with respect to uncertain parameters, in the framework of the Method of Moments. Kriging and radial basis functions metamodels have been used in conjunction with Monte-Carlo simulations to estimate statistics.

Both approaches have their strenght points: from the pure performance point of view, the Method of Moments is generally faster than metamodels and this fact will appear more evident when a larger number of uncertain variables will be considered. In fact, MoM requires only one single CFD simulation and the cost for the gradient and Hessian evaluation grows linearly with the number of uncertain variables, while metamodels, being non-

local approaches, require a larger number of CFD evaluations to build the database. However they can yield a more accurate estimate for the statistical quantities.

The Method of Moments requires first- and second-order derivatives of PDE-constrained functionals, and this task, even if accomplished with the help of Automatic Differentation tools, is not so straightforward. Moreover, the user must have the source code of the CFD solver available. Conversely, metamodels use CFD solvers as *black boxes* and the construction of the response surface could be done in a preprocessing phase.

Satisfactory results in terms of accuracy have been obtained for a simple testcase in subsonic and fish-tail transonic case, while Taylor-based approximations seem to be not suitable for the transonic case using our discretization with high-order accurate schemes and limiters.

Now, the main issue would be the application of these methodologies to problems with a more complex flow modelling, such as Navier-Stokes equations, in which a larger number of uncertain parameters is required (e.g. geometrical uncertainties).

Acknowledgments

This work was partially supported by the project NODESIM-CFD "Non-Deterministic Simulation for CFD-based Design Methodologies" funded by the European Community represented by the CEC, Research Directorate-General, in the 6th Framework Programme, under Contract No. AST5-CT-2006-030959.

A. Automatic differentiation

Consider a program that computes an output vector $v \in \mathbb{R}^m$ from an input vector $u \in \mathbb{R}^n$ as a function $v = \phi(u)$. The derivative of the function is provided by the Jacobian matrix $\frac{\partial \phi}{\partial u}$. The program is a sequence of elementary instructions that can be identified with a composition of elementary functions. The AD tool simply applies the chain rule to differentiate these elementary functions to obtain the desired Jacobian matrix. However, we are usually not interested by the knowledge of the full Jacobian matrix. Then, TAPENADE has two differentiation modes, that allow to compute the product of the Jacobian matrix by a given vector. We can perform this matrix-by-vector product in a twofold manner: by right (*tangent mode*) or by left (*reverse mode*):

- the *tangent mode* allows to compute, from an arbitrary direction $\dot{u} \in \mathbb{R}^n$, the derivative in the direction $\dot{u}$:

$$u, \dot{u} \mapsto \left.\frac{\partial \phi}{\partial u}\right|_u \dot{u} \ ,$$

- the *reverse mode* allows to compute, from an arbitrary direction $\bar{\phi} \in \mathbb{R}^m$, the following product:

$$u, \bar{\phi} \mapsto \left(\left.\frac{\partial \phi}{\partial u}\right|_u\right)^T \bar{\phi} \ .$$

  Note that if $\phi(u) \in \mathbb{R}$ (i.e. a functional), reverse mode differentiation gives the *gradient* of ϕ w.r.t. u (multiplied by the input scalar $\bar{\phi}$).

These two modes can be employed to easily compute the terms that are required for derivatives estimation.

Consider that the evaluation of functional $j = J(\alpha, W)$ is implemented by a Fortran subroutine `func`, whose input variables are `alpha` and `w` and output variable is `j`:

$$\underset{J}{\downarrow}\texttt{func}(\texttt{j}, \overset{\alpha}{\underset{\downarrow}{}}\texttt{alpha}, \overset{W}{\underset{\downarrow}{}}\texttt{w}).$$

If we perform a reverse mode differentiation with respect to the input variables `alpha` and `w`, we obtain a new subroutine:

$$\underset{J}{\downarrow}\texttt{func\_b}(\texttt{j}, \overset{\bar{J}}{\underset{\downarrow}{}}\texttt{jb}, \overset{\alpha}{\underset{\downarrow}{}}\texttt{alpha}, \underset{\bar{\alpha}}{\downarrow}\texttt{alphab}, \overset{W}{\underset{\downarrow}{}}\texttt{w}, \underset{\bar{W}}{\downarrow}\texttt{wb}),$$

where `jb` is a new input variable and `alphab` and `wb` are new output variables defined as:

$$\bar{\alpha} = \left(\frac{\partial J}{\partial \alpha}\right)^T \bar{J} \qquad \bar{W} = \left(\frac{\partial J}{\partial W}\right)^T \bar{J}. \tag{23}$$

If we evaluate the above subroutine with the input $\bar{J} = 1$, the quantities in (23) are the first term in the right hand side (r.h.s.) of (11) and the r.h.s. of the adjoint equation (10).

