Master Thesis:

Automatic optimisation of the Bloodhound SSC air intake duct

Manon Forey

Supervisor: Dr Ben J. Evans

Submitted to Swansea University in fulfillment of the requirements for the Master of Science in Computational Mechanics

Swansea University, 2012

Abstract

The design of an air intake duct is an extremely complex problem, and the aerodynamicists generally use the experience they have for certain designs to create a new intake duct.

The method proposed here tries to deviate from the usual ‘trial-error’ approach of such complex aerodynamics problems, by using an automatic optimisation technique.

In this paper, a genetic algorithm called ‘Modified Cuckoo Search’ is used to get, from a set of parameters defining the geometry to optimise, a final design which minimizes the chosen objective function.

The issue with these genetic optimisation algorithms is the great number of configurations to experiment, or run through a CFD solver. Here, the cost of the optimisation is hugely decreased using a reduction order method based on Proper Orthogonal Decomposition. The solutions for some initial designs are obtained by a CFD solver, which are then decomposed into a linear combination of ‘modes’. These modes can then be re-assembled to construct the solution for new configurations.

After some theoretical considerations about the optimisation and order reduction methods, their efficiency is assessed by optimising the air intake duct preceding the engine on the British 1000-mph car ‘Bloodhound SSC’.
The method is also compared to other techniques: the modified cuckoo search without a reduction order method, and the optimisation on the direct objective function interpolation.

Declarations and Statements

Declaration

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ....
Date..

Statement 1

This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ....
Date ....

Statement 2

I hereby give my consent for my dissertation, if relevant and accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ....
Date ....

Contents

List of Tables

List of Figures

Acknowledgements

I would first like to thank Dr. Ben Evans, and the whole Bloodhound team, who kindly offered my friend Caner Kara and I to work on this project.

Dr. Evans' enthusiasm in the whole Bloodhound SSC process as well as his expertise in CFD have been a real driving force in this Master Thesis.

Many thanks to Caner Kara (MSc. in Computational Mechanics), Sean Walton (PhD.) and Matt Kear (Level 3), which I had the pleasure to work with during this year. They all brought different and rewarding contributions to the project, professionally and humanly.

My final thought goes to Sylvain, always there for me, whatever the times.

Swansea, June 2012

Manon Forey

Definitions & Abbreviations

POD	Proper Orthogonal Decomposition
PO	Proper Orthogonal (used for modes/coefficients)
ROM	Reduction Order Method
SVD	Singular Value Decomposition
CS	Cuckoo Search
MCS	Modified Cuckoo Search
MMCS	Multiple Modified Cuckoo Search
RBF	Radial Basis Function

Ma	Mach number
α	Angle of attack
ρ	Density
u	x-velocity
v	y-velocity
p_{stat} or p	Static pressure
q	Dynamic pressure ($q = \frac{1}{2}\,\rho\,\boldsymbol{u}^2 = \frac{1}{2}\,\rho\,(u^2 + v^2)$)
p_t	Total pressure = static pressure + dynamic pressure
D	2D Distortion

1 Introduction

1.1 Motivation: the 'Bloodhound SSC'

The 'Bloodhound SSC' project, launched in 2007 by Andy Green and Richard Noble, at the instigation of Lord Paul Drayson (then UK Minister of Science), was created to be a new iconic scientific and engineering project for the United Kingdom, to inspire young people and struggle against the loss of British engineers.
It is designed to bring the current land speed record (held by Andy Green, with an average of 763 mph (1228 km/h) on 1 mile) to 1000 mph (1609 km/h), and go from 0 to 1050 mph (1690 km/h) in 42 seconds.

HAMPSON
umeco
EPSRC
serco
UWE BRISTOL
STP
BLOODHOUND SSC modelled in NX
SIEMENS

(a) Left view (February 2012)

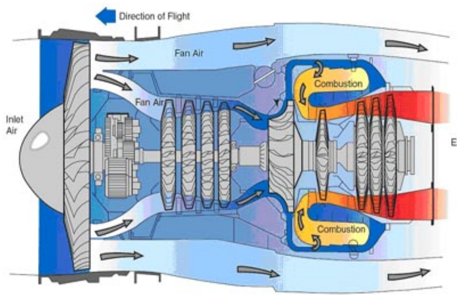
(b) Front Dynamic view (January 2012)

(c) Rear Dynamic view (February 2012)

Figure 1: Bloodhound SSC

To reach such supersonic speeds, the car will be propelled by a turbofan EJ200 (shown in figure 2a), plus a rocket engine.

(a) Eurofighter turbofan EJ200



(b) Pattern of a turbofan

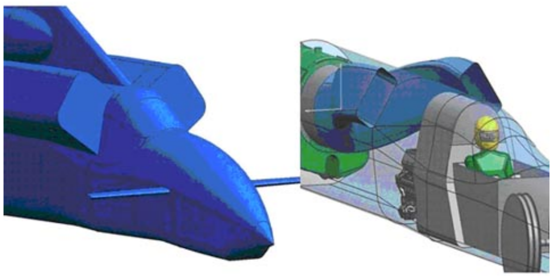
Figure 2: Turbofan

The part we are interested in is the air intake duct before the entry of the turbofan. Indeed, the intake geometry has an enormous and complex influence on the performance of the engine, and even on its integrity: a badly designed duct could well lead to some irreversible damage on the engine.

To insure the quality of the airflow at the inlet (more exactly, at the compressor face ; see figure 2b for a quick overview of the functioning of a turbofan), some aerodynamic performance parameters have been defined by aerodynamicists. These quantities are very much related to the exterior conditions, such as the mass flow rate, pressure, temperature . . .
The most important parameter is the 'distortion' of the air flow at the Aerodynamic Interface Plan (located a small distance forward the compressor face). It can be calculated with the variations in the total pressure on that particular aerodynamic reference plane.
Other parameters, such as the swirl (mean deviation from a purely axial flow upstream from the compressor face) or the flow stability (measure of the fluctuation of the flow in the duct) could be as well considered to define an optimal functioning point of the engine.
For more information about air intake for high speed vehicle, please read the article [1].

So, the shape of the duct will have a huge impact on the behaviour of the air flow entering the engine, and thus its smooth and optimal functioning.

The initial concept, shown in figure 3, had a double duct, joining behind the driver cockpit.
The poor quality of the air flow generated with this double-duct led the Bloodhoud engineers to another design, shown in figure 4. The comparison of the Mach contour for these two cases is presented in figure 5. The flow is indeed much more 'regular' in the second case, with the single intake duct.


(a) Scheme of the initial duct

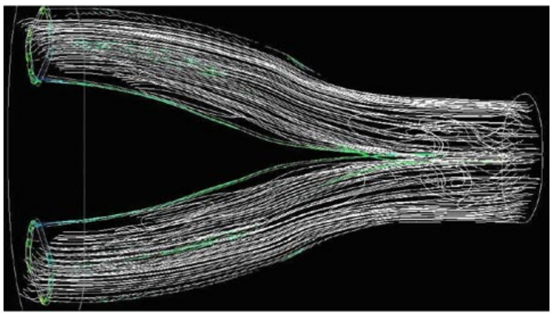

(b) CFD Simulation on the initial duct

Figure 3: Bloodhound SSC Air Intake Duct - Initial Design

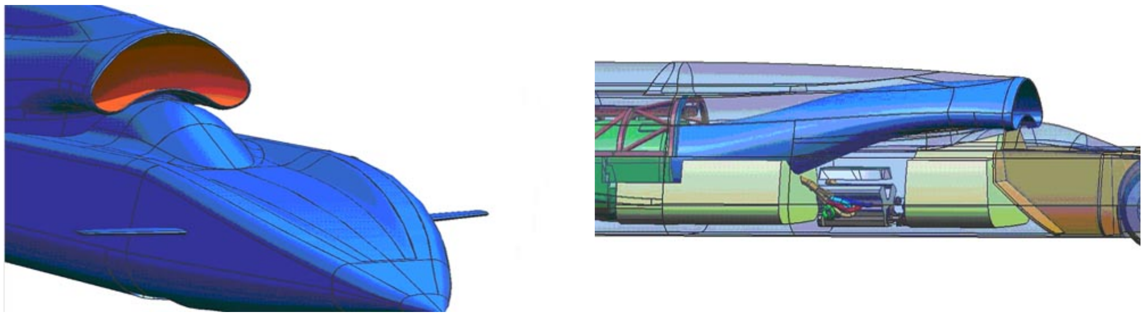


Figure 4: Bloodhound SSC Air Intake Duct - Present Design

However, this present duct has not been subject to an optimisation study, and is rather resulting from the experience and background knowledge of the Bloodhound

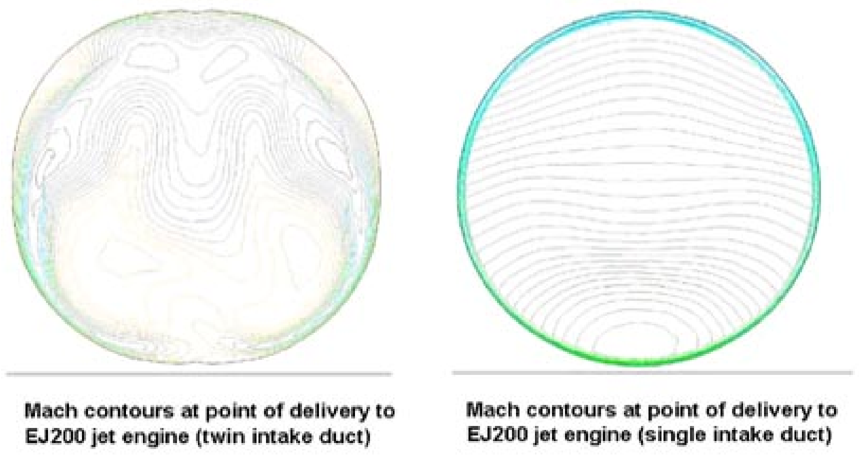


Figure 5: Comparison of the Mach contour at the compressor face for the twin (left) and single (right) intake ducts

aerodynamicists. It is this optimisation study that I have been offered to conduct by Dr Ben Evans (Swansea University), together with Caner Kara (MSc in Computational Mechanics), and with the help of Sean Walton (PhD) and Matt Kear (BEng in Aerospace Engineering).

1.2 General observations about optimisation

Optimisation problems tend to be more frequent in our world, of course in Engineering subjects (Mechanics, Control Engineering . . .), but also in the common life, for Economics or Strategy.

The engineer who has to deal with the search of an optimal solution generally knows nothing more than the values of the objective function he is trying to optimise at some certain points, either from experimental data or from a numerical solver. The usual approach is based on ‘trial and error’, where the results for a bunch of cases are analysed to get an idea of the ‘best way’ to get to the optimal solution.

However, this optimisation process reveals to be time-consuming, and there is usually no certainty that the absolute optimal solution has been reached.
To help in that process, many optimisation algorithms have been designed, which can be divided in two main parts:

- deterministic methods (gradient-based, Gauss-Seidel, ...): they are effective when the objective function is known, or when its value at some point is easy to obtain
- non-deterministic methods (Monte-Carlo method, genetic algorithms, ...): they are effective when the evaluation of the objective function at a point is difficult or/and time-consuming ; they also have the advantage of exploring the available design space more deeply, which leads to finding the global optimal solution on that space (and not stopping at a local optimum)

1.3 Aim of this project

To simplify the approach, the following study has been conducted in a 2-dimensional case. However, it can be extended to 3D, using the same methods.

As explained in part 1.1, the aim of this project is to develop and implement a Matlab code which would find the optimal 2D geometry of the intake duct, in order to get a good quality air flow at the entry of the engine fitted on Bloodhound SSC. The function to optimise is the distortion, calculated in 2D as follows:

$$\text{Distortion} = \frac{\left(\int_{\Gamma} P_t \, l \, \mathrm{d}l\right)}{\bar{P}_t L} \tag{1}$$

where Γ is the line where we want to know the value of the distortion, L its length, P_t the total pressure and $\bar{P}_t$ the average total pressure on Γ.

Here, the objective function is unknown and extremely complex with respect to the geometry of the duct, so a genetic algorithm has been used (as explained in part 1.2). The one proposed for this particular application, called 'Modified Cuckoo Search' (itself derived from 'Cuckoo Search' method, created by Yang and Deb [8]), was developed in Swansea University [6].

The following part will detail this optimisation method, and the techniques that have been developed consequently (mesh movement and reduction order methods). Later in this report, these algorithms will be implemented and the results analysed, and compared to more classical approaches.

2 Theoretical aspects, and implementation

2.1 Summary of the optimisation problem

Based on the 2D geometry defined in figure 6,

Find the geometry of the air intake duct, in 2 dimensions, which minimizes the distortion at the exit of the duct (Γ), defined as follows:

$$\text{Distortion} = D = \frac{\left(\displaystyle\int_\Gamma P_t\, l \,\mathrm{d}l\right)}{\bar{P}_t L}$$

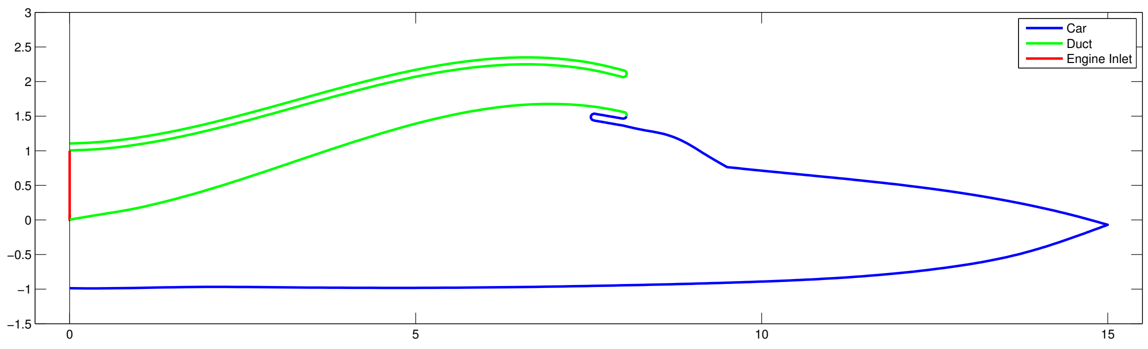


Figure 6: Geometry of the Bloodhound SSC air intake duct
Blue: car ; Green: duct ; Red: duct exit/engine inlet (for distortion computation)

2.2 Optimisation Algorithm: Modified Cuckoo Search

Choice of modified cuckoo search As the objective function (here, 2D distortion at the exit of the duct) is unknown and certainly very complex with regards to the design parameters (defining the geometry of the duct), a genetic algorithm has been chosen to solve this optimisation problem.

The selected method is called ‘Modified Cuckoo Search’ [6], which is itself derived from the ‘Cuckoo Search’ algorithm [8].

Summary of the method As any other genetic algorithm, it starts with an initial population, which moves at each iteration (called ‘generation’) following certain rules. For the modified cuckoo search, the population is divided in two parts: the individuals with the worse fitness are spread randomly on all over the domain to find better quality solutions ; for the best ones (and that is where it differs from the original cuckoo search), an improved solution is searched somewhere in the zone of these points.

Qualities The modified cuckoo search differs from other usual genetic algorithms (such as Particle Swarm, Differential Evolution or Cuckoo Search) on the following points (for more details, see [6]):

- it is robust, for very different applications
- it generally converges faster than the methods previously cited
- its implementation is easy and fast
- it allows to search in the whole design space, both locally (near the best design points) and globally (to find new interesting design points), and thus tries to reach a global minimum.

