Open Journal of Fluid Dynamics, 2024, 14, 83-99
https://www.scirp.org/journal/ojfd
ISSN Online: 2165-3860
ISSN Print: 2165-3852

Thrust Optimization of Flapping Wing via Gradient Descent Technologies

Jeshwanth Kundem

Department of Mechanical Engineering, IIT Guwahati, Guwahati, India
Email: jeshwanth.kundem@outlook.com

How to cite this paper: Kundem, J. (2024) Thrust Optimization of Flapping Wing via Gradient Descent Technologies. *Open Journal of Fluid Dynamics*, **14**, 83-99.
https://doi.org/10.4236/ojfd.2024.142004

Received: April 28, 2024
Accepted: May 25, 2024
Published: May 28, 2024

Abstract

The current work aims at employing a gradient descent algorithm for optimizing the thrust of a flapping wing. An in-house solver has been employed, along with mesh movement methodologies to capture the dynamics of flow around the airfoil. An efficient framework for implementing the coupled solver and optimization in a multicore environment has been implemented for the generation of optimized solutions—maximizing thrust performance & computational speed.

Keywords

Steepest Descent, CFD, Flapping Wing Airfoil, Thrust Performance

1. Introduction

The fact that flapping airfoil generates thrust was first recognized by Knoller, and later independently by Betz. They observed that a flapping wing creates an effective angle of attack, resulting in a normal force vector with both lift and thrust components. Due to the flapping wing, a relative motion is created and it is clear that there is a net force in the horizontal direction. If this thrust force is greater than the net viscous drag on the body there is a net thrust generated. This is known as Knoller Betz Effect.

This was later verified by Katzmayr by measuring the average thrust on a stationary airfoil placed in a sinusoid ally oscillating wind stream [1]. Dickinson *et al.* [2] talked about the enhanced aerodynamic performance of insects resulting from an interaction of three distinct yet interactive mechanisms: delayed stall, rotational circulation and wake capture. Delayed stall functions during the translation portions of the stroke, when the wings sweep through the air with a large angle of attack. In contrast, rotational circulation and wake capture generate aerodynamic forces during the stroke reversals, when the wing rapidly rotates and changes direction.

They observed that the delayed stall alone is not sufficient to explain the elevated aerodynamic performance. The rotational mechanisms we describe are necessary components of the basic unsteady aerodynamic toolkit in this species. They developed a more general theory of insect aerodynamics that incorporates both translational and rotational mechanisms.

In the present study based on the available literature, with special reference to [3] [4] [5] about the flapping wings, the following aspects are considered:

- Motion Optimization of an airfoil in plunge is considered. The justification for such an analysis may be had from the fact that plunging corresponds to the primary thrust generation mechanism of the flapping wing. It is well known that mimicking the plunging motion in a flapping device is a lot easier compared to pitching and plunging together. In the design of a flapping wing which involves very tight weight margins, there are two conflicting requirements about the size/weight of the actuator. On one hand, a more efficient flapping motion which involves both pitching and plunging, can result in a smaller propulsive system for a specified mission. On the other hand, such a motion, which is more complex as compared to a simple plunging motion can result in a more complex actuator, which adds weight to the system. Therefore in the present work, we have restricted ourselves to the optimization of plunging motion alone. The optimization involving pitch and plunge is a topic of future research.
- As also shown by Kaya *et al.* [5], the optimal motion for the highest efficiency for a plunging airfoil is non-sinusoidal. Even though our optimization efforts are to improve thrust rather than efficiency, we expect that the resulting optimal motion path is also to be far from sinusoidal motion.
- The motion of the wing can be mathematically represented as a spline function using NURBS [6]. The control points for this curve naturally become the optimization variables which are determined using a steepest ascent approach.
- The present computations involving unsteady NS computations are very expensive; generating the aerodynamic data on a given point in the search space is of the order of 20 - 30 hours on reasonably fine meshes for 2 - 3 cycles of motion. For very obvious reasons, parallel computing becomes the most natural choice for such an analysis.

All the unsteady flow computations in this work are done with High-resolution Flow solver on Unstructured Meshes (HIFUN) code. The code is a cell-centered finite volume compressible flow solver. The second-order accurate three-point backward difference scheme is employed for time discretization, whereas the spatial gradients are computed using diamond path reconstruction. The system of equations is solved using a dual-time stepping strategy [7], which allows for efficient and easy implementation of convergence acceleration techniques, without loss in temporal accuracy.

2. Gradient Descent Optimization Framework

Optimization is the process of obtaining the best results under given circumstances. The existence of optimization methods can be traced to the days of Newton, Lagrange, and Cauchy. Cauchy first applied the Steepest Descent Method to solve the unconstrained minimization problem 1847. Traditional optimization techniques are of three categories—Mathematical Programming Techniques, Stochastic Process Techniques, and Statistical Methods. The Steepest Descent method is an Unconstrained Optimization Technique which is a sub-category of Mathematical Programming. Unconstrained Optimization is of two types Direct Search Methods and Indirect Search (Descent) Methods. In this section, we discuss its validation & application with flapping wings.

2.1. Mathematical Formulation & Algorithm

The following steps summarize the steepest descent approach where $f(X)$ is the objective function. The objective function in our case the thrust coefficient $-C_t$

1) Start with an arbitrary initial point X_1. Set the iteration number as $i = 1$.

2) Find the search direction S_i as

$$S_i = \nabla f_i = \nabla f\left(X_i\right).$$

3) Determine the optimal step length λ_i^* in the direction S_i and set

$$X_{i+1} = X_i - \lambda_i^* S_i = X_i - \lambda_i^* \nabla f_i.$$

4) Test the new point, X_{i+1} for optimality. If X_{i+1} is optimum, stop the process. Otherwise, go to step 5.

5) Set the new iteration number $i = i + 1$ and go to step 2.

The method of Steepest Descent may appear to be the best-unconstrained optimization technique since each one-dimensional search starts in the optimal direction. It should be noted that evaluating the gradient components requires an unsteady flow solution over a few periods of the flapping motion until the periodic flow behavior is reached.

The step size and initial point are very crucial while using the Steepest Descent Algorithm. For a function having multiple optimum values, different initial values converge to different optimal solutions. This is discussed with an example below. Step size in the algorithm may be determined by one of the following methods.

1) Constant Step Size:

A constant step size is used irrespective of the objective function. The optimization is not in the user's control and this method is used very often for checking the convergence. This generally takes more computational cost when compared to other methods.

