INVESTIGATION OF ADJOINT BASED SHAPE OPTIMIZATION TECHNIQUES IN NASCART-GT USING AUTOMATIC REVERSE DIFFERENTIATION

A Thesis
Presented to
The Academic Faculty

by

Siddhartha Verma

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in the
School of Aerospace Engineering

Georgia Institute of Technology
May 2009

INVESTIGATION OF ADJOINT BASED SHAPE OPTIMIZATION TECHNIQUES IN NASCART-GT USING AUTOMATIC REVERSE DIFFERENTIATION

Approved by:

Professor Stephen M. Ruffin, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Professor Eric Feron
School of Aerospace Engineering
Georgia Institute of Technology

Professor Lakshmi N. Sankar
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 1 May 2009

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Automated shape optimization involves making suitable modifications to a geometry that can lead to significant improvements in aerodynamic performance. Currently available mid-fidelity Aerodynamic Optimizers cannot be utilized in the late stages of the design process for performing minor, but consequential, tweaks in geometry. High-fidelity shape optimization techniques are explored which, even though computationally demanding, are invaluable since they can account for realistic effects like turbulence and viscocity. The high computational costs associated with the optimization have been avoided by using an indirect optimization approach, which was used to dcouple the effect of the flow field variables on the gradients involved. The main challenge while performing the optimization was to maintain low sensitivity to the number of input design variables. This necessitated the use of Reverse Automatic differentiation tools to generate the gradient. All efforts have been made to keep computational costs to a minimum, thereby enabling hi-fidlity optimization to be used even in the initial design stages. A preliminary roadmap has been laid out for an initial implementation of optimization algorithms using the adjoint approach, into the high fidelity CFD code NASCART-GT.

CHAPTER I

INTRODUCTION

1.1 Motivation

This research work focuses on development of computational tools for designing aeroshell configurations with optimal characteristics for high mass hypersonic entry into the Martian atmosphere. The main problem related to hypersonic entry is the rapid deceleration required to slow the vehicle down sufficiently to achieve effective deployment of parachutes at supersonic speeds. It is desirable to achieve slower speeds as early as possible in the entry trajectory since it makes for easier subsonic maneuvering at the later stages closer to the surface.

Lighter entry vehicles with high drag-area (area exposed to oncoming flowstream) experience higher deceleration in the early stages, thereby reaching the parachute deployment region at a higher altitude, resulting in a secure landing. However heavier vehicles may never reach speeds low enough for the parachute to be deployed successfully before reaching the surface. Theoretical studies have shown [5] that manipulating the vehicle's geometry can result in increased vertical lift in the hypersonic flight regime and can lead to successful hypersonic deceleration into the parachute deployment region even with higher payload values. The ultimate goal of this research project is to develop aeroshell configurations that will maximize landed mass capability, which in turn, will ease the challenges related with high mass Mars entry systems for future robotic and human exploration missions.

1.1.1 Details on EDL (Entry, Descent, & Landing)

Any entry system's deceleration profile depends on its hypersonic ballistic coefficient [5] which is a function of the mass, the drag coefficient and the reference area of the

body as:

$$\beta = \frac{m}{C_D * A}$$

A lower value of ballistic coefficient for a vehicle implies that peak deceleration will occur at a higher altitude. This is desirable since it gives more time for subsequent descent and landing events, and can be achieved by either decreasing the payload of the entry vehicles or by increasing the reference area. The easiest way of increasing the reference area is by increasing the overall diameter of the aeroshell, and the only constraint on this is the maximum available cross sectional diameter of the launch vehicle used. Aerodynamic drag during liftoff and ascent is critical for determining the maximum possible allowable diameter of launch vehicles. Increasing the diameter will result in a corresponding increase in drag from the atmosphere which in turn would necessitate the use of more fuel. This might result in a *d*ecrease in the amount of net available payload! Using bigger rockets would also cause a multi-fold increase in cost of structure, and manufacturing and launch facilities in addition to the fuel.

Figure 1 shows the effect of increasing the ballistic coefficient from $25kg/m^2$ to $200kg/m^2$ on the descent profile. The red (biggest) region indicates the conditions (Mach number and altitude) necessary for optimum operation of the Viking era supersonic parachutes that have been used on almost all of the Mars missions till date [5]. Here we notice that the parachute deployment region becomes narrower with altitude. This happens because lower air density at higher altitudes makes it difficult for the supersonic parachute to generate sufficient deceleration to work effectively.

The blue (second biggest) triangular region in Figure 1 represents conditions ideal for deploying the subsonic parachute once the vehicle has decelerated to subsonic speeds. We see a similar narrowing with altitude of this region as is evident in the red (biggest) region. The green (smallest) region represents conditions optimum for thruster firing for attitude adjustments or for further terminal deceleration close to the surface.

