Summer 2003

Numerical Analysis of an Airfoil Response to an Impinging Gust

Claire M. Lessiau

Embry-Riddle Aeronautical University - Daytona Beach

Follow this and additional works at: http://commons.erau.edu/db-theses

Part of the Aerospace Engineering Commons

Scholarly Commons Citation

This thesis is brought to you for free and open access by Embry-Riddle Aeronautical University – Daytona Beach at ERAU Scholarly Commons. It has been accepted for inclusion in the Theses - Daytona Beach collection by an authorized administrator of ERAU Scholarly Commons. For more information, please contact commons@erau.edu.
NUMERICAL ANALYSIS OF AN
AIRFOIL RESPONSE TO AN
IMPINGING GUST

by
CLAIRE M. LESSIAU

A Thesis Submitted
to the Aerospace Engineering Department
in Partial Fulfilment of the Requirements for the Degree of
Master of Science in Aerospace Engineering

EMBRY-RIDDLE
AERONAUTICAL UNIVERSITY
DAYTONA BEACH, FLORIDA
SUMMER A 2003
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
NUMERICAL ANALYSIS OF AN
AIRFOIL RESPONSE TO AN
IMPENDING GUST

by
CLAIRE M. LESSIAU

This thesis was presented under the direction of the candidate's thesis committee chair, Dr. Vladimir V. Golubev, Department of Aerospace Engineering, and has been approved by the members of this thesis committee. It was submitted to the Department of Aerospace Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Aerospace Engineering.

THESIS COMMITTEE:

Dr. Vladimir V. Golubev
Chair

Dr. Eric Perrell
Member

Dr. Axel Rohde
Member

Dr John Watret,
Associate Chancellor for Academic Affairs

Chair, MSAE Program

Dean, College of Engineering

Date: 06/03/03
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>vi</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>RESUME</td>
<td>viii</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1. Single Airfoil Gust Response Problem</td>
<td>2</td>
</tr>
<tr>
<td>1.1. Presentation of the Problem</td>
<td>2</td>
</tr>
<tr>
<td>1.2. Airfoil Geometry</td>
<td>4</td>
</tr>
<tr>
<td>1.3. Expected Answers</td>
<td>5</td>
</tr>
<tr>
<td>2. The BASS code</td>
<td>7</td>
</tr>
<tr>
<td>2.1. Purpose of the Code</td>
<td>7</td>
</tr>
<tr>
<td>2.2. Mathematical Formulation</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1. Governing Equations</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1.1 Spatial Discretization</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1.2 Time Discretization</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2. Numerical Implementation of the Gust</td>
<td>12</td>
</tr>
<tr>
<td>2.2.3. Structure of the Code</td>
<td>13</td>
</tr>
<tr>
<td>2.3. Grid</td>
<td>17</td>
</tr>
<tr>
<td>2.3.1. Background on GridPro®</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2. Spatial Discretization of the Computational Domain</td>
<td>18</td>
</tr>
<tr>
<td>2.3.3. The “decomp” Code</td>
<td>20</td>
</tr>
<tr>
<td>2.4. Boundary Conditions</td>
<td>21</td>
</tr>
<tr>
<td>2.4.1. External Boundary Conditions</td>
<td>21</td>
</tr>
<tr>
<td>2.4.1.1 Radiation Boundary Condition</td>
<td>22</td>
</tr>
<tr>
<td>2.4.1.2 Outflow Boundary Condition</td>
<td>23</td>
</tr>
<tr>
<td>2.4.2. Internal Boundary Conditions</td>
<td>24</td>
</tr>
<tr>
<td>2.4.2.1 Wall Boundary Condition</td>
<td>24</td>
</tr>
<tr>
<td>2.4.2.2 Connectivity Boundary Condition</td>
<td>26</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Fig 1: Airfoil in a 2-D Gust – Parallel and Vertical Components ........................................ 2
Fig 2: Benchmark Joukovski Airfoil Geometry .................................................................... 5
Fig 3: Spatial Discretization ............................................................................................... 9
Fig 4: Structure of the Code .............................................................................................. 16
Fig 5: 605 × 240 Point Grid for the Cambered Airfoil .................................................... 19
Fig 6: Decomposition of the Computational Domain: 16 blocks .................................. 20
Fig 7: Boundary Conditions .............................................................................................. 21
Fig 8: PACL Net .................................................................................................................. 28
Fig 9: Radiated and Convected Waves Throughout the Domain .................................... 33
Fig 10: Speedup Expected from the Cluster ...................................................................... 34
Fig 11: Transient Profile for the Gust Initialization ......................................................... 38
Fig 12: Density Initialization (2D Gust, k=3.0, ε=20%) ..................................................... 39
Fig 13: ρu (a) and ρv (b) Initialization (2D Gust, k=3.0, ε=20%) ...................................... 39
Fig 14: Energy Initialization (2D Gust, k=3.0, ε=20%) ..................................................... 40
Fig 15: Velocity Magnitude Contours u for a 2D-2% Gust, k=1.0 ................................. 43
Fig 16: Velocity Magnitude Contours v for a 2D-2% Gust, k=1.0 .................................. 44
Fig 17: Pressure Contours ................................................................................................. 45
Fig 18: Pressure Contours: Zoom in the Airfoil Region .................................................. 45
Fig 19: Energy Contours v for a 2D-2% Gust, k=1.0 ....................................................... 46
Fig 20: Density Contours v for a 2D-2% Gust, k=1.0 ....................................................... 47
Fig 21: Density Contours .................................................................................................. 48
Fig 22: Energy Contours .................................................................................................... 48
Fig 23: Pressure Contours ................................................................................................. 49
Fig 24: Velocity Contours of the u Component ............................................................... 49
Fig 25: Velocity Contours of the v Component ............................................................... 50
Fig 26: Pressure Contours ................................................................................................. 51
Fig 27: Velocity Magnitude Contours ............................................................................. 52
Fig 28: Density Contours .................................................................................................. 53
Fig 29: Pressure Contours ................................................................................................. 54
Fig 30: Velocity Contours of the u Component ............................................................... 55
Fig 31: Velocity Contours of the v Component ............................................................... 55
Fig 32: Physical Grid for the GUST3D Domain Decomposition Approach ................ 58
Fig 33: Directivity Patterns for a 2-D Gust Impinging a Cambered Airfoil .................... 59
LIST OF SYMBOLS

\(\vec{a}\) \hspace{1cm} \text{Gust amplitude vector}
\(\alpha\) \hspace{1cm} \text{Angle of attack of the airfoil}
\(c\) \hspace{1cm} \text{Chord of the airfoil}
\(\gamma\) \hspace{1cm} \text{Ratio of heat capacity}
\(c_0\) \hspace{1cm} \text{Sound speed}
\(\varepsilon\) \hspace{1cm} \text{Gust amplitude}
\(\vec{k}\) \hspace{1cm} \text{Wave number vector}
\(\omega\) \hspace{1cm} \text{Pulsation}
\(p\) \hspace{1cm} \text{Pressure}
\(\rho\) \hspace{1cm} \text{Density}
\(t\) \hspace{1cm} \text{Time}
\(T\) \hspace{1cm} \text{Temperature}
\(\vec{U} = (u, \nu)\) \hspace{1cm} \text{Upstream velocity}
\(\vec{x}\) \hspace{1cm} \text{Position vector}
\(s\) \hspace{1cm} \text{Entropy}
\((\xi, \eta)\) \hspace{1cm} \text{Generalized curvilinear coordinates}

Subscripts and Superscripts:
\(BC\) \hspace{1cm} \text{Boundary condition}
\(gust\) \hspace{1cm} \text{Gust parameter}
\(mean\) \hspace{1cm} \text{Mean flow condition}
\(ND\) \hspace{1cm} \text{Nondimensionalized symbols}
\(0\) \hspace{1cm} \text{Steady mean flow}
\(1\) \hspace{1cm} \text{x - component}
\(2\) \hspace{1cm} \text{y - component}
\(\infty\) \hspace{1cm} \text{Freestream conditions}
\(\cdot\) \hspace{1cm} \text{Unsteady perturbation}
Thanks to my thesis advisor Dr. Vladimir V. Golubev for his guidance and support.

Thanks to Dr. Reda Mankbadi for providing this project. And to Dr. Ray Hixon for his support when issues appeared with the code and for the explanations he provided on the way the code was written.

Thanks to Dr. Eric Perrell and Dr. Axel Rohde for being members of the thesis committee. Dr. Perrell provided very useful critics on the thesis’ draft.

From the Information Technology department, thank you to Rich Riendeau for his patience and knowledge. Rich taught how to manage the Linux resources of the Propulsion and Aerodynamics Computational Lab. And thanks to Glen Dosey who took over after Rich’s departure.

From ESTACA (Ecole Supérieure de Techniques Aéronautiques et de Construction Automobile, Paris, France), I would like to thank Miss Elodie Cavalier Donzel thanks to who I received funds for the thesis’ credit hours during the summer.

At last, but not least, very special thanks go to my parents Francoise Schultz and Michel Lessiau. And Alexandra Lascoux. Thank you for your love, support, understanding and patience...
ABSTRACT

AUTHOR: CLAIRE M. LESSIAU
TITLE: NUMERICAL ANALYSIS OF AN AIRFOIL RESPONSE TO AN IMPINGING GUST
INSTITUTION: EMBRY RIDDLE AERONAUTICAL UNIVERSITY
DEGREE: MASTER OF SCIENCE IN AEROSPACE ENGINEERING
YEAR: 2003

The BASS code, a nonlinear high-order prefactored compact code is validated on a benchmark problem. The nonlinear response of a loaded airfoil to an impinging vortical gust is investigated in the parametric space of gust intensity and frequency. Computational resources, involving a Linux cluster, were set up and maintained. The code was corrected and adapted to this particular problem. Results are compared with linear solution from the GUST3D solver.

