60M60

Turbomachinery Airfoil Design Optimization Using Differential Evolution

Nateri K. Madavan

NASA Advanced Supercomputing Division,
NASA Ames Research Center, Moffett Field, CA 94035-1000, USA

Abstract. An aerodynamic design optimization procedure that is based on a evolutionary algorithm known at Differential Evolution is described. Differential Evolution is a simple, fast, and robust evolutionary strategy that has been proven effective in determining the global optimum for several difficult optimization problems, including highly nonlinear systems with discontinuities and multiple local optima. The method is combined with a Navier-Stokes solver that evaluates the various intermediate designs and provides inputs to the optimization procedure. An efficient constraint handling mechanism is also incorporated. Results are presented for the inverse design of a turbine airfoil from a modern jet engine. The capability of the method to search large design spaces and obtain the optimal airfoils in an automatic fashion is demonstrated. Substantial reductions in the overall computing time requirements are achieved by using the algorithm in conjunction with neural networks.

1 Introduction

Remarkable progress has been made in recent years in the ability to design turbomachinery airfoil shapes that are optimal with regard to certain desired characteristics. This progress has been achieved by combining improved methods for predicting the complicated flow fields in turbomachinery with efficient numerical optimization techniques and by harnessing the powerful capabilities of modern computers. Both steady and unsteady Navier-Stokes and Euler solvers have been combined with various optimization techniques (gradient-based methods [1], [2], response surfaces, etc.) to optimize the design of turbomachinery airfoils.

More recently, there has been considerable interest in the development of turbomachinery airfoil design optimization techniques that are based on nontraditional approaches such as evolutionary algorithms and neural networks. Various approaches based on neural networks (see, for example, [3], [4], [5]), neural networks in conjunction with response surfaces [6], [7], genetic algorithms [8], [9], and genetic algorithms in conjunction with neural networks [10],[11],[12] have been reported in the literature. These techniques offer several advantages over traditional optimization methods.

This paper deals with the development of a turbomachinery airfoil design optimization procedure that is based on a relatively new evolutionary algorithm known as Differential Evolution [13] developed for single-objective optimization in continuous search spaces. It is conceptually simple and possesses good convergence properties that have been demonstrated in a variety of applications. Differential Evolution (DE) is best characterized as an evolutionary strategy (ES) rather than as a genetic algorithm (GA), although the distinction between GAs and ESs has blurred in recent years. Perhaps the main ideological difference lies in the relative importance given to the two main evolutionary operators, recombination (crossover) and mutation, with GA-based approaches relying heavily on the former and ES-based approaches on the latter. DE has proven to be an effective approach in determining the global optimum for several difficult optimization problems in a variety of applications. Its application in aeronautics, however, has been rather limited. It has been used in the predictive control of aircraft dynamics [14]. DE has been used in conjunction with a potential flow solver in the inverse design of turbomachinery airfoils [15]; the same authors have also presented a hybridized version [16] that combined DE with a local search method to minimize the number of objective function evaluations using the potential flow solver.

In this paper the DE algorithm is combined with a Navier-Stokes solver that provides inputs to the optimization procedure. An efficient constraint handling mechanism is also incorporated in the algorithm. An airfoil geometry parametrization that uses a minimal number of variables is also used to minimize the number of objective function evaluations. The procedure is also combined with neural networks that are incrementally trained on the Navier-Stokes simulation data and can then be used in the objective function evaluation. This results in substantial reductions in the overall computing time. Additionally, the procedure has been implemented on a distributed parallel computer in a straightforward manner that relies on the simultaneous computation of multiple, independent aerodynamic simulations on separate processors. The procedure is primarily script-based and allows for a variable number of processors to be used depending on the size of the population used in the DE algorithm. Details of the method and its implementation and results for the inverse design of a turbine airfoil to demonstrate its capabilities are described.

2 Design Optimization Method

The main ingredients of the design optimization method are discussed in this section. The DE algorithm, the airfoil geometry parametrization procedure, and the CFD flow solver used for evaluating the objective function are discussed briefly. Some details regarding the implementation of the method on parallel distributed computers are also provided. A hybrid version that helps reduce computational cost by combining the DE algorithm with a neural network approach is also described.

2.1 Differential Evolution

Differential Evolution is an ES-based approach developed for single-objective optimization in continuous search spaces. It is conceptually simple and possesses good convergence properties that have been demonstrated in a variety of applications [18]. Details of the algorithm can be found elsewhere [13], [17]; only its main features are summarized here.