Algorithm This 'modified cuckoo search' method begins with a random set of initial 'nests' (chosen for example with a Latin Hypercube sampling), and then goes through the following algorithm for each 'generation' (G being the number of the current generation, and A a constant usually fixed to 1):

1. 25% 'best nests' $\longrightarrow$ For each nest $\boldsymbol{x_j}$:

   Pick a random nest $\boldsymbol{x_k}$ in the 'top' nests:

   (a) If $\boldsymbol{x_j} = \boldsymbol{x_k}$ (same nest): do a Lévy flight from $\boldsymbol{x_j}$, with step size $\alpha = \frac{A}{G^2}$, to find a new nest, which will replace $\boldsymbol{x_j}$

   (b) If $\boldsymbol{x_j} \neq \boldsymbol{x_k}$ (different nests):

   - if $F(\boldsymbol{x_j}) \neq F(\boldsymbol{x_k})$ (different fitnesses): move from the worst nest to the best one with distance $\frac{|\boldsymbol{x_j} - \boldsymbol{x_k}|}{\phi}$ ($\phi = \frac{1+\sqrt{5}}{2}$: golden ratio)
   - if $F(\boldsymbol{x_j}) = F(\boldsymbol{x_k})$ (same fitness): move half-way to get a new nest

   Replace any random nest by the new nest if this new fitness is better, and the new nest inside the domain

2. 75% 'worst nests' $\longrightarrow$ For each nest $\boldsymbol{x_j}$:

   Do a Lévy flight from $\boldsymbol{x_j}$, with step size $\alpha = \frac{A}{\sqrt{G}}$, to find a new nest, which will replace $\boldsymbol{x_j}$ if it is inside the domain defined previously.

After a rather small number of iterations ('generations'), a good approximation of the value of $\boldsymbol{x}$ which optimises the fitness is obtained.

Reduction of cost There is however a remaining problem: the high number of fitness evaluations means that this process could take weeks, which cannot be tolerated for industrial applications.

Indeed, if the value of the distortion had to be known at a particular design point, a computation through a CFD solver should first be performed, which would take quite a long time (minimum of one hour). If we start with an initial set of 20 cases, and run the 'modified cuckoo search' for 50 generations (minimum to have a correct idea of the optimal solution) and on one processor, the process would not be over before one month. Of course, much more computational resources could be used (with a cluster of PCs), but some new methods to reduce running time can be used there, as explained in the next section.

2.3 Reduction-Order Method: Proper Orthogonal Decomposition

The difficult part is thus to obtain the total pressure field (only field needed to compute the distortion) for any set of parameters, without going through a complete CFD computation for each case, to then be able to compute the distortion.
For this purpose, an approach based on Proper Orthogonal Decomposition (POD) has been used. The idea is to decompose the total pressure fields that are already known for different geometries, extract modes from these data, and then 'reconstruct' the pressure field for new geometries.

The method of POD presented here is called 'method of snapshots'.

2.3.1 Prerequisite: Singular Value Decomposition (SVD)

From the N initial snapshots (or cases), each containing p values (with $N < p$), a matrix of snapshots is created:

$$\boldsymbol{A} = \begin{pmatrix} a_1^{\textcircled{1}} & a_1^{\textcircled{2}} & \cdots & a_1^{\textcircled{N}} \\ a_2^{\textcircled{1}} & a_2^{\textcircled{2}} & \cdots & a_2^{\textcircled{N}} \\ \vdots & \vdots & \ddots & \vdots \\ a_p^{\textcircled{1}} & a_p^{\textcircled{2}} & \cdots & a_p^{\textcircled{N}} \end{pmatrix}$$

$$\begin{array}{ccc} \uparrow & \qquad & \uparrow \\ \textit{Snapshot } \textcircled{1} & \qquad & \textit{Snapshot } \textcircled{N} \end{array}$$

where each column corresponds to one snapshot.

In our case, each column corresponds to a case/geometry (defined with a set of geometrical parameters), and contains the values of the total pressure at the nodes of the mesh: $\boldsymbol{A}_{i,j} = P_t(\text{geometry } j, \text{node } i)$.

As the matrix is real, a singular value decomposition (SVD) can be carried out, to obtain:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$$

where $\boldsymbol{U}$ $(p \times p)$ and $\boldsymbol{V}$ $(N \times N)$ are orthogonal and $\boldsymbol{S}$ $(p \times N)$ is pseudo-diagonal:

$$\boldsymbol{S} = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_N \\ & \boldsymbol{0} & \end{pmatrix}$$

Consequently:

$$\boldsymbol{A}^T\boldsymbol{A} = \left(\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\right)^T\left(\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\right) = \boldsymbol{V}\boldsymbol{S}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{S}^T\boldsymbol{S}\boldsymbol{V}^T = \boldsymbol{V}\,\boldsymbol{diag}(\lambda_i)\,\boldsymbol{V}^T$$

so that:

$$\left(\boldsymbol{A}^T\boldsymbol{A}\right)\boldsymbol{V} = \boldsymbol{V}\,\boldsymbol{diag}(\lambda_i)$$

which means that the i^{th} column of $\boldsymbol{V}$ is the eigenvector corresponding to the eigenvalue λ_i^2 of $\boldsymbol{A}^T\boldsymbol{A}$.

The i^{th} column of $\boldsymbol{V}$ is then the right singular vector corresponding to the singular value λ_i of $\boldsymbol{A}$.

From this decomposition, a Proper Orthogonal Decomposition can be carried out.

2.3.2 Decomposition of the known pressure fields using Proper Orthogonal Decomposition (POD)

This technique enables to decompose some initial results (obtained here with a CFD solver), for different sets of parameters, in ‘modes’.

The ‘physical’ idea of the POD is to find a basis which minimizes the projection of the data points on its axis.
The problem we now have to solve is to find a good approximation of the vector containing the total pressure at the mesh nodes ($\boldsymbol{p}$), which depends only on the set of parameters ($\boldsymbol{x}$) selected for the parametric study (geometrical parameters), by a finite sum of the following form:

$$\boldsymbol{p}(\boldsymbol{x}) \simeq \sum_{k=1}^{N} c^{(k)}(\boldsymbol{x})\ \boldsymbol{\phi}^{(k)} \tag{2}$$

where the $\left(c^{(k)}(\boldsymbol{x})\right)_{k=1...N}$ are the coefficients and $\left(\boldsymbol{\phi}^{(k)}\right)_{k=1...N}$ the new basis created.

The problem can then be summed up as: find $\left(c^{(k)}(\boldsymbol{x})\right)_{k=1...N}$ and $\left(\boldsymbol{\phi}^{(k)}\right)_{k=1...N}$ that minimizes:

$$\sum_{j=1}^{N} \left|\left|\boldsymbol{p}(\boldsymbol{x}_j) - \sum_{k=1}^{N} c^{(k)}(\boldsymbol{x}_j)\ \boldsymbol{\phi}^{(k)}\right|\right|_2 \tag{3}$$

Coefficients determination The orthogonality of the basis imposes that:

$$\boldsymbol{\phi}^{(m)} \cdot \boldsymbol{\phi}^{(n)} = \delta_{mn} = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{if } m \neq n \end{cases}$$

which means that:

$$\boldsymbol{p}(\boldsymbol{x}_j) \cdot \boldsymbol{\phi}^{(i)} = \sum_{k=1}^{N} c^{(k)}(\boldsymbol{x}_j)\ \boldsymbol{\phi}^{(k)}\boldsymbol{\phi}^{(i)} = c^{(i)}(\boldsymbol{x}_j)$$

Then, equation (2) can be rewritten:

$$\boldsymbol{p}(\boldsymbol{x}) \simeq \sum_{k=1}^{N} \left(\boldsymbol{p}(\boldsymbol{x}) \cdot \boldsymbol{\phi}^{(k)}\right) \ \boldsymbol{\phi}^{(k)} \tag{4}$$

Basis determination The problem stated in (3) can then be rewritten using (4):

$$\min \sum_{j=1}^{N} \left|\left|\boldsymbol{p}(\boldsymbol{x_j}) - \sum_{k=1}^{N} \left(\boldsymbol{p}(\boldsymbol{x_j}) \cdot \boldsymbol{\phi}^{(k)}\right) \ \boldsymbol{\phi}^{(k)}\right|\right|_2 \tag{5}$$

We now introduce the Forbenius norm $||.||_F$, defined as follows:

for a matrix $\boldsymbol{M} \in \mathbb{R}^{p\times N}$,

$$||\boldsymbol{M}||_F = \sum_{i=1}^{N} (||\boldsymbol{M}_{:,i}||_2)^2$$

where $\boldsymbol{M}_{:,i}$ is the i^{th} column of $\boldsymbol{M}$.

If we define $\boldsymbol{\Phi}$ such that $\boldsymbol{\Phi}_{:,i} = \boldsymbol{\phi}^{(i)}$, we get:

$$\sum_{k=1}^{N} \left(\boldsymbol{p}(\boldsymbol{x_j}) \cdot \boldsymbol{\phi}^{(k)}\right) \ \boldsymbol{\phi}^{(k)} = \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{A}\right)_{:,j} \qquad \text{and} \qquad \boldsymbol{\Phi}^T\boldsymbol{\Phi} = \boldsymbol{I}_N(\text{orthogonality})$$

Then, as we have $\boldsymbol{p}(\boldsymbol{x_j}) = \boldsymbol{A}_{:,j}$ with the snapshot matrix $\boldsymbol{A}$ defined in part 2.3.1:

$$\begin{aligned}\sum_{j=1}^{N} \left|\left|\boldsymbol{p}(\boldsymbol{x_j}) - \sum_{k=1}^{N} \left(\boldsymbol{p}(\boldsymbol{x_j}) \cdot \boldsymbol{\phi}^{(k)}\right) \ \boldsymbol{\phi}^{(k)}\right|\right|_2 &= \sum_{j=1}^{N} \left|\left|\boldsymbol{A}_{:,j} - \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{A}\right)_{:,j}\right|\right|_2 \\ &= \sum_{j=1}^{N} \left|\left|\left(\boldsymbol{A} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{A}\right)_{:,j}\right|\right|_2\end{aligned}$$

As the $\left|\left|\left(\boldsymbol{A} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{A}\right)_{:,j}\right|\right|_2$ are all positive,

$$\text{Find } \boldsymbol{\Phi} \text{ which minimizes} \quad \sum_{j=1}^{N} \left|\left|\left(\boldsymbol{A} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{A}\right)_{:,j}\right|\right|_2$$

is equivalent to:

$$\text{Find } \boldsymbol{\Phi} \text{ which minimizes} \quad \sum_{j=1}^{N} \left(\left|\left|\left(\boldsymbol{A} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{A}\right)_{:,j}\right|\right|_2\right)^2$$

which is again equivalent to:

Find $\boldsymbol{\Phi}$ which minimizes $\left|\left|\boldsymbol{A} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{A}\right|\right|_F$.

Thus, we can rewrite our problem (5) in terms of the Frobenius norm: find $\boldsymbol{\Phi}$ such that:

$$\min \left|\left|\boldsymbol{A} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{A}\right|\right|_F \tag{6}$$

We now use Eckart–Young theorem, which states that, for a matrix $\boldsymbol{A} \in \mathbb{R}^{p\times N}$, the matrix $\boldsymbol{A}_k \in \mathbb{R}^{p\times N}$ of rank k that approximates best $\boldsymbol{A}$, ie $\min\limits_{rank(X)=k} ||\boldsymbol{A} - \boldsymbol{X}|| = ||\boldsymbol{A} - \boldsymbol{A}_k||$, is given by:

$$\boldsymbol{A}_k = \boldsymbol{U}\boldsymbol{S_k}\boldsymbol{V}^T = \boldsymbol{U}_k\boldsymbol{S}\boldsymbol{V}_k^T$$

where the SVD of $\boldsymbol{A}$ (with singular values sorted in descending order) is $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ and where $\boldsymbol{S}_k$ is the $N \times p$ pseudo-diagonal matrix containing the k^{th} first singular values of $\boldsymbol{A}$ ($\boldsymbol{S}_k(i,i) = s_i = \boldsymbol{S}(i,i)$ for $1 \leq i \leq k$, 0 otherwise), and $\boldsymbol{U}_k$ and $\boldsymbol{V}_k$ are the matrices containing the k first columns of respectively $\boldsymbol{U}$ and $\boldsymbol{V}$.

Here, it can then be deduced that: $\boldsymbol{\Phi} = \boldsymbol{U}_k\boldsymbol{S}_k$, to get an approximation of $\boldsymbol{A}$ of rank k. Here, the rank does not need to be reduced ; then $\boldsymbol{\Phi} = \boldsymbol{U}\boldsymbol{S}$.

To avoid computing the whole $\boldsymbol{U}$ matrix (usually very large), and because $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ (SVD) and $\boldsymbol{V}$ is orthogonal, the previous expression for the modes can also be re-written as:

$$\boldsymbol{\Phi} = \boldsymbol{A}\boldsymbol{V}$$

For more simplicity, the modes have been normalized, which finally gives the best approximation of $\boldsymbol{p}(\boldsymbol{x})$:

$$\boldsymbol{p}(\boldsymbol{x}) \simeq \sum_{k=1}^{N} c^{(k)}(\boldsymbol{x})\ \boldsymbol{\phi}^{(k)}$$

with

$$\boldsymbol{\phi}^{(k)} = \frac{\sum_{i=1}^{N} v_i^{(k)} \boldsymbol{p}(\boldsymbol{x}_i)}{\left|\left|\sum_{i=1}^{N} v_i^{(k)} \boldsymbol{p}(\boldsymbol{x}_i)\right|\right|} \qquad \text{and} \qquad c^{(k)}(\boldsymbol{x}_i) = \boldsymbol{\phi}^{(k)} \boldsymbol{p}(\boldsymbol{x}_i)$$

It can also be written under the matrix form:

$$\boldsymbol{A} = \boldsymbol{\Phi C}$$

with

$$\boldsymbol{\Phi} = \left[\boldsymbol{\phi}^{(1)} \dots \boldsymbol{\phi}^{(N)}\right] = \left[\frac{\boldsymbol{\psi}^{(1)}}{||\boldsymbol{\psi}^{(1)}||} \dots \frac{\boldsymbol{\psi}^{(N)}}{||\boldsymbol{\psi}^{(N)}||}\right] \quad \text{where} \quad \boldsymbol{\Psi} = \boldsymbol{AV} \quad ; \text{and} \quad \boldsymbol{C} = \boldsymbol{\Phi}^T \boldsymbol{A}$$

For more information about POD and SVD, the reader may refer to the papers [3], [2] and [5].

The POD is then obtained from the SVD of the matrix of snapshots. It enables to extract some 'behaviours' from the snapshots, as shown in figure 7.

2.3.3 Reconstruction of a new pressure field

This decomposition can now be used to 'compose' new results, for almost any set of parameters (in the same range as the initial ones), by interpolating the known coefficients for a new snapshot.

If we have the N POD modes $\left(\boldsymbol{\phi}^{(k)}\right)_{k=1\dots N}$ and the corresponding coefficients $\left(c^{(k)}(\boldsymbol{x}_j)\right)_{k=1\dots N}$ extracted from the N initial snapshots ($1 \leqslant j \leqslant N$) defined each by a set of parameters ($\boldsymbol{x}_j$), the solution at a new snapshot(defined by $\hat{\boldsymbol{x}}$) can be 'guessed' as: $\boldsymbol{p}(\hat{\boldsymbol{x}}) = \sum_{k=1}^{N} c^{(k)}(\hat{\boldsymbol{x}})\, \boldsymbol{\phi}^{(k)}$.