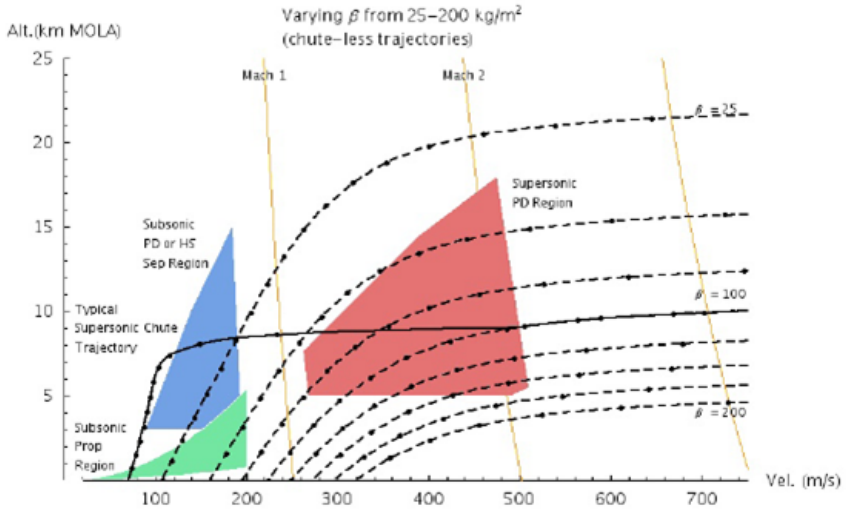
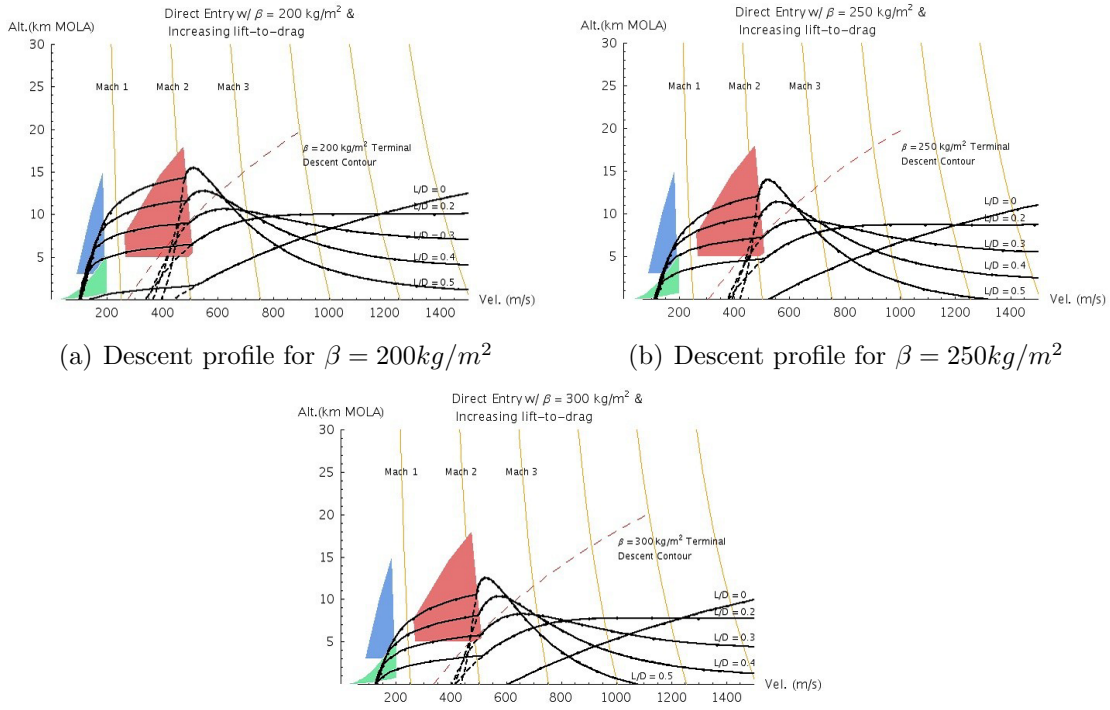


Figure 1: Effect of increasing β on descent profile.[5]

We notice from Figure 1 that for higher values of beta most of the deceleration occurs at higher altitudes. Take for example the $\beta = 25$ line, it is obvious from this descent contour that the aeroshell decelerates from a speed of above Mach 3 to a speed below Mach 1 at an altitude of around 12 km. This is in stark contrast to the deceleration produced with a β value $= 100kg/m^2$, which is close to the typical value of ballistic coefficients of Martian entry vehicles used recently ([5]: Table 1). We notice that such vehicles enter the subsonic region (the region to the left of the line labeled Mach1) very close to the surface, around an altitude of 2km without the aid of lifting techniques.

1.2 Lifting Trajectory

The speed at which the entry vehicle hits the surface is a critical parameter for ensuring the safety of the sensitive equipment included in the payload. We see that the dashed line (non-lifting trajectory) labeled $\beta = 100$ in Figure 1 shows the descent profile of the aeroshell with no other technique employed to augment deceleration, whereas the solid line shows the profile if a lifting trajectory is used. Notice that the vehicle following a non-lifting trajetory will hit the surface at speeds close to Mach 1



(a) Descent profile for $\beta = 200kg/m^2$

(b) Descent profile for $\beta = 250kg/m^2$

(c) Descent profile for $\beta = 300kg/m^2$

Figure 2: Effect of increasing L/D on descent profile.[5]

since the trajectory never really enters the regions of subsonic deceleration (the blue (second biggest) and the green (smallest) regions). The lifting trajectory denoted by the solid line, however, goes through further deceleration caused by the vertical lifting force and subsequent subsonic deceleration. In this case, the vehicle hits the surface at speeds close to merely $\frac{1}{5}$th of the non-lifting speeds.

Figure 2 illustrates descent profiles of entry vehicles employing various different lifting trajectories. The individual graphs represent descent profiles for three different relatively high values of β. From the graph in 2(a), we see that with the use of high enough L/D values, it is still possible to land at safe subsonic speeds with β values almost three times as high as those currently used. This will result in almost a threefold rise in the payload delivery capability of the entry vehicle without a very significant increase in launch costs. Designing entry vehicles with such lifting characteristics via the means of automatic geometry optimization forms the main

focus of this research.

