Contemporary Engineering Sciences, Vol. 15, 2022, no. 1, 13 – 39
HIKARI Ltd, www.m-hikari.com
https://doi.org/10.12988/ces.2022.91908

Self-learning, Adaptive Software: An Example from Aerospace Engineering

Mohamed Jrad

Kevin T. Crofton Department of Aerospace and Ocean Engineering
Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

Currently at M4 Engineering, Inc., Long Beach, CA, USA

Rakesh K. Kapania

Kevin T. Crofton Department of Aerospace and Ocean Engineering
Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

Joseph A. Schetz

Kevin T. Crofton Department of Aerospace and Ocean Engineering
Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

Layne T. Watson

Department of Computer Science, Department of Mathematics, and Kevin T. Crofton Department of Aerospace & Ocean Engineering
Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

Abstract

A novel application of machine learning concepts towards making software learn from its own experience in solving similar problems is proposed. The objective is to continuously improve the computational performance as the software solves more problems, avoid the cost of repeated runs, and make software able to take effective

decisions for rapidly solving complex nonlinear problems (e.g., which numerical method and initial guess to employ for solving a set of nonlinear equations). The illustrative problem consists of finding oblique shock solutions of supersonic flow over a wedge, which involves four, coupled, nonlinear equations with six variables. The key idea of the proposed self-learning concept consists of constructing local surrogate functions that help determine a good approximation of the expected solution as a starting initial guess. Three different approaches (Delaunay triangulation, cubic interpolation, and locally trained neural network) have been implemented and compared. Moreover, the software is given the ability to implement and compare different nonlinear solver techniques across the parameter space. A demonstration of the implementation of the new high-dimensional Delaunay triangulation in a machine learning context is performed. Moreover, the concept of locally trained neural network with moving weights is presented as a way to allow continuous growth of the constructed database used in the training. The self-learning approach is shown to be very efficient, especially since it reduces the burden of finding appropriate initial guesses. Although this approach is applied for solving nonlinear equations, as a proof of concept, the concept can be extended to other engineering applications.

Keywords: machine learning, artificial neural networks, Delaunay triangulation, oblique shock, normal shock, MACH wave

Nomenclature

k: Ratio of specific heats
M: Mach number
p: Pressure
ρ: Mass density
σ: Angle of shock to incoming flow in degrees
δ: Turning angle across shock in degrees
LB_i: Lower bound of the output variables
UB_i: Upper bound of the output variables
r: Residue
f: Flag
n_f: Number of function evaluations
n_j: Number of Jacobian matrix calls
n_{it}: Number of iterations
Δt: CPU time
t_s: Analysis start time

I. Introduction

The design of modern, advanced aerospace systems with significant interactions amongst various disciplines, requires that high-fidelity analyses be used as early as possible in their design optimization. This is because a substantial fraction (roughly 70%) of the cost of developing a new system is committed in the early stages. If not done right, substantial cost overruns could occur as many design changes are needed during the testing stage. While performing the multi-disciplinary design optimization of an efficient supersonic vehicle, completely different configurations could be obtained using low- and high-fidelity models. Despite this need of employing high-fidelity models, very few entities (government, industry, and academia) are able to do so due to the extremely high computational cost of running such models at the preliminary, let alone at the conceptual, design stage.

A number of researchers have employed surrogate models, namely response surface approximations, Kriging, and artificial neural networks (deep learning) that are generated by using high-fidelity models [1, 2, 3, 4]. Development of these surrogate models is not without substantial computational cost as a large number of computational runs are required to generate the data needed for their development (``learning'') [5]. Caixeta and Marques [4] conducted multidisciplinary optimization of aircraft wing structures using genetic algorithms. They have constructed metamodels using artificial neural networks for the critical flutter speed prediction and employed Latin hypercube sampling. A review of metamodeling techniques used in engineering design optimization has been conducted by Wang and Shan [6]. As new data (output of a response quantity for a given set of input data) becomes available, these surrogate models require retraining or updating, further increasing the cost of their use for multidisciplinary design optimization. There is thus a need for studying alternative computational paradigms that can use high-fidelity models at the early design stage avoiding redesigns and resulting cost overruns. The general tendency of the available commercial software is to consider each simulation independently. Previously run simulation history is generally neither saved nor used in future analyses. We propose, in this work, a new methodology for solving problems by making the software learn from its own experience. The final objective is to improve the computational performance as the software solves more problems, avoid the cost of repeated runs, and give the software the ability to make effective decisions (e.g., select a better initial guess, select appropriate algorithm or tuning parameters, etc.) towards improving the process for finding the solution for complex nonlinear problems.