2) Linear Search Methods:

Famous linear search techniques like Fibonacci Search, Golden Section, and

Quadratic Interpolation Methods can be used to find the optimum step size for every perturbation. These techniques require the calculation of the objective function value at every experiment. However, the step size changes dynamically which decreases the computational cost when compared to the constant step size.

3) Adaptive Step:

In this method, the previous step size is incremented or decremented depending on certain conditions. This is a blend of Newton-Marquardt's Approach where the step size is increased when we are going in the correct path and decreased otherwise. The decision can be taken based on two successive values of the Objective Function.

If $f(X_{i+1}) < f(X_i)$ then SET $\lambda_{i+1} = c_1 \lambda_i$ where $c_1 > 1$

Else SET $\lambda_{i+1} = c_2 \lambda_i$ where $0 < c_2 < 1$

The constants are chosen as 1.2 and 0.7 respectively. This technique is more optimal when compared to the methods discussed earlier. It is the other way round for a Steepest Descent Approach.

4) Momentum Term Approach:

This technique is proposed by Rumelhart which uses the previous update to compute the current one by introducing a momentum term $\eta \in]0,1[$. The new point is then

$$X_{i+1} = X_i + \eta\left[X_i - X_{i-1}\right] + \lambda_i^* S$$

This is a non-traditional approach without theoretical support which gives good results. It can be traced as an average gradient descent that is sometimes used to improve gradient descent.

In the present problem, we are encountered with a situation where a continuous presentation of the Objective Function and we calculate its value at a particular point numerically using the flow solver. The solver takes a lot of time to calculate the function value; therefore, the linear search methods are not employed in step-size computation.

2.2. Convergence Criteria & Validation

There are three convergence criteria for the Steepest Ascent Algorithm for terminating the iterative process.

1) The first criterion is based on changes in functional values

$$\left|\frac{f(X_{i+1}) - f(X_i)}{f(X_i)}\right| \le \varepsilon_1$$

2) The second criterion is based on the change in functional derivative

$$\left|\frac{\partial f}{\partial x_i}\right| \le \varepsilon_2$$

3) The third criterion is based on the change in the design vector

$$\left|X_{i+1} - X_i\right| \le \varepsilon_3$$

Out of the three, the first one is the most frequently used convergence criteria. In some cases, the process can be terminated after the desired number of iterations.

Both Steepest Ascent and Steepest Descent Algorithms are validated using well-known functions as objective functions, out of which three of them are discussed here. The optimization is done on the Windows Platform in Dev-C++ software which uses a "*gcc*" compiler and the computational cost in terms of no. of iterations is calculated by three methods outlined earlier

1) Basic Function

$$y_1 = \sin(x_1) + 2\cos(x_2) - \sin(x_3)$$

2) Himmelblau's Function

$$y_2 = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2$$

3) Trid Function

$$y_3 = \sum_{i=1}^{n}\left(x_i - 1\right)^2 - \sum_{i=2}^{n} x_i x_{i-1}$$

Without loss of generality number of variables is chosen to be $n = 6$.

The convergence criteria and the initial step length are taken to be 0.000001 and 0.001 respectively. The number of iterations taken for a particular function to optimize is calculated and is found that the Adaptive Step is the best regarding computational cost. The dependence of the algorithm on the starting point can be spotted below.

The adaptive step size approach shows the near problem-independent convergence for all test functions and is chosen due to its simplicity and lower cost in the present work to decide on the dynamic step size.

The plots of Himmelblau's function and Trid function (for two variables) are plotted in MATLAB to visualize the occurrence of the optimum solution (**Figure 1** & **Figure 2**). As discussed in **Table 1** the validation of Himmelblau's function which has multiple optima depending on the initial guess and the Trid Function which has one global minimum for two variables can be made.

3. Optimization Framework

In this section, the numerical tools and techniques used in trajectory definition and cost function evaluation as well as the development of the integrated solver-optimization framework were discussed. It begins with a clear description of the airfoil regime followed by a clear description of the cost function being optimized during the study. Additionally, the use of NURBS is taken up in greater detail along with a description of the parallel architecture that has been implemented in the framework. Both of these would form crucial components of the entire study at a later stage when the flow solver and optimization routine would run in tandem to generate results.

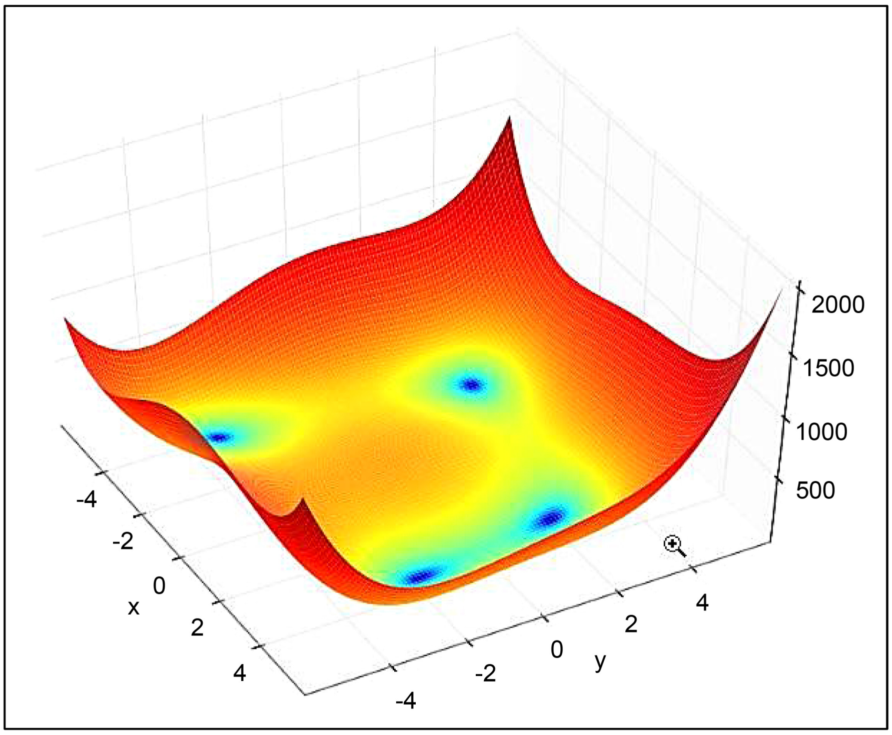

Figure 1. Himmelblau's function.