The approach uses a population $\boldsymbol{P}^Q$ that contains N K-dimensional real-valued parameter vectors in generation Q, where K is the number of parameters or decision variables:

$$P^Q = P_i^Q = p_{i,k}^Q \ , \ i = 1, .., N \ ; \ k = 1, ..., K \tag{1}$$

The population is usually initialized in a random fashion and the population size N is maintained constant throughout the optimization process. Differential evolution is thus similar to a $(\mu + \lambda)$ ES [19] with μ and λ equal to N [20]. The method however differs from standard ES approaches in several respects as described below.

As with all ES-based approaches, mutation is the key ingredient of differential evolution. The basic idea is to generate new parameter vectors for the subsequent generation by using weighted differences between two (or more) parameter vectors selected randomly from the current population to provide appropriately scaled perturbations that modify another parameter vector (or, comparison vector) selected from the same population. This idea has been implemented in various forms but the form discussed and used here is the classical implementation where new trial parameter vectors $P_{i,k}^{Q+1}$ for the next generation $Q + 1$ are generated according to the following mutation scheme:

$$\tilde{p}_{i,k}^{Q+1} = p_{a_i,k}^Q + F \cdot (p_{b_i,k}^Q - p_{c_i,k}^Q) \ , \ i = 1, ..., N \ ; \ k = 1, ..., K \tag{2}$$

where

$$a \ , \ b \ , \ c \in \{1, ..., N\} \ ; \ a_i \neq b_i \neq c_i \neq i$$

The integers a_i, b_i, and c_i represent three random individuals of the population that are mutually different from each other and from the running index i. The mutation parameter $F \in [0, 2]$ is a real, constant, user-supplied parameter that controls the amplification of the differential variation. Other variants that either use the difference between more than two parameter vectors or keep track of the best parameter vector at each generation and use it in the mutation scheme have also been developed [9] and used with varying success in specific applications. Thus, differential evolution differs from other ESs most notably in that the mutation operator is derived from the current population and not by probability density functions that are defined separately.

DE is similar to other recombinative ES approaches in that it also uses discrete recombination. While various recombination strategies exist [19] the strategy adopted in differential evolution is to modify the trial parameter vectors $\tilde{P}_i^{Q+1}$ as follows:

$$\tilde{p}_{i,k}^{Q+1} = \tilde{p}_{i,k}^{Q+1} \ \text{ if } \ r_{i,k} \leq C_r \ \text{ or } \ k = d_k \tag{4}$$

and

$$\tilde{p}_{i,k}^{Q+1} = p_{i,k}^Q \ \text{ otherwise} \tag{5}$$

In the above, $r \in [0,1]$ is a uniform random number, $C_r \in [0,1]$ is the crossover parameter, and d_k is a randomly chosen index in $\{1, ..., K\}$ that ensures that $\tilde{p}_{i,k}^{Q+1}$ gets at least one parameter from the mutated $\tilde{p}_{i,k}^{Q+1}$ and not all from $p_{i,k}^{Q}$. Note also that the mutation and recombination operations described above can lead to new vectors that may fall outside the boundaries of the variables. Various repair rules can be used to ensure that these inadmissible vectors do not enter the population. A simple strategy, which is the one adopted here, is to delete these inadmissible vectors and form new ones until the population is filled.

The selection scheme used in DE is deterministic but differs from methods usually employed in standard ES approaches. Selection is based on local competition only, with the child trial vector (or, child) $\tilde{P}_i^{Q+1}$ competing against one population member P_i^Q (the comparison vector) and the survivor entering the new population P_i^{Q+1}. In other words, if $\tilde{P}_i^{Q+1}$ yields a better objective function value than P_i^Q then P_i^{Q+1} is set to $\tilde{P}_i^{Q+1}$. Otherwise, the old value P_i^Q is retained. This greedy selection criterion results in fast convergence; the adaptive nature of the mutation operator, in general, helps safeguard against premature convergence and allows the process to extricate itself from local optima.

An efficient constraint handling mechanism has been incorporated into the algorithm. Details are given elsewhere; briefly, it is a parameter-less penalty function approach where infeasible solutions are penalized and help guide the algorithm away from these regions. Physical constraints, e.g., maximum airfoil thickness, etc., are imposed, as well as aerodynamic constraints (wavy surfaces, etc.). Airfoil geometries that meet the constraints but for which the CFD solver is unable to converge or "blows up" are also deemed infeasible and penalized accordingly.