Among several existing definitions for self-adaptive software, one is provided in a DARPA Broad Agency Announcement (BAA) [7]: “Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.” Scientists and engineers have made significant efforts to

design and develop self-adaptive systems. A review of self-adaptive software presented by Salehie and Tahvildari [8] shows how an open-loop system can be converted to a closed-loop system using feedback, and they proposed a taxonomy of adaptation.

The problem addressed in the present work consists of solving the nonlinear system of equations that governs oblique shocks formed in a supersonic flow over a wedge [9]. This problem is selected due to its moderate level of complexity. The number of obtained solutions (strong shock, weak shock) depends on the problem parameters and is unknown beforehand. The proposed algorithm makes the software learn from similar problems studied earlier and use the lessons learned from those problems for solving a completely new problem. One key idea of the proposed self-learning concept consists of constructing local surrogate functions from solutions of earlier problems that determine a good approximation of the expected new solution as an initial guess. Such an approach is shown here to be very efficient, especially since it reduces the difficulty of finding appropriate initial guesses when solving a nonlinear problem numerically. To efficiently construct the local surrogate functions, three different approaches (Delaunay triangulation [10], cubic interpolation, and locally trained neural network [11] with moving weights) have been implemented and compared. An implementation of the new high-dimensional Delaunay triangulation, developed by Chang et al. [10], in a machine learning application is demonstrated in this work. Moreover, unlike the traditional use of the neural network, the concept of locally trained neural network with moving weights is presented herein as a way to allow continuous growth of the constructed database used in the training. Second, a few numerical techniques are also compared in each region of the parameter space to collect information that serves subsequently in determining which numerical solver to consider for a new problem. For this purpose, the nonlinear solver *nleqslv* [12] has been employed to compare several numerical techniques and line searches. It is shown that the choice of the *nleqslv* numerical technique and line search depends on the problem and how far the initial guess is from the solution.

II. Example Case of Oblique Shocks in Supersonic Flow

Consider a steady, two-dimensional (plane) supersonic flow passing over a turn or wedge with a certain angle δ with the flow direction as shown in Figure 1 [9]. An oblique shock is then observed with an angle σ. An oblique shock is a special form of pressure discontinuity within the fluid, which is inclined to the direction of the oncoming flow. This problem was selected here to apply and test the proposed algorithm because of several factors. First, the complexity of the resulting system of equations makes it difficult to solve four, highly nonlinear equations with four unknowns numerically in an efficient way. The user will have to keep making initial guesses in a 4-D space. Furthermore, depending on other problem inputs described below, a different number of solutions could be obtained (either two solutions: a weak

shock and a strong shock, one unique solution, or no solutions at all). Hence, it becomes tougher, more tedious, and more time consuming, since one is required to find two initial guesses: one for the weak shock, and the other for the strong shock. Since this number of solutions is not known *a priori*, it is a challenge to know when to stop searching for a solution if a user has tried several initial guesses, but the algorithm did not converge. The question that arises naturally is: Can one make use of the solutions of previously solved similar problems so as to determine: (a) how many solutions there might be for a problem at hand, (b) what are the most appropriate initial guesses that are close to these expected solutions to make the numerical solution converge faster, and (c) what numerical equation solver approach is most appropriate.

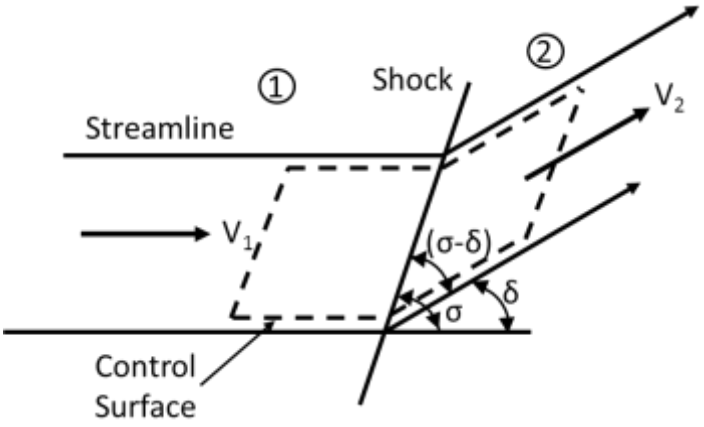


Figure 1. Oblique Shock

The solutions for the particular case of $\delta = 0°$ are two simple, limiting cases of this problem, where there is no change observed in the flow direction. The first solution, $\sigma = 90°$, is called a normal shock, whereas the second solution, $\sigma = \sin^{-1}\frac{1}{M_1}$, a Mach wave. The Mach wave corresponds to the weak shock with the least possible shock angle σ.