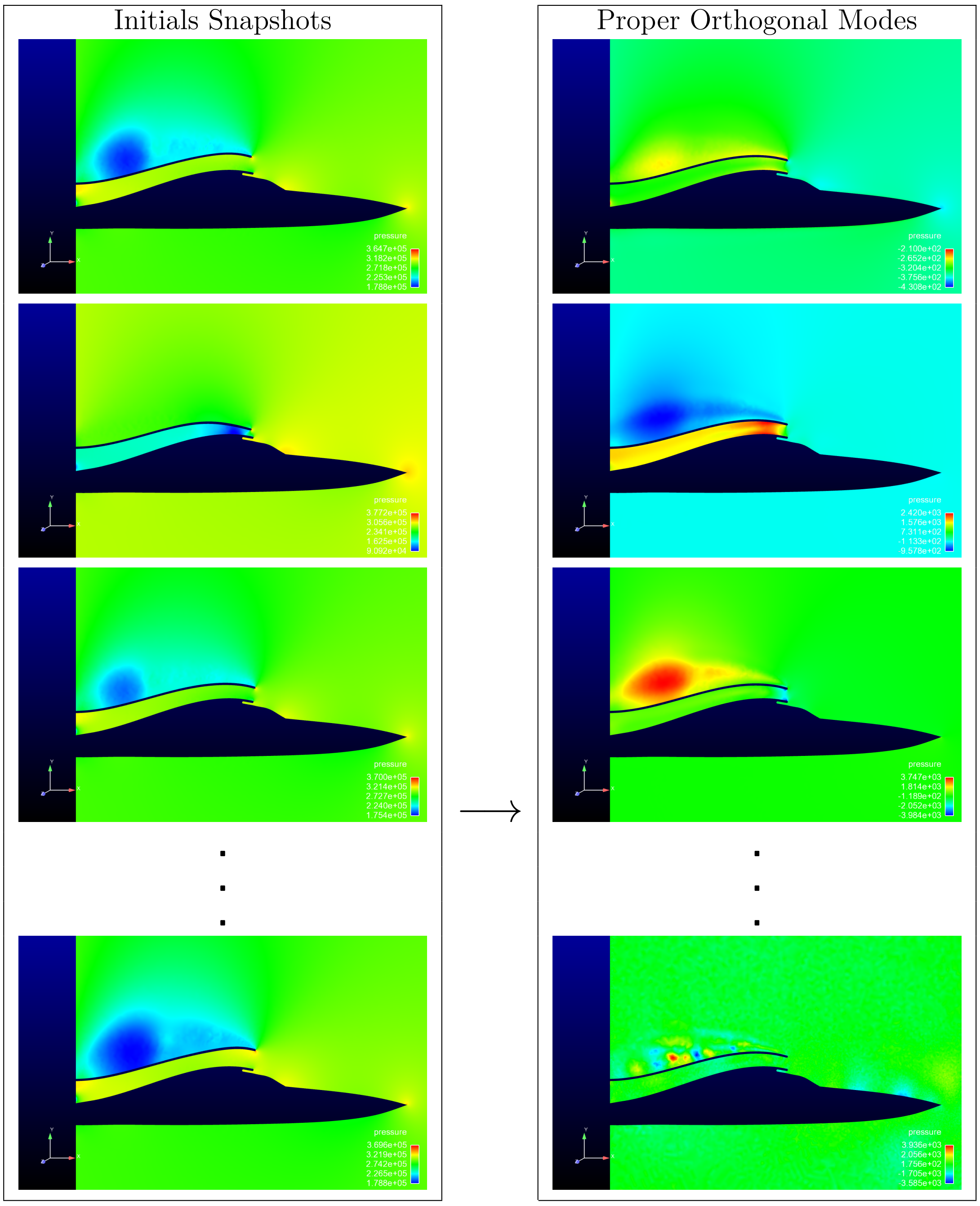


Figure 7: Initial snapshots and PO modes obtained (total pressure, plotted for a particular design)

The coefficients $\left(c^{(k)}(\hat{\boldsymbol{x}})\right)_{k=1\ldots N}$ can be obtained using an interpolation with a 'Radial Basis Function' (RBF), based on the coefficients known for the intial snapshots. Here, the multiquadric RBF approach (giving a certain type of interpolation function, as explained further), described by Rolland L. Hardy, has been used.

We first consider that the value needed (in our case, the PO coefficients $c^{(k)}(\boldsymbol{x})$; here it is generalized with $f(\boldsymbol{x})$) can be expressed as:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \lambda_i \, \zeta(||\boldsymbol{x} - \boldsymbol{x_i}||)$$

where the multiquadric RBF is defined as: $\zeta(r) = \sqrt{r^2 + \delta^2}$, where constant δ is called the shape parameter.

The interpolation function must fit the points $\boldsymbol{x}_j$ where the value $f(\boldsymbol{x_j})$ is known, which leads to:

$$f(\boldsymbol{x_j}) = \sum_{i=1}^{N} \lambda_i \, \zeta(||\boldsymbol{x_j} - \boldsymbol{x_i}||) \qquad , \; j = 1 \ldots N$$

$$\text{ie} \quad \boldsymbol{F} = \boldsymbol{B\Lambda}$$

where $\boldsymbol{F} = (f(\boldsymbol{x}_j))_{j=1\ldots N}$, $\boldsymbol{B} = (\zeta(||\boldsymbol{x}_i - \boldsymbol{x}_j||))_{i,j=1\ldots N}$ and $\boldsymbol{\Lambda} = (\lambda_i)_{i=1\ldots N}$.

The interpolation coefficients $(\lambda_i)_{i=1\ldots N}$ can then be obtained solving $\boldsymbol{F} = \boldsymbol{B\Lambda}$.

Thus, the k^{th} PO coefficient for any new configuration $(\hat{\boldsymbol{x}})$ can be computed easily from the k^{th} coefficient of the known snapshots $(\boldsymbol{x_j}, \; j = 1 \ldots N)$.
A reconstruction using these new PO coefficients $(c^{(k)}(\hat{\boldsymbol{x}}), \; k = 1 \ldots N)$ and the previously calculated PO modes $(\boldsymbol{\phi}^{(k)}, \; k = 1 \ldots N)$ will give the values of the total pressure at the nodes for the new parametrised geometry:

$$\boldsymbol{p}(\hat{\boldsymbol{x}}) = \sum_{k=1}^{N} c^{(k)}(\hat{\boldsymbol{x}}) \; \boldsymbol{\phi}^{(k)}$$

2.4 Mesh movement

Finally, using this method raises a last difficulty: when the geometry is modified, the connectivity must remain the same so that the nodes are always in the same place. This can be achieved with some particular mesh-movement techniques, to conserve the connectivity of the mesh.
The mesh movement implementation, performed by Caner Kara (MSc student in Computational Mechanics, 2010-2012), is briefly explained in appendix A.

The geometry will then be controlled by moving some ‘control points’, located on the duct boundary, on the x- and y-axis. The x- and y-displacements of the control points will be the parameters of the study.

2.5 Summary of the method

In figure 8 is shown the MCS/POD process that has been used for this project.
The algorithm of the whole main optimisation code is summed up in appendix B.

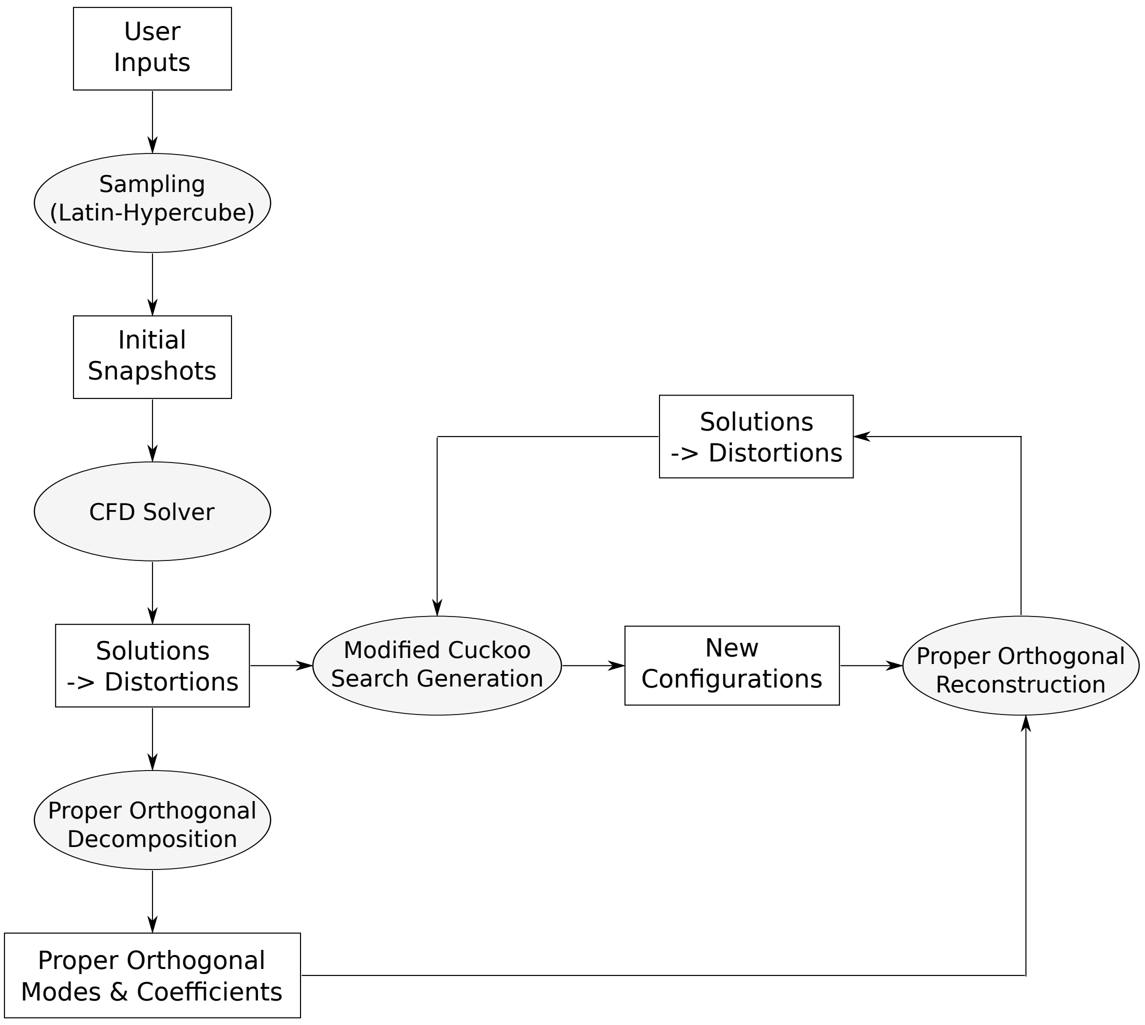

User Inputs
Sampling (Latin-Hypercube)
Initial Snapshots
CFD Solver
Solutions -> Distortions
Proper Orthogonal Decomposition
Proper Orthogonal Modes & Coefficients
Modified Cuckoo Search Generation
New Configurations
Proper Orthogonal Reconstruction
Solutions -> Distortions

Figure 8: MCS/POD process

3 Results and observations

3.1 Some numerical aspects

3.1.1 Computational aspects

This project has been developed using Matlab.
All the important scripts (solver or optimisation code) have been run on the cluster `cvcluster` in the College of Engineering of Swansea University. It uses PBS (Portable Batch System), a UNIX work-load management system developed initially by NASA, and then acquired by Altair Engineering. It enables to manage the jobs run by the users on a supercomputer.

3.1.2 Flite2D Preprocessor and Solver

Presentation of the preprocessor and the solver The preprocessor and CFD solver used here were implemented in Fortran by Kaare A. Sorensen (then PhD in Swansea University). The versions used for this Master thesis dates back from August 1998 for the preprocessor and September 1998.

The geometry and mesh are contained in a formatted `.dat` file, which details the nodes, connectivity and boundaries of the mesh.
It is then passed into the preprocessor which creates an unformatted `.sol` file.
The solver then takes this `.sol` file, as well as a formatted `.inp` file which contains all the data for the computation (farfield Mach number, angle of attack, CFL number, number of iterations . . .). The solver (which is not parallelized) finally creates a result file (`.res`) as well as a residual file (`.rsd`).

However, for this particular project, two main changes have been done to the preprocessor and the solver.

Modification of the input/output files Some files/parts of files which were not useful for the project have been removed, to save a great amount space. In the result files, the data given were the density, x- and y- velocities, the total energy as well as a data concerning the turbulence. As this particular data was not exploited, it has been replaced by the value of the static pressure, computed by the solver, but which was not printed into the result files. To differentiate these two types of result files, the new one, with the static pressure, has been renamed `.resp`

Addition of a new boundary type In the 2D solver, some boundary types were already implemented: inviscid wall, symmetry surface, far field, isothermal or adiabatic viscous wall, internal outflow.

However, for this project, the 'engine inlet' boundary type had to be included in the solver.

It consists in enforcing a given mass flow at the engine inlet. The process is the following:

- compute the total length $L_\Gamma = \int_\Gamma \mathrm{d}\Gamma$ (for 2D) of the engine inlet (Γ), as well as: $\text{ptFunc} = \int_\Gamma \frac{\sqrt{T}}{p}\,\mathrm{d}\Gamma$, where T is the temperature and p the pressure
- compute: engine mass flow $= \int_\Gamma \rho \boldsymbol{u} \cdot \boldsymbol{n}\,\mathrm{d}\Gamma$
- compute: mass flow function $=$ (engine mass flow) $\times \frac{\text{ptFunc}}{L_\Gamma}$
- compute: engine mass factor $= \frac{\text{imposed engine front mass flow}}{\text{mass flow function}}$
- modify density and velocity: $\rho_{new} =$ (engine mass factor) $\times\, \rho_{old}$

  $\boldsymbol{u}_{new} =$ (engine mass factor) $\times\, \boldsymbol{u}_{old}$
- recompute total energy: $\epsilon = e + \frac{1}{2}\boldsymbol{u}_{new}^2 = \left(\epsilon - \frac{1}{2}\boldsymbol{u}_{old}^2\right) + \frac{1}{2}\boldsymbol{u}_{new}^2$, where e is the internal energy
- recompute pressure: $p = (\gamma - 1)\left(\rho\epsilon - \frac{1}{2}\rho\boldsymbol{u}_{new}^2\right)$

After few iterations, the solver finally makes the computed mass flow function reach the imposed engine front mass flow.

3.1.3 Computation of the distortion

The Matlab code performing the computation of the distortion at the engine inlet has been implemented by Matt Kear (Level 3 Student).

This code can be split into the following steps:
once the user has entered the 2D intercept 'plane' (in fact a line, as the distortion has a real meaning only in 3D), defined by a set of points, :

1. search for each given point ($\hat{\boldsymbol{x}}$) on the 'plane' the three nearest nodes of the mesh ($\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$), so that that point is inside this new defined node triangle

2. compute the value of the total pressure at the points on the intercept plane, by weighting the total pressure at the 3 mesh nodes around ($\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$, defined at the previous step) with its barycentric coordinates

3. get the average total pressure on the interface using the pressure (p_i) at the *internodes* interface points, and the length (L_i) between each midpoint of the segments defining the interface:

$$P_{ave} = \frac{\displaystyle\sum_{i=2}^{internodes-1} p_i L_i}{\displaystyle\sum_{i=2}^{internodes-1} L_i}$$

4. compute the distortion on the 'intercept plane' (or interface):

$$D = \frac{\displaystyle\sum_{i=2}^{internodes-1} L_i \, \frac{|P_t - P_{ave}|_{i+1} - |P_t - P_{ave}|_i}{2}}{L \, P_{ave}}$$

3.1.4 Parallelization of the initial cases computations

For the CFD computation of the initial nests, a Matlab script has been written so that all the cases are all automatically computed in parallel.
After choosing the sets of parameters using a particular sampling (here, a Latin Hypercube sampling), the Matlab script follows this scheme:
For each set of parameters,

1. create a 2D geometry and mesh (`.dat` file) corresponding to the chosen geometrical parameters (mesh movement technique coded by Caner Kara)

2. pass the mesh into the preprocessor to create a side-based data structure (`.sol` file)

3. write an input file (`.inp` file), specifying, for example the Mach number, the number of iterations for the solver and the engine front mass flow

4. write a batchfile, specifying the queue, the number of nodes and processors to be used, the walltime and the memory needed for the computation

5. write the submit instruction (`qsub` for the PBS system used for this project) in a text file (`BatchSolv`)

Once this has been made for each case, and that every `.sol`/`.inp` and batchfile has been created, the file `BatchSolv` containing all the submission instructions is run, which enables to run all the cases in the same time, in parallel.