2.2 Airfoil Geometry Parametrization

Geometry parametrization and prudent selection of design variables are among the most critical aspects of any shape optimization procedure. The ability to represent various airfoil geometries with a common set of geometrical parameters is essential. Variations of the airfoil geometry can be obtained then by smoothly varying these parameters. Geometrical constraints imposed for various reasons, such as structural, aerodynamic (e.g., to eliminate flow separation), etc., should be included in this parametric representation as much as possible. Additionally, the smallest number of parameters should be used to represent the family of airfoils. Here, the airfoil geometry parametrization method described in [6] that uses a total of 13 parameters to define the turbine airfoil geometry is used. Figure 1 illustrates the method for a generic airfoil. The geometric parameters used are the leading edge and trailing edge airfoil metal angles (2 parameters), eccentricity of upper leading edge ellipse (1 parameter), angles defining the extent of the leading edge ellipses (2 parameters), semi-minor axes values at the leading edge (2 parameters), angles defining the extent of the trailing edge circle (2 parameters), airfoil y-coordinate values at about 50% chord on the upper and lower surfaces (2 parameters), and airfoil y-coordinate values at about 25% and 75% chord on the upper surface (2 parameters).

2.3 Flow Solver for Objective Function Evaluation

A two-dimensional Navier-Stokes solver is used to perform the flow simulations (direct function evaluations) that serve as inputs to the optimization process. Multiple grids are used to discretize the flow domain; an inner "O" grid that contains the airfoil and an outer "H" grid that conforms to the external boundaries as shown in Fig. 3. The flow parameters that are specified are the turbine pressure ratio, inlet temperature and flow angle, flow coefficient, and unit Reynolds number based on inlet conditions.

2.4 Parallel Implementation Details

In order to reduce overall design time, the procedure has been implemented on distributed parallel computers. The results in this article were obtained on the SGI Origin 3000 and the Cray SV1 at NASA Ames Researach Center. The implementation of the method is quite straightforward and relies on the simultaneous computation of multiple, independent aerodynamic simulations on separate processors. The procedure is primarily script-based and allows for a variable number of processors to be used depending on processor availability and the size of the population used in the DE algorithm. The number of processors can also be adjusted as the design proceeds. The current setup is based on a "master-slave" arrangement, with the master handling the tasks of setting up the simulations, neural network training for the hybrid method, and farming out of the aerodynamic computations to the other "slave" processors. Since the aerodynamic computations are independent of each other, no communication between the processors is required until the computations are completed. The slave processors then communicate their results to the master which then performs the necessary calculations to determine the members of the next population.

2.5 Hybrid Differential Evolution–Neural Network (DE-NN) Approach

While the DE algorithm is quite efficient in terms of exploring the entire design space in its search for the optimum solution, there is a tendency for the algorithm to slow down after it approaches the vicinity of the optimal solution and several function evaluations are required in order to obtain the exact optimal solution. This can be remedied by a hybrid approach that combines the DE algorithm with neural networks that are incrementally trained on the Navier-Stokes simulation data. The trained neural network can then be used to evaluate the objective function evaluation with little or no computational expense instead of using the Navier-Stokes solver. While hybridization is always useful, it must be done with caution. Here we resort to it only in the latter stages after the entire population has evolved to the general vicinity of the optimal solution. Thus the neural network is used as a "local" response surface with validity only in a small region of the design space. This makes it easier to train the neural network and improves its generalization abilities.

The results obtained in this article were obtained using a feed-forward neural network with two hidden layers as shown in Fig. 3. The first node in the input layer is a bias node (input of 1.0) and the remaining input nodes are used to specify the various design parameters. For the inverse design optimization problem, the neural network is trained on the sum-of-squares error between the actual pressure and the target pressure at various points on the airfoil. For the sake of brevity, details such as the number of neurons in each layer, etc. are not included here.

3 Results

The design method was used in the inverse design of a turbine airfoil with a specified pressure distribution. The target pressure distribution was obtained at the midspan of a turbine vane from a modern jet engine and was supplied by Pratt and Whitney (Private Communication, F. Huber, 1997). Several flow and geometry parameters were also supplied and used in the design process. The design objective function was formulated as the equally-weighted sum-of-squares error between the target and actual pressure obtained during the optimization process at 45 locations on the airfoil.