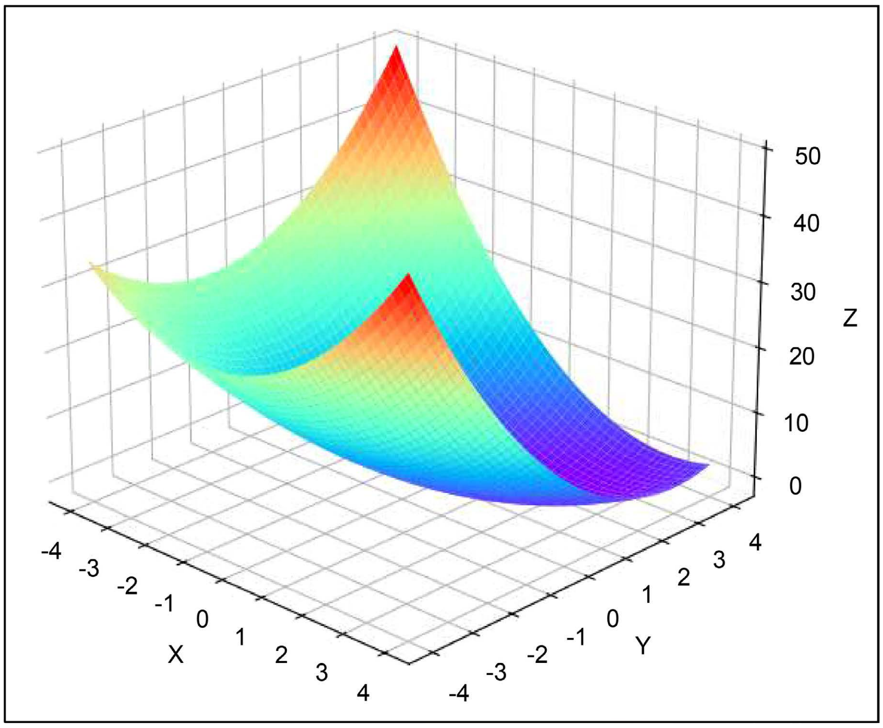

Figure 2. Trid function.

3.1. Objective Function

The present work aims at maximizing the time-averaged thrust being generated for the given flow conditions and characteristics. This average quantity is defined

Table 1. Various function characteristics with steepest descent algorithm.

Function	Initial Point	Optimal Solution		Number of Iterations		
		Expected	Calculated	Case 1	Case 3	Case 4
Basic Function (*Steepest Ascent*)	$\begin{Bmatrix}1\\1\\1\end{Bmatrix}$	$\begin{Bmatrix}1.570\\0\\-1.570\end{Bmatrix}$	$\begin{Bmatrix}1.570\\0\\-1.571\end{Bmatrix}$	3084	68	1546
Himmelblau's Function (*Steepest Descent*)	$\begin{Bmatrix}1\\1\end{Bmatrix}$	$\begin{Bmatrix}3\\2\end{Bmatrix}$	$\begin{Bmatrix}2.999\\2.000\end{Bmatrix}$	2305	91	1156
	$\begin{Bmatrix}-1\\1\end{Bmatrix}$	$\begin{Bmatrix}-2.805\\3.131\end{Bmatrix}$	$\begin{Bmatrix}-2.805\\3.131\end{Bmatrix}$	2.882	73	1442
	$\begin{Bmatrix}1\\-1\end{Bmatrix}$	$\begin{Bmatrix}3.584\\-1.848\end{Bmatrix}$	$\begin{Bmatrix}3.584\\-1.848\end{Bmatrix}$	3165	90	1588
	$\begin{Bmatrix}-1\\-1\end{Bmatrix}$	$\begin{Bmatrix}-3.779\\-3.283\end{Bmatrix}$	$\begin{Bmatrix}-3.780\\-3.283\end{Bmatrix}$	3793	87	1899
Trid Function (*Steepest Descent*)	$\begin{Bmatrix}1\\1\\1\\1\\1\\1\end{Bmatrix}$	$\begin{Bmatrix}6\\10\\12\\12\\10\\6\end{Bmatrix}$	$\begin{Bmatrix}5.999\\9.999\\11.999\\11.999\\9.999\\5.999\end{Bmatrix}$	21,303	85	10,623

as:

$$\underline{T} = \frac{1}{P}\int_{mP}^{(m+1)P} T(t)\,\mathrm{d}t$$

where T is the time-varying thrust function, and P is the period for the periodic thrust function (the motion being executed by the wing repeats itself after time P). The initial values when the motion is initiated are ignored and when the flow properties reach a stationary state, attaining a periodic nature, a time-averaged value is generated for the period P. The constraints added include a minimum amount of lift to be generated to counter the weight for level flight and the thrust being sufficient to overcome the drag acting on the vehicle.

Computation of the time-averaged thrust coefficient would require the use of a method that would provide for replacing the integral with a summation since the thrust coefficient is available only at discrete time intervals (0.01 non-dimensional time units for the current study). Thus, employing a quadrature rule, which allows approximating definite integrals by stating them as a weighted sum of function values at specified points within the domain of integration, becomes critical. An n-point Gaussian quadrature rule is a quadrature rule constructed to yield an exact result for polynomials of degree $2n - 1$ or less by such choice of sampling points. The time-varying thrust coefficient was assumed to be represented by a polynomial of degree 19 and therefore, by employing a 10-point Gaussian quadrature rule, an integral of acceptable accuracy was obtained for optimization.

3.2. Path Representation Using NURBS

The optimization of the plunge for time can result in non-sinusoidal motion and thus, there is a requirement to able to accurately and lucidly represent different motion paths using a fixed set of parameters. Thus, Non-uniform Rational Bezier Splines (NURBS) are used to represent the different paths. In NURBS, a n^{th} degree Bezier curve is used to represent the required curve. The nth-degree Bezier curve is defined as:

$$C(u)=\frac{\sum_{i=0}^{n}\left[B_{i,n}(u)w_i P_i\right]}{\sum_{i=0}^{n}\left[B_{i,n}(u)w_i\right]}$$

$$0<u<1$$

where $B_{i,n}(u)$ is the n^{th} degree Bernstein's polynomial which is the basis of the blending functions, P_i are the control points, w_i are the scalar weights. To ensure smooth curves, we choose the w_i's as 1, 0.6, 0.8, 0.8, 0.6 and 1. The n^{th} degree Bernstein's polynomial is defined as:

$$B_{i,n}(u)=\frac{n!u^i(1-u)^{n-i}}{i!(n-i)!}$$

$$0<u<1$$

NURBS are piecewise rational polynomial representations of a curve. Bernstein's polynomials or the blending functions used in NURBS are used to blend different parts of the curve to give a smooth and continuous curve. Each control point influences the final curve according to the assigned blending function. A blending function defines the weight of the control point at each point of the curve.