3.1.5 Main problems encountered

During this project, a series of problems has been encountered, which were finally solved:

- The initial solver provided was not the final version, and soon demonstrated some important issues. One of these problems was, for example, the fact that the solution for a geometry with angle of attack 0° was not giving the same results has the symmetrical geometry with angle of attack 180°. This problem has finally been solved after few months by the use of another version of the solver.
- The Modified Cuckoo Search Matlab code, written by Sean Walton, had also to be modified, to fit in the whole POD/MCS process, but also to correct some mistakes. The most important one was to initialize the seed for the random process, otherwise the Modified Cuckoo Search code would have given the exact same results at each run, which is not what is asked for a random genetic algorithm.
- Using the PO Decomposition and Reconstruction, some negative pressure fields were recovered. The PO coefficients were also very badly reconstructed, as it can be seen in figure 9, which may explain such negative pressure. It seems from this figure that the last modes are too much represented, which led to keeping only the ten first modes.
  However, this problem was apparently due to some mistakes in the main code, where the real value of the parameters were mixed with their normalized values.

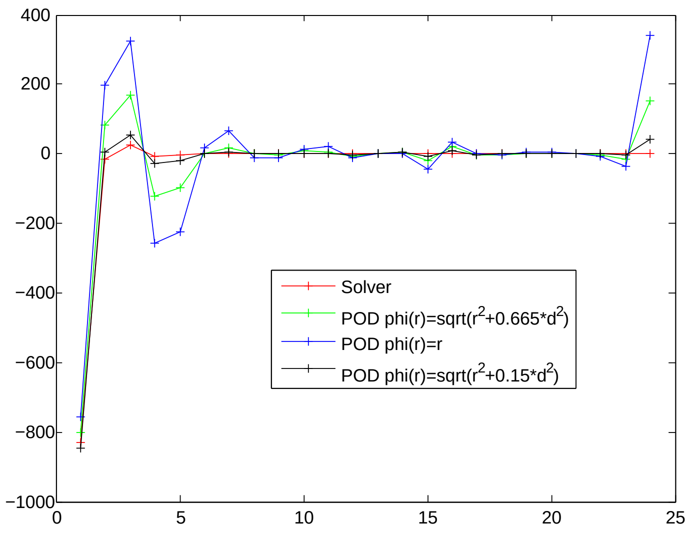


Figure 9: Badly reconstructed PO coefficients

3.2 Proper Orthogonal Decomposition and Reconstruction

3.2.1 Influence of the user inputs

To test the POD and the following reconstruction, a series of runs has been performed, while changing the different user inputs:

- number of snapshots
- number of control points to move the geometry
- range of displacement of the control points
- Mach number

The following study has been performed as such: for the chosen user inputs listed above, all the cases are run with the solver ; then, for each case, a Proper Orthogonal Decomposition is performed with all the other snapshots, to reconstruct a PO approximation of the case isolated. The distortion is then calculated, for each case, from both the real solution (obtained with the solver) and the approximated solution (obtained with the PO reconstruction). The error made on the distortion by the POD is calculated as: $\varepsilon = \dfrac{|\mathrm{D}_{\mathrm{POD}} - \mathrm{D}_{\mathrm{solver}}|}{\mathrm{D}_{\mathrm{solver}}}$.

Number of control points In table 1 are listed the tests done on the number of control points.

Here, going from one to seven control nodes (with the same ranges for displacement) increases a lot the error. The average error is multiplied from 5 to 10 when adding these 6 additional control nodes.

This is perfectly understandable, as 20 and 40 snapshots are clearly not enough to have a good 'idea' of a 14-parameter space (7 control points, with 2 prescribed displacements each, on x and y).

No Snapshots	No Control Points	x_{max}	y_{max}	Ma	Distortion Error (%)		
					Maximum	Minimum	Average
20	1 7	0.1	0.3	0.8	11.02 66.26	0.28 1.19	3.41 17.25
40	1 7	0.1	0.3	0.8	10.17 91.69	0.03 0.62	2.11 22.89

Table 1: Influence of the number of control points

Number of snapshots Table 2 sums up some tests on the influence of the number of snapshots on the POD reconstruction.

At the first glance, it seems that increasing the number of snapshots does not have an important effect on how good the approximation of the solution is.

However, the three tests using 7 control points may not mean a lot, as the space is not enough sampled anyway.

Based on the 1-control point test, the error does not seem to decrease by much. 20 snapshots will then be enough for a 1-control point parametrization.

Range of control point displacements Table 3 indicates a test made by changing the range of the control point displacement. Here, each control point can move between $-x_{max}$ and x_{max} on the x-direction (here, no x-displacement), and between $-y_{max}$ and y_{max} on the y-direction. For simplicity's sake, the maximum displacements are the same for all the control points.

It seems here that increasing the range of the displacement leads to an increase in

No Snapshots	No Control Points	x_{max}	y_{max}	Ma	Distortion Error (%)		
					Maximum	Minimum	Average
20 40	1	0.1	0.3	0.8	11.02 10.17	0.28 0.03	3.41 2.11
20 40	7	0.1	0.3	0.8	66.26 91.69	1.19 0.62	17.25 22.89
40 80	7	0.05	0.1	0.8	30.58 32.90	0.00 0.11	9.41 8.32
20 40	7	0	0.3	0.8	43.71 78.54	2.02 5.92	17.81 22.42

Table 2: Influence of the number of snapshots

the error: for three times the maximum displacement, the error is also increased by three times.

This can be explained very simply by the fact that we are enlarging the design domain: thus, for the same number of snapshots, this domain is less ‘known’.

No Snapshots	No Control Points	x_{max}	y_{max}	Ma	Distortion Error (%)		
					Maximum	Minimum	Average
40	7	0	0.1 0.3	0.8	26.38 78.54	0.41 5.92	8.73 22.42

Table 3: Influence of the number of control point displacement range

Mach Number Finally, the tests on the influence of the Mach number are listed in table 4.

First, let us study the average error. For Mach 0.5, it is much much lower than for Mach 0.8, 1.1 or 1.3, where the average error is quite similar.

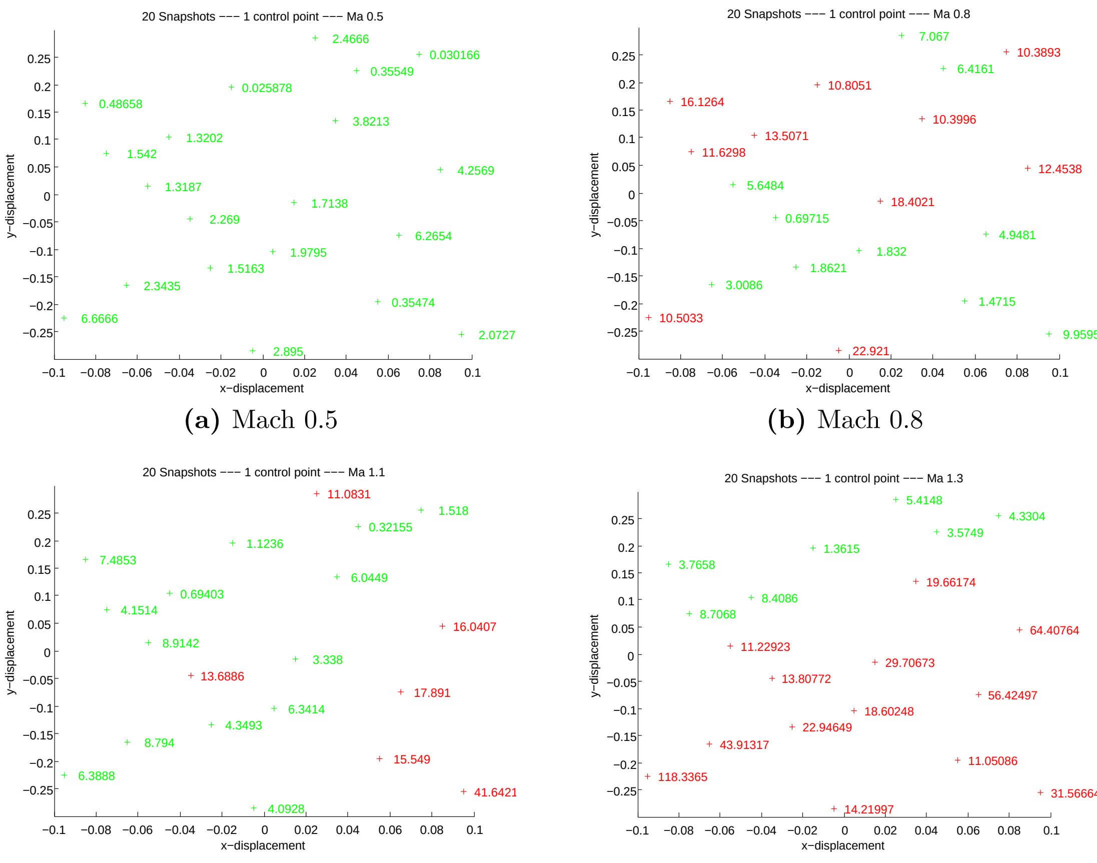
This phenomenon can be understood by noticing that Mach 0.8, 1.1 and 1.3 are all in a transonic regime, in which some complex phenomena appear (such as shocks) that the POD may not catch with the sampling imposed here. At Mach 0.5, however, the regime is subsonic, and thus the solutions are much more ‘continuous’ and quite similar.

We can also note that the maximum distortion error increases with the farfield Mach number: indeed, the highest the speed of the car, the more complex the behaviour of the air flow becomes.

No Snapshots	No Control Points	x_{max}	y_{max}	Ma	Distortion Error (%)		
					Maximum	Minimum	Average
20	1	0.1	0.3	0.5	4.59	0.05	1.18
				0.8	11.02	0.28	3.14
				1.1	18.97	0.16	4.85
				1.3	19.69	0.11	3.52

Table 4: Influence of the Mach number

Figure 10 shows the error of the POD reconstruction at each snapshot (not the same sampling as for table 4), and for different Mach numbers.

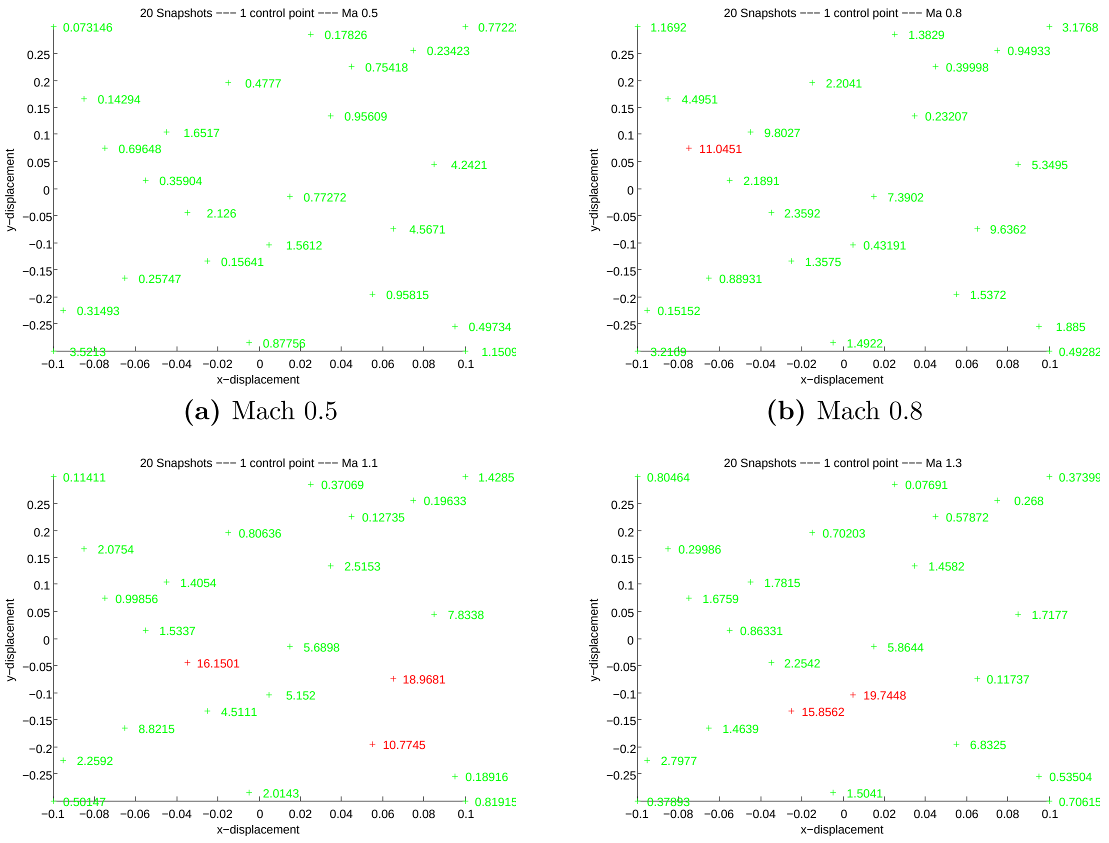


(a) Mach 0.5 **(b)** Mach 0.8 **(c)** Mach 1.1 **(d)** Mach 1.3

Figure 10: Percentage error on the POD reconstruction at each snapshot
Green: less than 10% - Red: more than 10%

This error could be reduced by adding the four 'corners' of the design space into the snapshots, as shown in figure 11. This solution has been studied, as explained in the following parts.

Note : From now on, only the top of the mouth of the duct will be moved,



(a) Mach 0.5 **(b)** Mach 0.8

(c) Mach 1.1 **(d)** Mach 1.3

Figure 11: Percentage error on the POD reconstruction at each snapshot, with the corners Green: less than 10% - Red: more than 10%

which corresponds to two parameters: the x-displacement ($-0.1 \leq x \leq 0.1$) and the y-displacement ($-0.3 \leq y \leq 0.3$) of the point at the top of the entrance of the duct.

3.2.2 Influence of the interpolation function

To be able to use an interpolation on the PO coefficients for a high number of parameters / dimensions, a radial-basis function (RBF) has been used.

However, the choice of the RBF, as well as its own coefficients, is quite delicate, as the whole optimisation process depends on the good quality of the interpolation of the PO coefficients.

Here, a multiquadric RBF, based on Hardy's theory, has been used:

$$\phi(r) = \sqrt{r^2 + c\,d^2}$$

where d is the mean distance between all the (known) points used for the interpolation, and c a coefficient which should be varied.

A series of values for the coefficient c, as well as another RBF (Gaussian: $\phi(r) = \frac{1}{c\,d^2}\, e^{-\frac{r^2}{c\,d^2}}$) have been tested, and figure 12 shows the reconstruction of the PO coefficients for a particular case with different RBFs, as well as the theoretical ones (computed using the solution obtained from the solver).

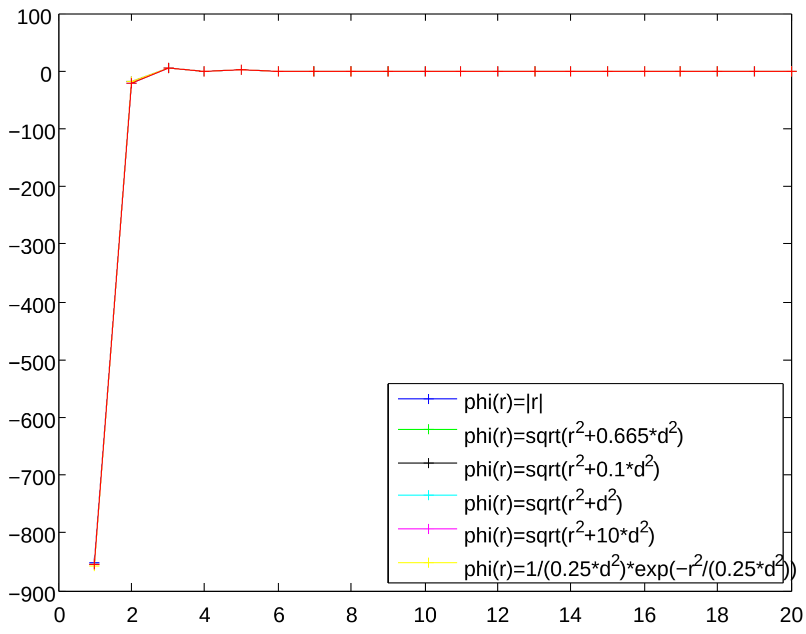


Figure 12: Original and reconstructed coefficients for a particular design

It seems from this study that the chosen RBFs all seem enough to get a good quality reconstruction of the PO coefficients, and then of the solution for the considered new configuration.