Using conservation of mass, momentum, and energy, and assuming the fluid to be a perfect gas, the six oblique shock parameters of a supersonic flow (Mach numbers M_1 and M_2, density ratio $\frac{\rho_2}{\rho_1}$, pressure ratio $\frac{p_2}{p_1}$, shock angle σ, flow turning angle δ) are governed by the nonlinear system of four equations

$$f_1(X) = \frac{p_2}{p_1} - \frac{\frac{k+1}{k-1}\frac{\rho_2}{\rho_1} - 1}{\frac{k+1}{k-1} - \frac{\rho_2}{\rho_1}} = 0, \tag{1}$$

$$f_2(X) = \frac{\rho_2}{\rho_1} - \frac{tan\,\sigma}{tan(\sigma-\delta)} = 0, \tag{2}$$

$$f_3(X) = \frac{p_2}{p_1} - \left(1 + kM_1{}^2(sin^2\sigma)(1 - \frac{\rho_1}{\rho_2})\right) = 0, \tag{3}$$

$$f_4(X) = \frac{p_1}{p_2} - 1 - kM_2{}^2\left(1 - \frac{\rho_2}{\rho_1}\right) sin^2(\sigma - \delta) = 0, \tag{4}$$

where the subscript "1" denotes "before the shock", and "2" "after the shock". The parameter vector for each problem is $X = \left(M_1, \delta, M_2, \frac{\rho_2}{\rho_1}, \frac{p_2}{p_1}, \sigma\right)$ and this is solved for by picking two parameters (in this work, we choose M_1 and δ) and solving for the remaining four parameters using the four nonlinear equations presented above. The following physical inequalities govern our problem parameters:

$$0 \leq M_2 \leq M_1, \tag{5}$$

$$1 \leq \frac{\rho_2}{\rho_1}, \tag{6}$$

$$1 \leq \frac{p_2}{p_1}, \tag{7}$$

$$0 \leq \sigma \leq 90°, \tag{8}$$

$$0 \leq \delta \leq 90°, \tag{9}$$

In what follows, the problem is solved numerically to have a minimal residual $r = \max(|f_i(X)|) \approx 0$.

III. Numerical Equation Solver Methods

The numerical equation solver package considered in this work is *nleqslv* [12], a FORTRAN 77 library that solves a system of nonlinear equations using quasi-Newton and Newton type methods with a choice of global strategies such as line search and trust region. There are options for using a numerical or user-supplied Jacobian matrix, for specifying a banded numerical Jacobian matrix and for allowing a singular or ill-conditioned Jacobian matrix. Broyden's method starts with a computed Jacobian matrix

of the function and then updates this Jacobian matrix after each successful iteration using the so-called Broyden update, and has super-linear convergence. When *nleqslv* determines that it cannot continue with the current Broyden matrix, it computes a new Jacobian matrix. Newton's method calculates a Jacobian matrix of the function at each iteration, and has quadratic convergence. The following options are available within the package:

1. Cubic line search.
2. Quadratic line search.
3. Geometric line search.
4. Double dogleg: A trust region method using the double dogleg method as described in Dennis and Schnabel [13].
5. Powell dogleg: A trust region method using the Powell dogleg method as developed by Powell [14], [15].
6. Hookstep: A trust region method described by Dennis and Schnabel [13].