Keywords: Computational Aeroacoustics, BASS, GUST3D, code validation, single airfoil benchmark problem, nonlinear solver.

RESUME

BASS, un code nonlinéaire de haute précision développé par la NASA, est validé sur un cas d’étude. La réponse nonlinéaire d’un profil d’aile porteur à une rafale est étudiée en fonction de l’amplitude et de la fréquence de la perturbation incidente. Les ressources informatiques nécessaires à la réalisation de cette recherche ont été mises au point et maintenues. Le code a également été adapté et corrigé pour ce problème particulier. Les résultats sont comparés à ceux donnés par la théorie linéarisée du solveur GUST3D.

Mots clefs: Computational Aeroacoustics, BASS, GUST3D, validation d’un code, étude d’un profil d’aile, solveur nonlinéaire.
INTRODUCTION

For the last thirty (30) years, the science of acoustics has expanded in many directions. Products of the jet age have added economic incentive to the solution of problems related to the generation and transmission of noise.

Such complex problems cannot be solved by hand. With the enhancement of computational tools, the field of computational aero acoustics (CAA) is focused on obtaining long-term, time-accurate numerical solutions to unsteady flow and acoustic problems. In order to accomplish this, a high-accuracy time-marching scheme is combined with high-resolution spatial derivatives.

Many flow fields that occur in aerospace applications involve upstream flow disturbances which propagate downstream, interact with structural components, and radiate sound. A particular problem of interest is the noise radiated when a vortical gust impinges on an airfoil with realistic geometry. This problem appears in helicopter and turbo machinery noise, e.g. rotor-stator interactions.

The purpose of this research is to test the ability of the BASS code, a computational fluid dynamics / computational aero acoustics (CFD/CAA) code developed by NASA, to accurately predict the unsteady aerodynamic aero acoustic response of a single airfoil to a two-dimensional, periodic vortical gust. This work involved setting up the computational resources needed for this purpose, generating the inputs, running and adapting the code to this particular problem, and analyzing the results.

These different steps are described in this paper. The problem is first defined. Then the way the BASS code works is explained. Numerical implementations that were achieved in the Propulsion and Aerodynamics Computational Lab (PACL) at Embry Riddle Aeronautical University (ERAU), Daytona Beach campus, FL, are developed before the results are analyzed.
1. **Single Airfoil Gust Response Problem**

The benchmark problem: “Single Airfoil Gust Response Problem”, from the sound generation by interaction with a gust benchmark set is described in this section.

1.1. **Presentation of the Problem**

Consider the airfoil configuration shown in figure 1. The airfoil has chord length c and angle of attack \( \alpha \). The upstream velocity is the combination of a uniform upstream velocity \( U_\infty \) with a small amplitude gust \( \vec{u}_\infty = \vec{a} \cos[\vec{k} \cdot (\vec{x} - \vec{U}_\infty t)] \):

\[
\vec{U} = U_\infty \hat{\vec{U}} + \vec{a} \cos[\vec{k} \cdot (\vec{x} - \vec{U}_\infty t)]
\]

Where \( \vec{x} = (x_1, x_2) \) denotes the spatial coordinates,

\( \vec{a} = (a_1, a_2) \) is the gust amplitude vector, with:

\[
a_1 = -\frac{\varepsilon U_\infty k_2}{|\vec{k}|}
\]

\[
a_2 = \frac{\varepsilon U_\infty k_1}{|\vec{k}|}
\]

\( \vec{k} \) is the wave number vector: \( \vec{k} = (k_1, k_2) \)

\( \varepsilon \) is a small parameter satisfying \( \varepsilon \ll 1 \)

*Fig 1: AIRFOIL IN A 2-D GUST – PARALLEL AND VERTICAL COMPONENTS.*
Equations are nondimensionalized. If the linear theory is applied, disturbance values will be nondimensionalized as described below:

\[
\begin{align*}
\bar{x} & \quad \text{by} \quad \frac{c}{2} \\
\bar{U} & \quad \text{by} \quad U_\infty \\
c_0 \text{ (sound speed)} & \quad \text{by} \quad U_\infty \\
\rho & \quad \text{by} \quad \rho_\infty \\
p & \quad \text{by} \quad \rho_\infty U_\infty^2 \\
T & \quad \text{by} \quad T_\infty \\
t & \quad \text{by} \quad \frac{c}{2U_\infty} \\
k & \quad \text{by} \quad \frac{2}{c}
\end{align*}
\]

For the following two cases, the gust response problem has to be solved for a Joukovsky airfoil in a two-dimensional gust with:

- \( k_1 = k_2 \) for reduced frequencies:
  - \( k_1 = 0.1 \)
  - \( k_1 = 1.0 \)
  - \( k_1 = 2.0 \)
  - \( k_1 = 3.0 \) (this case is not required for the benchmark problem; however, it allows comparison to previous results, as it will be discussed in section 4.).

- The nondimensional upstream velocity is \( \bar{U}_{ND} = \bar{U} + a_{ND} \cos[k \cdot \bar{x} - \omega t] \)

where \( a_{ND} = \left( \frac{a_1}{U_\infty}, \frac{a_2}{U_\infty} \right) = \frac{\epsilon}{|k|}(-k_2, k_1) \), with \( k_1 = k_2 \), then \( a_{ND} = \frac{\epsilon \sqrt{2}}{2}(-1,1) \).
Thus: \( |a_{ND}| = \varepsilon \) and \( \varepsilon \) is the gust amplitude relative to the mean flow:

- \( \varepsilon = 2\% \)
- \( \varepsilon = 20\% \) (this case is not required for the benchmark problem; however, it allows comparison to previous results, as it will be discussed in section 4.).

For case 1, the airfoil has a 12% thickness ratio, free stream Mach number \( M_{\infty}=0.5 \), angle of attack \( \alpha=2^\circ \), and a camber ratio of 0.02. For case 2, the camber ratio is zero and \( \alpha=0 \).

1.2. Airfoil Geometry

The airfoil geometry was generated as follows:

\[
\zeta_1 = r_0 e^{i\theta} + \zeta_0',
\]

where \( \zeta_0' = -\varepsilon_1 + i\varepsilon_2 \) is a complex constant.

Letting \( z = x + iy \) denotes the airfoil coordinates in the complex \( z \)-plane. The transformation:

\[
z = \left( \frac{\zeta_1 + \frac{d^2}{\zeta_1}}{\zeta_1} \right) e^{-i\alpha}
\]

transforms the \( \zeta_1 \) circle defined above into the desired airfoil shape. The \( \zeta_1 \) circle was discretized in \( \theta \), starting from \( \theta = -\beta \) and going to \( \theta = 2\pi - \beta \). Then, the above equation is applied to obtain the airfoil coordinates. The values \( \theta = -\beta \) and \( \theta = 2\pi - \beta \) map into the trailing edge point.
According to the case studied, different values for the above constants were used. They are defined in the following table.

<table>
<thead>
<tr>
<th></th>
<th>$r_0$</th>
<th>$\epsilon_1$</th>
<th>$\epsilon_2$</th>
<th>$d^2$</th>
<th>$\alpha$</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>0.273163765</td>
<td>0.02531002</td>
<td>0</td>
<td>0.0614314775</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Case 2</td>
<td>0.283382215</td>
<td>0.02185310</td>
<td>0.0614314775</td>
<td>0.034906585</td>
<td>0.039978687</td>
<td>0</td>
</tr>
</tbody>
</table>

The above procedure for generating the airfoil geometries generates Joukovsky airfoil of chord length 1, situated very nearly between $x = -0.5$ and $x = 0.5$, where $x$ is the nondimensional horizontal spatial coordinate. The airfoil geometries for the two cases are shown on figure 2.

For both cases, the discretized equations are expected to march in time until the solution becomes periodic. The following values must be computed:

1.3. Expected Answers

For both cases, the discretized equations are expected to march in time until the solution becomes periodic. The following values must be computed:
• on the airfoil surface: the root mean square (RMS) pressure $\sqrt{\langle p' \rangle^2}$

• in the far field: the intensity $\langle p' \rangle^2$ on a circle centered at the origin (the airfoil center) of radius:
  - $R = 1$ (one (1) chord length),
  - $R = 2$ (two (2) chord lengths),
  - $R = 4$ (four (4) chord lengths).

Also, the number of complete periods computed, the CPU time per period and the type of machine on which the calculations were run must be given.

As it is hardly possible to generate such a gust in reality, experimental results are not available. Results from this benchmark problem will be compared to the linearized theory results. These were validated by comparison with the analytical solution for gusts presenting small amplitudes and reduced frequencies. This will be discussed in section 4.
2. The BASS code

2.1. Purpose of the Code

The BASS code is designed to solve unsteady 2-D or 3-D nonlinear flow and noise problems in complex geometry domains, using structured multiblock grids.

To do so, the code marches in time, solving the nonlinear Euler or Navier-Stokes equations in generalized curvilinear coordinates on a block structured grid. The numerical formulation follows [2] and [3] and uses the low-storage 4th order 5-6 Low Dispersion and Dissipation Runge-Kutta scheme [5] for time marching, and pre-factored 6th order compact scheme and explicit boundary stencils for spatial derivatives [6]. A 10th order explicit filter is used at every stage of the Runge-Kutta solver to provide dissipation.

In order to reduce the amount of user time required to obtain a solution, the BASS code is designed to be run on a distributed-memory parallel computer, communicating between processors by message passing.