The initial design space was chosen to be quite large to allow for a wide range of airfoil shapes to be explored. In order to hold the CFD function evaluations to a reasonable number, 6 (instead of 13) design variables were used in the initial stages of the design. The population of 50 members were then evolved using the DE algorithm. The DE mutation and crossover parameters were both chosen to be 0.8 based on prior experience with other problems; no attempts were made to optimize these parameters. Figure 4 shows the pressure distribution for an intermediate airfoil that represents the best airfoil obtained after 10 generations using the DE optimization method. The algorithm is able to approach the target distribution within roughly 500 function evaluations (note that the actual number of function evaluations is much less, because many of the initial airfoil geometries that were infeasible were not evaluated by the CFD solver. After 10 generations, the number of design variables is increased to 13 and the population evolved further. As is typical with other genetic algorithms, once in the vicinity of the optimal solution, convergence of the DE algorithm slows down considerably. The final design shown in Fig. 5 was obtained after another 10 generations.

In order to reduce the computational cost of the CFD function evaluations the population after the first 10 generations was used to train a neural network. Figure 6 shows the band around the target pressure distribution in which all the data used to train the neural network lie. Note, however, that the network was trained directly on the sum-square-error and not on the individual pressure data. The data band in Fig. 6 gives an idea of the local nature of the neural network response surface in the vicinity of the optimal solution. The trained neural network was used in lieu of the CFD solver to perform the function evaluation with negligible computational cost. The final design obtained was close to that shown in Fig. 5. Finally, Fig. 7 shows the optimal airfoil geometry obtained by the DE algorithm.

Attempts to quantify the computational costs and obtain more detailed evaluation of the various parameters are currently underway.

4 Summary

An aerodynamic design optimization procedure that is based on a evolutionary algorithm known as Differential Evolution is described. The method is combined with a Navier-Stokes solver that evaluates the various intermediate designs and provides inputs to the optimization procedure. Results are presented for the inverse design of a turbine airfoil from a modern jet engine. The capability of the method to search large design spaces and obtain the optimal airfoils with a reasonable number of CFD function evaluations in an automatic fashion is demonstrated. The airfoil geometry parametrization and constraint handling procedure helps limit the number of CFD function evaluations required by disallowing airfoil geometries that are infeasible from a physical or aerodynamic standpoint. Substantial reductions in the overall computing time requirements are achieved by the hybrid DE-NN algorithm that uses a neural network trained on the CFD data to evaluate the objective function in the latter stages of the design evolution.

References

1. S. Y.Lee, K. Y. Kim: 'Design Optimization of Axial Flow Compressor Blades with a Three-Dimensional Navier-Stokes Solver'. *Proceedings of the ASME Turbo Expo 2000, Munich, Germany, May 8–11, 2000*
2. J. M. Janus, J. C. Newman: 'Aerodynamic and Thermal Design Optimization for Turbine Airfoils'. *AIAA Paper No. 2000-0840, Jan. 2001*
3. J. M. Sanz: 'Development of a Neural Network Design System for Advanced Turbo-Engines'. *4th U.S. National Congress on Computational Mechanics, San Francisco, CA, Aug. 1997*
4. M. M. Rai: 'A Rapid Aerodynamic Design Procedure Based on Artificial Neural Networks'. *AIAA Paper No. 2001-0315, Jan. 2001*
5. S. Pierret, R. A. V. Braembussche: Journal of Turbomachinery, **121**, 326 (1999)
6. M. M. Rai, N. K. Madavan: AIAA Journal, **38**, 173 (2000)
7. N. Papila, W. Shyy, L. Griffin, D. J. Dorney: 'Shape optimization of supersonic turbines using response surface and neural network methods'. *AIAA Paper No. 2001-1065, Jan. 2001*
8. S. Obayashi, S. Takanashi: AIAA Journal, **34**, 881 (1996)
9. B. H. Dennis, I. N. Egorov, Z. X. Han, G. S. Dulikravich, C. Poloni: 'Multi-Objective Optimization of Turbomachinery Cascades for Minimum Loss, Maximum Loading, and Maximum Gap-to-Chord Ratio'. *AIAA Paper No. 2000-4876, Sept. 2000*