Then, the RBF that has been chosen for the continuation of the optimisation process was the simple $\phi(r) = \sqrt{r^2} = |r|$.

As the first mode seems to have the higher importance (as shown in figure 13, with the normalized energy of each mode, the total being equal to 1), a plot of the surface of the first coefficient, with respect to the two parameters (x and y displacement of the top of the duct entrance), might be interesting in this study. The resulting surface is shown in figure 14.

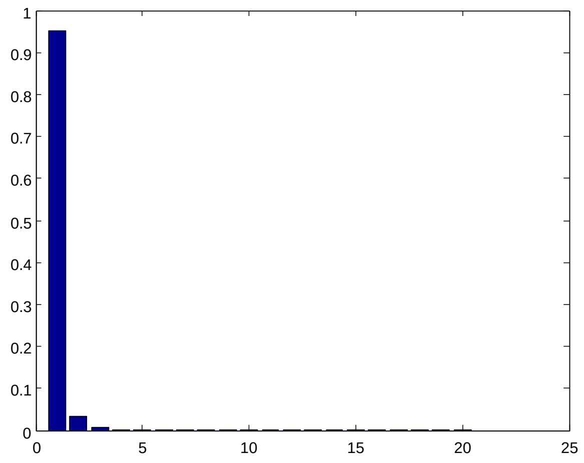


Figure 13: Normalized energy of each PO mode

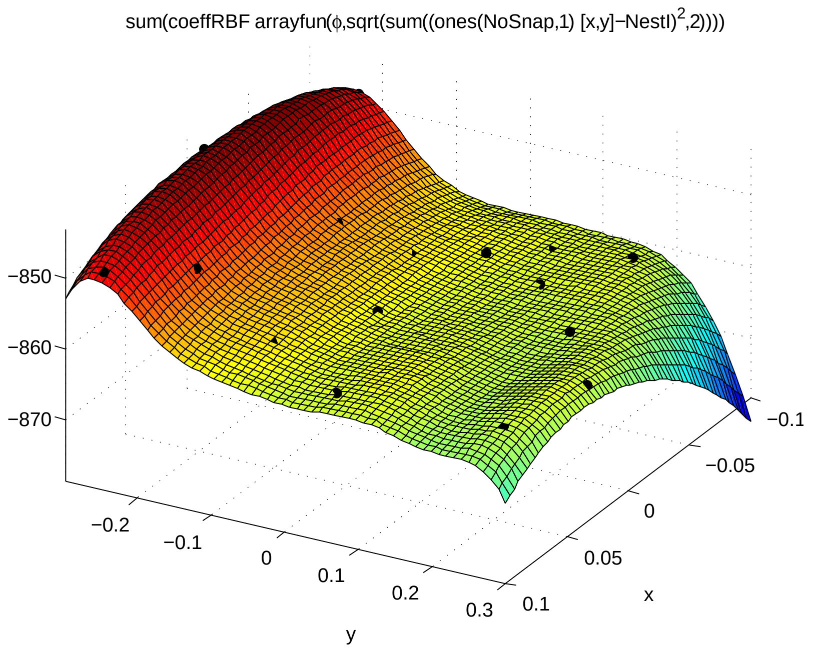


Figure 14: First PO coefficient with respect to the two design parameters

The surface fits the known points (snapshots) as expected, but ‘falls’ in the areas where there are no given points. Thus, the reconstruction could be very poor in these areas, particularly in the corners (and the whole boundary of the domain), which explains the difference observed for the error on POD reconstruction in figures 10 and 11.

To solve this problem, the four corners of the design space have then also been computed with the solver, and inserted in the PO decomposition.

The surface obtained with the initial twenty cases, plus these four additional corners, plotting the first coefficient with respect to the two design parameters, is given in figure 15.

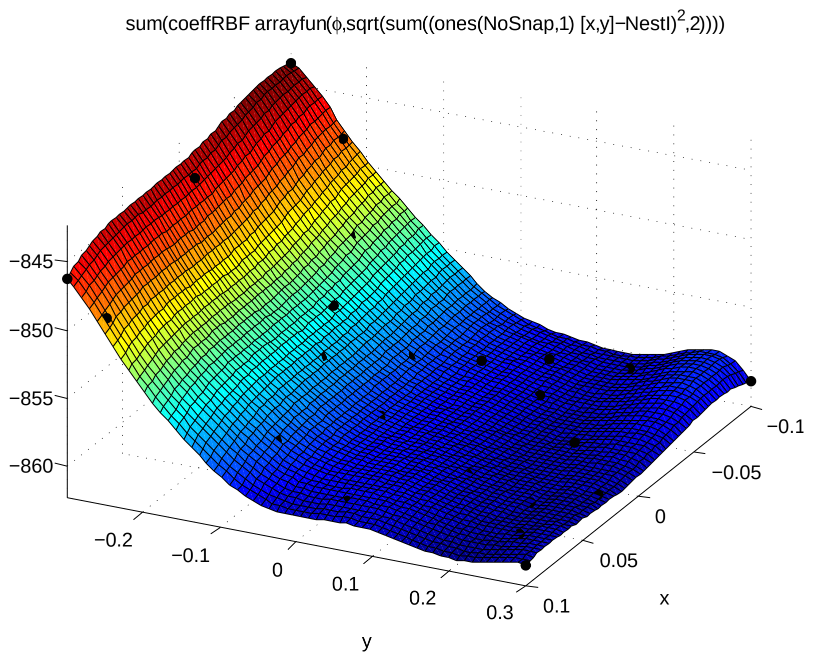


Figure 15: First PO coefficient with respect to the two design parameters, reconstructed adding the four corners to the initial snapshots

This new surface seems much more valid. It even seems that the x-displacement of the control node does not have an important impact on this first PO coefficient, but some more study should be performed on that subject.

For the continuation, this particular sampling, with twenty initial snapshots chosen with a Latin-Hypercube sampling, plus the four corners of the design space

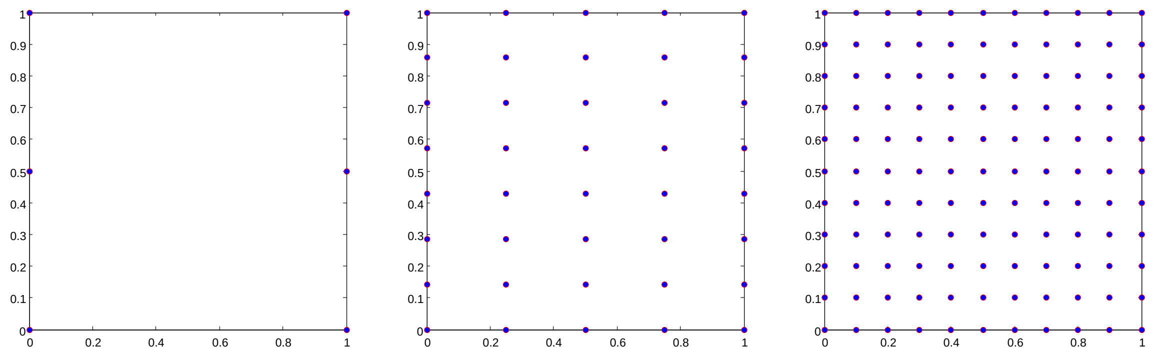
$((x, y) = (-0.1; -0.3), (-0.1; 0.3), (0.1; 0.3), (0.1; -0.3))$, has been kept.

However, this particular sampling can only be possible for a small number of dimensions (like here, with two dimensions). As soon as the number of control points increases, to be able to obtain almost 'random' geometries for the duct, the number of snapshots will have to increase inside the design space, but also on the 'corners' of that domain.
If we use five control points to move the geometry, ten parameters will be needed. If the corners were all taken as the only snapshots, it would mean that we would need to run $2^{10} = 1024$ different geometries, which is absolutely irrelevant.
For this purpose, new types of sampling could be considered, as described in the following part.

3.3 Influence of the initial sampling

Several methods are available to get a 'smart' sampling of the design domain:

- Full Factorial (figure 16): a grid is created, and the sampling points are the grid points ; the size of the grid can be adjusted



(a) Full Factorial 2×3 **(b)** Full Factorial 5×8 **(c)** Full Factorial 11×11

Figure 16: Full Factorial sampling

- Latin Hypercube (figure 17): for the number of samplings n stated, each dimension is divided into n equal segments ; the sampling points are then picked randomly so that there is one and only one point in each column and line (in 2D)

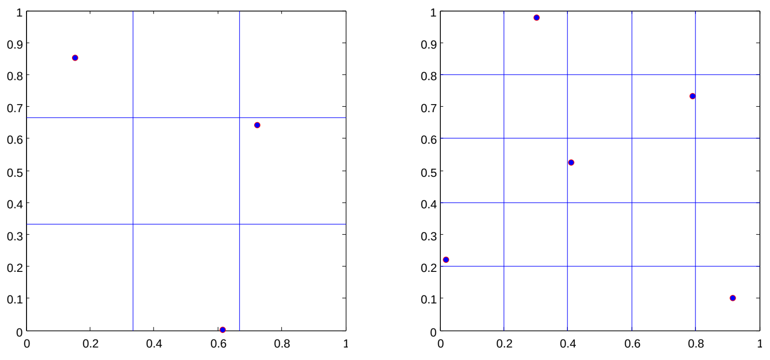

(a) Latin Hypercube - $n = 3$ (b) Latin Hypercube - $n = 5$

Figure 17: Latin Hypercube sampling in 2D

- Box-Behnken (figure 18): incomplete but equilibrated full factorial sampling with two levels for each variable, plus the central point

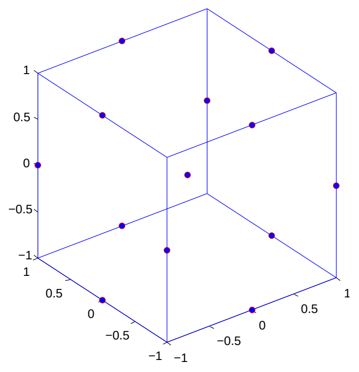
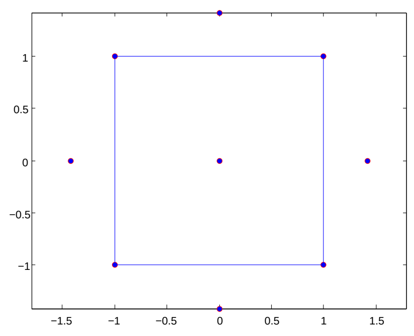


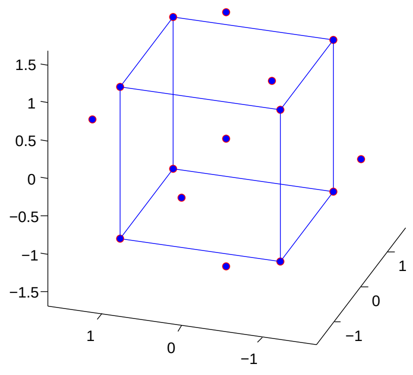
Figure 18: Box-Behnken sampling in 3D

- Central Composite (figure 19): incomplete but equilibrated full factorial sampling with two levels for each variable, plus some ‘axial’ points, plus the central point

Here, the Latin Hypercube sampling has been used, coupled with the ‘corners’ of the design space (2 dimensions, so four corners).
However, for much more parameters, the use of a Box-Behnken or a Central Composite sampling could be the only viable solution. Indeed, for ten dimensions, there



(a) 2 dimensions



(b) 3 dimensions

Figure 19: Central Composite sampling

are $2^{10} = 1024$ corners to compute, while the Box-Behnken sampling would give only 161 cases, and the Central Composite, 149 cases.

Unfortunately, there has not been enough time to investigate on that subject. It could be studied in a future work.

3.4 Optimisation algorithm

3.4.1 Modified Cuckoo Search with POD

Simple MCS The process described in figure 8 is now run, with the following characteristics:

- design parameters: x and y displacement of the top of the duct mouth ; $-0.1 \leq x \leq 0.1$ and $-0.3 \leq y \leq 0.3$
- 20 initial nests, chosen by Latin Hypercube sampling, plus the four corners of the design space: $(x, y) = (-0.1; -0.3), (-0.1; 0.3), (0.1; 0.3), (0.1; -0.3)$ $\longrightarrow$ see figure 20
- 100 MCS generations
- $Re = 6.5\,10^6$; CFL $= 1.0$; (artificial 2D) engine front mass flow $= 1.0$

A first study at Mach 0.5 is performed. Several simple MCS are launched one after the other, and the 'optimal' solutions for each test are listed in table 5, with

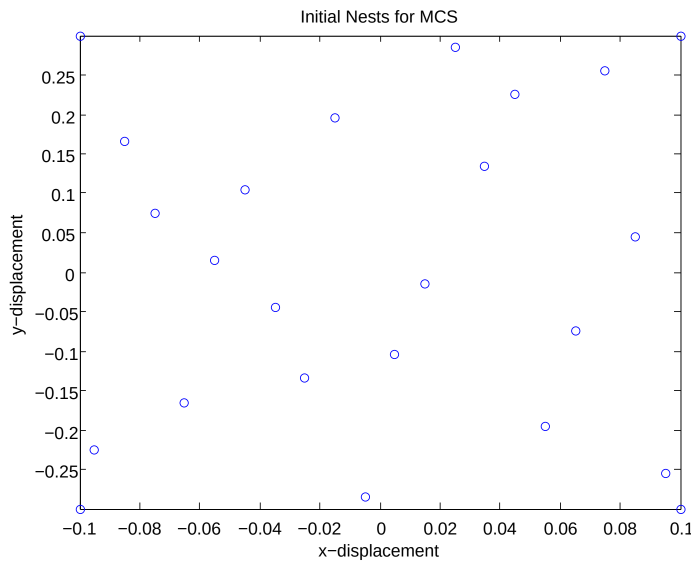


Figure 20: Initial nests for MCS

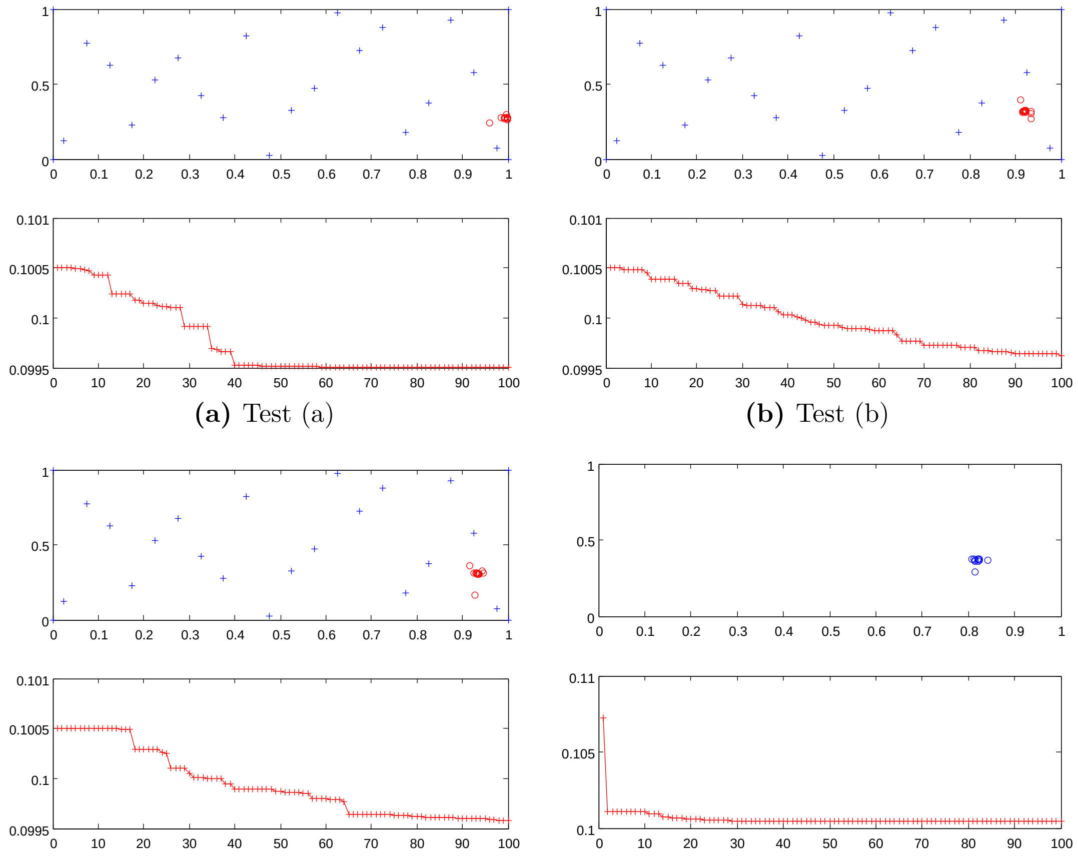
the optimal parameters and the corresponding distortion obtained with the POD. Figure 21 also shows the graphics obtained for each of these tests, with the final position of the nests, and the evolution of the distortion of the best nest (computed with the solver at the first generation, and with the POD for the following ones) with the generations.