Hence, there are a total of 12 methods that can be employed as shown in

Table **1**,

Table 1. Equation Solver Methods in *nleqslv* [12]

Approach	Method	Approach	Method
1	Newton with cubic line search	7	Broyden with cubic line search
2	Newton with quadratic line search	8	Broyden with quadratic line search
3	Newton with geometric line search	9	Broyden with geometric line search
4	Newton with double dogleg	10	Broyden with double dogleg
5	Newton with Powell dogleg	11	Broyden with Powell dogleg
6	Newton with hookstep	12	Broyden with hookstep

As described in the *nleqslv* package manual, "*Which global strategy to use in a particular situation is a matter of trial and error. When one of the trust region methods fails, one of the line search strategies should be tried. Sometimes a trust region will work and sometimes a line search method; neither has a clear advantage but in many cases the double dogleg method works quite well*". Therefore, a self-learning algorithm was developed here such that the software initially compares several methods in different regions of the parameter space when its database is fresh and then becomes able to find the best method to use for each considered problem. Since some of the built-in approaches are similar, only Approaches 1, 7, and 11 were selected and compared in this work. The numerical Jacobian matrix option was selected.

IV. Normal Shock and Mach Wave Cases

The two sample limiting cases of oblique shocks are normal shocks and Mach waves, corresponding to the strong shock and weak shock solutions of $\delta = 0°$, respectively. Considering as fixed parameters $P = \left(\widehat{M}_1, \hat{\delta}\right) = \left(\widehat{M}_1, 0\right)$, the solution for the Mach wave is, $X_0 = \left((M_2)_0, \left(\frac{\rho_2}{\rho_1}\right)_0, \left(\frac{p_2}{p_1}\right)_0, \sigma_0\right) = \left(\widehat{M}_1, 1, 1, \sin^{-1}\frac{1}{\widehat{M}_1}\right)$. The normal shock solution, however, requires the nonlinear system of Equations (1)--(4) to be modified and then solved numerically. The first step consists of replacing $\sigma = 90°$ and $\delta = 0°$ in Eqs. (3) and (4). Note that Eq. (2) can be dropped, since the number of unknowns is reduced to three. Using the new parameter vector $X' = \left(M_1, M_2, \frac{\rho_2}{\rho_1}, \frac{p_2}{p_1}\right)$, the new obtained equations are

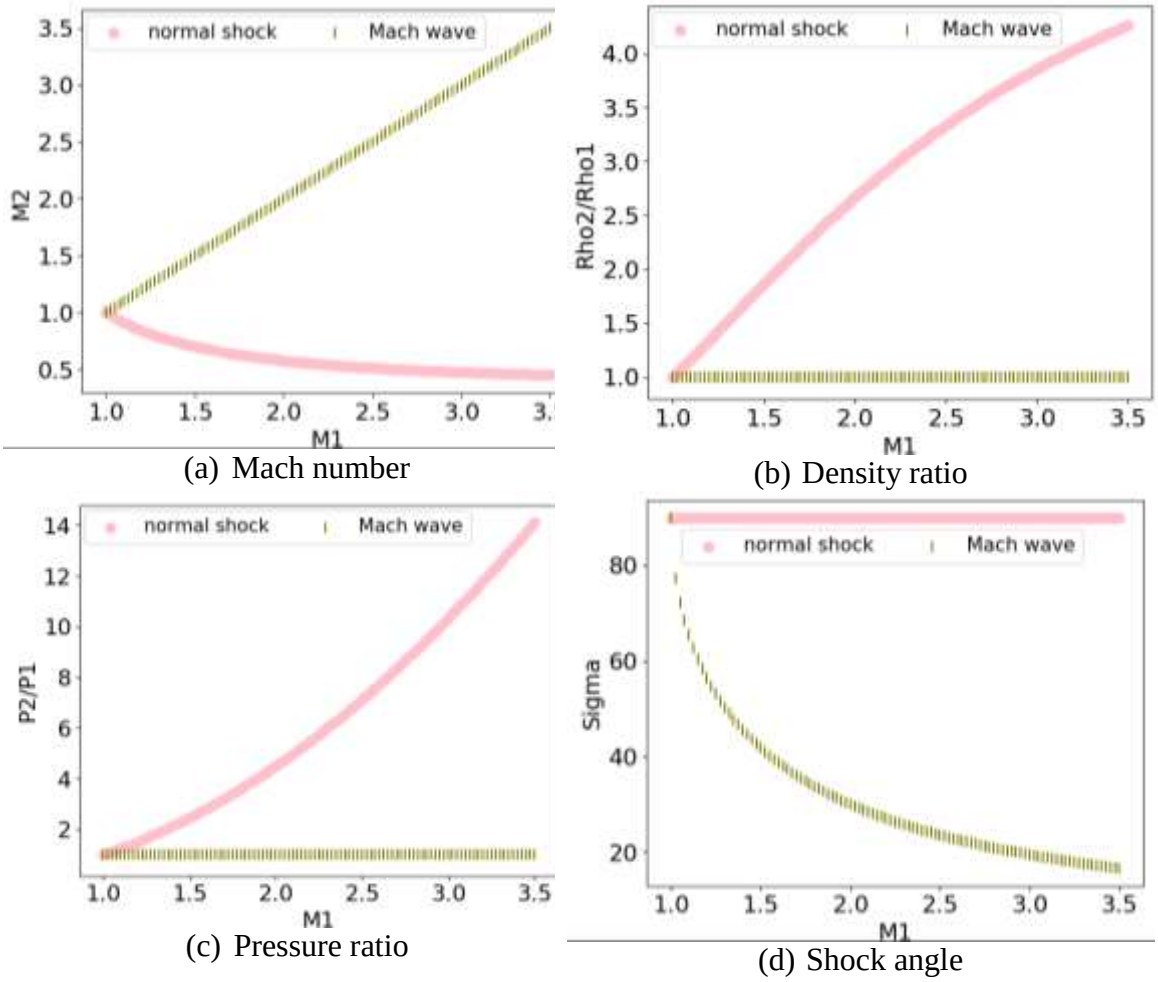
$$\hat{f}_1(X') = \frac{p_2}{p_1} - \frac{\frac{k+1}{k-1}\frac{\rho_2}{\rho_1} - 1}{\frac{k+1}{k-1} - \frac{\rho_2}{\rho_1}} = 0, \tag{10}$$