1.3 Software for Analysis and Design Optimization

The primary software used for computational analysis and design optimization in this project is NASCART-GT (Numerical Aerodynamics Simulation via CARTesian Grid Techniques). NASCART is capable of variable fidelity flow analysis, and has been developed by Dr. Stephen Ruffin and his team over the past few years at Georgia Tech. This project focuses on direct coupling of NASCART to numerical optimization algorithms, which will give us the capability of precise high-fidelity design optimization. For this purpose, adjoint approaches ([9], [21], [22], [11]) are used in conjunction with Automatic Differentiation techniques for accurate and efficient computation of gradient information indirectly. Adjoint methods coupled with AD result in substantially lower computational costs compared to finite difference methods and thus are especially suitable for shape optimization of the aeroshells, which involve a large number of design variables. This approach has been implemented on several simpler CFD codes like Euler2D [26] which deal mainly with inviscid flows. However, not many fully Navier-Stokes capable codes use this technique. Such codes (similar in scope and scale to NASCART) tend to present several difficulties in employing Automatic Differentiation due to their complexity. The large file structure and the extensive interlinking of subroutines also present certain challenges with AD.

1.4 Steps to Completion

The following steps have been or need to be taken for the completion of the ultimate goals of this line of research:

1. The gradient of the cost function will be generated using the reverse differentiation mode of TAPENADE.

    - Making use of automatic differentiation is a necessity since this is the only

method capable of generating derivatives in the reverse mode. Further details are presented in Section 2.2

2. An iterative gradient descent method will then be used to find the new geometry node locations which minimize/maximize the gradient obtained in Step 1.

    - It should be noted that this method will cause the gradient to move towards the closest local minima/maxima, thus it is neessary to start with a shape that is close to the desired final shape.
    - Initially, a first order descent method will be employed. Later, higher order non-linear optimization methods like BFGS may be needed.
    - The starting guess for the locations of the nodes in the iterative gradient minimization process will be the starting non-optimized node locations.

3. The gradients will then need to be validated using Finite Difference approximations.

    - This step is **essential** to ensure that the differentiated code works as intended.
    - This can also be useful for debugging purposes.

CHAPTER II

METHODOLOGY

2.1 Methodology

2.1.1 Cost function

A Cost Function (J) is defined for the given geometry, depending on the parameter (C_D or C_L) that needs to be optimized. This allows us to obtain a geometry with the minimized (or maximized) Cost Function using gradient descent methods, even if the exact geometry with specified objective does not exist [21]. For our unconstrained test cases, we will define the Cost Function as the drag coefficient calculated on the surface nodes of the body. These specific calculations have been have been included in a file called 'adjoint.f'. This allows us to work with just the sections of the code that are necessary for the calculation of the Cost Function, thereby avoiding the need to feed the entire source code to the differentiator.

2.1.2 The Gradient descent method

The gradient of a function gives the direction and magnitude of the greatest rate of increase of the function and hence can be used to ascend iteratively to the point of the nearest local maxima. For a minimization problem we will need to take a step proportional to the negative of the gradient, whereas for maximization we take a step proportional to the positive of the gradient. The main working equation for our initial test case is shown in Equation 1.

$$\alpha_{new} = \alpha_{old} - \gamma \nabla J(\alpha_{old}) \tag{1}$$

Here α collectively represents the parameters defining the body geometry (in our case the co-ordinate locations of the surface nodes). The step size γ is set to a

suitable value arbitrarily and may be allowed to change at every iteration to make sure that the minimization step does not keep oscillating about the local minima as a result of repeated overshooting. The initial value of α_{old} will be set equal to the α corresponding to the original body node locations.

The iterations will be stopped when the value of $(\alpha_{new} - \alpha_{old})$ becomes sufficiently small, indicating that the iterations are no longer having a significant effect on the α values and hence, that an optimal solution may have been reached.

2.1.3 Gradient of the Cost function

All numerical calculations of aerodynamic properties depend on two main variables: the state vector, and variables defining the body geometry. Let us represent the state vector by the variable u. The Cost Function (C_D) can hence be expressed as:

$$J = J(\alpha, u) \tag{2}$$

To study the effect of a change in α on the Cost function, we can write the differential of J as:

$$\delta J = \frac{\partial J}{\partial u}\delta u + \frac{\partial J}{\partial \alpha}\delta\alpha \tag{3}$$

We can clearly see that the change in Cost Function is affected by changes in both the geometry as well as in the state vector. Please note that the change in the state vector is non-zero since even a slight modification to the geometry ($\delta\alpha$) might result in significant changes in the flowfield (δu) when simulating supersonic cases. This indicates that the gradient ∇J would be dependent on two variables, α and u, as opposed to only α as was assumed in Equation 1. The equation for gradient minimization would then become:

$$\alpha_{new} = \alpha_{old} - \gamma \nabla J(\alpha_{old}, u_{old}) \tag{4}$$

We can see here that each successive $((n+1)^{th})$ iteration would require the state vector from the modified geometry created by the previous (n^{th}) iteration. This

would neccessitate the calculation of the flow field variables at each iteration step for feeding in to the next iteration step. This process can prove to be quite cumbersome computationally and may not lead to the optimal solution in a feasible amount of time. The alternative is to make use of the Control Theory approach (also referred to as the 'adjoint method') developed by Jameson in reference [21], and implemented by Giles, Duta and Ghate in reference [9], which essentially decouples the dependence of the Cost function gradient on the flow solution (u) and the geometric parameters (α). This method is discussed in detail in the following section.