10. D. Quagliarel a, A. D. Cioppa: 'Genetic Algorithms Applied to the Aerodynamic Design of Transonic Airfoils'. *AIAA Paper 94-1896-CP, 1994*
11. M. Uelschen, M. Lawerenz: 'Design of Axial Compressor Airfoils with Artificial Neural Netwo ks and Genetic Algorithms'. *AIAA Paper No. 2000-2546, June 2000*
12. C. Poloni, A. Giurgevich, L. Onesti, V. Pediroda: Comp. Meth. Appl. Mech. Engr., **186**, 403 (200)
13. R. Storn, K. Price: Dr. Dobb's Journal, **22**, 18 (1997)
14. K. Nho, R. K. Agarwal: 'Fuzzy Logic Model-Based Predictive Control of Aircraft Dynamics Usi ig ANFIS'. *AIAA Paper 2001-0316, Jan. 2001*
15. T. Rogalsky, R. W. Derksen, S. Kocabiyik: Can. Aero. Space Inst. Jour., **46**, 183 (2000)
16. T. Rogalsky, R. W. Derksen: 'Hybridization of Differential Evolution for Aerodynamic Desi gn'. *Proceedings 8th Ann. Conf. Comp. Fluid Dyn. Soc. Can., pp. 729–736, June 11–13, 2000*
17. K. V. Price: 'Differential Evolution: A Fast and Simple Numerical Optimizer'. In: *Biennial Conference of the North American Fuzzy Information Processing Society, (NAFIPS), June 1996*, ed. by M. Smith, M. Lee, J. Keller, J. Yen (IEEE Press, New York 1996) pp. 524–527
18. J. Lampinen: A Bibliography of Differential Evolution Algorithm. Technical Report. Lappeenranta University of Technology, Department of Information Technology, Laborato y of Information Processing, Lappeenranta, Finland (2001)
19. T. Back, U. I ammel, H.-P. Schwefel: IEEE Transactions on Evolutionary Computation, **1**, 3 (1997)
20. M. A. Shokr llahi, R. Storn: 'Design of Efficient Erasure Codes with Differential Evolution'. In *Proceedings of ISIT 2000, International Symposium on Information Theory, Sorr nto, Italy, June 25-30, 2000*

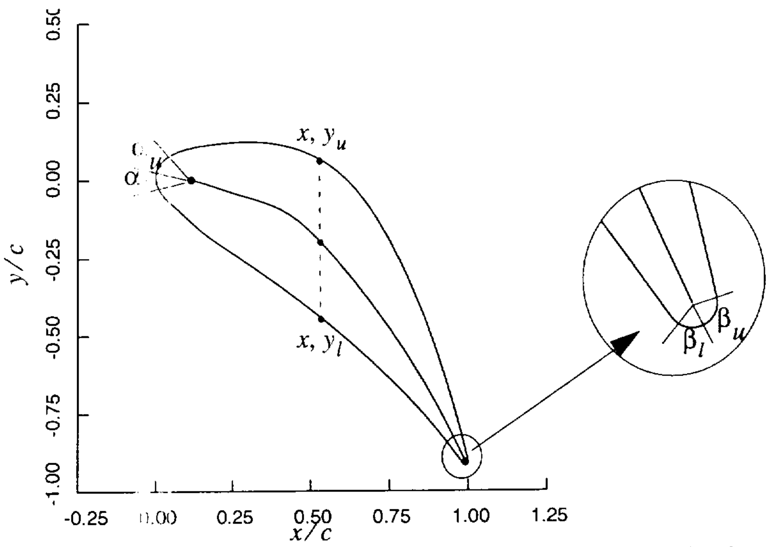

Figure 1. Schematic of a generic airfoil showing location of control points on the airfoil surface and the defining angles used in the parameterization of the airfoil geometry.

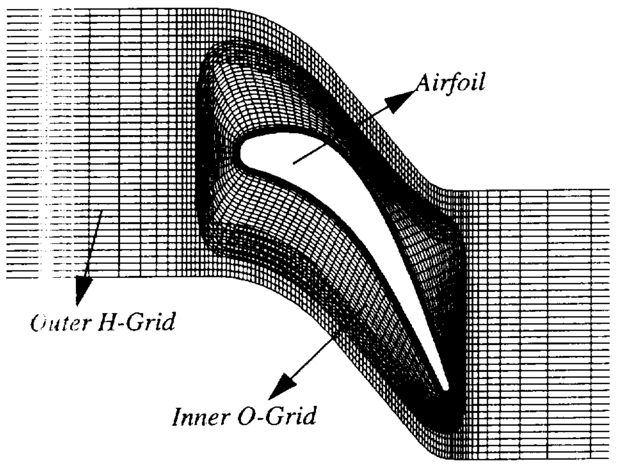

Figure 2. Representative turbine airfoil geometry and computational grid used in the CFD simulations.