$$\hat{f}_3(X') = \frac{p_2}{p_1} - \left(1 + k{M_1}^2\left(1 - \frac{\rho_1}{\rho_2}\right)\right) = 0, \tag{11}$$

$$\hat{f}_4(X') = \frac{p_1}{p_2} - 1 - k{M_2}^2\left(1 - \frac{\rho_2}{\rho_1}\right) = 0, \tag{12}$$

Note that Eqs. (11) and (12) are coupled while, Eq. (10) is independent. Therefore, to compute the normal shock solution for fixed $P = \left(\widehat{M}_1, \hat{\delta}\right) = \left(\widehat{M}_1, 0\right)$, first compute $(M_2)^*$ and $\left(\frac{\rho_2}{\rho_1}\right)^*$ with Eqs. (11) and (12) and then obtain $\left(\frac{p_2}{p_1}\right)^*$ from Eq. (10). The quasi-Newton with Powell dogleg method is employed for this purpose. However, one first needs appropriate initial guesses $(M_2)_0$ and $\left(\frac{\rho_2}{\rho_1}\right)_0$, a challenging task that requires several attempts. To that end, the segment $\Gamma_{M_1} = [1, 3.5]$ is first discretized using a sequence $(M_1)_{0 \le i \le n} = 1 + \frac{2.5 \times i}{n}$, with $n \ge 2$ being a pre-selected positive integer. The normal shock is then solved for the right end-point $\left(\widehat{M}_1, \hat{\delta}\right) = \left((M_1)_n, 0\right)$, and a backward sweep is finally used to solve for each new input $(M_1)_i$ using the solution of $(M_1)_{i+1}$as an initial guess until reaching the left end-point $(M_1)_0$.

Figure 2 presents the obtained Mach wave and normal shock solutions, showing the limiting cases for the Mach number M_2, density ratio $\frac{\rho_2}{\rho_1}$, pressure ratio $\frac{p_2}{p_1}$, and the shock angle σ. It can be seen that the upper limit of M_2 and σ (normal shock) are $M_2 = M_1$ and $\sigma = 90°$, respectively, while the lower limit of $\frac{\rho_2}{\rho_1}$ and $\frac{p_2}{p_1}$ (Mach wave) are $\frac{\rho_2}{\rho_1} = 1$ and $\frac{p_2}{p_1} = 1$, respectively. In what follows, this obtained data will be used and interpolated to compute the Mach wave and normal shock for any problem, if needed.