Figure 3 shows a NURBS curve with control points. By joining the control points we obtain a control polygon. The shape of the control polygon controls the shape of the NURBS curve. As seen in the figure the NURBS curve has a starting slope equal to the slope of the control polygon line, joining the first two control points. Then the shape of the NURBS curve evolves according to the control polygon lines, ensuring a smooth and continuous curve. Again the slope of the NURBS curve at the end of the curve is equal to that of the control polygon line. By choosing different sets of P_i's, we get different motion curves.

These control points are the optimization parameters as the right combination of these control points determines the motion path or curve with the best efficiency under the given constraints. The control points determine the plunge path of the airfoil. Hence, certain constraints are imposed on the control points so that the motion paths obtained are practically achievable. Constraints are continuity, amplitude, and time period.

These control points are the optimization variables. Each motion curve is defined by six control points. Since the first and the last control points are fixed, only the remaining four control points are the optimization variables. The x and y coordinates of these four control points make the number of optimization

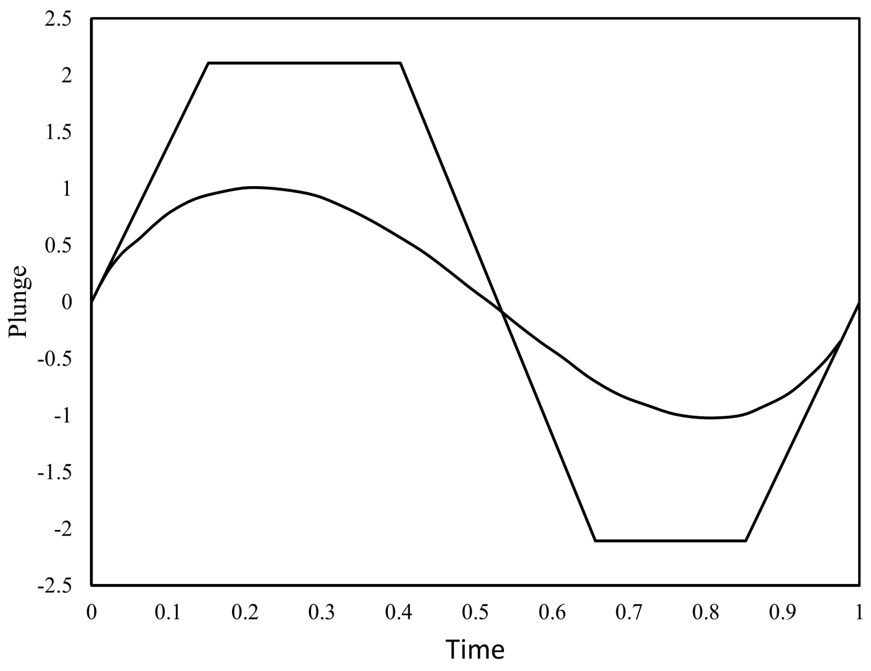

Figure 3. NURBS control polygon.

variables equal to eight. Also, the continuity constraint fixes the y coordinate of the fifth control point. Hence the total number of optimization variables is reduced to seven. These seven optimization variables can take any value in the search space subject to the above constraints.

4. Overall Validation & Results

Steepest descent optimization was taken up for implementation within the current framework involving parallel process management and NURBS. In this section, an attempt is made to outline the flow of information and the sequence of steps while running the optimization algorithm.

4.1. Optimization Procedure Applied

The objective function evaluation is performed on "n" slave processors (where n has been limited to seven by a choice of seven decision variables). Hence, each slave processor performs an unsteady flow computation for a plunge motion path during every optimization cycle. The other optimization steps are carried out by the master processor.

1) Initialization: The initial plunge motion path is randomly initialized by starting with a control point coordinate which then generates the path based on the method of NURBS. The constraints are enumerated before enforced while evaluating these points. The time-averaged thrust value is calculated at the base value using the unsteady flow solver.

2) Perturbation: The cost function is discrete in this case, so the gradients are obtained by perturbing the control variables and re-evaluating the cost function.

3) Objective Function Evaluation: The perturbed control points indicate marginally different motion paths and the CFD simulation for these motions gives the thrust coefficients for the perturbed motion trajectory.

4) Gradient Computation: The gradients needed for optimization are calculated using numerical differentiation by using the thrust coefficients obtained in b. and c. In practice, a perturbation of $h = 10^{-3}$ suffices to obtain accurate gradients.

5) Update: A new set of control points is calculated, with the help of the optimum step length obtained by the Adaptive Scheme.

Convergence: If the convergence criterion is not satisfied they are sent as the initial values for the next optimization iteration. The process is continued till the optimum curve is evolved.

4.2. Optimization with NURBS Excluded

The first batch of runs was aimed at validating the successful implementation of the steepest descent optimization algorithm in the current parallel framework designed using shell scripting and relying on numerical differentiation for the computation of gradients. This phase excluded the use of the NURBS routine for generating curves that acted as an intermediary between the actual decision variables and the output function.

The mathematical program replacing the CFD solver generated output based on the Trid function. This simplified routine was plugged in instead of the actual CFD solver and multiple runs were completed based on random initial values. The program converged satisfactorily to the optimal function value in each case. Figure 4 presents the evolution of the function value with increasing optimization iterations for initial values of (1, 2, 3, 4, 5, and 7) and its subsequent convergence to the optimal value of 0 after 69 optimization cycles. Thus, the optimization framework developed stands validated with its efficacy displayed in optimizing the Trid Function.

4.3. Optimization with Full Functionality NURBS Included

The next step towards full validation of the motion optimization framework running in conjecture with the steepest descent optimization algorithm was to incorporate the usage of curves generated using NURBS. The mathematical routine mimicking the CFD program was modified accordingly to receive input in the form of a curve (limited to a set of discrete points describing the curve) generated from the set of control points (designated as the decision variables). The fake program generated an output that was equal to the total absolute area under the curve. The area under the curve was approximated using the trapezoidal rule (1-point Gauss Quadrature rule).