Test	x-displacement	y-displacement	Distortion
(a)	0.1000	-0.1355	0.0995072
(b)	0.0870	-0.1173	0.0996158
(c)	0.0871	-0.1164	0.0995839
(d)	0.0643	-0.0768	0.100465

Table 5: Optimal solutions obtained with simple MCS at Mach 0.5

We can see from figure 21 that these four tests show different behaviours:

- Test (a) shows the distortion decreasing quite regularly, and then stopping at a certain value, which should mean that a convergence has been reached, and that the position obtained is the minimal one.
- Tests (b) and (c) show the distortion decreasing regularly too, but the MCS is stopped before the convergence ; here, a convergence criterion could be



(a) Test (a) **(b)** Test (b)

(c) Test (c) **(d)** Test (d)

Figure 21: Simple MCS at Mach 0.5 — Several tests
Top: blue crosses = initial nests ; red circles = final nests
(except for case (d): only final nests, in blue circles)
Bottom: best nest POD distortion evolution with the generations

used, to stop the MCS when it comes to the same optimal point at successive generations.

- Test (d) shows the distortion dropping at the second generation, and then decreasing regularly, but with a much lower slope ; the distortion computed at the final optimal point with the solver is 0.0942534, and not 0.100465 as predicted by the POD, which leads to an error of 6.5%. The big gap between the minimal distortion at generations 1 and 2 can be explained by the fact that the reconstruction with the POD is not good enough, and then gives an imprecise evaluation of the distortion. Then, once the distortion is not well reconstructed, the MCS could lead to a solution which may not be optimal, where the distortion has been under-evaluated.

Multiple MCS To avoid this sort of problem, where the reconstruction of the distortion could well lead to a 'false' optimal solution, a new approach has been used.

This time, several MCS schemes (called 'cycles' in what follows) are applied. The 'optimal' solution found at the end of each of these cycles is recomputed with the solver, to have the exact corresponding distortion, and is then added to the initial snapshots for the POD (and to the initial nests for the next MCS cycle). This method enables to add more snapshots in the area where the optimal solution seems to be, to be able to compute the distortion at the new configurations more accurately.

This process, called 'multiple MCS' (MMCS) in what follows, is summed up in figure 22.

The MMCS cycles are repeated until the optimal solution found at the end of each cycle is the same five consecutive times (and if the distortion computed with the POD is similar to the one from the solver). The number of cycles has been limited to 20, and the number of generations inside each cycle to 50.

This process is run for Mach 0.5, 0.8, 1.1 and 1.3 (maximal Mach number for the Bloodhound SSC).

Figures 23 to 26 show, for each Mach number listed above, the MMCS process, with the position of the final nests (top) and the evolution of the distortion with the MMCS cycles (bottom), as well as the geometry obtained from the optimal param-

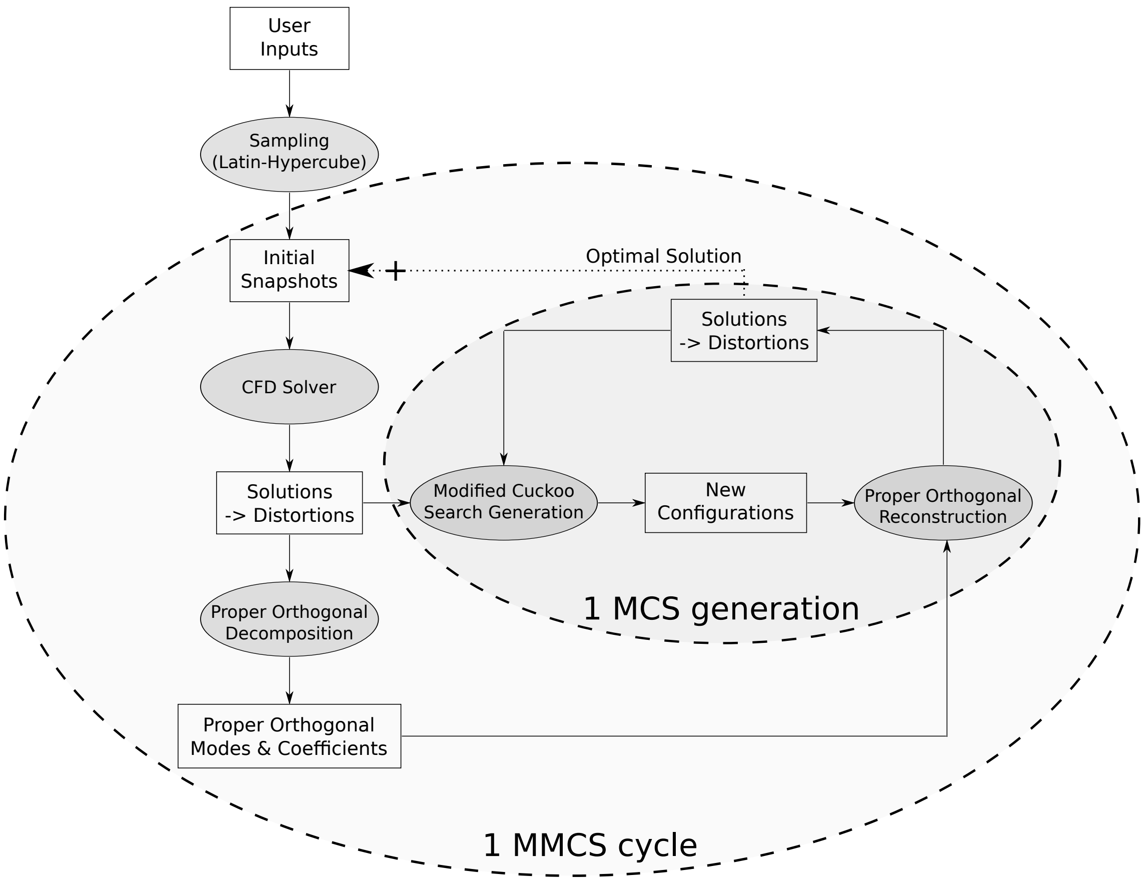
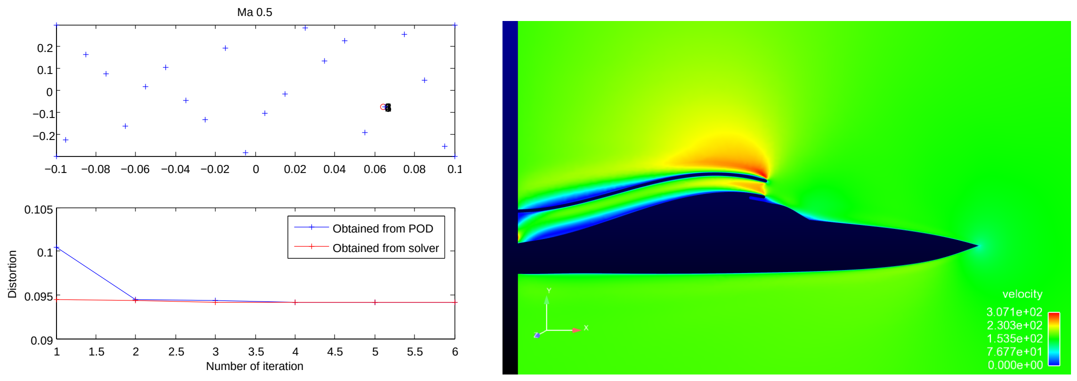

User Inputs
Sampling (Latin-Hypercube)
Initial Snapshots
Optimal Solution
Solutions -> Distortions
CFD Solver
Solutions -> Distortions
Modified Cuckoo Search Generation
New Configurations
Proper Orthogonal Reconstruction
1 MCS generation
Proper Orthogonal Decomposition
Proper Orthogonal Modes & Coefficients
1 MMCS cycle

Figure 22: Multiple MCS (MMCS) / POD process

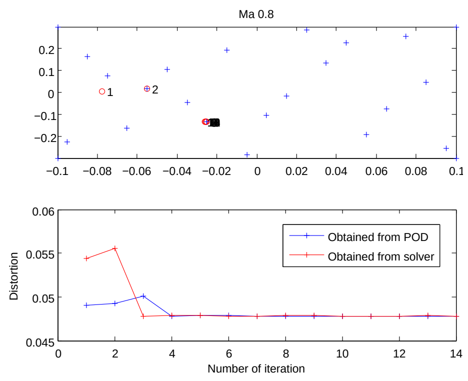
eters found, on which the velocity (computed with the solver) is plotted.



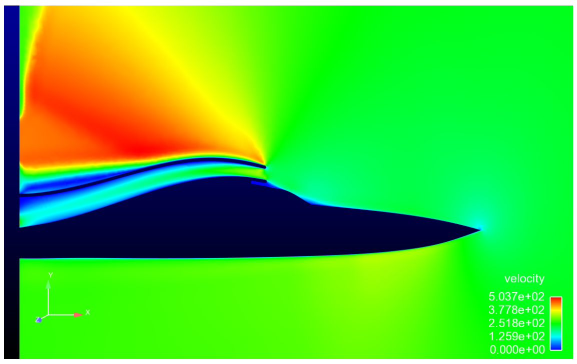
(a) MMCS Process **(b)** Optimal solution — Velocity

Figure 23: MMCS at Mach 0.5 — Optimal solution: (0.0643 ; -0.0770)

These MMCS processes were run on one processor each, with a required 1 Gb memory, a mesh of 82868 nodes and 163419 triangular elements, and with a conver-

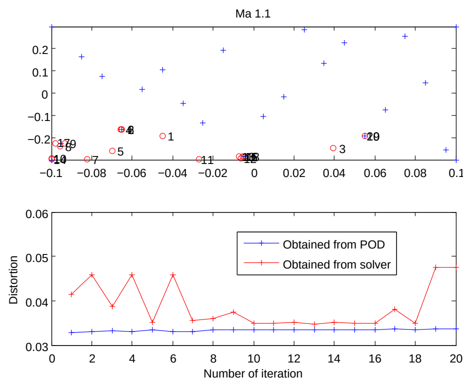


(a) MMCS Process

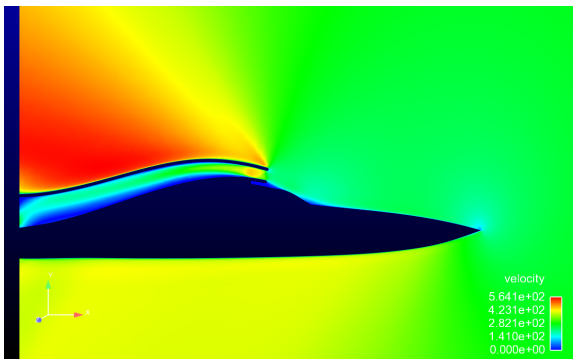


(b) Optimal solution — Velocity

Figure 24: MMCS at Mach 0.8 — Optimal solution: (-0.0255 ; -0.1350)

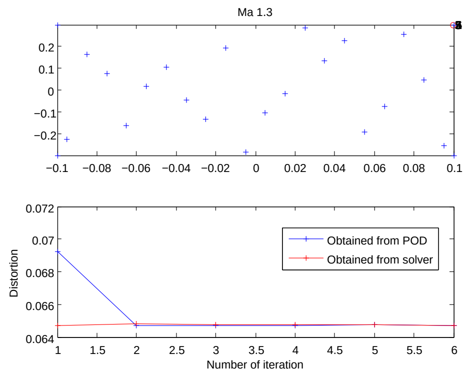


(a) MMCS Process

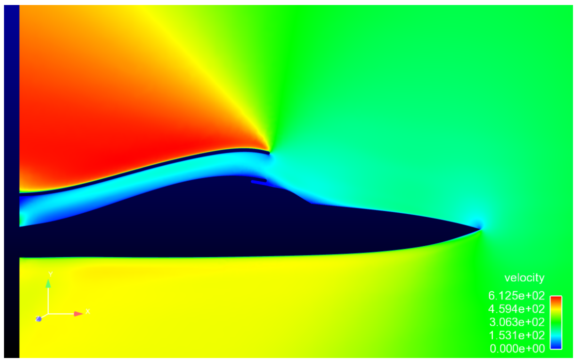


(b) Optimal solution — Velocity

Figure 25: MMCS at Mach 1.1 — Optimal solution: (0.0550 ; -0.1950)



(a) MMCS Process



(b) Optimal solution — Velocity

Figure 26: MMCS at Mach 1.3 — Optimal solution: (0.1000 ; 0.2999)

gence criterion of -2 (ratio of the current residual with the initial one).

The computation requirements for each run (at each Mach number) are listed in table 6.

Mach number	Number of cycles	Walltime	Memory used (Mb)
0.5	6	06:46	395
0.8	14	16:16	460
1.1	20	28:03	495
1.3	6	04:53	409

Table 6: Computational requirements of the MMCS at different Mach numbers

The MMCS process for Mach 0.5, 0.8 and 1.3 seems quite good, contrary to Mach 1.1.

At this Mach number (1.1), the MMCS does not seem to find an optimal solution. Worse, when a new optimal solution is found at the same position than a previous one (that has then been added to the initial snapshots), the value obtained for the distortion from the POD is not the same as from the solver.

The optimal solution at Mach 1.3 could also be discussed: indeed, it is located on the boundary of the design domain, which is not really well represented by the POD. Moreover, it could mean that the design space may need to be extended, to get an 'improved' optimal solution.

If we only concentrate on the y-displacement, it seems that lowering the top of the duct can be beneficial for Mach numbers from 0.5 to 1.1. At Mach 1.3 however, the optimal solution would be to open the 'mouth' of the duct as much as possible.

As the distortion is a value which is extremely difficult to predict (variations of the total pressure at the engine inlet) a direct physical analysis of the optimality of the solutions found with the MMCS/POD approach is not obvious.
However, some other techniques can be used alternatively to check the quality of

the optimal solution, but also to compare the computational requirements and thus highlight the high interest of the MMCS/POD method in this type of applications.

3.4.2 Modified Cuckoo Search without POD

A first alternative approach to the MMCS/POD would be the use of the (simple) MCS optimisation algorithm, but without the POD. Instead of trying to 'guess' the distortion for the new configurations, these configurations are run with the solver to get an exact value of the distortion. This can ensure there is no computational mistake when calculating the distortion.

For this purpose, a Python script has been written. Indeed, it is not possible to submit a job from one of the node of the cluster used in Swansea University (cvcluster), so the previous Matlab code for MCS, modified to run each of the new configurations with the solver, could not be used directly.