(a) Mach number (b) Density ratio

(c) Pressure ratio (d) Shock angle

Figure 2. Mach wave and normal shock solutions

V.Continuation Method

Before developing the self-learning algorithm, it is important to develop a method that is able to solve the nonlinear system of equations (1)--(4) for any problem, which can be used as a reference method. While the choice of the reference method is not crucial and is problem dependent, the continuation method is employed here. Our objective is to solve the problem for any $\hat{P} = \left(\widehat{M_1}, \hat{\delta}\right)$ using only the Mach wave and normal shock solutions to approximate the required initial guesses and obtain both the weak and strong shock solutions. We start with the weak shock, then obtain results for the strong shock. If during the search for the weak shock solution, we first find the strong shock, we save that result to avoid duplicating the effort.

The proposed algorithm is as follows:

1. Find the Mach wave solution X_{01} and the normal shock solution X_{02} for fixed $M_1 = \widehat{M_1}$ and $\hat{\delta} = 0$ from the data computed in section IV. These will be the two extreme points that make the search line for the initial guesses. For the weak shock, start with initial guess $\widehat{X_0} := X_{01}$, and for the strong shock, start with initial guess $\widehat{X_0} := X_{02}$.
   $X_{01} = \left((M_2)_1,\ \left(\frac{\rho_2}{\rho_1}\right)_1,\ \left(\frac{p_2}{p_1}\right)_1,\ (\sigma)_1\right)$ and $X_{02} = \left((M_2)_2,\ \left(\frac{\rho_2}{\rho_1}\right)_2,\ \left(\frac{p_2}{p_1}\right)_2,\ (\sigma)_2\right)$.
2. Solve the system for X_0 using one of the three selected *nleqslv* methods.
3. If the system solution is obtained, stop. Otherwise,
   Define $d = \begin{cases} 1,\ for\ weak\ shock, \\ -1,\ for\ strong\ shock. \end{cases}$
   $\widehat{X_0} := \widehat{X_0} + d * \frac{X_{02} - X_{01}}{10}$

Go to Step 2.

The efficiency of this method and the number of attempted initial guesses depend on how close the problem at hand is to the projection of the Mach wave and normal shock solutions, as presented in
Figure **3**. The oblique shock solutions are then computed for several points covering the space $1 \leq M_1 \leq 3.5$ and $0 \leq \delta \leq 90°$ and presented in
Figure **4** with different colors to visually link the different regions of the space for the four parameters $\left(M_2, \frac{\rho_2}{\rho_1}, \frac{p_2}{p_1}, \sigma\right)$. Note from
Figure **4** that the pressure ratio (b), density ratio (c), and shock angle (d) are larger for the strong shock than for the weak shock while the Mach number (a) is larger for the weak shock than for the strong shock. A value of $M_2 = 0.93$ is used to approximate the line that separates weak shocks from strong shocks [9]. Therefore, comparing $(M_2)^*$ to 0.93 can be used as a way to decide whether an obtained shock solution is weak or strong. Note also that the weak and strong shocks coincide with the Mach wave and normal shock, respectively, for $\delta = 0$ and keep approaching each other with increasing δ until merging at a certain value $\delta = \delta_{max}$, beyond which there is no solution.