The major features that are designed into the code are to provide to the user a wide range of temporal and spatial differencing methods, and high-accuracy boundary conditions.

2.2. Mathematical Formulation

Governing equations are given. Then a background on the different discretization schemes that are implemented in BASS follows. The implementation of the gust is also detailed.

2.2.1. Governing Equations

The governing equations for flows such as the one studied for the benchmark problem are the unsteady Navier-Stokes equations. However, viscous effects are often confined to small regions of the flow, and the unsteady Euler equations can be solved instead. The governing equations are the 2-D nonlinear Euler equations, written in the Cartesian coordinates as:
\[
\frac{\partial Q}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} = 0
\]  
\(Q = \begin{bmatrix}
\rho \\
\rho u \\
\rho v \\
E
\end{bmatrix},
E = \begin{bmatrix}
\rho u \\
\rho u^2 + p \\
\rho uv \\
u(E + p)
\end{bmatrix},
F = \begin{bmatrix}
\rho v \\
\rho uv \\
\rho v^2 + p \\
v(E + p)
\end{bmatrix}
\]

where: \(\rho, u, v, p, E\) denote the fluid density, velocity, pressure and internal energy per unit volume

\[
p = (\gamma - 1) \left( E - \frac{1}{2} \rho (u^2 + v^2) \right)
\]  
Equation (5) comes from the conservation of the stored energy. The stored energy is composed of the internal energy, the kinetic energy and the potential energy which is neglected: \(E = E_i + E_v = C_v T + \frac{1}{2 \rho} (u^2 + v^2) = \frac{p}{(\gamma - 1) \rho} + \frac{1}{2 \rho} (u^2 + v^2)\).

The gust response is investigated for the Joukowksi airfoil, which requires recasting the equations in the generalized curvilinear coordinates, with the chain-rule curvilinear Euler equations written as:

\[
\frac{\partial Q}{\partial t} + \frac{\partial E}{\partial x} \frac{\partial \xi}{\partial x} + \frac{\partial E}{\partial \eta} \frac{\partial \xi}{\partial \eta} + \frac{\partial F}{\partial x} \frac{\partial \xi}{\partial y} + \frac{\partial F}{\partial \eta} \frac{\partial \xi}{\partial \eta} = 0
\]  
(6)

To march in time, the equation is re-written as:

\[
\frac{\partial Q}{\partial t} = - \left( \frac{\partial \xi}{\partial x} \frac{\partial E}{\partial \xi} + \frac{\partial \eta}{\partial x} \frac{\partial E}{\partial \eta} + \frac{\partial \xi}{\partial y} \frac{\partial F}{\partial \xi} + \frac{\partial \eta}{\partial y} \frac{\partial F}{\partial \eta} \right)
\]  
(7)

The code uses finite differences to obtain accurate spatial derivatives. To obtain high accuracy, the finite difference at a grid point uses data from seven neighboring grid points.
2.2.1.1 Spatial Discretization

The solver uses a sixth order pre-factored compact scheme for spatial differencing, with explicit stencils at the block boundaries. The unresolved components in the solution are damped using an explicit tenth order filter which simulates dissipation.

**Fig 3: SPATIAL DISCRETIZATION**

The interior sixth order compact differencing scheme solves for the spatial derivatives. A standard sixth order compact scheme difference is written as:

$$
\frac{1}{5} \left[ \frac{\partial f}{\partial x} \bigg|_{i+1} + \frac{\partial f}{\partial x} \bigg|_{i-1} \right] + \frac{4}{5} \left( \frac{\partial f}{\partial x} \bigg|_{i} \right) = \frac{1}{\Delta x} \left[ \frac{1}{60} (f_{i+2} - f_{i-2}) + \frac{7}{15} (f_{i+1} - f_{i-1}) \right]
$$

In order to reduce the stencil size (from five (5) to three (3) points), a pre-factorization is used which splits the above equation into forward and backward biased stencils which are solved separately and then added together:

$$
\alpha \left( \frac{\partial f}{\partial x} \bigg|_{i+1} \right) + \left( \frac{\partial f}{\partial x} \bigg|_{i} \right) = \frac{1}{\Delta x} \left( \frac{\beta}{2(1-\alpha)} (f_{i+1} - f_{i}) + \frac{1-\beta}{2(1-\alpha)} (f_{i} - f_{i-1}) \right)
$$

$$
\alpha \left( \frac{\partial f}{\partial x} \bigg|_{i-1} \right) + \left( \frac{\partial f}{\partial x} \bigg|_{i} \right) = \frac{1}{\Delta x} \left( \frac{\beta}{2(1-\alpha)} (f_{i} - f_{i-1}) + \frac{1-\beta}{2(1-\alpha)} (f_{i+1} - f_{i}) \right)
$$

$$
\frac{\partial f}{\partial x} \bigg|_{i} = \frac{\partial f}{\partial x} \bigg|_{i}^{f} + \frac{\partial f}{\partial x} \bigg|_{i}^{b}
$$

where: \( \alpha = \frac{1}{2} - \frac{1}{2\sqrt{5}} \) and \( \beta = \alpha^2 - \frac{\alpha}{30} \)
At the boundary, explicit stencils are used. In this way, each block derivative can be calculated separately. The explicit stencils are also defined as forward and backward stencils, and used to start and end the compact difference sweeps.

Since the equations that are to be solved are nonlinear, and that arbitrary grids will be used, the waves and flow gradients that are resolved by the scheme at one step may well become unresolved as the computation proceeds. Since unresolved waves will only contaminate the solution and destroy its accuracy, they have to be removed while not damping the resolved ones. To do so, a tenth-order explicit filter is employed at each stage. It is used only on the conserved variables, before the fluxes are calculated.

At a boundary where a one-sided stencil is appropriate, the order of the filter reduces to a minimum of fifth-order. So the interior filter used is modified four (4) points away from the boundary and closer. Higher order filters are possible, but stability is compromised by the large one-sided stencils at the boundary.

2.2.1.2 Time Discretization

Acoustics problems involve accurate time-dependent wave propagation. The dissipation and dispersion properties of the numerical method are very important for computing wave solutions of systems of partial differential equations. The explicit Runge-Kutta methods are widely used to discretize the time derivative because of their advantages that include flexibility, large stability limits, and ease of programming. Dissipation and dispersion properties of the Runge-Kutta methods depend on their coefficients. They can be optimized for the convective wave equation, obtaining what is called low-dissipation and dispersion Runge-Kutta methods.

For large size physical problems, memory requirements can be decreased using special Runge-Kutta schemes that can be written such that only 2N-storage is required, where N is the number of degrees of freedom of the system (i.e., number of grid points*number of variables). To design such Runge-Kutta schemes, enough free coefficients must exist such that additional conditions hold between them. It was shown that fourth-order 5-6 Runge-Kutta methods can be written in 2N-storage form [5].
The optimized low-storage Runge-Kutta scheme used in the code is called the 5-6 scheme because it alternates between five (5) and six (6) stages per time step.

The fourth order Runge-Kutta method is one of the standard algorithms to solve differential equations. It is generally considered to provide an excellent balance of power, precision and simplicity to program. To better understand the way BASS handles the time discretization, some insights about the Runge-Kutta method are given below.

Let us start with the original differential equation and integrate it formally.

\[
\frac{dy}{dt} = f(t, y) \Rightarrow y(t) = \int_0^t f(t, y) dt
\]

\[
\Rightarrow y_{n+1} = \int_0^{t_{n+1}} f(t, y) dt + \int_{t_n}^{t_{n+1}} f(t, y) dt
\]

\[
\Rightarrow y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y) dt
\]

(13)

The task to perform is now a differentiation instead of an integration. To do this, \( f(t) \) is expanded in a fourth order Taylor series around the midpoint of the integration subinterval: \( (t_{n+1/2}, y_{n+1/2}) \). Here only the second order is described:

\[
f(t, y) \approx f(t_{n+1/2}, y_{n+1/2}) + (t - t_{n+1/2}) \frac{df}{dt}
\]

(14)

The integral of \( (t - t_{n+1/2}) \) vanishes when evaluated about the midpoint, then:

\[
f(t, y) \approx f(t_{n+1/2}, y_{n+1/2})
\]

\[
\Rightarrow y_{n+1} = y_n + hf(t_{n+1/2}, y_{n+1/2}) = y_n + hf(t_n + \frac{h}{2}, y_{n+1/2})
\]

(16)

This algorithm cannot be applied immediately since it requires a knowledge of \( y_{n+1/2} \) which is not in the scheme of things. Thus, it is approximated with Euler's algorithm:
\[ y_{n+1/2} = y_n + \frac{\Delta y}{\Delta t} = y_n + \frac{h}{2} \frac{dy}{dt} = y_n + \frac{h}{2} f(t_n, y_n) \] (17)

\[ y_{n+1} = y_n + hf(t_n + h/2, y_{n+1/2}) = y_n + hf(t_n + h/2, y_n + \frac{h}{2} f(t_n, y_n)) = y_n + hf(t_n + h/2, y_n + \frac{k_1}{2}) \Rightarrow y_{n+1} = y_n + k_2 \] (18)

where: \[ k_1 = hf(t_n, y_n) \quad k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \]

The fourth order Runge-Kutta follows exactly the same procedure. However, as a fourth order Taylor serie expansion is used, it requires four gradients (or \( k \) terms) to calculate \( y_{n+1} \):

\[ y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \] (19)

where: \[ k_1 = hf(t_n, y_n) \quad k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \]

\[ k_3 = hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \quad k_4 = hf(t_n + h, y_n + k_3) \]

At the moment, only RK56 is implemented. This is a single time stepping method: all spatial grid points march at the same time. Another method of the same kind is being implemented: fourth order Adams-Bashforth (AB4), single time step. AB4, multiple time step is to be coded: each grid block will march at some multiple of the minimum time step.