2.1.4 Jameson's Control Theory approach

The Control Theory approach makes use of the fact that a converged solution of the fluid dynamic equations (for example the Euler or the Navier Stokes equations) ideally implies that the residual is zero, and can be written as:

$$R(u, \alpha) = 0 \tag{5}$$

We multiply this Residual with a Langrangian (v^T) which we will refer to as the 'adjoint variable'. We can then add the resulting expression to the Cost function without affecting it (since the product to be added will be zero).

$$J(\alpha, u) = J(\alpha, u) + v^T R(\alpha, u) \tag{6}$$

We rewrite Equation 6 without the function arguments for brevity during differentiation:

$$J = J + v^T R \tag{7}$$

Differentiating Equation 7, we get:

$$dJ = \left(\frac{\partial J}{\partial \alpha} d\alpha + \frac{\partial J}{\partial u} du\right) + v^T \left(\frac{\partial R}{\partial \alpha} d\alpha + \frac{\partial R}{\partial u} du\right) \tag{8}$$

Rearranging Equation 8 so as to group the $d\alpha$ and the du terms together we get:

$$dJ = \left(\frac{\partial J}{\partial \alpha} + v^T \frac{\partial R}{\partial \alpha}\right) d\alpha + \left(\frac{\partial J}{\partial u} + v^T \frac{\partial R}{\partial u}\right) du \tag{9}$$

To make dJ independent of the du term, we force its coefficient to zero:

$$\left(\frac{\partial J}{\partial u} + v^T \frac{\partial R}{\partial u}\right) = 0 \quad \text{which implies that} \quad v^T \frac{\partial R}{\partial u} = -\frac{\partial J}{\partial u} \tag{10}$$

Taking the transpose on both sides:

$$\left(\frac{\partial R}{\partial u}\right)^T v = -\left(\frac{\partial J}{\partial u}\right)^T \tag{11}$$

Equation 11 can be used to find the value of the variable 'v' which can then be used to find the value of the gradient using the following equation:

$$\nabla J = \frac{dJ}{d\alpha} = \left(\frac{\partial J}{\partial \alpha} + v^T \frac{\partial R}{\partial \alpha}\right) \quad \text{under the condition that} \quad \left(\frac{\partial R}{\partial u}\right)^T v = -\left(\frac{\partial J}{\partial u}\right)^T \tag{12}$$

The numerical values of the 4 derivatives shown in Equation 12 $\left(\frac{\partial J}{\partial \alpha}, \frac{\partial R}{\partial \alpha}, \frac{\partial R}{\partial u}, \frac{\partial J}{\partial u}\right)$ are calculated by feeding in the state vector from a partially converged solution, and the original body node locations, to the differentiated routines. To obtain these differentiated routines, we make use of Automatic Differentiation in the reverse mode as explained in Section 2.2.

It is important to reiterate here how efficient Jameson's control theory approach is compared to performing the optimization in a brute force manner. The gradient of the Cost Function with respect to the shape parameters can hence be found without the need for additional flow field evaluations [22] due to the decoupling of the dependence of the final gradient from the effect of changes in the state vector.

2.1.5 Computational cells vs. Body surface cells

NASCART calculates aerodynamic forces using the state vector at the center of cells on a computational grid instead of the original body cells. Using derivatives at the computational cell centers or nodes will not lead to the optimal result since it will not accurately reflect the change in body surface node locations needed to obtain the optimized Cost Function. Hence NASCART variables containing the physical

co-ordinates of the body surface nodes have been used in the calculation of the Cost Function in the file called 'adjoint.f'.

2.1.6 Constrained vs. Unconstrained Optimization

Using the currently proposed algorithm, the modification of the geometry to minimize the Cost Function will be done irrespective of its effect on other aerodynamic parameters. This might at times be an undesirable effect, and can be avoided by introducing certain constraints in the Cost Function expression as done by Modammadi et al. in [14] and [24], and Jameson et al. in [2]. These constraints ascertain that some given aerodynamic parameter values are left unchanged during the optimization process, and can be useful in several scenarios, such as minimizing the drag coefficient while keeping the lift coefficient constant. Mohammadi and Jameson discuss the introduction of such constraints to maintain the shape of an airfoil or a wing within specified limits during the optimization. For the sake of simplicity in the initial stages, the present research deals only with unconstrained optimization problems.

2.1.7 Test Cases

The basic test case for the optimization algorithm would use a supersonic wedge similar to that shown in Figure 3 with the Cost Function minimized being the Drag Coefficient. Successful implementation of the optimization algorithm should transform the wedge into a geometry closely resembling the von Karman Ogive which can be analytically proven to have the minimum drag at supersonic speeds.