The framework was initialized using the set of control points: (0, 0), (1.256, 1.1), (2.512, 1.1), (3.768, −1.1), (5.024, −1.1) and (6.28, 0), that produced an approximately sinusoidal curve. Figure 5 presents the value of the output function

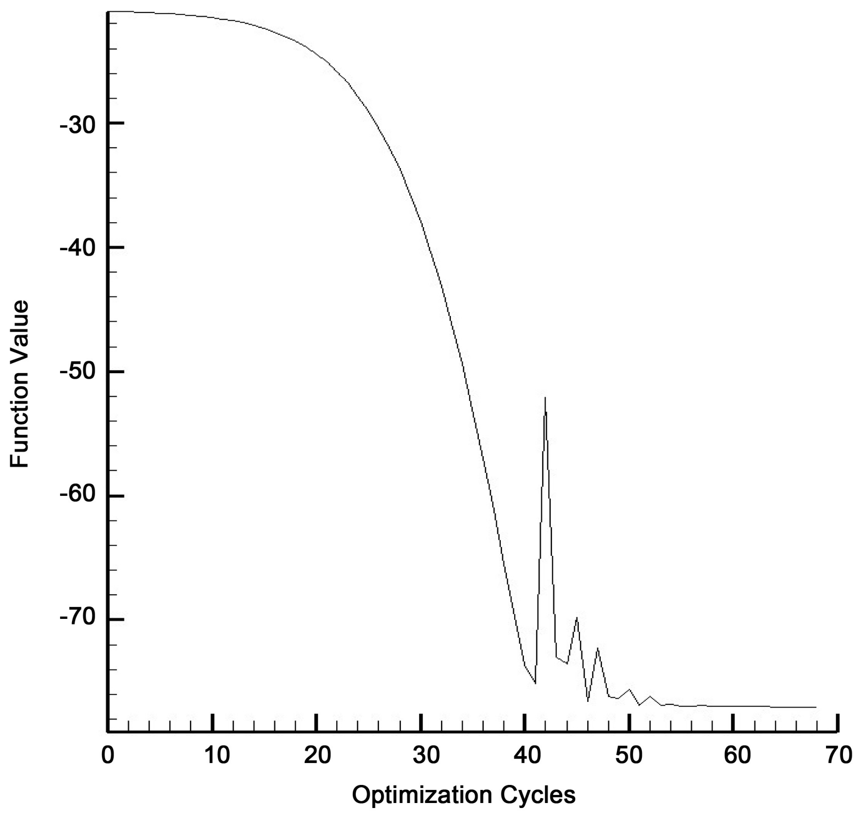

Figure 4. Steepest descent on trid function.

and its subsequent convergence to zero within the given error threshold after 61 cycles. Figure 6 presents the evolution of the curve with increasing optimization iterations. The entire exercise, therefore, successfully demonstrates the usage of curves generated using NURBS as a means of providing input to the output function with only the control points used by NURBS being the decision variables.

4.4. Case Study: Sinusoidal & Non-Sinusoidal Motion

To illustrate the need and possible mechanism of thrust production in pure plunge, we consider the simulations of the NACA 0014 airfoil at Re = 10,000 and M = 0.1 for two different motion trajectories. The first is a sinusoidal motion with a plunge amplitude of 0.3 and the second is a non-sinusoidal motion as shown in Figure 7.

The thrust history (represented by the instantaneous values of the reversed drag coefficient) for two plunge cycles are shown for either case in Figure 8. The non-sinusoidal motion is more thrust-producing than the sinusoidal motion, which is reflected in the fact that the thrust coefficient in the former case is four times that of the latter.

The vorticity contours for either case at the mean and maximum amplitude positions (beginning of downstroke and upstroke) are shown in Figures 9-14 for both cases. Although not very evident from these plots, an analysis of the vorticity

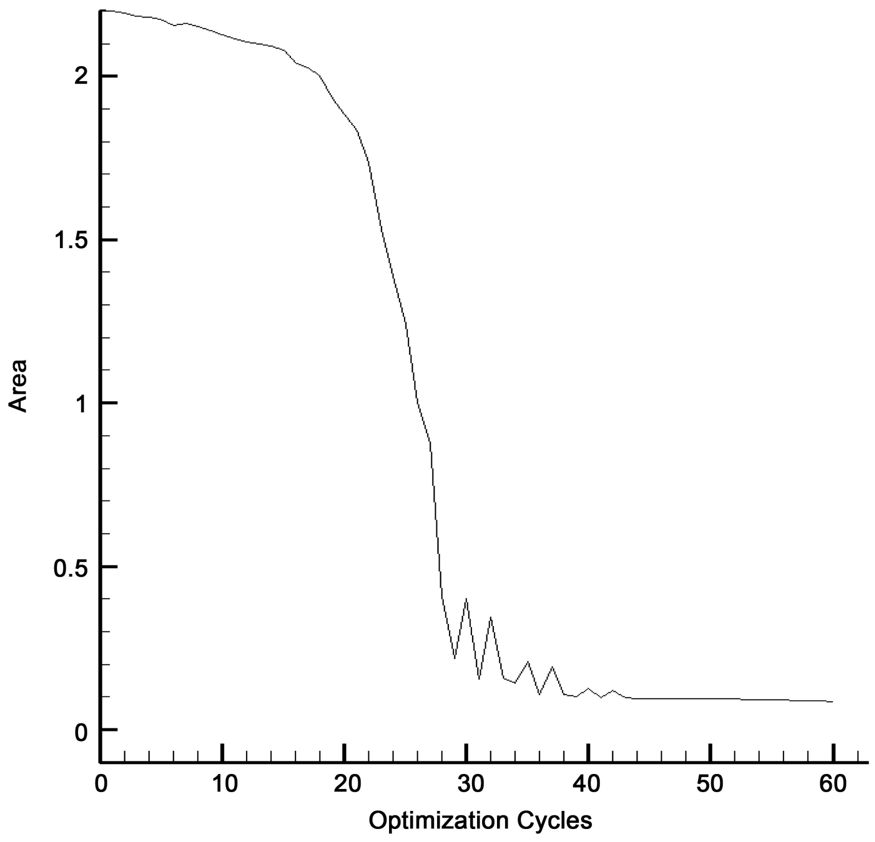

Figure 5. Steepest descent with NURBS for area optimization.