2.2.2. Numerical Implementation of the Gust

In the current work, the nonlinear gust response is examined for a series of imposed gust intensities and frequencies.

The mean flow is defined far upstream from the airfoil as:

12
\[
\bar{\rho} = 1 \\
\bar{u} = M \\
\bar{v} = 0 \\
\bar{p} = \frac{1}{\gamma} \left( \bar{\rho}' \right)
\]

where: \( M \) is the upstream mean flow Mach number: \( M = 0.5 \)
\( \gamma \) is the ratio of specific heat: \( \gamma = 1.4 \)

In nondimensionalized form, from equation 1: \( \bar{U}_{\text{gust}}^{ND} = \bar{a}_{\text{ND}} \cos[\vec{k} \cdot \vec{x} - k_x t] \).

According the mean flow velocity, each vortical gust harmonic is initially imposed on the mean flow with the following distribution:

\[
\begin{align*}
\bar{u}_{\text{gust}} &= -\varepsilon \times \cos(k(x + y) - \omega t) \\
\bar{v}_{\text{gust}} &= \varepsilon \times \cos(k(x + y) - \omega t)
\end{align*}
\]

where: \( \varepsilon \) is the gust intensity relative to the mean flow,
\( k \) is the gust wave number in the \( x \) and \( y \) directions,
\( \omega \) is the imposed gust frequency: \( \omega = k \bar{U}_m \).

### 2.2.3. Structure of the Code

The code is written in Fortran 90 to take advantage of the improved memory management and data structures of Fortran 90 as compared to Fortran 77. This code had already been validated on benchmark problems, and had been found to perform very well.

However, its structure was totally re-done, in part to enhance parallel capabilities of the code, and also for development reasons: as more and more people are involved in programming and testing this code which aims at being very extensive, a robust structure was to be adopted. This new version was available for download on the twenty fourth
(24th) of March, 2003. Its structure is described on figure 4 and a detailed description follows:

- **/CAA-work**: this is the root directory.

- **/CAA-work/Makefile**: all makefiles are made from here. Subdirectories—which are not detailed here, because they only serve the purpose of the current analyses—were created as explained in section 4.1.1.1.

- **/CAA-work/Code_Run**: this is where the code is run. For convenience, restart, input and grid files are stored in the corresponding subdirectories. Subdirectories of **/CAA-work/Code_Run** were also created to house the BASS executable just for the purpose of the current analyses. Naming conventions will also be explained in section 4.1.1.1.

- **/CAA-work/Pre-Process**: this directory is not used by BASS. It contains some routines to pre-process grid and input files for cascade problems: one that reads single-passage grids and outputs multiple-passage grids, one that reads an input file that is designed for single-passage and modifies it to become a multiple-passage input file.

- **/CAA-work/Post-Processor**: this directory is not used by BASS. It contains some routines to post-process results from BASS: one that outputs the frequency amplitudes of all grid point locations as produced by a Fast Fourier Transform (FFT) as well as the real part of the complex amplitude which provides a snapshot of the amplitudes at the initial time.

- **/CAA-work/Solver**: here live all the routines that allow the computation of the results. Its subdirectories and the functions they perform are briefly described below, as directory names are self explained.
  
  - Initialize_Flow: this directory only contains the flow initialization file. It will be described later on in section 4.1.1.3.
  
  - Error_Checking: this directory contains routines for patch and block checks.
- **Main_Routines**: this is where the start, run and stop are defined. The actual time stepping occurs in the main routine.

- **Solver_Data**: this is where general data are defined: coefficients for the different schemes (e.g. the sixth order compact scheme), as well as MPI and input/output information.

- **Solver_Definitions**: in this directory are defined the files that will be used during the computation for the different modules (boundary conditions, numerical schemes...).

- **Solver.Include**: this directory only contains the Format Statement.f90 file. It formats the output.

- **Solver_Parameters**: here are defined the constants for the $k - \varepsilon$ turbulent model as well as the ones for Sutherland's law for laminar viscosity.

- **Input_Output**: this directory contains two (2) subdirectories where everything about reading data—from the input file for instance—and writing results is handled.

- **Topology**: routines contained in this directory link the code to the topology.

- **Flow_Solver**: this is where the equations are actually defined and solved: time stepping scheme, spatial derivative module, boundary conditions... For the spatial derivative module, the choice made was to standardize the data structure, this way, a single module can be used to perform all the spatial derivatives calculated in the code.
Fig 4  STRUCTURE OF THE CODE
2.3. Grid

The mesh was generated with the GridPro™ [22] commercial package by Program Development Corporation. A background about this code focused on the aspects needed for BASS is given to help understand how the grid was generated. Then a description of the spatial discretization follows.

2.3.1. Background on GridPro™

GridPro™ [22] is a very powerful system that generates block structured hexahedral meshes, once the geometry has been prepared. GridPro™ partially automates topology generation by reducing the user task to the generation of a coarse wire frame of the topology in which only imprecise corner and edge information is required; while the blocks and block faces are automatically generated from the wire frame.

The GridPro™ software consists of the main grid generator (a batch code), a graphical user interface (GUI), and about twenty (20) utility programs. The main code is controlled by the GUI or may be run separately in batch mode. In either mode, the main code is controlled by a script written in Topology Input Language (TIL) code.

The TIL reduces the labor involved in domain decomposition by providing the user with a programming language of sorts, allowing for complex configurations to be built up from simpler components. Eiseman's implementation of TIL, included in the GridPro™/az3000 package [22], is able to automate surface grid generation, zone construction, and intersection of surfaces. Simple components are built up from primitive elements: surfaces, corners, and vectors. Components which represent more complicated configurations can be built up by including simpler components in their description along with the primitive elements.
The code has very powerful domain decomposition (blocking) and grid generation functions. It features excellent point spacing and grid smoothing functions. It produces really smooth and high quality grids. GridPro™ contains proprietary algorithms that optimize orthogonality throughout the entire grid. The grid generation process is iterative. Grid points are allowed to move along the edges (2-D) or surfaces (3-D) until their surface or volume grid converges.

The major weakness of GridPro™ is the lack of geometry preprocessing capability. However, this was not a big issue considering the geometry that was to be meshed.

It must also be stated that the learning curve is pretty steep.

The way of generating a grid with GridPro™ follows:

1. Import Surfaces
2. Approximate position of blocks
3. Add fake surfaces to influence grid points
4. Choose densities in each block
5. Let GridPro™ process

2.3.2. Spatial Discretization of the Computational Domain

For the analysis of the nonlinear gust response problem, a C-grid 2D topology is used for a 12% thick Joukowski airfoil, either cambered or symmetric. One could argue that an O-grid would have been more appropriate. It would have better defined the trailing edge geometry. However, the C-grid was chosen due to the excessive number of grid points and the associated small time steps that the O-grid would have required to accurately resolve the sharp trailing edge.

For every case, two different domains were created and meshed. An algebraic clustering was used around the profile in the normal direction ($\Delta n = 0.01$) and near the trailing edge ($\Delta x = 0.01$). A stretching ratio of 1.05 was then used to expand the grid to the farfield. Available geometries are:
• The larger domain extends ten (10) chords away from the surface in each direction and its grid has $605 \times 240$ (145200 total) points,

• The smaller one extends five (5) chords away from the surface in each direction and its grid has $433 \times 125$ (54125 total) points.

As it was previously shown that the effect of larger domain sizes on the near-field solution was invariant once the domain boundary was at least ten (10) chords away from the airfoil [2], only the first grid was used.

However, as nonlinearities are expected with the highest frequencies and amplitudes of gust, it may be necessary to refine this domain, as eight (8) mesh points per wavelength are needed to be able to resolve the nonlinearities. This will be discussed in section 4.1.3.1.

Grids do not present any singularities, hence only one block was created, as singularities are allowed at the interface between different blocks only.

As it can be seen on the following figures, grids are smooth and orthogonal, hence accuracy and calculation stability from the CFD/CAA code are most likely improved.

Another file containing two (2) blocks is available. Blocks are divided along the (1,0) vector. It allows some fancy plots: one can represent the contours of one of the variables on the upper part, and another one, or the mesh below, as shown on figure 13.

Fig 5: $605 \times 240$ POINT GRID FOR THE CAMBERED AIRFOIL
2.3.3. **The “decomp” Code**

The one (1)-block grid was divided into the sixteen (16)-block grid shown below, using the “decomp” code. This code serves the purpose of decomposing a block into as many blocks as needed. By asking for sixteen (16) blocks, analyses were run on the sixteen (16)-node cluster with the parallel version of the Very Large Eddy Simulation (VLES) code (cf Section 4.1.).

It will also be used for the BASS code. Before being able to run it in parallel, a parallelized version of BASS has to be delivered. Then, “decomp” will be used, and the BASS input file will be modified accordingly. As “decomp” divides the domain into many blocks, connectivity patches must be defined at the different interfaces.

![Decomposition of the Computational Domain: 16 Blocks](image)

*Fig 6: Decomposition of the Computational Domain: 16 Blocks.*
2.4. Boundary Conditions

It has been established over the last few years that the accuracy requirements of the boundary conditions as well as the solution schemes are much more stringent for aeroacoustics problems [16]. For this reason, internal boundary conditions are different from the ones used in CFD codes, and external boundary conditions have to be carefully implemented.