2.2 Reverse Automatic Differentiation (RAD)

'Automatic Differentiation' (AD) has been selected over conventional numerical differentiation methods (such as Finite Differences and Complex Differences) for two main reasons. Firstly, AD generates analytical derivatives from the source code. This

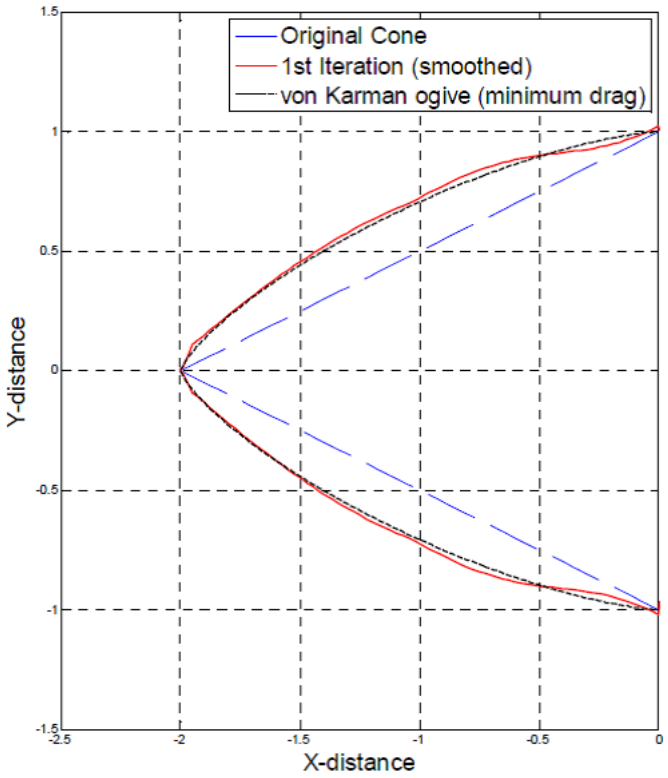


Figure 3: Optimization of a wedge in supersonic flow [8]

eliminates the presence of any inherent Truncation Error associated with certain numerical differentiation methods. Secondly, the generation of derivatives using reverse differentiation is possible only by using Automatic differentiation tools. It is for this reason that Complex Differences, though lacking any associated Truncation Error, cannot be used for derivative generation.

The term 'reverse differentiation' refers to the special algorithm employed by the differentiator where the derivatives are computed in the reverse order of the normal program flow. The main advantage of the reverse mode is that the calculation of derivatives becomes dependent on the number of output variables as opposed to the number of input variables as is the case in the more conventional 'forward mode'. This is especially advantageous when dealing with optimization problems involving thousands of input parameters (for example: the body surface node locations on a 3D geometry, as shown in Figure 4).

The most computationally efficient way to generate the required derivatives would be to go through the code and manually write out the expressions for the derivatives. However this is a tedious and error-prone process and would need to be repeated everytime the program source code undergoes an upgrade. This repeated derivative generation can be done relatively easily and rapidly using source transformation based automatic differentiation tools.

The automatic differentiation tool used for our purpose is TAPENADE which has been developed by INRIA, France [16]. A separate file called *adjoint.f* has been created specifically for the purpose of feeding into TAPENADE and includes calculations using sections of code that were isolated from several files of the source code. Please keep in mind that adjoint.f still includes calls to other subroutines which will also need to be given to TAPENADE for processing, as discussed in Section 2.5.

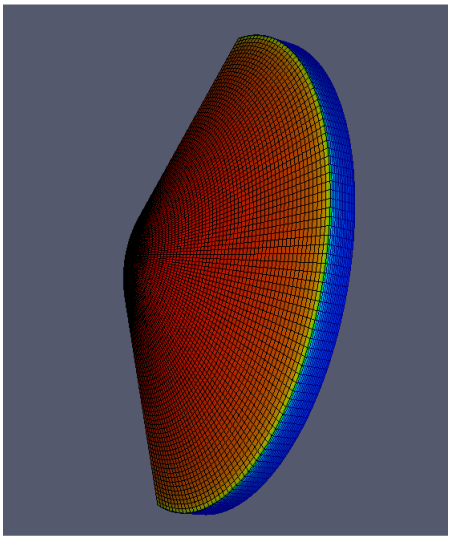


Figure 4: Body Surface nodes read in by NASCART

2.2.1 TAPENADE

TAPENADE was selected for performing the differentiation since it is capable of parsing Fortran code which is the language that NASCART is written in, and also due to its capability to produce differentiated code in both Forward and Reverse

modes.

The default output of TAPENADE is a Fortran77 file. However since the CFD code uses certain constructs that are not compatible with older versions of Fortran, it might become necessary at some point to specify that the output be generated using Fortran95 syntax when using TAPENADE. This can be done using the '-*outputlanguage*' command-line option when using TAPENADE locally installed on a computer instead of using the differentiator directly from the web server.

To compile files differentiated in the reverse mode, it will be necessary to link the definition of the PUSH* and POP* subroutines that TAPENADE generates. These subroutines handle the management of the stack during the forward and the reverse sweep as explained in the User's manual [16].

2.2.2 TAPENADE RAD algorithm

The reverse differentiation algorithm is introduced briefly here. For a more detailed working, please refer to Section 4.2 in the TAPENADE User's guide [16]. We will briefly discuss the following example taken directly from the User's guide (Figure 5):

$$a(i) = x * b(j) + cos(a(i)) \tag{13}$$

Here, we will consider x and $b(j)$ to be our input variables, and $a(i)$ to be the output variable. Figure 5 shows the results of the differentiated program in both the tangent (forward) mode, and the reverse mode. The algorithm behind each of the two modes of differentiation is discussed in the following section.