The Python code is the main script: it has to be run on the master node, and must thus consume as little CPU (and memory) as possible.
Here, this main Python code only consists in submitting some Matlab jobs on the cluster.
To enable some 'communication' between all the Matlab codes (run separately), the user inputs, such as the range of the displacements, the number of nests to consider, the Mach number, etc., have been listed in a file. The data computed by each of the Matlab scripts have been recorded in a Matlab `.mat` file.

The main Python script (given in appendix C) starts by submitting a Matlab code which gets a sample of the design space, and record the values in the `.mat` file. Then, an iteration loop submits the following Matlab scripts, for each generation, from a set of positions in the design space:

- create the mesh files (`.dat`)
- preprocess
- solve (parallel solve, one case on one processor)
- from the distortions obtained with the solver, find new configurations using the (simple) MCS process

Here, each job must wait for the previous one to finish, which can be done with the PBS system (using `afterany` or `afterok` in the submission command).

This whole code has been implemented and tested, and it has been run on the master node. For 163419 triangular elements and a convergence criterion of -3 (ratio between current and initial residuals), each generation was running in three hours, which would have surely given a good approximation of the optimal solution after $50 \times 3 = 150$ hours, or 6 days and 6 hours (after 50 generations).
However, because of the lack of time as well as some issues with the cluster, the process had to be stopped after only 20 generations.

The comparison with the MMCS/POD approach can only be made on the computational requirements (summed up in table 7), and not on the final optimal solution. For 163419 triangular elements and a convergence criterion of -3:

- the MMCS/POD asks for the same number of processors as the number of initial nests (24 here), but only during 3 hours (it can take longer if this number of processors is not available) ; after this step, it only requires one processor, and each cycle is run in three or four hours (mainly for the computation of the optimal solution with the solver at the end of each cycle)
- the first step is the same for the MCS without POD ; the second one requires 3 hours for each generation, but using again as many processors as initial nests.

	Computation of the n initial nests	Optimisation process
MMCS/POD	3 hours n processors	3 hours / cycle 1 processor
MCS without POD	3 hours n processors	3 hours / generation n processor

Table 7: Compared computational requirements of the MMCS/POD and the MCS without POD approaches (best configuration)

The MMCS/POD has then a real interest on the simple MCS without POD, particularly when the number of initial nests / snapshots is high (as it can easily happen if the number of geometrical parameters is increased).

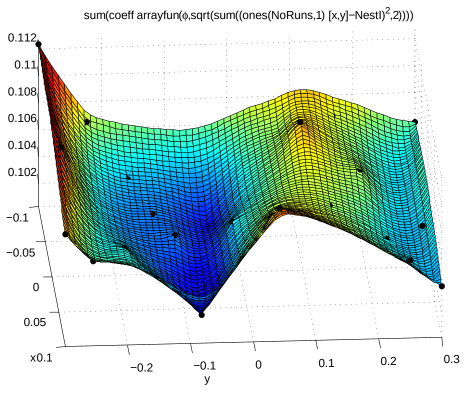
3.4.3 Design Of Experiment (DoE) and direct distortion interpolation

Another approach to this optimisation is to make a smart sampling of the design space, interpolate the distortion and get directly its minimum on the design space, without using a genetic optimisation method.
In fact, the aim here is to get an analytical description of the distortion on the design space under the form of a function, and then use a non-linear optimiser to get the optimal solution.

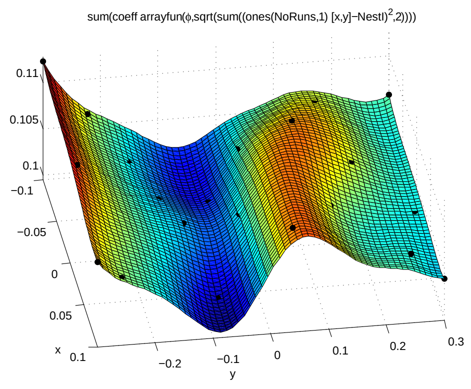
Initial sampling The choice of the sampling used to get the configurations that will be run with the solver is crucial. Indeed, as it has been seen in the previous sections, some 'holes' in the sampling of the design domain may lead to badly reconstructed function, and then artificial optima.
In this case, with a 2-dimensional space (parameters: x- and y-displacements), the Central Composite design does not have any interest (there is no need to decrease the number of sampling points), and the Box-Behnken design does not even exist in 2D. They could however reveal extremely efficient for higher number of dimensions (4 and more).

Here, the sampling previously introduced (20 snapshots chosen with Latin Hypercube sampling, plus the four corners of the design space) has been kept for the direct interpolation of the distortion.

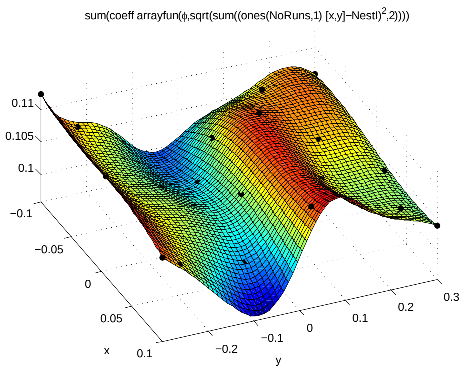
Interpolation of the distortion The interpolation from the points where the distortion is known is performed using an RBF approach, as explained in part2.3.3. Here, the multiquadric RBF ($\phi(r) = \sqrt{r^2 + \delta^2}$) has also been used, and a brief study on the shape parameter δ has been carried out at Mach 0.5, as shown in figure 27.



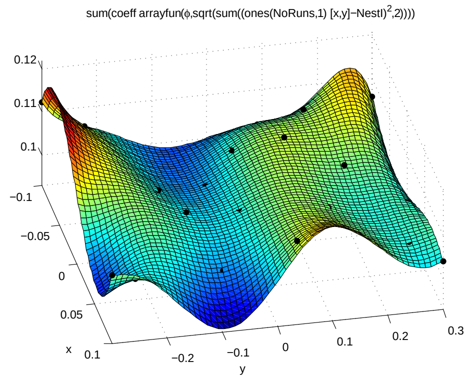
(a) $\phi(r) = \sqrt{r^2} = |r|$



(b) $\phi(r) = \sqrt{r^2 + 0.1\, d^2}$



(c) $\phi(r) = \sqrt{r^2 + d^2}$



(d) $\phi(r) = \sqrt{r^2 + 10\, d^2}$

Figure 27: Direct interpolation on the distortion with multiquadric RBF at Ma 0.5 (d: mean distance between interpolation points)

It can first be noticed that the choice of the shape parameters here is crucial, unlike the one needed for the interpolation of the PO coefficients (see figure 12). The interpolation using $\delta^2 = 0$ (figure 27a) and $\delta^2 = 10\, d^2$ (figure 27d) does not seem to be the best ones, contrary to the ones with $\delta^2 = 0.1\,d^2$ (figure 27b) and $\delta^2 = d^2$ (figure 27c).

However, they all show a 'valley' in the distortion surface, near the line $y = -0.08$, where the optimal value could be located. It is in accordance with the optimal position found at Mach 0.5 with the MMCS/POD approach: (0.0643 ; -0.0770).

Now that the distortion has been reconstructed under the form of a function on the design space, a minimization process can be carried out.

Minimization of the reconstructed distortion function To minimize the function of the reconstructed distortion, Matlab optimisation tool `fmincon` has been used.

`fmincon` function enables to find the minimum of a nonlinear function on a constrained domain, using a gradient-based method (three different algorithms available).

The user just needs to enter the function, the boundaries of the domain, a starting point (here, the center of the domain), and some options (tolerances, maximum number of iterations and evaluations, . . .), and the function returns a position in the domain where the function is minimal.

Here, for the two 'good-quality' interpolations ($\delta^2 = 0.1\,d^2$ and $\delta^2 = d^2$), two different solutions have been returned by `fmincon` at Mach 0.5:

- for $\delta^2 = 0.1\,d^2$: (-0.0635 ; -0.0669), with a distortion of 0.1011
- for $\delta^2 = d^2$: (-0.1000 ; -0.0484), with a distortion of 0.0995

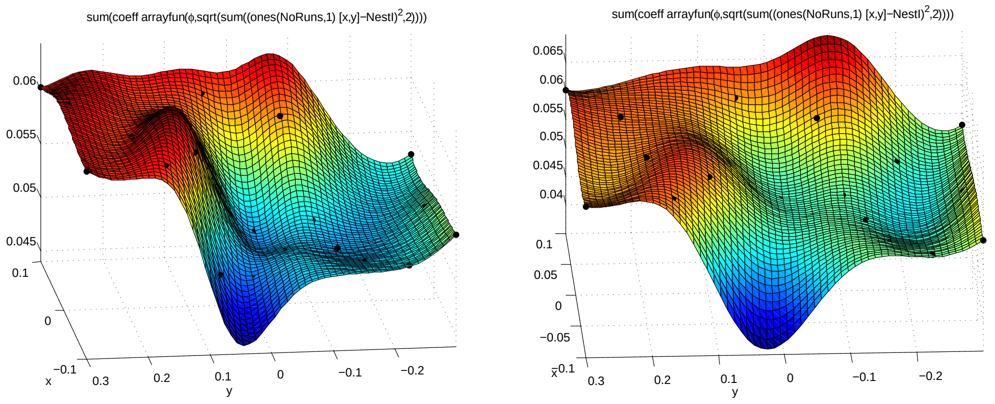
Here, the optimal solutions are in the same area, but not quite similar.

It would not be easy to state which one is nearest to the 'real' optimal solution, as both of the interpolated surfaces seem quite relevant.

However, even if this method does not indicate clearly an optimal design, it helps understand were it could be located ; in this case, at Mach 0.5, a valley is observed near $y = -0.08$.

The same process has been applied to other Mach numbers:

- Mach 0.8 (see figure 28): the optimal position found is near (0 ; -0.02), which is one of the minima, but not the global one ; the ‘global’ minimum observed near (-0.1 ; 0.02) may be only artificial and due to the lack of interpolation points in this area and to the choice of the shape parameter. The MMCS/POD was giving an optimal position at (-0.0255 ; -0.1350), for a distortion of 0.0477.



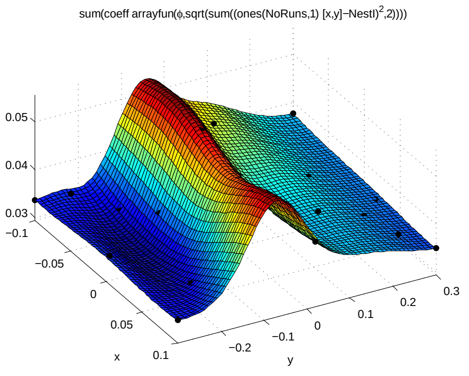
(a) $\phi(r) = \sqrt{r^2 + 0.1\,d^2}$
Minimum: (-0.0058 ; -0.0178) for 0.0496

(b) $\phi(r) = \sqrt{r^2 + d^2}$
Minimum: (0.0076 ; -0.0269) for 0.0500

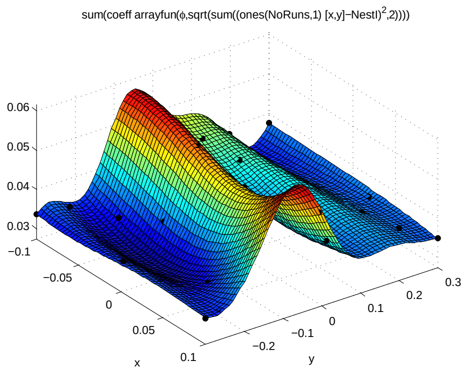
Figure 28: Direct interpolation on the distortion with multiquadric RBF at Ma 0.8 (d: mean distance between interpolation points)

- Mach 1.1 (see figure 29): the two optimal positions found are near the corners (0.1 ; 0.3) and (0.1 ; -0.3) ; from the plot of the surfaces (and also from the distortion at those two points), it seems that the global minimum is located in (0.1 ; -0.3). Two valleys are also observed near y=-0.3 and 0.3. The MMCS/POD was giving an optimal position at (0.0550 ; -0.1950), for a distortion of 0.0476 (with the solver ; POD: 0.0336).

- Mach 1.3 (see figure 30): the optimal position found is near (0.08 ; 0.29), which seems to be, from the plots, a local minimum, and which also corresponds to



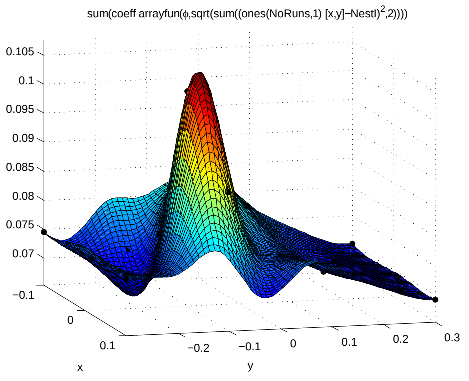
(a) $\phi(r) = \sqrt{r^2 + 0.1\,d^2}$
Minimum: (0.0988 ; 0.3000) for 0.0349



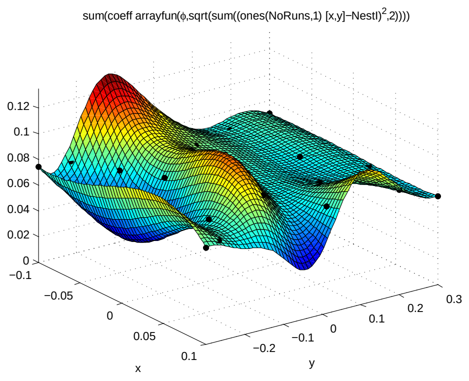
(b) $\phi(r) = \sqrt{r^2 + d^2}$
Minimum: (0.1000 ; -0.2790) for 0.0336

Figure 29: Direct interpolation on the distortion with multiquadric RBF at Ma 1.1 (d: mean distance between interpolation points)

the value found with the MMCS/POD ((0.1 ; 0.3) for a distortion of 0.0647). However, the value near (-0.05 ; -0.25) could be the global minimum on the design domain.



(a) $\phi(r) = \sqrt{r^2 + 0.1\,d^2}$
Minimum: (0.0833 ; 0.2944) for 0.0691



(b) $\phi(r) = \sqrt{r^2 + d^2}$
Minimum: (0.0703 ; 0.2938) for 0.0678

Figure 30: Direct interpolation on the distortion with multiquadric RBF at Ma 1.3 (d: mean distance between interpolation points)

So, this method seems more useful to find an area were the minimum distortion could be, by plotting the function of the interpolated distortion, rather than getting an automatic optimal solution.
It enabled to check the results obtained with the MMCS/POD, and, for some cases, to have a rough idea of the influence of the parameters on the distortion.

4 Conclusion

4.1 Brief summary of the work done

The following points have then been covered during this work:

1. Implement the MCS coupled with the POD into Matlab
2. Run this code for different configurations (changing the number of snapshots, the number of control points / parameters, the Mach number . . .) and study the results to have a better understanding of the influence of the various optimisation parameters on the POD reconstruction and on the optimisation process
3. From this study, propose solutions to improve the optimisation process, and implement them (or part of them): MMCS/POD
4. Compare this optimisation process to other usual approaches (genetic algorithm without reduction order methods, and direct interpolation over the objective function) in terms of optimal solution found and computational requirements

In this report, the quality of the POD reconstruction has been studied first. The influence of the number of snapshots, the number of control points and parameters, the range of each design parameter and the Mach number have been understood, as well as the way to reconstruct new configurations from the initial snapshots. From this initial work, it has been decided to restrict the movement of the duct to the displacement of the top of its inlet, on x and y directions. The twenty initial snapshots, obtained from a Latin Hypercube sampling, have also been completed with the four corners of the design domain, to have a better understanding of the values on the boundaries of the domain.