2.4.1. External Boundary Conditions

As the spatial domain is unbounded, a need for artificial boundary conditions arises to make the computational domain finite: numerical boundary treatment needs to be applied at the external boundaries to depict conditions at infinity.
The artificial boundary conditions should be such that the numerical boundaries are transparent to out-going disturbances, while preventing the entrance of nonphysical waves: waves should exit the computational domain without significant reflections.

In fact, an incorrect specification of the boundary conditions gives rise to spurious reflections, which are entirely numerical in nature. These numerical reflections propagate into the interior domain as the solution progresses in time, eventually contaminating the entire solution.

As the Euler equations support three (3) types of wave, a combination of acoustic, vorticity and entropy waves disturbs the flow. By nature acoustic waves are radiated and propagate at sound speed relative to the mean flow. Vorticity as well as entropy waves are frozen patterns convected downstream by the mean flow. Because of the presence of the three (3) types of wave disturbances, each having distinct propagation characteristics, the outgoing disturbances present at the inflow and outflow boundaries are very different. Thus the need for two (2) different external boundary conditions: the acoustic radiation one at the inflow and the Tam & Webb [7] one at the outflow.

2.4.1.1 Radiation Boundary Condition

At an inflow boundary, the only outgoing disturbances are acoustic waves. The conventional acoustic radiation (ACRAD) condition based on the asymptotic analysis of the wave equation applies. This treatment was tested and found to be robust, producing no reflection at radiation boundaries. It is applied to the upstream boundary so that the outgoing disturbances leave the computation domain smoothly.
The ACRAD boundary condition is the standard linearized farfield condition. A uniform mean flow is assumed. The way this boundary condition is implemented in the code is as follows:

\[ \{Q\}_{t} + V(\theta) \times \left( \frac{x-x_{s}}{R} \{Q\}_{x} + \frac{y-y_{s}}{R} \{Q\}_{y} + \frac{1}{2R} \{Q - Q_{\text{mean}}\} \right) = 0 \]  (21)

with:
\[ R = \sqrt{(x-x_{s})^{2} + (y-y_{s})^{2}} \]

\[ V(\theta) = \frac{x-x_{s}}{R} u_{\text{mean}} + \sqrt{a_{\text{mean}}^{2} - \left( \frac{y-y_{s}}{R} u_{\text{mean}} \right)^{2}} \]

In this aero acoustics problem, unsteady incoming vorticity waves had to be defined. The inflow boundary condition also allows the incoming disturbances to propagate in the domain. The gust was coded into the ../Solver/Flow_Solver/Boundary_Conditions/Acoustic_Radiation/ACRAD_inflow.f90 file. Its parameters were defined, and formulae were set up to allow the propagation and dispersion of the gust as explained in section 4.1.1.4.

2.4.1.2 Outflow Boundary Condition

At the outflow, the outgoing disturbances are a combination of acoustic, entropy and vorticity waves. Tam and Webb [7] derived the radiation boundary condition by means of the asymptotic solutions of the governing equations, while accounting for density and velocity variations due to others than acoustic waves.

The Tam & Webb boundary condition is similar to the standard linearized farfield condition. It is only applied to the pressure:

\[ \{p\}_{t} + V(\theta) \times \left( \frac{x-x_{s}}{R} \{p\}_{x} + \frac{y-y_{s}}{R} \{p\}_{y} + \frac{1}{2R} \{p - p_{\text{mean}}\} \right) = 0 \]  (22)
The code implements this boundary condition as written above, and corrects the energy time derivative using:

\[
\frac{\partial E}{\partial t}\bigg|_{\text{corrected}} = \frac{1}{\gamma - 1} \frac{\partial p}{\partial t}_{\text{uncorrected}} + \frac{1}{2} \frac{\partial p}{\partial t}_{\text{uncorrected}} (u^2 + v^2) \quad (23)
\]

2.4.2. **Internal Boundary Conditions**

There are internal boundaries inside the computation domain. On these internal boundaries, conditions simulating the presence of the airfoil and its wake with specific acoustic characteristics are applied.

2.4.2.1 Wall Boundary Condition

On solid surfaces, the slip, no-through-flow boundary condition is applied. Wall boundary points are treated as being in the interior of the domain. However, there is a difference with the usual wall boundary condition used in CFD where reduced equations are solved as high order finite difference schemes are used in CAA. This way, there is numerical dispersion, but spurious solutions that have no relationship to the original partial differential equation are induced. For the accuracy at the wall to be consistent with the solver high accuracy, a different approach is used.

For aero acoustics problems, the spurious waves are of two types:

- propagating waves with short wave lengths,
- spatially damped waves.

Thus when an acoustic wave pulse impinges on a wall, in addition to the reflected waves, both spurious short waves and spatially damped waves which decay as they propagate away from the wall will also be emitted in a high order finite difference solution.

There are two major difficulties in developing wall boundary conditions for high order finite difference schemes.
First, high order finite difference equations require additional boundary conditions—beyond the physical boundary conditions of the original problem—to define a unique solution. The way to handle the need for these boundary conditions must be found so that only very small amplitude spurious waves are excited.

Second, in the discretized system, each flow variable at a point is governed by an algebraic equation discretized form of the partial differential equation. The number of unknowns is exactly equal to the number of equations. Thus there will be too many equations if it is insisted that the boundary conditions at the wall mesh point are satisfied also.

Tam and Dong [17] proposed a method to include the wall boundary points in the computational domain. To do this, they allowed the flow at the wall points to evolve as interior points, solving the tangential flow equations in the usual way while setting the boundary condition using only the normal spatial derivatives. This eliminates the need for extra boundary conditions. To provide enough unknowns to enforce the physical wall boundary conditions as well as to allow the discretized governing equations to be satisfied at mesh points on the wall, they suggested including ghost values at ghost points. Ghost points are mesh points immediately outside the computation domain which correct the normal derivatives of the pressure so that the normal velocity at the wall at the next time level will be zero (0).

Hixon [19] recasted the flow equations into generalized curvilinear coordinates and extended it to nonlinear, viscous walls.

The reason for this particular approach is due to the separation of the filtering and spatial derivative functions—since the filter has no boundary condition, it may well change the normal velocity at the wall.
For inviscid wall, the boundary condition that is implemented in the code is:

\[ \vec{V} \cdot \hat{n} = 0 \]  

or

\[ \frac{\partial \vec{V}}{\partial t} \cdot \hat{n} = 0 \]  

where \( \hat{n} \) is the unit vector normal to the wall.

This condition is set using the normal derivative at the wall, while leaving all other derivatives fixed. It allows the solver to represent the wall to the accuracy of the grid and solver.

2.4.2.2 Connectivity Boundary Condition

A connectivity condition is applied to its wake. This condition just connects two (2) patches together. It is also widely used when the decomposition of the grid is performed with "decomp".
3. Numeric Implementation

All the research work has been conducted in the PACL at ERAU. Actually, the first step was to set up correctly the different resources needed for the research: re-imaging the systems, re-installing the software, fixing the hardware, maintaining the lab. A description of the resources available follows.

3.1. Resources

Previously, the VLES code was designed to run in parallel and numerical solutions were obtained through parallel implementation on the Beowulf-type Athlon cluster. The code was written to demonstrate optimum performances on such a cluster. The fewest and the longest possible messages are passed to the nodes. So cluster with a high computation speed and a lower communication speed are more appropriate to run this code. The BASS code was re-written in part to enhance its parallel capabilities. Even if the parallel version is still to be delivered, computational resources are up and ready to run.

Several systems are available in the PACL to run the code: Linux only machines (explorer micro systems), dual-boot Dell PCs, and Head0 and the Linux cluster. This way, codes can be studied, tested and run. Pre and post processing tools are also available, such as grid generators and visualization software.

The way these systems are linked is described on figure 8. Their implementation is explained in the following subsections. As hardware evolves quickly, the configuration of the lab’s network changes often (new computers are bought, older ones are thrown away...).

So the way to set up the resources was implemented in order to facilitate the management of the lab, mainly by minimizing software installation. To do so, software is installed only once on Head0, new IP addresses just have to be added in or removed from “/etc/exports” on the head node when a piece of hardware changes (cf section 3.2.)...
Linux 8.0 from Red Hat is run on Dell's computers, and 7.2 or 7.3 is used on the oldest machines.

3.2. Head0 and the cluster

Head0 or figaro is the controller node. Users login to Head0 to run their parallelized codes. Some tools were implemented in order to allow users to submit and monitor the cluster jobs. It serves filesystem to cluster nodes via Network File System (NFS). NFS was developed to allow machines to mount a disk partition on a remote machine as if it were on a local hard drive. This allows for fast, seamless sharing of files across a network.
Cluster nodes are eight (8) dual processor nodes –1.4GHz AMD– stored on a rack, located in LB170.

Head0 is linked to the compute nodes on a private network, and it acts as a firewall for compute nodes.

The following software is installed on the head node for general purpose:

- NFS Server plus automounter
- RSH (remote shell: allows node interoperability without SSH overhead, while providing secure encrypted communications over the network)
- OpenPBS (Open Source version of the Portable Batch System, a flexible batch queuing and workload management system to start and monitor jobs)
- Ipchains (for firewalling and security)

The following software is installed on the head node for special applications:

- GridGen (another grid generator delivered with a floating license)
- Matlab (university license)
- Absoft Fortran 90/95
- MPICH 1.2.2 (a portable implementation of Message Passing Interface (MPI), with shared memory enabled)

This software is installed under “/export/bin/”. The “/export” directory is exported to the lab’s network as specified in the “/etc/exports” file which gives the IP addresses of the computers to which the directory should be exported as well as the path of the concerned directory.
However, it should be stressed that these codes are not supposed to be used from the head node. It is installed here only to make it easier to manage the software resources. This way, software is installed only once on the head node. The "/export" directory is mounted to the other Linux systems.