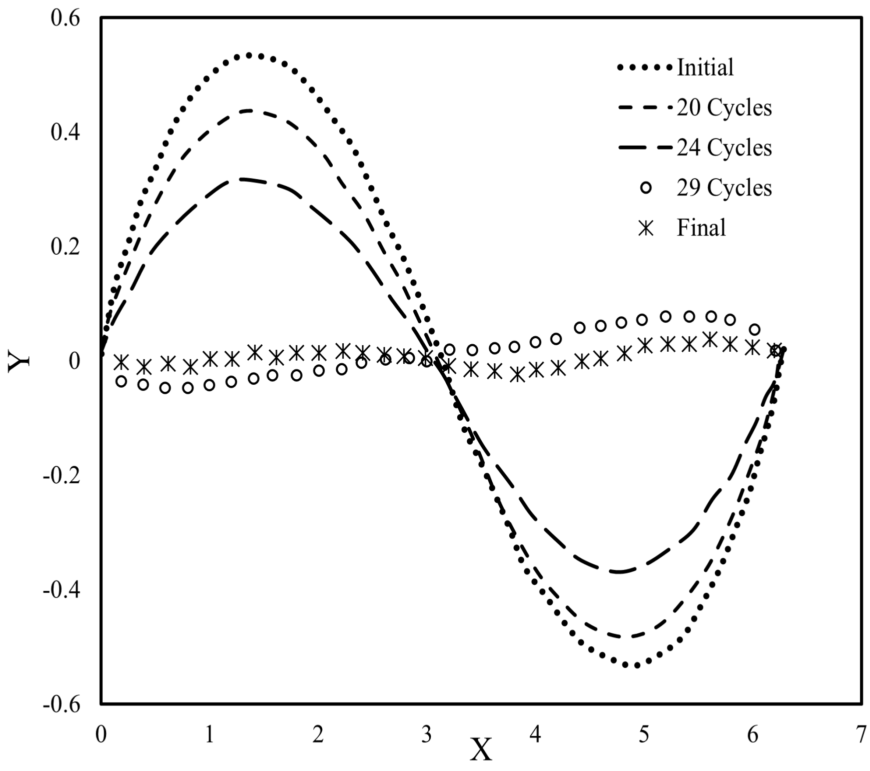

Figure 6. Evolution of input curves with optimization cycles.

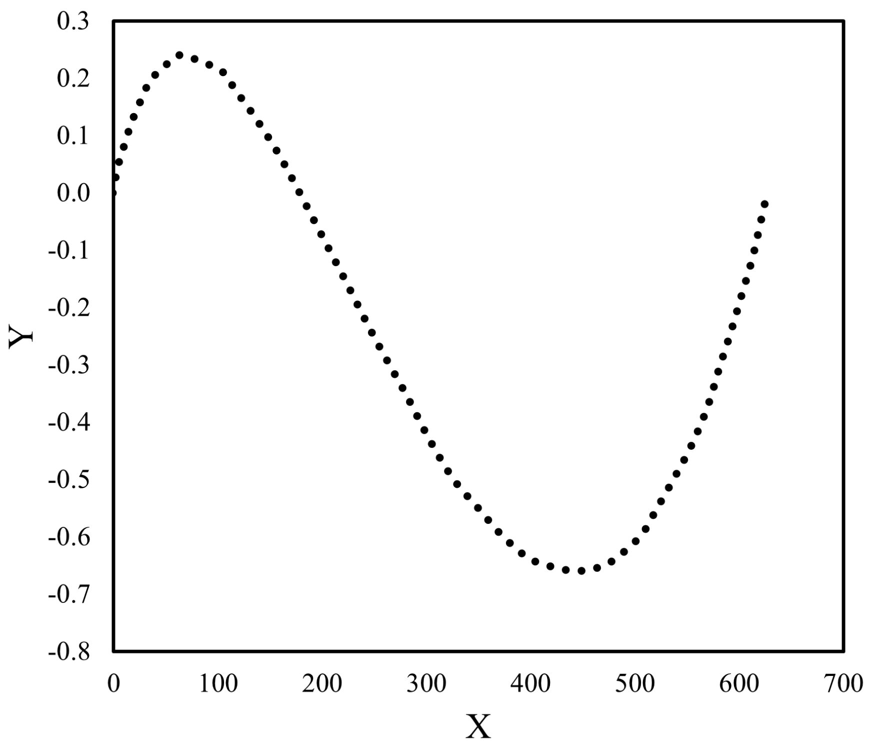

Figure 7. Non-sinusoidal motion curve.

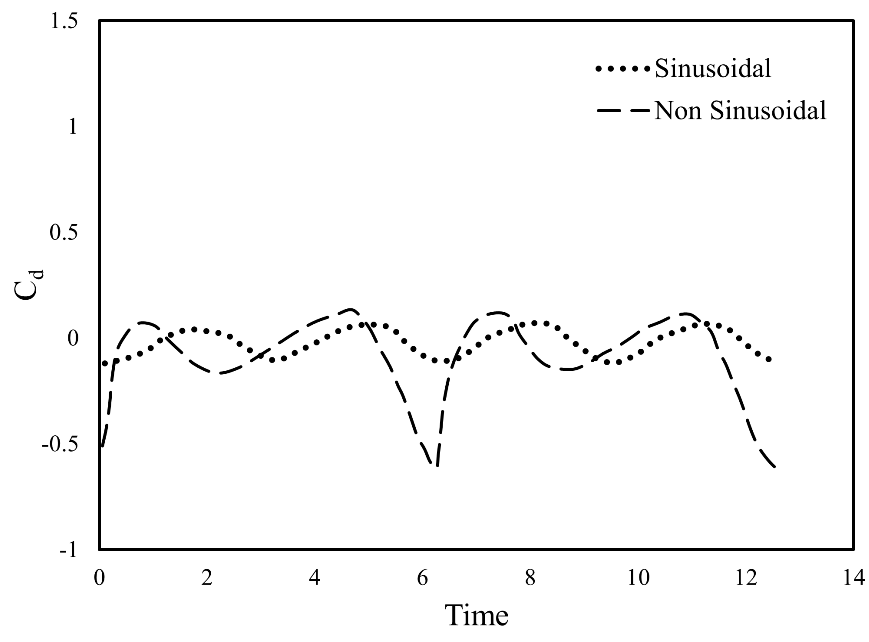

Figure 8. Time-varying drag for sinusoidal & non-sinusoidal.

patterns in a complete cycle indicates that while they are similar for both motion trajectories, the vortex strength in the non-sinusoidal case is greater than that in the symmetric sinusoidal case. This reaffirms the fact that the momentum excess in the wake when plunging in a non-sinusoidal trajectory is larger compared to the sinusoidal trajectory, which appears as a larger time-averaged thrust.