To compute the terms required to estimate the second-order derivatives, we need to perform two successive tangent-mode differentiations. For example, considering the tangent-mode differentiation of the subroutine `func` with respect to `alpha` and `w` we obtain:

$$\underset{J}{\downarrow}\underset{\dot{j}}{\downarrow}\texttt{func\_d}(\texttt{j}, \texttt{jd}, \overset{\alpha}{\underset{\downarrow}{}}\texttt{alpha}, \overset{\dot{\alpha}}{\underset{\downarrow}{}}\texttt{alphad}, \overset{W}{\underset{\downarrow}{}}\texttt{w}, \overset{\dot{W}}{\underset{\downarrow}{}}\texttt{wd}),$$

where `alphad` and `wd` are new input variables and `jd` a new output variable. These input variables are provided by the user, whereas the output variable

is the directional derivative:

$$\dot{J} = \frac{\partial J}{\partial \alpha}\dot{\alpha} + \frac{\partial J}{\partial W}\dot{W}.$$

Then, the differentiation of the output variable `jd` in the subroutine `func_d` with respect to input variables `alpha` and `w` gives:

$$\underset{\downarrow\, J \;\downarrow\, \dot{J} \quad \downarrow\, \dot{\dot{J}}}{\texttt{func\_dd}(\texttt{j}, \texttt{jd}, \texttt{jdd}, \overset{\alpha}{\underset{\downarrow}{\texttt{alpha}}}, \overset{\dot{\alpha}_0}{\underset{\downarrow}{\texttt{alphad0}}}, \overset{\dot{\alpha}}{\underset{\downarrow}{\texttt{alphad}}}, \overset{W}{\underset{\downarrow}{\texttt{w}}}, \overset{\dot{W}_0}{\underset{\downarrow}{\texttt{wd0}}}, \overset{\dot{W}}{\underset{\downarrow}{\texttt{wd}}})},$$

where `jdd` is the new output variable, that represents:

$$\begin{aligned}\dot{\dot{J}} &= \frac{\partial}{\partial \alpha}\left(\frac{\partial J}{\partial \alpha}\dot{\alpha} + \frac{\partial J}{\partial W}\dot{W}\right)\dot{\alpha}_0 + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial \alpha}\dot{\alpha} + \frac{\partial J}{\partial W}\dot{W}\right)\dot{W}_0 \\ &= \frac{\partial}{\partial \alpha}\left(\frac{\partial J}{\partial \alpha}\dot{\alpha}\right)\dot{\alpha}_0 + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial \alpha}\dot{\alpha}_0\right)\dot{W} + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial \alpha}\dot{\alpha}\right)\dot{W}_0 + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial W}\dot{W}\right)\dot{W}_0\ ,\end{aligned}$$

where we have used the commutativity of the order of differentiation.

If one calls this subroutine with the following input parameters:

$$\dot{\alpha}_0 = e_k \quad \dot{\alpha} = e_i \quad \dot{W}_0 = \frac{dW}{d\alpha_k} \quad \dot{W} = \frac{dW}{d\alpha_i}, \tag{24}$$

one obtains as output variable $\dot{\dot{J}} = D^2_{i,k}J$ as defined in (13).

B. Radial Basis Functions

Radial Basis Functions [15] (RBF) are a non-polynomial interpolation method, that have been found to be very accurate for highly non-linear data in high dimension [2]. RBFs seek an approximation of the function $j(\alpha)$, $\alpha \in \mathbb{R}^n$ of the form:

$$\hat{j}(\alpha) = \sum_{i=1}^{N} \omega_i \phi_i(\alpha) \tag{25}$$

where:

$$\phi_i(\alpha) = \Phi(||\alpha - \alpha^i||) \tag{26}$$

$(\alpha^i)_{i=1,\ldots,N}$ are called RBFs centers and correspond to the points stored in a database. Several radial functions Φ can be considered. For the present study a Gaussian function is employed:

$$\Phi(r) = e^{-\frac{r^2}{s^2}}, \tag{27}$$

where s is a parameter called attenuation factor, that controls the extend of the basis functions.