On the compute nodes, the following software is installed:

- NFS Client plus automounter
- RSH
- OpenPBS
- Configure paths correctly to access software installed on NFS shared folders
- Configure routes correctly for inter-node communication

Software was customized:

- jtop (to monitor cluster status by calling jstat, sleeps, clears screen, loops),
- pullaccounts (to synchronize data between the cluster and the head node: downloads the password and shadow file from the head node over the private network; downloads MPICH node list used with mpirun; synchronizes the system clock from the head node; runs every night at midnight),
- noderoutes (used instead of a standard network startup; sets routes and backup routes for all six (6) interfaces based upon the node name; /etc/noderoutes contains the scripts to bring each node up or down according to the way they are connected),
- usersync (on system boot it simply calls pullaccounts.pl).
Regarding the node interconnection, nodes are nearly fully connected—all nodes have a direct connection to all other nodes except two (2) of them. Interconnect gives a ½ gigabit connection plus out-of-band management (because at the time of the implementation, going gigabit was too expensive).

3.3. Other Linux Systems

Explorer micro systems were bought and implemented for the first study of this benchmark problem with the VLES code. They served their purpose and are now being replaced by Dell PCs because they are more reliable.

The only difference between explorer micro systems and Dell PCs—apart from obvious hardware ameliorations—is that the first ones run Linux only whereas the others are dual-boot. Both systems run pullaccounts, usersync and automount the head node. Each system was put in the head node’s firewall configuration.

These systems are used for code development, testing, and visualization of the results. For this reason, the following software can be used from the explorer micro systems:

- GridPro™ [22] to create the structured grid (locally installed on Paisley as only one (1) node locked license is owned)
- TecPlot to display the results and post-process them (locally installed on Red as only one (1) node locked license is owned)
- MPI to be able to run BASS
- Absoft FORTRAN to compile the code
- Gridgen
- Matlab
All software is either installed or linked to its installation directory (cf section 3.2.) under 
"/usr/local/". The path is set up under the "/etc/profile" file to avoid typing the full path 
for the executable everytime it is to be run.

Issues appeared with TecPlot: the license manager has to be re-launched very often. And 
root permission is needed to do so. So a "sudo" command is now used. This gives root 
permission to anyone who needs to re-start the TecPlot's license manager by typing 
"sudo /usr/local/tecplot/setuplic".

3.4. Computational Effort

The computational effort depends on the running time. And the running time is induced 
by the propagation phenomena, so it depends on the physics involved in the problem.

3.4.1. Physics Involved

As explained in section 2.4.1., this problem involves the propagation of different types of 
waves. The nondimensionalized time (NDT) gives an idea of the time needed to get a 
stable response from the airfoil. It is here computed from the physical time.

The angular velocity of the motion $\omega$ is expressed in radians per second. Hence, $\omega \times t$ is 
nondimensionalized. By considering the variables nondimensionalized as stated in section 
1.1., then:

\[ \omega \times t = \omega_{ND} \times t_{ND} \]

\[ \omega_{ND} = \omega \frac{c}{2U_\infty} \Rightarrow \omega = \omega_{ND} \frac{2U_\infty}{c} \]

\[ \omega t = \omega_{ND} \frac{2U_\infty}{c} t = \omega_{ND} t_{ND} \Rightarrow \frac{t_{ND}}{t} = \frac{2U_\infty}{c} = 1 \] (26)

Hence, the NDT is equal to the physical time.
As the different waves do not propagate at the same speed, they will exit the domain at different times.

\[ \text{Acoustic Radiation} \]
\[ \text{Gust} \]

\[ \text{Fig 9: RADIATED AND CONVECTED WAVES THROUGHOUT THE DOMAIN} \]

It is now possible to determine the NDT when the first waves of each kind will exit the domain:

- The gust propagates at mean flow velocity:

\[
NDT_{\text{gust}} = \frac{10c}{U_\infty} \Rightarrow NDT_{\text{gust}} = \frac{10}{0.5} \Rightarrow NDT_{\text{gust}} = 20 \quad (27)
\]

- The acoustic radiations propagate at sound speed:

\[
NDT_{\text{upstream}} = \frac{10c}{c_\infty - U_\infty} \Rightarrow NDT_{\text{upstream}} = \frac{10}{1 - 0.5} \Rightarrow NDT_{\text{upstream}} = 20 \quad (28)
\]

\[
NDT_{\text{downstream}} = \frac{10c}{c_\infty + U_\infty} \Rightarrow NDT_{\text{downstream}} = \frac{10}{1 + 0.5} \Rightarrow NDT_{\text{downstream}} = 6.67 \quad (29)
\]
First of all, this means that for \( NDT = 6.67 \) and on, as well as for \( NDT = 20 \) and on, boundaries will receive a particular attention to make sure that there are no reflections and that they are set up properly. The outflow boundary (Tam & Webb) will be checked first, then, the inflow (ACRAD) one.

It also means that the computational time will be at least up to \( NDT = 200 \). Then, it will be based on experience gained from the previous researches [1] and analyses.

### 3.4.2. Computational Time

From the NDT analyses in section 3.4.1. and previous studies [2], it was assumed that a NDT of 280 corresponds to a stabilized flow. A NDT of at least 360 is however necessary when spurious modes are studied [1]. The reason is that a stable solution has to be found for imposed gust frequencies and higher harmonics and combination tones.

In this paper, results correspond to \( NDT = 360 \). This corresponds to more than two (2) weeks of running time on Plaid, Paisley or Red (the cluster was not used because the parallelized version of the code was not available). To compare to the potential time needed with the parallel version, a sixteen (16)-hour numerical run was required to march to the same NDT. And the code was re-formatted to enhance its parallel capabilities.

For the moment, only the lifting airfoil (case 1) has been studied. It is expected that analyses on the non-lifting one (case 2) will converge faster, as observed previously [1].

Here on figure 10 is a graph of the speed up that can be expected from the cluster [1].

![Graph of speed up expected from the cluster](image)

**Fig 10:** SPEEDUP EXPECTED FROM THE CLUSTER.
4. Results

The evolution of the BASS code is discussed first, including a comparison between present results and previous ones from the VLES code. Then, results from BASS are compared to results from another CFD/CAA code: GUST3D (cf Section 4.2.).

4.1. Evolution of the BASS Code

The way the code was handled is explained. Then, results from BASS are analyzed and compared to the ones from its previous version VLES. This way the consistency of BASS can be tested.

4.1.1. Troubleshooting

As the goal of this paper is to evaluate the performances of the BASS code which is still in the process of being coded more extensively, several problems appeared while running it. Here follow some details about the troubles encountered and the way they were assessed.

Also, as this research involves several research works conducted by different students and doctors, this paper tends to give a reference background and way of proceeding to facilitate future work. So the way of running the code and the conventions used are detailed below.

4.1.1.1 Makefile

The code downloaded consisted in several directories, including a makefile one. Originally, this directory contained only a file named "makefile". This is a description file which contains every command and dependency explicitly, as well as the compiler command that makes the executable "BASS.exe".
Before running the BASS executable, it has to be created, or made. With the above architecture, different files from the code can be modified or created, then one just has to specify which files have to be used for the intended run into the makefile, and a specific executable will be made. The first step in the process of making the executable is to check if any of the code files is newer than BASS.exe. If so, only the newer files and their dependencies are re-compiled. Then "o" and "mod" files are generated in the makefile directory from the "f90" files. And the executable is created and moved to the target directory.

Several errors were spotted and corrected.

- The path to the MPI package had to be changed to "/usr/local/mpich/include/mpi.f.h".

- The way to compile the files was customized to the lab’s system. The f90 compiler was used. An optimization -o3- was used to save time during the iteration process.

- The “Format_Statement” file could not be reached because its path was wrong. So it was changed to "../Solver/Solver_Include/" in a good amount of files.

- Two (2) time variables were not defined correctly –inconsistency in the input/output definitions– in the

  
  \texttt{INTENT(OUT) current\_time} \\
  \texttt{INTENT(INOUT) time0\_RK}

The way of creating the executables was standardized. A new directory corresponding to the type of analysis (1D or 2D gust, type of initialization, optimized compiler or not...) to run was created under Code_Run. Let’s name it “type-of-analysis”. Then, the more appropriate Makefile directory was copied as “Makefile_type-of-analysis” and “.o” and “.mod” files removed so that just the makefile file remained. It was customized to the
type of analysis wanted, and the path to show where to create the BASS executable was pasted into the makefile `./Code_Run/type-of-analysis`.

This way, several makefiles can be made at the same time, because they do not live in the same directory, and BASS.exe files are not generated at the same place. It is also easier to keep track of the differences between the different analyses, so a different user can find his way.

### 4.1.1.2 Inputs

After making the executable, the “type-of-analysis” directory only contains the BASS.exe file. The input `-Input_File.dat-` and grid files have to live at the same place as the BASS.exe file. So the input file has to be customized to be consistent with the type of analysis and results wanted.

The input file defines:

- the geometry of the problem through the grid file
- the initial guess of the problem through the restart file
- the boundary conditions to be used (which one and where)
- the time stepping method to be used (at this time, RK56 is the only available)
- the different flow constants
- the name of the output files, how often to save them and how many to keep
- the name of the restart file, and how often to save it
- the frequency at which the flow time is computed

Once the input file is set up and the corresponding grid file imported to the same directory, the code can be run on only one processor (for now): “mpirun –np 1 BASS.exe”.