TAPENADE **tangent**	TAPENADE **reverse**

`ad(i) = xd*b(j) &`
`& + x*bd(j) &`
`& - ad(i)*SIN(a(i))`
	`xb = xb + b(j)*ab(i)`
	`bb(j) = bb(j) + x*ab(i)`
	`ab(i) = -SIN(a(i))*ab(i)`

Figure 5: Comparison of FAD and RAD algorithms ([16]:Figure 8)

2.2.2.1 Forward mode

This expression is obtained using the chain rule of differentiation in the regular manner:

$$\dot{a}(i) = [\dot{x} * b(j) + x * \dot{b}(j)] - sin(a(i)) * \dot{a}(i) \tag{14}$$

Each of these $\dot{a}(i), \dot{x}$, and $\dot{b}(j)$ is represented as the variable $ad(i), xd$, and $bd(j)$ in the actual differentiated code.

2.2.2.2 Reverse mode

The algorithm used in this mode is somewhat unconventional. Since the differentiation is done in reverse of the original program flow, the first step is to inspect the output variable, which in our case is $a(i)$. The reverse 'derivatives' (represented by $\bar{a}(i), \bar{b}(j)$, and $\bar{x}$ in the following discussion) actually represent the partial derivative of the output variable with respect to that specific variable. That is:

$$\begin{aligned} \bar{a}(i) &= \frac{\partial a(i)}{\partial a(i)} \\ \bar{b}(j) &= \frac{\partial a(i)}{\partial b(j)} \\ \bar{x} &= \frac{\partial a(i)}{\partial x} \end{aligned} \tag{15}$$

Logically $\bar{a}(i)$ in our case should always be equal to 1 since $\bar{a}(i)$ represents $\frac{\partial a(i)}{\partial a(i)}$, however the algorithm shown below in Equation 16 is followed instead. These results can be verified from the matrix calculations shown in Section 4.3 of the User's guide.

$$\begin{aligned} \bar{a}(i) &= 0 + \frac{\partial [x * b(j) + cos(a(i))]}{\partial a(i)} \bar{a}(i) \\ &= -sin(a(i)) * \bar{a}(i) \end{aligned} \tag{16}$$

This was the first step. The next steps now involve the partial derivatives with respect to the input variables. These derivatives follow a slightly different algorithm

as shown below. Instead of a zero (as shown in Equation 16), we now have the derivative itself as the first term.

$$\begin{aligned} \bar{b}(j) &= \bar{b}(j) + \frac{\partial[x * b(j) + cos(a(i))]}{\partial b(j)} \bar{a}(i) \\ &= \bar{b}(j) + x * \bar{a}(i) \end{aligned} \tag{17}$$

The derivative of the last remaining term can similarly be given as:

$$\begin{aligned} \bar{x} &= \bar{x} + \frac{\partial[x * b(j) + cos(a(i))]}{\partial x} \bar{a}(i) \\ &= \bar{x} + b(j) * \bar{a}(i) \end{aligned} \tag{18}$$

These results can be compared to the derivatives listed in Figure 5

2.3 Problems with Fortran pre-processor statements (fpp)

Fortran pre-processor (fpp) statements are processed by the pre-processor before compilation of the actual code. If the instruction given during the 'make' (for example: 'make double', 'make parallel') operation matches the #ifdef statement then the code inside the fpp statements becomes 'activated' and will be executed by the compiler. If the specific #ifdef condition was not provided during the 'make' statement then the pre-processor comments out the statements in between the #ifdef and its corresponding #endif marker before the code is executed by the compiler.

TAPENADE is not capable of handling fpp statements and will return an error unless all the lines containing the fpp markers in the source code are commented out. A Perl script (provided in full in Chapter A.1) was used for this process of commenting out the lines of code contained in between the fpp markers. Another advantage of commenting out these sections of code is that there will be no need to provide TAPENADE with as many files as before since some of the files dealing with MPI (Message Passing Interface) during parallel processing are no longer required.

The script requires the correct permission (*chmod 755 FppCommenter.pl*) before it can process the source code. Once the correct permissions have been set, the script can then be run using the following command ('filename' should be replaced with the name of the file to be processed):

./FppCommenter.pl filename

2.4 Running TAPENADE locally from the computer

The web based TAPENADE differentiation server can handle only limited file sizes and hence it becomes necessary to use TAPENADE locally installed on the system. Another advantage of using the locally installed version of TAPENADE is that the main MakeFile used for compiling NASCART can include instructions to generate the derivative code required. Some of the steps essential for this process are provided in the Appendix in Sections A.2.1 and A.2.2.

2.5 Source code files necessary to be given to TAPENADE for differentiation

Tabel 1 lists all the source files that will be need to be given to TAPENADE for differentiation of the main file ('adjoint.f'). The user will need to ensure that '*FPP-Commentor.pl*' is run on all these files before passing on to TAPENADE.

Table 1: Files to be provided to TAPENADE for differentiation

adjoint.f	head.f	areaccl.f
inout.f	updater.f	luscheme.f
movegeom.f	soladapt.f	bound.f
procmap.f	borderflag.f	restout.f
borderflag.f	nbodytonc.f	solidbc.f
plane.f	icross.f	thermolib.f

CHAPTER III

FUTURE WORK

Once the files containing the derivatives have been generated the next step would be piecewise validation of these derivatives using Finite Difference approximations. Several approaches to validation of the transformed source code have been discussed in [17] and [16].

An additional Fortran file will need to be written to perform the gradient descent step iteratively until the optimized conditions have been reached. This file will require the derivatives from the transformed source code as inputs for calculations involving the gradient of the cost function. The output of this file will be the modified node locations.