The training of the RBFs consists in determining the weights $(\omega_i)_{i=1,\ldots,N}$ to fit the data. Suppose that the function value is known for a set of N points that correspond to the RBF centers $(\alpha^k)_{k=1,\ldots,N}$. Then, the weights $(\omega_i)_{i=1,\ldots,N}$ are determined from the interpolation conditions:

$$j(\alpha^k) = \hat{j}(\alpha^k) = \sum_{i=1}^{N} \omega_i \ \phi_i(\alpha^k) \quad k = 1, \ldots, N. \tag{28}$$

Thus, $(\omega_i)_{i=1,\ldots,N}$ is the solution of the following linear system:

$$\begin{pmatrix} \phi_1(\alpha^1) & \ldots & \phi_N(\alpha^1) \\ \phi_1(\alpha^2) & \ldots & \phi_N(\alpha^2) \\ \ldots & \ldots & \ldots \\ \phi_1(\alpha^N) & \ldots & \phi_N(\alpha^N) \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \ldots \\ \omega_N \end{pmatrix} = \begin{pmatrix} j(\alpha^1) \\ j(\alpha^2) \\ \ldots \\ j(\alpha^N) \end{pmatrix}. \tag{29}$$

The matrix of the system is obviously symmetric. It is also positive-definite if the RBF centers $(\alpha^i)_{i=1,\ldots,N}$ are distinct. According to Ref. [18], we employ the *leave-one-out* technique to determine a suitable attenuation coefficient.

B.1. Kriging

Gaussian process models [11, 19] (also called *kriging*) treat the response of some experiment as if it were a realization of a stochastic process.

The vector of known function values $J_N = (j(\alpha^i))_{i=1,\ldots,N}$ is assumed to be one realization of a multivariate Gaussian process with joint probability density:

$$p(J_N) = \frac{\exp\left(-\frac{1}{2} {J_N}^T C_N^{-1} J_N\right)}{\sqrt{(2\pi)^N \det(C_N)}} \tag{30}$$

where C_N is the $N \times N$ covariance matrix. The element C_{ik} of the covariance matrix gives the correlation between the function values $j^i = j(\alpha^i)$ and $j^k = j(\alpha^k)$ obtained for different points. We assume that these values are correlated, since they correspond to underlying physical phenomena. This is expressed in terms of a covariance function c, i.e. $C_{ik} = c(\alpha^i, \alpha^k)$.

Now, we suppose that we would like to evaluate the functional value at a new point α^{N+1}. It can be shown that the probability density for the function value at the new point is:

$$p(j^{N+1}|J_N) \propto \exp\left[-\frac{(j^{N+1} - \hat{j}^{N+1})^2}{2\sigma^2_{j^{N+1}}}\right]$$

where:

$$\hat{j}^{N+1} = k^T C_N^{-1} J_N, \qquad \sigma^2_{j^{N+1}} = \kappa - k^T C_N^{-1} k \tag{31}$$

and:

$$k = [c(\alpha^1, \alpha^{N+1}), c(\alpha^2, \alpha^{N+1}), \ldots, c(\alpha^N, \alpha^{N+1})]^T \qquad \kappa = c(\alpha^{N+1}, \alpha^{N+1}) \tag{32}$$

Thus the probability density for the function value at the new point α^{N+1} is also Gaussian with mean $\hat{j}^{N+1}$ and standard deviation $\sigma_{j^{N+1}}$. Therefore,

the most likely value for the function at the new point is $\hat{j}^{N+1}$. This value will be considered as the prediction of the kriging model. The variance $\sigma_{j^{N+1}}$ can be interpreted as a measure of uncertainty in the value prediction.

The covariance function must reflect the characteristics of the output of the computer code. In the absence of any knowledge regarding the unknown function, the most commonly used correlation function is an exponential:

$$c(\alpha^i, \alpha^k) = \theta_1 \exp\left[-\frac{1}{2}\sum_{l=1}^{N}\frac{(\alpha_l^i - \alpha_l^k)^2}{r_l^2}\right] + \theta_2$$

where $\Theta = (\theta_1, \theta_2, r_1, r_2, \ldots, r_N)$ are some parameters to be determined. This task is performed by maximizing the joint probability density $p(J_N)$, using a Particle Swarm Optimization (PSO) method.

References

[1] H.-G. Beyer and B. Sendhoff. Robust optimization – A comprehensive survey. *Comput. Methods Appl. Mech. Engrg.*, 196:3190–3218, 2007.

[2] M.D. Buhmann. Radial basis functions. *Acta Numerica*, 9:1–38, 2000.