37
4.1.1.3 Flow Initialization

The initialization process consists in giving an initial guess. Theoretically, this should not influence the final results, but just the time to attain convergence. There are basically two ways to look at the problem: either the mean flow is computed and it becomes the initial guess for the gust problem, or the gust is added to the mean flow for the initialization, and the flow is computed. These two ways were explored.

In the first case, the flow was initialized with a density equal to 1.0 and a Mach number of 0.5. The analyses were run so that the influence of the airfoil on the mean flow became a given. This result became the initial guess for the gust problem: the gust was added to the inflow boundary condition, and the initialization file was not accounted for.

Considering the results, it seemed that the response was unstable. It was assumed that the gust was just hitting the boundary too strongly—especially for a 20% intensity—to be handled properly by the code. So the gust was introduced the same way, but gradually: a transient profile was defined. It had to be smooth and to equal one (1) after two (2) seconds:

\[
f(t) = \left(\frac{t - t_0}{2}\right)^2 \times (3 - (t - t_0)) \quad \text{(30)}
\]

where \(t_0\) is the current time for which the mean flow restart file was saved.

Hence, the gust amplitude was multiplied to this function, and the origin was moved to \(t_0\). Its profile is shown below.

![Fig 11: Transient Profile for the Gust Initialization](image-url)
In the second case, the flow was initialized with a built-in gust. The gust was defined in the flow initialization file. Of course, it was also defined as before in the inflow boundary condition. So the gust was a given, and the response of the airfoil to either the mean flow or the gust was unknown.

It appears that this way of proceeding produced a faster convergence. The initialization files are shown below for a 20%-amplitude-2D gust with a reduced frequency of 3.0 interacting with the cambered airfoil.

![Density Initialization](image1)

**Fig 12:** DENSITY INITIALIZATION (2D GUST, $k=3.0$, $\varepsilon=20\%$)

![Velocity Components](image2)

**Fig 13:** $\rho u$ (A) AND $\rho v$ (B) INITIALIZATION (2D GUST, $k=3.0$, $\varepsilon=20\%$)
4.1.1.4 Acoustic Radiation Boundary Condition

The ACRAD boundary condition is described in section 2.4.1.1. Formulae have to be modified to account for the gust. It is subtracted from the mean flow. The gust induces a change in the velocities, and by consequence in energy. Only the first order is considered:

\[ U_{BC} = u_{mean} - u_{gust} \]  
\[ V_{BC} = v_{mean} - v_{gust} \]  
\[ E_{BC} = E_{mean} - \rho u_{mean} u_{gust} - \rho v_{mean} v_{gust} \]

Then equation 4 holds with:

\[ Q = \begin{bmatrix} \rho \\ \rho U_{BC} \\ \rho V_{BC} \\ E_{BC} \end{bmatrix} \]
Hence, the formula for $\rho$ is not affected by the introduction of the gust. But formulae for $\rho u$, $\rho v$ and $E$ have to be modified. For instance, the one for $U_{BC}$ becomes:

$$\frac{\partial}{\partial t} (\rho U_{BC}) = -V(\theta) \times \left( \frac{x-x_s}{R} \frac{\partial}{\partial x} [\rho U_{BC}] + \frac{y-y_s}{R} \frac{\partial}{\partial y} [\rho U_{BC}] + \frac{1}{2R} \rho U_{BC} \right) \tag{35}$$

Mean flow corrections are then added to these derivatives.

The code was delivered without this boundary condition having been checked. So the ACRAD_Inflow.f90 file was modified extensively. Above formulae were set up correctly.

With the NDT calculated in section 3.4.1, it is possible to make sure very efficiently that everything is set up correctly with this file. Otherwise, a reflection problem or a wrong gust appearing at the inflow boundary condition can be observed.

4.1.2. Post-Processing

As stated in section 2.2.3., some post-processing routines are available. However, velocities $u,v$ and pressure $p$ were obtained by transferring data to MATLAB [21], and by computing them to make sure of the consistency of the included routines. The ASCII format from the “fast” files saved by BASS was transferred to MATLAB. Data were converted to vectors of $1\times145200$ for every variable, computations were executed, as results from BASS are $\rho, \rho u, \rho v, E$.

So to get the pressure, equation 5 has to be modified:

$$p = (\gamma - 1) \left( E - \frac{1}{2\rho}((\rho u)^2 + (\rho v)^2) \right) \tag{36}$$

The same kind of modifications is applied to get the velocities:

$$u = \frac{\rho u}{\rho} \tag{37}$$

$$v = \frac{\rho v}{\rho} \tag{38}$$

Then, new vectors $u,v,p$ were grouped to be exported as an ASCII file to TecPlot.
4.1.3. **BASS Results for a 2D Gust**

According to the benchmark problem defined in Section 1, the following configurations for the airfoil geometry were considered in the previous study [1]:

- Unloaded symmetric 12%-thick Joukowsk airfoil,
- Loaded 12%-thick 2%-cambered Joukowski airfoil at a two-degree angle of attack.

Two configurations for the impinging gust were examined:

- One-dimensional (1-D) transverse gust with \( k_2 = 0 \) in (1);
- Two-dimensional (2-D) gust.

where \( k \) is the reduced frequency of the gust nondimensionalized by the half-chord and the upstream flow velocity. In the computations, both the gust intensity and reduced frequency are varied.

In the current study, results obtained with the latest version of the BASS code were compared to the results from the previous study by Golubev & Crivellini [1] on the VLES code – previous version of BASS. Hence, the 1-D gust was first implemented, and used to spot inconsistencies. Then, work was focused on the 2-D gust only. Different frequencies and amplitudes were tested. As a perfect agreement was found, here are presented results for the 2-D gust case with \( k = 1.0 \) on the loaded airfoil only.

### 4.1.3.1 2%-amplitude, k=1

Results for the 2%-amplitude gust corresponding to the velocity magnitude, pressure, density and energy contours are presented.

On the following figures (figure 15-17), no reflection of neither vortical nor acoustic waves can be spotted. This confirms that the boundary conditions were well treated. The definition of the gust also appears nice and clean.
Figure 15 and 16 show the velocity magnitude contours of the unsteady flow. As the disturbance on the mean flow has to be examined to study aeroacoustics, the mean flow values were subtracted from the result obtained by BASS. This explains the range of values observed on the different scales.

Wake instabilities can be observed on the velocity magnitude contours for both $u$ and $v$ components, as shown on figures 15 and 16. So directivity patterns may indicate a wake effect—as on figure 33.

This effect is not due to acoustic waves, but to hydrodynamic pulsations caused by these instabilities. This is the reason why this effect appears on the near field and usually not on the far field. Hydrodynamic waves die out more rapidly than acoustic waves.
On the pressure contour plot (figure 17), two (2) circles show where the near and far fields are defined to get the directivity patterns observed on figure 33.

If one pays carefully attention to figure 17, one can see that the small radius circle hits a vortex in the wake region, whereas the bigger one does not. This observation refers to the wake effect explained above.

Figure 17 also reveals that the wake is not the dominant source of noise. The structural interactions of the gust with the leading edge are.

Once again, no reflections are observed at the boundaries. Pressure waves leave the domain without any scratches. So the corrected boundary conditions can be trusted.
On figure 18, a zoom in the airfoil region shows the pressure contours. One could be surprised not to see the well-known pressure pattern showing the depression on the extrados and the smaller suppression on the intrados creating the lift on this airfoil. Once again, results from BASS were treated in order to show the unsteady components only, and not the mean flow characteristics.
The energy contours follow (see figure 19). As one can see, the stored energy is influenced by the gusts. Once again, modifications to the ACRAD boundary condition proves to be relevant (as on early runs, the gust was not accounted for in the energy formulation).

![Figure 19: Energy Contours for a 2D-2% Gust, k=1.0.](image)

At last, density contours are displayed on figure 20. Density is not affected by the gust for the gust reduced frequency and amplitude studied in this section. Only structural interactions with the airfoil induce a change in density.
4.1.3.2 20%-amplitude, $k=1$

Results for a 20%-amplitude gust are now presented for a NDT of 150.

For a 20%-amplitude gust, the unsteady vortical flow velocity component is not too small compared to the steady mean flow velocity. So nonlinear effects are expected to be more important than on the 2%-amplitude case.

One can notice that the density is now affected by the gust (see figure 21). This plot was obtained by increasing the number of contour levels, so one should not be mistaken between post-processing inaccuracies and actual patterns. However, the point is that the density plot is a lot different from the one observed on figure 20.
Energy contours (see figure 22) show the wake effect.
Fig 23  PRESSURE CONTOURS

Fig 24  VELOCITY CONTOURS OF THE u COMPONENT
Previous plots have to be examine cautiously, because the analysis is not converged yet. A NDT of 360 should be attained. However I could not get this NDT on time because of a network issue that caused the computer to crash.

4.1.3.3 2%-amplitude, \( k=3 \)

As specified before, the mesh is not refined enough to handle such a frequency. However, BASS does not crash. Results presented here correspond to \( NDT = 260 \). So results are assumed to be converged. As no nonlinear studies will be conducted on this case, there was no use pushing the iterations until a NDT of 360 was attained.
Pressure contours show an unexpected pattern. The range is also very different from what was expected. The gust does not seem to be totally accounted for. An inspection of the trailing edge region shows that the gust is not resolved with the current mesh.