The aim was then to try to obtain an optimal solution at various Mach numbers. For this purpose, a new method has been applied: the MCS/POD was run several times, and the optimal solution at the end of each run was added to the snapshots, to improve the quality of the reconstruction in the areas where the optimal solution was supposed to be located. This new approach was called Multiple MCS/POD (MMCS/POD).

To ensure the quality of the optimal solution found, two alternative techniques have been used. The first one was the MCS without POD, which was implemented and tested, but which unfortunately was developed too late to be run properly. It would have certainly given a more accurate optimal solution in fewer generations than the MMCS/POD, but would have also taken much more time and resources. The second one consisted in a minimization using the direct interpolation of the distortion on the design domain. This method is of course the quickest, and only needs a good initial sampling with the corresponding cases solved. However, it requires a deeper study of the RBF used than for the interpolation of the PO coefficients. The second problem is that, when the function representing the distortion on the domain is minimized using a gradient-based method, it is very difficult to ensure the result returned is the global minimum. This method should then be seen more as a way to visualize (only for 1 or 2 parameters) the distortion in the design space than a method to get the optimal solution for a problem.

The MMCS/POD approach thus seems to be a smart method to get an optimal solution for a defined objective function, at least for a small number of parameters.

Some future work could then be carried out in the continuation of this 1-year project ; some suggestions are listed in this last part.

4.2 Future scopes

The following studies could be carried out in the continuation of this work:

- **improvement of the MMCS/POD**: it has been indeed noticed that the final solution seemed to be really close in most cases to an initial nest ; this behaviour may be due to the reconstruction of the PO coefficients ;
- **addition of some geometry control points, extension of the range of the parameters and addition of the position of the control nodes into the parameters**: this would enable a real flexibility in the movement of the geometry, but it can be limited by the quality of the deformed mesh ;
- **influence of the type of sampling**: Latin-Hypercube was enough for two parameters, but if the number of control points is increased, a sampling like Central Composite or Box-Behnken would be much more sensible ;
- **launch and study of the MCS without POD**: the code for this method is already implemented and working ; it should require at least one week of computation on a parallelized cluster, with as many processors as initial nests, to run 50 generations with a convergence criterion for the solver of -3 ;
- **extension to 3D**: once the 2D case will be working for a large number of parameters, the 3D optimisation process will be very similar to the one in 2D ; the real obstacle there should be the very high number of parameters, which the MCS manages with no difficulty, but may require more work on the POD part.

Appendices

A Mesh Movement Method

In this project, the initial connectivity must be kept whatever the configuration. To achieve this, a mesh movement technique has been developed by Caner Kara (MSc. in Computational Mechanics 2010-2012).

The mesh movement technique is based on an initial mesh that will then be deformed by the displacement of some control points (selected boundary nodes). The technique employed here can be split into two parts.

- **First, on the initial mesh, in a 'neutral' position** (figure 31):

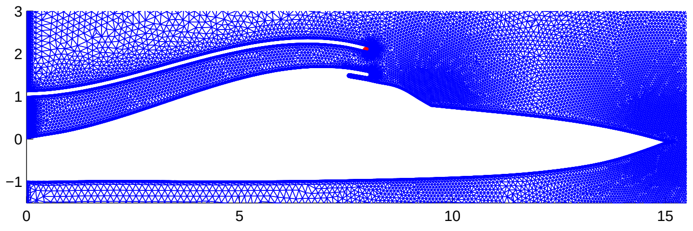
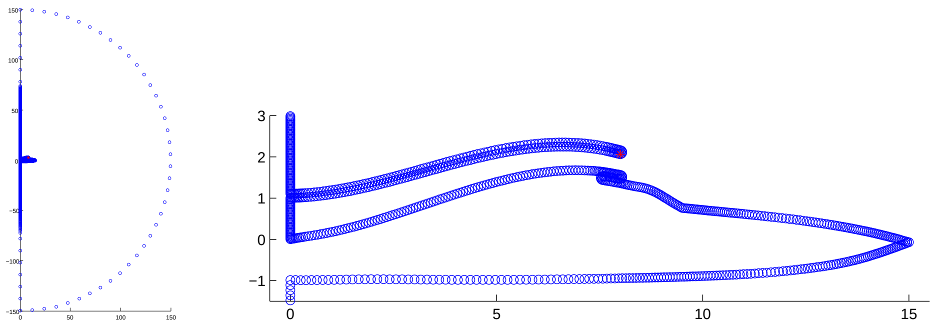


Figure 31: Initial mesh and control point (red)

1. Define the boundaries with the boundary nodes (figure 32)



(a) Whole domain

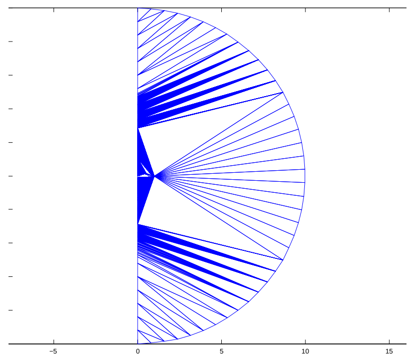
(b) Zoom on the car

Figure 32: Boundary nodes

2. Create a Delaunay graph from the boundary nodes (figure 33)

- **Then, construct the new deformed mesh**:

1. Displace the control nodes as prescribed


(a) Whole domain

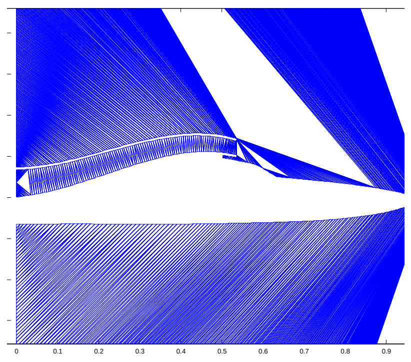
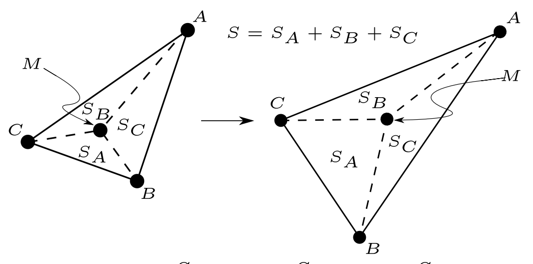

(b) Zoom on the car

Figure 33: Delaunay graph on boundary nodes

2. Displace the boundary nodes using on Radial Basis Function (RBF) approach. This is the main part of the mesh-movement work done by Caner Kara: defining the ranges and calculating the proper RBF coefficient and moving the control point with it.
3. Displace the internal nodes using the barycentric coordinates (Fast Dynamic Grid Deformation Based on Delaunay Graph Mapping, detailed in [7]), as shown in figure 34



$$\vec{x}_M = \frac{S_A}{S}\vec{x}_A + \frac{S_B}{S}\vec{x}_B + \frac{S_C}{S}\vec{x}_C$$

Figure 34: Internal nodes displacement using barycentric coordinates

The new deformed mesh is then obtained (figure 35).

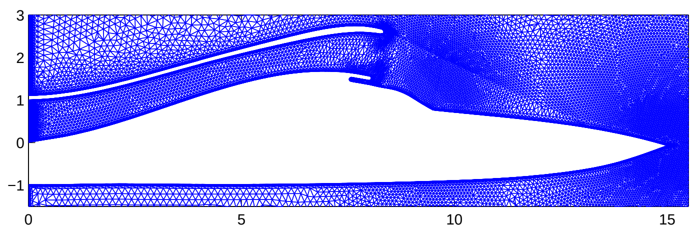


Figure 35: Deformated mesh (top of the mouth displaced of (0.3;0.5)

For more information about this mesh movement technique, the reader may refer to Caner Kara's Master thesis report ([4]) and to [7].

B Main code

Clear all

I. Generate an initial set of eggs

↪ Data given by user:

- number of dimensions (`NoDim`)
- domain for each parameter (`vardef`)
- number of nests/eggs (`NoNests`)
- number of iterations for the solver (`NoIter`)
- Mach number (`Ma`), for the solver
- engine front mass flow (`engFMF`), for the solver
- path to the executables (`pathexec`)
- nodes (identifiers and coordinates) defining the plane to compute distortion
- (if needed: whether or not the solver will be run on cluster (`runOnCluster`))

1. Generate random sets of parameters

   ```
   NestI = LHC(vardef, NoNests)
   ```

2. Generate corresponding `.dat` files

   ```
   @meshfunc(del, mesh, Nest(i,:), rect, fnames{i})
   ```
   , for i=1 . . . NoNests

   Names of `.dat` files: `duct‘i’.dat` ; all contained in cell `fnames`

3. Preprocess (Flite2D preprocessor)

   ```
   preproFlite2D(fnames, pathexec)
   ```

   Creates unformatted ‘.sol’ files

4. Solve (Flite2D solver)

   - On a PC:

     ```
     solveFlite2D(runOnCluster,fnames,NoIter,Ma,engFMF,pathexec)
     ```

     Creates `.inp` files, and solves each case

   - On the cluster:

     ```
     solveFlite2Dc(fnames, NoIter, Ma, engFMF,pathexec)
     ```

     Creates `.inp` files, writes a file `batchSolv`, which contains all the ‘qsub’ commands, and should be executed by the user on the master

5. Get fitness for initial nests

   ```
   F(i,1) = getDistortion([fnames{i} ‘.resp’], engInl)
   ```

   (only for nests inside the domain, if needed)

6. Get PO modes and coefficients for initial nests
   `[modes, coeff] = pod_imodes(NestI, fnames,NoNodes)`
   Does a proper orthogonal decomposition (POD), using singular value decomposition

II. Cuckoo Search iterations

↪ Data given by user:
- number of generations (`K`)
- maximum number of steps for a random walk (`MaxNoSteps`)
- maximum distance for a random walk (`A`)
- fraction of discarded eggs (`pa`)
- constrained to domain (`constrain`)

Here, the coordinates (parameters) are sealed between 0 and 1.

For each generation G:

1. Sort nests by fitness

2. For each discarded egg:

   - Do a Lévy flight with step $\alpha = \frac{A}{\sqrt{G}}$
   - Replace position and fitness (evaluated with function evalFitness, described below) if inside the domain (if needed, ie `constrained`$= 1$)

3. For each 'top' egg:

   - Pick a random egg among 'top' eggs
     * if same egg: do a Lévy flight with step $\alpha = \frac{A}{G^2}$
     * if different eggs: move from worst ($\boldsymbol{x_w}$) to best ($\boldsymbol{x_b}$) egg with distance $\frac{\|\boldsymbol{x_w}-\boldsymbol{x_b}\|}{\phi}$ where $\phi = \frac{1+\sqrt{5}}{2}$ (golden ratio) ; if same fitness, move half way
   - Replace position and fitness (evaluated with function evalFitness), if inside the domain (if needed, ie `constrained`$= 1$), and if better fitness than a random nest.

Function evalFitness(pos, del, mesh, NestI, modes, coeff)

Returns fitness F for given position pos

- (Create geometry and mesh from position pos: `[xy, connec] = ductsnap3b(del, mesh, pos)`) → useless if the plane to compute distortion is fixed (here, contained in engInl)
- Reconstruct results with POD (j=1...5):
    - interpolate coefficients in function of position:
      `newcoeff = interpol(NestI, coeff', pos)'`
    - find results: `pressField = modes * newcoeff`
- Get fitness: `F = getDistortion2(pressField,engInl)`

III. Final solution

Take the best nest at the end of the final generation

C Main Python script for MCS without POD

totalMCS.py

```
#!/usr/bin/python
# -*-coding:utf-8 -*

import os, time, string,subprocess

#---------------------------------------------------------------------#
# Function 'waitJob', to wait for job 'idJob' to finish
def waitJob(idJob) :
    while True:
        p = subprocess.Popen(["qstat",idJob],stdout=subprocess.PIPE,
                             stderr=subprocess.PIPE)
        if p.wait()!=0 :
            break
        time.sleep(10)
#---------------------------------------------------------------------#

# User Inputs
startScratch = "n"
NoGen = 50

# Choose: use initial nests or compute new ones
if startScratch == "y" :
    print "Starting␣from␣scratch"
    istart = 1
    print "Creating␣a␣random␣sampling"
    # Sampling
    p = subprocess.Popen(["qsub","batchSampl"],stdout=subprocess.PIPE)
    p.wait()
    IDSAMPL = p.communicate()[0]
    IDSAMPL = IDSAMPL[0:-1]
    # Wait for the end of sampling
    waitJob(IDSAMPL)

elif startScratch == "n" :
    print "Starting␣from␣initial␣nests"
    print "Generation␣1"
    # First generation of MCS
    p = subprocess.Popen(["qsub","batchMCSGen1"],stdout=subprocess.PIPE)
    p.wait()
    IDMCS = p.communicate()[0]
    IDMCS = IDMCS[0:-1]
    istart = 2
    # Wait for the end of MCS 1st generation
    waitJob(IDMCS)



# General MCS process (iteration = generation)
for i in range(istart,NoGen+1) :
    print "Generation␣",i

    # Meshing
    if i!=1:
        p = subprocess.Popen(["qsub","-W","depend=afterany:"+IDMCS,"batchMesh"],
                                            stdout=subprocess.PIPE)
    else:
        p = subprocess.Popen(["qsub","batchMeshGen1"],stdout=subprocess.PIPE)
    p.wait()
    IDMESH = p.communicate()[0]
    IDMESH = IDMESH[0:-1]

    # Preprocessing
    p = subprocess.Popen(["qsub","-W","depend=afterany:"+IDMESH,"batchPrepro"],
                                        stdout=subprocess.PIPE)
    p.wait()
    IDPREPRO = p.communicate()[0]
    IDPREPRO = IDPREPRO[0:-1]

    # Prepare Solving
    p = subprocess.Popen(["qsub","-W","depend=afterany:"+IDPREPRO,"batchPreSolv"],
                                        stdout=subprocess.PIPE)
    p.wait()
    IDPRESOLV = p.communicate()[0]
    IDPRESOLV = IDPRESOLV[0:-1]

    # Wait for the end of solving preparation
    waitJob(IDPRESOLV)

    # Solving (1 qsub/case to solve)
    os.chdir('./Data')
    p = subprocess.Popen(["sh","./batchSolv"],stdout=subprocess.PIPE)
    os.chdir('..')
    p.wait()
    IDsSOLV = p.communicate()[0]
    IDsSOLV = string.replace(IDsSOLV,'\n',':')
    IDsSOLV = IDsSOLV[0:-1]
    print IDsSOLV

    # MCS Generation
    if i!=1:
        p = subprocess.Popen(["qsub","-W","depend=afterany:"+IDsSOLV,"batchMCS"],
                                            stdout=subprocess.PIPE)
    else:
        p =
            subprocess.Popen(["qsub","-W","depend=afterany:"+IDsSOLV,"batchMCSGen1"],
                                            stdout=subprocess.PIPE)
    p.wait()
    IDMCS = p.communicate()[0]
    IDMCS = IDMCS[0:-1]
```

References

[1] AGARD (Advisory Group for Aerospace Research & Development), NATO. Air intakes for high speed vehicles (AGARD advisory report 270). September 1991.

[2] L. Cordier and M. Bergmann. Proper orthogonal decomposition: an overview. Lecture series 2003-04 on post-processing of experimental and numerical data, February 2003.

[3] L. Cordier and M. Bergmann. Réduction de dynamique par décomposition orthogonale aux valeurs propres (POD). March 2006. in French.

[4] C. Kara. A generic mesh movement method via radial basis function. Master thesis, Swansea University, June 2012.

[5] G. Kerschen, J.-C. Golinval, A. Vakakis, and L.A. Bergman. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: An overview. *Nonlinear Dynamics*, 41:141–170, 2005.

[6] S. Walton et al. Modified cuckoo search: A new gradient free optimisation algorithm. *Chaos, Solitons & Fractals*, 2011.

[7] X. Liu et al. Fast dynamic grid deformation based on delaunay graph mapping. *Journal of Computational Physics*, 211:405–423, 2006.

[8] X.-S. Yang and S. Deb. Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, 1(4):330–343, 2010.