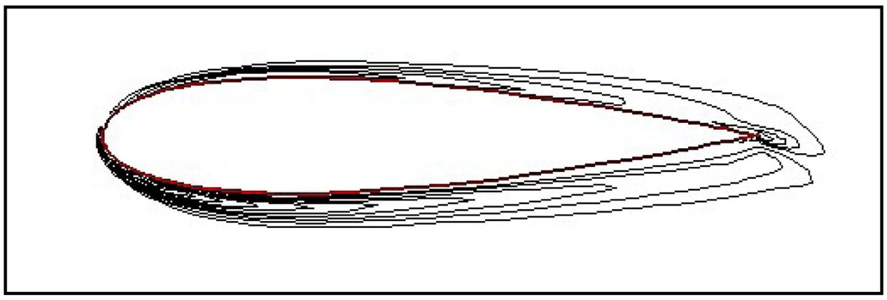

Figure 9. Mean vorticitycontours—sinusoidal motion.

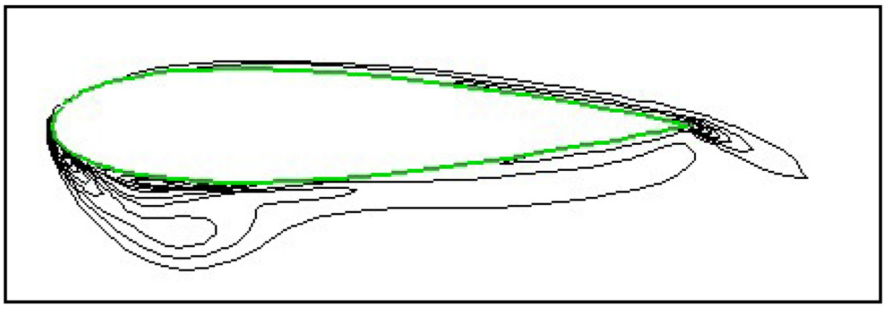

Figure 10. Mean vorticitycontours—non sinusoidal motion.

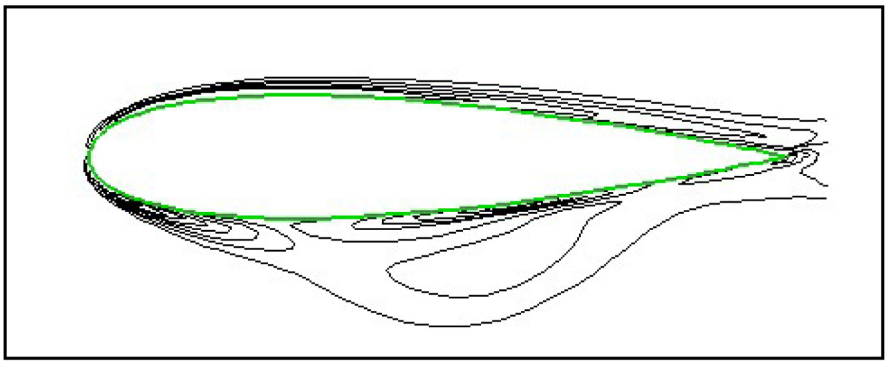

Figure 11. Vorticitycontours—sinusoidal motion—downstroke.

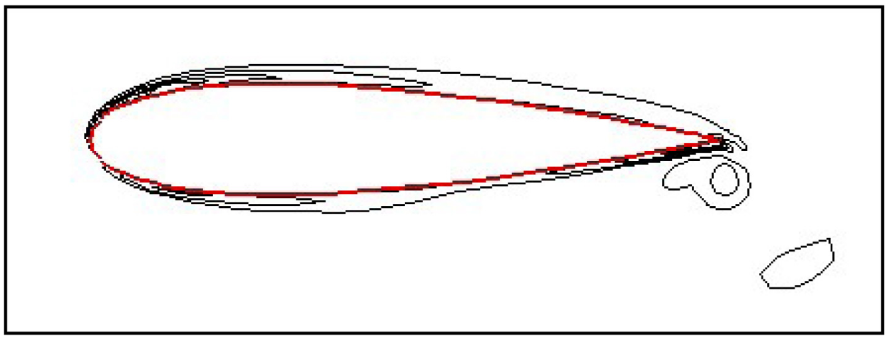

Figure 12. Vorticitycontours—non-sinusoidal motion—downstroke.

Based on an Initial input curve that was sinusoidal and generated by the control points (0, 0), (1.256, 1.3), (2.512, 1.3), (3.768, −1.3), (5.024, −1.3) and (6.28, −1.3). The motion was expected to turn increasingly non-sinusoidal with every

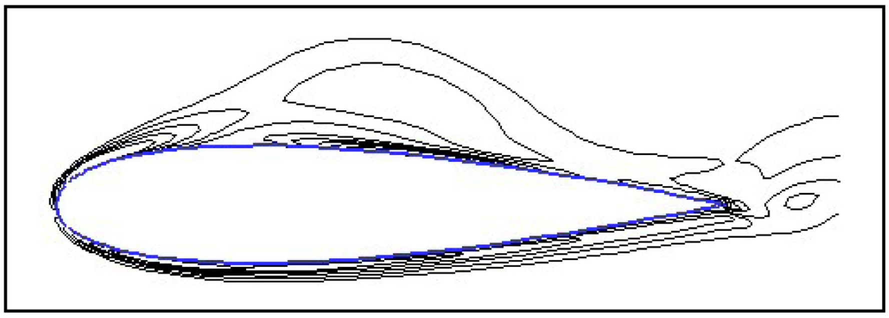

Figure 13. Vorticitycontours—sinusoidal motion—upstroke.