On velocity magnitude contours (figure 27) some unexpected phenomenon are also noticeable: the gust seems to disappear from the domain. This is not due to the ACRAD boundary conditions, as the gust seems to keep on coming from the inflow. A discontinuity appears across the wake, as observed previously. The fact that the gust is disappearing from the domain could be due to a reflection on the outflow boundary.
A closer inspection of the mesh shows that the cell size is about 0.08 on the trailing edge and on the domain boundaries. This corresponds to about twelve (12) mesh points per wavelength for a reduced frequency of one (1), if only the fundamental frequency is accounted for. But for a reduced frequency of three (3), there are only four (4) mesh points per wavelength. At least six (6) mesh points per wavelength should be present.
4.1.3.4 20%-amplitude, k=3

This analysis does not converge at all: the code crashes, giving a null field for every variable. This occurs for \( NDT = 3.32 \). Results are shown here to try to understand the reason why BASS cannot handle it.

![Density Contours](image)

Fig 28: Density Contours.

The density contours on figure 28 show that the good range of value remains for the density: most of the air in the domain as a density of one (1). However, when one pays attention to the legend, one can notice that there must be a very large gradient somewhere in the domain. By looking carefully, it appears to be on the extrados of the airfoil. By analyzing results corresponding to anterior NDT, it was observed that this vortex was generated in the wake and was expelled from it, going upstream on the extrados.
Concerning the general shape of the contours, the gust seems to disappear also. And huge reflections can be seen along every external boundary. The way the gust is introduced could be discussed, as it does not seem to come in anymore. Also, a huge vortex seems to envelope the airfoil. This could be due to some radiations of some sort.

The same vortex on the extrados can be seen on figure 29, on the pressure contours, as well as the big vortex around the airfoil. This huge recirculation can also be spotted on the velocity contours on figures 30 and 31. As expected while looking at figures 28 and 29, unsteady velocity disturbances in the wake also shows up.

However, the mean value of the $u$ velocity magnitude is not anymore around $U_\infty = 0.5$, but below 0.1. The $v$ velocity magnitude is also way off the zero (0) expected.
If the wrong results obtained for a 2%-amplitude gust are due to a too coarse grid resolution, the discrepancies observed for a 20%-amplitude gust suggest that the
implementation of the boundary conditions should be revised. Nonlinear effects seem to be really too important in this case.

4.1.3.5 Comparison to VLES Results

Agreement is obtained with the previous results from the VLES code [1] for the case \( k = 1.0 \).

Hence it is assumed that results for the case \( k = 0.1 \) should be similar too. For reference and discussion purpose the directivity patterns for both cases are shown on figure 33. Results were scaled to the gust amplitude of 2% for proper comparison.

High frequencies were further investigated. But, it seems that the grid used is too coarse to handle properly the \( k = 3.0 \) case. The gust seems to disappear which means that there are not enough grid points to resolve it.

For the high frequency gust to be handled properly, at least six (6) mesh points per wavelength would be needed. The largest grid presents to few mesh points per wavelength near the upstream/downstream boundaries. So the grid resolution is not good enough, and the large domain grid has to be refined to get a more appropriate grid density.

This result is however promising because BASS seems to handle it. Even if final results cannot be right because of this coarse grid, the code seems not to fail to converge, unlike with the VLES code.

4.2. Comparison to GUST3D

GUST3D is another CFD/CAA code developed by NASA. As it is a frequency-domain solver which gives linear solution, its approach is different from BASS. As results of such a test case are unknown for a high intensity, high frequency impinging gust, a comparison between BASS and GUST3D results is of primary importance.
4.2.1. Purpose of the Code

On the same benchmark problem, it will be further assumed that the convected disturbances are not too large ($|\tilde{a}| \ll U_\infty$), and that the flow moves at high speed. The "rapid distortion" [14, 15] approximation holds and the linearized unsteady Euler equations are solved –instead of the nonlinear ones that are solved by BASS. Hence, GUST3D solves the linearized Euler equations.

In this case, one obtains the zeroth-order steady mean flow first, and then obtains the unsteady flow as a first-order perturbation.

4.2.2. Governing Equations

Let the flow field be represented by:

$$U(x,t) = U_0(x) + u(x,t) \quad (39)$$
$$p(x,t) = p_0(x) + p'(x,t) \quad (40)$$
$$\rho(x,t) = \rho_0(x) + \rho'(x,t) \quad (41)$$
$$s(x,t) = s_0(x) + s'(x,t) \quad (42)$$

where the entropy $s_0$ is constant, and $u', p', \rho', s'$ are the unsteady –unknown– perturbation velocity, pressure, density and entropy, respectively. Quantities with "0" subscripts are the steady mean flow quantities which are independently solved and assumed to be known. Substituting into the nonlinear Euler equations and neglecting products of small quantities, one obtains the linearized continuity, momentum, and entropy conservation equations:

$$\frac{D_0 \rho'}{Dt} + \rho' \nabla \cdot U_0 + \nabla \cdot (\rho_0 \tilde{u}) = 0 \quad (43)$$
$$\rho_0 \left( \frac{D_0 \tilde{u}}{Dt} + \tilde{u} \cdot \nabla U_0 \right) + \rho' \tilde{U}_0 \cdot \nabla \tilde{U} = -\nabla p' \quad (44)$$
$$\frac{D_0 s'}{Dt} = 0 \quad (45)$$
where \( \frac{D_0}{Dt} = \frac{\partial}{\partial t} + U_0 \cdot \nabla \) is the material derivative associated with the mean flow.

The flow field is then divided into inner and outer regions where the velocity is decomposed according to different methods (Goldstein's in the outer and Atassi-Grzedzinski's in the inner). Velocities are then plugged in the above equations.

This is the domain decomposition approach. This approach uses each formulation where it is best suited. In the inner region, the Atassi-Grzedzinski formulation cancels the singularity in Goldstein's vortical velocity, and provides a boundary value problem with regular boundary conditions. In the outer region, far away from the airfoil singularity, Goldstein's formulation provides a boundary value problem which is better suited for wave propagation in an open domain.

**Fig 32: PHYSICAL GRID FOR THE GUST3D DOMAIN DECOMPOSITION APPROACH**

This domain decomposition approach is largely insensitive to the location of the outer grid boundary, and provides an acceptably grid independent solution for reduced frequencies ranging from 0.1 to 3.0. Hence, this result will be compared to the ones from BASS.
4.2.3. **Comparisons Between BASS and GUST3D**

As explained above, GUST3D solves the linearized Euler equations. Hence, it does not account for nonlinear effects induced by the gust. Deviations from the linear theory predictions of GUST3D will be observed carefully.

Fig 33: **DIRECTIVITY PATTERNS FOR A 2-D GUST IMPINGING A CAMBERED AIRFOIL**
Results from BASS and VLES match well with GUST3D predictions. For a low reduced frequency the shape of the lobes as well are their volumes are very similar in the near field. In the far field, the shape is conserved as well, but it looks like the intensity magnitudes are more important from the nonlinear analysis. If both pictures are compared together, it seems that the directivity patterns retain the same shape in the near and far field, and the intensity magnitude decreases while going away from the airfoil.

For the high reduced frequency case $k=1.0$, discrepancies between GUST3D and BASS results are more evident. In the near field, even though the intensity magnitudes of the lobes are quite different, the general shape remains. But in the far field, both intensity magnitude and directivity patterns do not match. Nonlinear effects appear: an extra lobe seems to show. It can be linked to the wake effect due to inviscid wake instability triggered by the breakdown of the gust at the airfoil surface. An increasing vorticity can be noticed downstream from the airfoil with higher amplitude gusts. This takes place in the wake.

So when one observes the pressure contours, one can see that the extra lobe comes from unsteady pressure distribution in the wake. Then, the wake instability acts as a new non-compact source. It can also been seen that this effect appears in the far field. This is the reason why it cannot be seen on the lobe corresponding to $R=1$.

4.3. Future Work

This research will be presented in October, 2003 during the fourth CAA workshop on benchmark problems (http://www.math.fsu.edu/caa4). Some more results will be presented at this conference.

Symmetric airfoil cases will be included. As observed in previous studies, these cases are expected to run faster than the loaded airfoil ones.

Also, the current grid will be refined to support the high intensity gust. However, a reduced frequency of two (2) –instead of three (3)– will be high enough according to the new definition of this benchmark. So the current grid should be tested for this new case.
before anything else. Then Fast Fourier Transform (FFT) should be applied in order to check the frequency-domain content of these results. This would pursue the work started by Golubev & Crivellini [1].

Lastly, post-processing routines will have to be implemented efficiently. This would save a huge amount of time.

All input files are already set up as well as executables. So this work will just involve running the cases. Then, results will have to be analyzed. As higher frequencies will be studied, nonlinear effects will be highlighted. In general, such effects may include generation of higher harmonics, which is the reason why analyses have been run until a NDT as long as 360.
CONCLUSION

Results were very satisfying. The BASS code proved to perform as well as the VLES code, and the latest results were confirmed. Other analyses are needed to be presented at the fourth CAA workshop on benchmark problems, in October 20-22, 2003 at the Ohio Aerospace Institute in Cleveland. Especially analyses involving higher gust frequencies, more typical of rotor-stator unsteady interactions, have to be conducted.

As the BASS code really needs much more documentation, this paper was written in part to leave a reference for future researchers. The way the code was fixed and set up is detailed, as well as the implementation of the computational resources of the PACL.

This research involved more than setting up a new CFD/CAA code. As the code is still under development, it was tested. This allowed me to explore another very important issue in CFD and CAA: the code validation. CFD/CAA is a very powerful tool from which enormous time and money savings can be gained. However, before being used in the industry, codes have to be thoroughly tested and validated. Not only one benchmark problem as this one has to be checked, but many more involving many diverse conditions.
REFERENCES


13. Computational Aeroacoustics.