Future work may also entail the requirement to expand optimization capability to include Constrained optimization. This will require the inclusion of additional terms in the cost function expression as mentioned in Section 2.1.6

The calculation of the adjoint variable in the initial test cases (2D) can be done using a basic linear algebra solver. However, a matrix independent method, such as GMRES [28], will need to be implemented for this purpose when dealing with optimization of 3D cases.

A first order gradient descent method has been proposed for the optimization steps mentioned in Section 2.1.2. This linearized optimization implies that large changes in the geometry between iterations may lead to a faulty result. Higher order optimization methods like BFGS might need to be used to overcome this limitation in the later stages of development.

CHAPTER IV

CONCLUSION

An initial effort has been made to implement algorithms involving an indirect approach for high fidelity aerodynamic optimization. The adjoint approach was used to generate the gradient of the cost function. This essentially decouples the effect of the flow field variables and the design variables on the gradient. A gradient descent algorithm is then used to reach the optimal solution. It becomes indispensible to make use of Reverse Automatic Differentiation tools to minimize the sensitivity of the optimization on the large number of design variables involved. The current research takes a step towards performing unconstrained optimization of relatively complicated 3 dimensional geometries while keeping computational costs to a minimum. The complexity of the high-fidelity CFD code (NASCART-GT) in question raises some difficulties while dealing with the 'blackbox' automatic differentiation program TAPENADE.

APPENDIX A

A.1 Perl Script for Source code modification

[1]

```
1  #!/usr/bin/perl
2
3  #use warnings;
4  use strict;
5
6  #filename to be read in and output
7
8  #my $filein = "adjoint.f";
9  #uncomment the following line to allow for user to input the name of the
10 input file
11 my $filein = shift @ARGV;
12 my $fileout = "adjointCommentedFpp.f";;
13
14 #read in text file and store in the array called @lines
15 open (TXTFILE, $filein);
16 my @lines=<TXTFILE>;
17 close(TXTFILE);
18
19 #declare variables for use inside the loop
20 my $ind;
```

[1]The code provided here has been wordwrapped

```
21  my $loopline;
22  my $increment=1;
23  my @newarray;
24  my $indcomm;
25  my $skipper=0;
26
27
28
29  #use this for-loop to go through all the lines in the code
30  for ($ind=0; $ind<=@lines; $ind=$ind+$increment){
31
32          if (@lines[$ind]=~ /#ifdef/ ){
33  #using print statements for debugging
34          #print "#ifdef at line $ind\n";
35          push(@newarray,"c..@lines[$ind]");
36
37
38  #now write another for loop to check the lines following this starting
39  point
40                  MIDCHUNK : for ($indcomm=$ind+1; $ind<=@lines;
41  $indcomm=$indcomm+1){
42
43  #if '#endif' is not present then keep commenting
44                          unless(@lines[$indcomm]=~ /#endif/){
45
46  #first check if there is another #ifdef present. If yes then increase
47  $skipper by 1
```

```
48                                  if(@lines[$indcomm]=~/#ifdef/){
49                                  $skipper=$skipper+1;
50                                  }
51                                  #print "not #endif at line $indcomm\n";
52                                  push(@newarray, "c..@lines[$indcomm]");
53
54
55  #if '#endif' is present
56
57                          }else{
58
59  #if $skipper is not 0 then keep commenting and decrease $skipper by one
60                                  if(!$skipper==0){
61                                  #print "non-final endif at line
62  $indcomm\n";
63                                  push(@newarray, "c..@lines[$indcomm]");
64                                  $skipper=$skipper-1;
65  #if $skipper=0 (meaning the last #endif has been reached) then comment the
66  line and break the for-loop named MIDCHUNK
67
68                                  }else{
69                                  $skipper=0;
70                                  push(@newarray, "c..@lines[$indcomm]");
71                                  last MIDCHUNK;
72                                  }
73                  }
74          }
```

```
75          $increment=$indcomm-$ind+1;
76
77  # if string comparison for '#ifdef/' returned false in the topmost 'if'
78  conditional
79  # then push the line without commenting
80
81          }else{
82          push(@newarray,"@lines[$ind]");
83          $increment = 1;
84          }
85  }
86
87  #open file for writing to
88  open (OUTFILE, ">adjointCommentedFpp.f");
89  print @newarray;
90  print OUTFILE @newarray;
91  close (OUTFILE);
```

A.2 TAPENADE Commandline

A.2.1 Steps for running TAPENADE locally

The latest version of JAVA jre must be installed to be able to run TAPENADE:

sudo apt-get install sun-java6-jre

The following two lines will need to be executed in the terminal before starting up each session of TAPENADE

export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.07/

export TAPENADE_HOME=/'TapenadeDirectory'/tapenade3.1

The pathways will need to be modified depending on where the target files (Java-6 folder and tapenade3.1) are located. The differentition is then initiated using the commands provided in section A.2.2

A.2.2 Command to run TAPENADE on the files listed in Table 1:

./tapenade -reverse -root func_body -vars "x_body" adjoint.f head.f areaccl.f inout.f updater.f luscheme.f movegeom.f soladapt.f bound.f procmap.f borderflag.f restout.f borderflag.f nbodytonc.f solidbc.f plane.f icross.f thermolib.f

REFERENCES

[1] Alonso, J. J., Kroo, I. M., and Jameson, A., "Advanced algorithms for design and optimization of quiet supersonic platforms," in *AIAA Paper 02-0114, 40th AIAA Aerospace Sciences Meeting and Exhibit*, vol. 14, 2002.