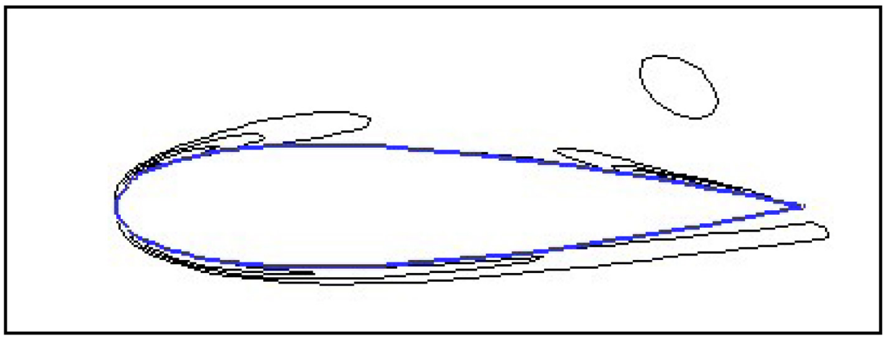

Figure 14. Vorticitycontours—nonsinusoidal motion—upstroke.

subsequent optimization cycle and register a definite increase in thrust generation. But despite all attempts at ensuring the correct operation of each tool and the overall integrated framework by repeated validation at every step of the process, the steepest descent failed to work satisfactorily in tandem with the CFD solver within the given programming architecture. **Figure 15** which presents the time-averaged drag coefficient (negative of the thrust coefficient, as a negative value of drag indicates net thrust being generated) to the total optimization cycles ran displays no clear trend and therefore, it is possible to conclude that this technique fails to function correctly when applied in the manner envisaged during the study. Additionally, it was observed that the evolution of the motion curves with increasing optimization cycles also showed no tendency to converge onto a certain kind of motion and continued to oscillate around the initial curve that was inputted.

Multiple attempts were made to enable the correct functioning of the steepest descent algorithm for obtaining optimized results but were largely unsuccessful. A change in the choice of initial value for the motion curves also was unable to resolve the issue. In light of the validation attempts (that were very successful) described before, it was concluded that the improper functioning of the algorithm was due to incorrect computation of the gradient vector at each optimization step. The gradient vector computation is based on an accurate time-varying value of the thrust coefficient obtained from the CFD solver and the subsequent time-averaging. Thus, it has been strongly speculated that the order of accuracy obtained from these two operations working in combination is not high enough

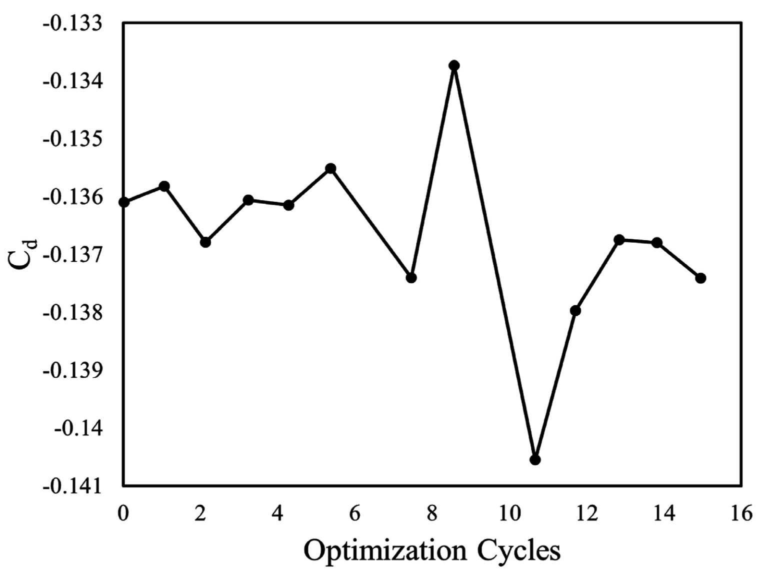

Figure 15. Change in drag with optimization cycles.

to allow for accurate gradient vector computation paving the way for an overall failure of the technique. It is worthwhile to note that a switch to a trapezoidal rule (*1*-point Gauss Quadrature) from a 10-point Gauss Quadrature was also implemented to overcome these shortcomings but the results obtained continued to be inconclusive.

5. Conclusion

In this article, we studied the optimization aspects of flapping wing airfoil. Even though there are multiple aspects of the flapping wing such as pitch, plunge, and rolling—only the plunging aspect in 2D was studied as a simple first step. A few types of thrust profiles were considered to mimic the plunge motion, which were then optimized via the Gradient descent optimization technique. The optimization worked in parallel with an in-house FVM solver that enables dynamic meshing. It has been concluded that the algorithm being a derivative-based technique, was unable to reach an optimized state to calculate the maximum thrust for a 2D airfoil, even with various stepping procedures, and starting points. Further advanced techniques that are derivative-free will be studied as the next step to aid in comparison and form a path forward.

Statement

"Department of Mechanical Engineering, IIT Guwahati, Guwahati, India" is the affiliation when work was conducted. I hereby state that the mentioned affiliation is no longer my current affiliation.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Katzmayr, R. (1992) Effect of Periodic Changes of Angle of Attack on Behavior of Airfoils. https://api.semanticscholar.org/CorpusID:107198920

[2] Dickinson, M.H., Lehmann, F.-O. and Sane, S.P. (1999) Wing Rotation and Aerodynamic Basis of Insect Flight. *Science*, **284**, 1954-1960. https://doi.org/10.1126/science.284.5422.1954

[3] Tuncer, I.H. and Kaya, M. (2005) Parallel Optimization of Flapping Airfoils in a Biplane Configuration for Maximum Thrust. In: Winter, G., Ecer, A., *et al*., Eds., *Parallel Computational Fluid Dynamics* 2004, Elsevier Science, 137-144. https://www.sciencedirect.com/science/article/abs/pii/B978044452024150018X https://doi.org/10.1016/B978-044452024-1/50018-X

[4] Tuncer, I.H. and Kaya, M. (2003) Thrust Generation Caused by Flapping Airfoils in a Biplane Configuration. *Journal of Aircraft*, **40**, 509. https://doi.org/10.2514/2.3124

[5] Tuncer, I.H. and Kaya, M. (2005) Optimization of Flapping Airfoils for Maximum Thrust and Propulsive Efficiency. *AIAA Journal*, **43**, 2329-2336. https://doi.org/10.2514/1.816

[6] Piegel, L. and Tiller, W. (1997) The NURBS Book. Springer, Berlin. https://doi.org/10.1007/978-3-642-59223-2

[7] Jameson, A. (1991) Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings. Honolulu, USA. https://doi.org/10.2514/6.1991-1596