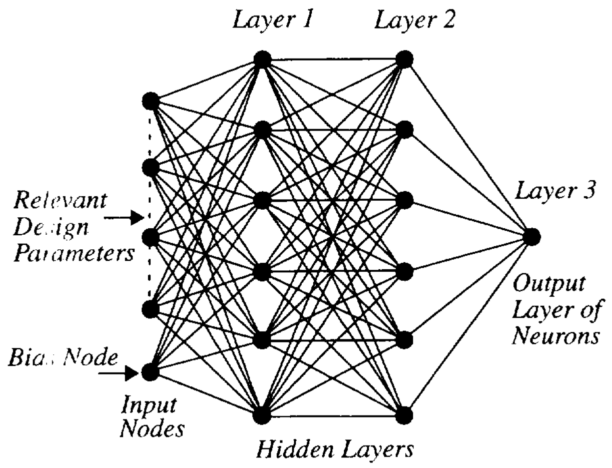

Figure 3. Schematic of the three-layer feed-forward neural network used in this study.

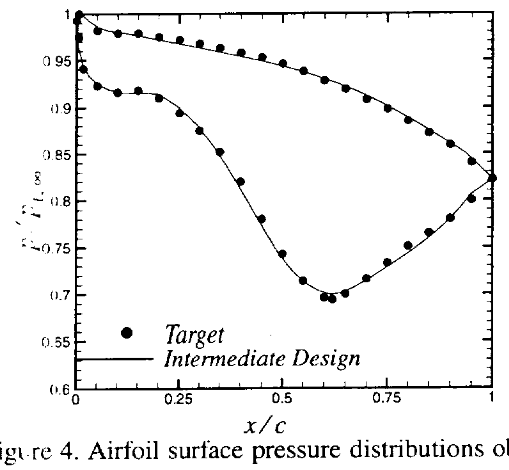

Figure 4. Airfoil surface pressure distributions obtained from CFD simulations for the intermediate design (using 6 design variables).

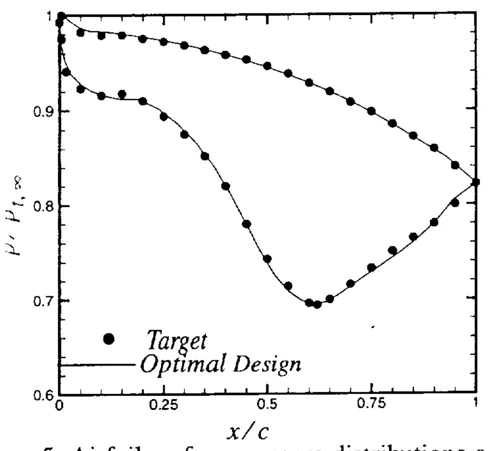

Figure 5. Airfoil surface pressure distributions obtained from CFD simulations for the final optimal design using 13 design variables.

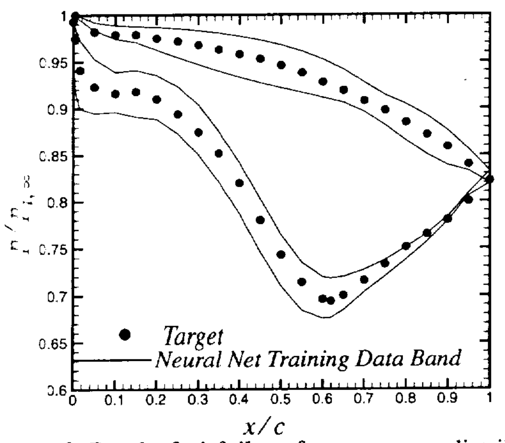

Figure 6. Band of airfoil surface pressure distributions obtained from CFD simulations used in training the neural network for the hybrid approach.

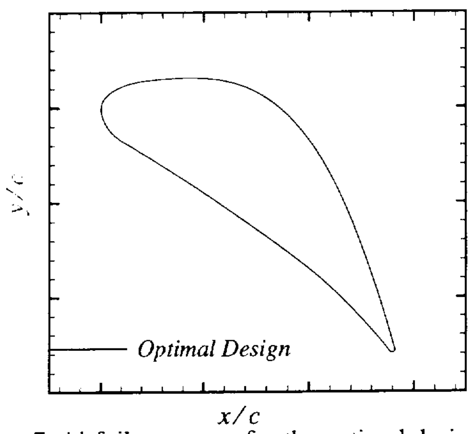

Figure 7. Airfoil geometry for the optimal design.