[2] Alonso, J. J., Reuther, J., Rimlinger, M., Sauders, D., and Jameson, A., "Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers," in *Aerospace Sciences Meeting and Exhibit, 35th, Reno, NV, Jan. 6-9, 1997*, 1997.

[3] Anderson, W. K. and Bonhaus, D. L., "Aerodynamic design on unstructured grids for turbulent flows," tech. rep., NASA TM, 1997.

[4] Anderson, W. K. and Bonhaus, D. L., "Airfoil design on unstructured grids for turbulent flows," *AIAA J.*, vol. 37, no. 2, pp. 185–191, 1999.

[5] Braun, R. D. and Manning, R. M., "Mars exploration entry, descent and landing challenges," in *2006 IEEE Conference*.

[6] Carpentieri, G., Koren, B., and van Tooren, M. J. L., "Adjoint-based aerodynamic shape optimization on unstructured meshes," *J. Comput. Phys.*, vol. 224, no. 1, pp. 267–287, 2007.

[7] Courty, F., Dervieux, A., Koobus, B., and Hascoët, L., "Reverse automatic differentiation for optimum design: from adjoint state assembly to gradient computation," *Optimization Methods and Software*, vol. 18, no. 5, pp. 615–627, 2003.

[8] Flaherty, K., *Initial Implementation of an Adjoint CFD Code for Aeroshell Shape Optimization*. Masters, Georgia Institute of Technology, 2008.

[9] Giles, M., Ghate, D., and Duta, M., "Using automatic differentiation for adjoint CFD code development," in *Recent Trends in Aerospace Design and Optimization* (Uthup, B., Koruthu, S., Sharma, R., and Priyadarshi, P., eds.), pp. 426–434, Tata McGraw-Hill, New Delhi, 2006.

[10] Giles, M. B., Duta, M. C., Mller, J.-D., and Pierce, N. A., "Algorithm developments for discrete adjoint methods," 2001.

[11] Giles, M. B., Duta, M. C., and Pierce, N. A., "Report no. 01/15 algorithm developments for discrete adjoint methods."

[12] Giles, M. B. and Pierce, N. A., "On the properties of solutions of the adjoint euler equations," in *Numerical Methods for Fluid Dynamics VI. ICFD*, 1998.

[13] Giles, M. B. and Pierce, N. A., "An introduction to the adjoint approach to design," 2000.

[14] Hafez, M., Mohammadi, B., and Pironneau, O., "Optimum shape design using automatic differentiation in reverse mode," in *Lecture Notes in Physics, Berlin Springer Verlag* (Kutler, P., Flores, J., and Chattot, J.-J., eds.), vol. 490 of *Lecture Notes in Physics, Berlin Springer Verlag*, pp. 83–+, 1997.

[15] Hascoët, L., "Tapenade: a tool for automatic differentiation of programs," in *Slideshow presented at SIAM Annual Meeting, July 7, 2008*, 2008.

[16] Hascoët, L. and Pascual, V., "Tapenade 2.1 user's guide," Technical Report 0300, INRIA, 2004.

[17] Hascoët, L., "Automatic differentiation by program transformation," technical report, INRIA, 2007.

[18] Hascoët, L., Vázquez, M., and Dervieux, A., "Automatic differentiation for optimum design, applied to sonic boom reduction."

[19] In, R. S., Nadarajah, S. K., Jameson, A., and Alonso, J., "Aiaa-2002-0261 an adjoint method for the calculation of remote sensitivities in supersonic flow," in *AIAA paper 2002-0261, AIAA 40th Aerospace Sciences Meeting*, 2002.

[20] Jameson, A., Martinelli, L., and Pierce, N. A., "Optimum Aerodynamic Design Using the Navier-Stokes Equations," *Theoretical and Computational Fluid Dynamics*, vol. 10, pp. 213–237, 1998.

[21] Jameson, A., "Aerodynamic design via control theory," *J. Sci. Comput.*, vol. 3, no. 3, pp. 233–260, 1988.

[22] Jameson, A., "An investigation of the attainable efficienty of flight at mach one or just beyond, 45th aiaa aerospace sciences meeting and exhibit, aiaa paper 2007-37, reno, nv," *AIAA*, vol. 37, 2007.

[23] Martinelli, M., *Sensitivity Evaluation in Aerodynamic Optimal Design*. Phd, Scuola Normale Superiore di Pisa and Université de Nice Sophia-Antipolis, 2007.

[24] Mohammadi, B., "Optimal shape design, reverse mode of automatic differentiation and turbulence," in *Aerospace Sciences Meeting and Exhibit, 35th, Reno, NV, Jan. 6-9,*, 1997.

[25] Nadarajah, S. K. and Jameson, A., "A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization," 2000.

[26] Praveen, C., "Adjoint code development and optimization using automatic differentiation," tech. rep., NAL, 2006.

[27] Reuther, J., Alonso, J. J., Rimlinger, M. J., and Jameson, A., "Aerodynamic shape optimization of supersonic aircraft configurations via an adjoint

formulation on distributed memory parallel computers," *Computers & Fluids*, vol. 28, no. 4-5, pp. 675 – 700, 1999.

[28] Saad, Y. and Schultz, M. H., "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[29] Verma, A., *Structured automatic differentiation*. PhD thesis, Ithaca, NY, USA, 1998. Chair-Coleman,, Thomas F.

[30] Wikipedia, "Gradient descent — wikipedia, the free encyclopedia," 2009. [Online; accessed 27-April-2009].