[3] D. Ghate and M. B. Giles. Efficient Hessian Calculation Using Automatic Differentiation. Number 2007-4059. AIAA, June 2007. 25th Applied Aerodynamics Conference, Miami (Florida).

[4] R. Giering, T. Kaminski, and T. Slawig. Generating efficient derivative code wit TAF: adjoint and tangent linear Euler flow around an airfoil. *Future generation computer systems*, 21(8):1345–1355, 2005.

[5] A. Griewank, D. Juedes, H. Mitev, J. Utke, O. Vogel, and Walther A. ADOL-C: A Package for the Automatic Differentiation of Algorithms

Written in C/C++. *ACM TOMS*, 22(2):131–167, 1996. The updated article is available online at http://www.math.tu-dresden.de/∼adol-c/adolc110.ps.

[6] L. Hascoët and V. Pascual. TAPENADE 2.1 user's guide. Technical Report 0300, INRIA, Sep 2004.

[7] L. Huyse and R.M. Lewis. Aerodynamic shape optimization of two-dimensional airfoils under uncertain conditions. Technical Report 2001–1, ICASE, January 2001.

[8] A.J Keane and P.B. Nair. *Computational Approaches fo Aerospace Design: The Pursuit of Excellence*. John Wiley and Sons, 2005.

[9] O.M. Knio and O.P. Le Maitre. Uncertainty propagation in CFD using polynomial chaos decomposition. *Fluid Dynamics Research*, 38(9):616–640, September 2006.

[10] G. Lin, C.-H. Su, and G.E. Karniadakis. Predicting shock dynamics in the presence of uncertainties. *Journal of Computational Physics*, (217):260–276, 2006.

[11] D. MacKay. Gaussian Processes: A Replacement for Supervised Neural Networks? Lecture notes for a tutorial at NIPS 1997, http://www.inference.phy.cam.ac.uk/mackay/gp.pdf, 1997.

[12] M. Martinelli and L. Hascoët. Tangent-on-tangent vs. tangent-on-reverse for second differentiation of constrained functionals. In *Proceedings of AD2008*, 2008.

[13] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[14] W. L. Oberkampf and F. G. Blottner. Issues in computational fluid dynamics code verification and validation. *AIAA Journal*, 36(5):687–695, 1998.

[15] M. J .D Powell. Radial basis function methods for interpolation to functions of many variables. *HERMIS: Internl. J. Computer Maths. & and its Applics.*, 3:1–23, 2002.

[16] M. M. Putko, P. A. Newman, A. C. Taylor III, and L. L. Green. Approach for uncertainty propagation and robust design in CFD using sensitivity derivatives. Technical Report 2528, AIAA, 2001.

[17] M. Riedmiller. Advanced supervised learning in multi-layer perceptron - from backpropagation to adaptative learning algorithms. *Computer standards and interfaces*, (5), 1994.

[18] S. Rippa. An algorithm for selecting a good value for the parameter c in radial basis function interpolation. *Advanced Computational Mathematics*, 11, 1999.

[19] J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.

[20] L. L. Sherman, A. C. Taylor III, L. L. Green, and P. A. Newman. First and second-order aerodynamic sensitivity derivatives via automatic dif-

ferentiation with incremental iterative methods. *Journal of Computational Physics*, 129:307–331, 1996.

[21] A. C. Taylor III, L. L. Green, P. A. Newman, and M. M. Putko. Some advanced concepts in discrete aerodynamic sensitivity analysis. Technical Report 2529, AIAA, 2001.

[22] E. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer, 1999.

[23] J. Utke. OpenAD: Algorithm implementation user guide. Technical Memorandum ANL/MCS–TM–274, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL., 2004. Avalaible online at ftp://info.mcs.anl.gov/pub/tech_reports/reports/TM-274.pdf.

[24] B. van Leer. Flux-vector splitting for the Euler equations. *Eighth International Conference of Numerical Methods in Fluid Dynamics, Lecture Note in Physics*, 170:505–512, 1982.

[25] B. van Leer. Computational methods for ideal compressible flow. *Von Karman Institute for Fluid Dynamics. Lecture Series 1983-84. Computational Fluid Dynamics*, 1983.

[26] R. W. Walters and L. Huyse. Uncertainty analysis for fluid mechanics with applications. Technical Report 2002-211449, NASA, Feb 2002. ICASE Report No. 2002-1.

[27] D.B. Xiu and G.E. Karniadakis. Modeling uncertainty in flow simulations via generalized polynomial chaos. *Journal of Computational Physics*, (187):137–167, 2003.