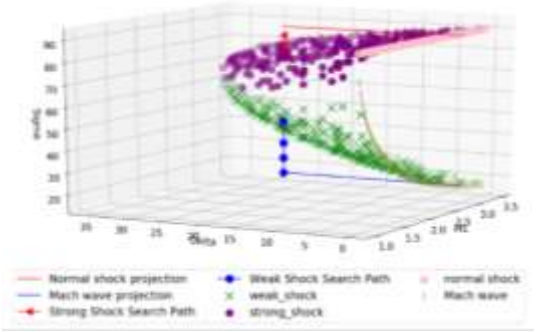
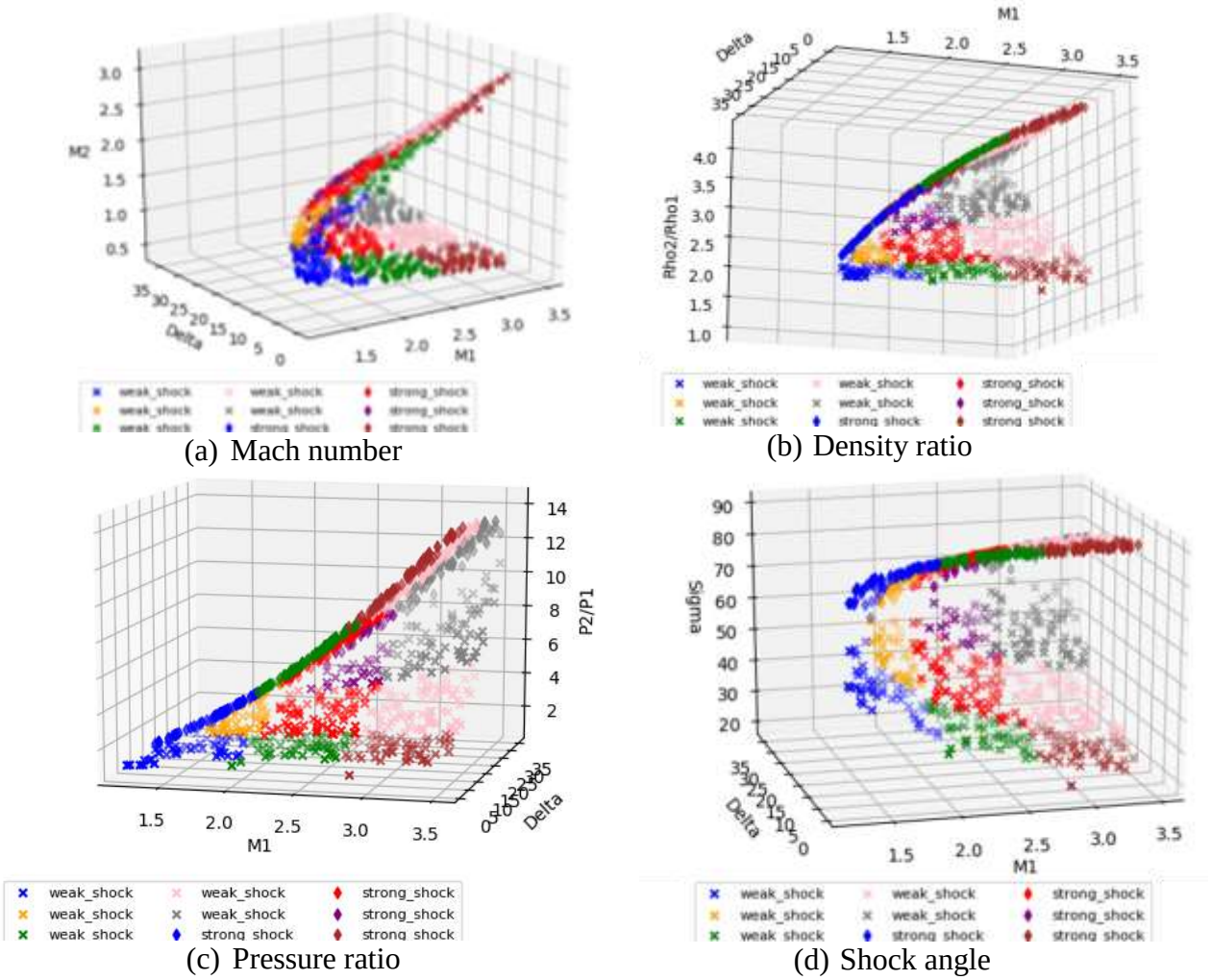


Figure 3. Continuation method.



(a) Mach number (b) Density ratio

(c) Pressure ratio (d) Shock angle

Figure 4. Oblique shock solutions obtained with the continuation method.

The objective of the present work is to develop a self-learning approach that solves the problem in a more optimized way and only calls the continuation method when required.

VI. The Learning Approach

The learning approach is based on the concept of "recycling" available information from solutions to previous cases and employing it to:

- predict the existence or lack of a solution,
- decide which solver is more efficient,
- compute an initial guess as close to the solution as possible.

The created database has the following record (row) format,

$\widehat{M_1}$	$(M_2)^*$	$\left(\frac{\rho_2}{\rho_1}\right)^*$	$\left(\frac{p_2}{p_1}\right)^*$	σ^*	$\hat{\delta}$	r	n_f	n_j	n_{it}	f	a	t_s	Δt	$\lvert X^* - X_0 \rvert$	A

where $\widehat{M_1}$ and $\hat{\delta}$ are the problem inputs, while $X^* = \left((M_2)^*, \left(\frac{\rho_2}{\rho_1}\right)^*, \left(\frac{p_2}{p_1}\right)^*, \sigma^*\right)$ constitutes the obtained oblique shock solution, and X_0 denote the selected initial guess. The normalized distance between the selected initial guess X_0 and the obtained solution X^* is $|X^* - X_0| = \max_i\left(\left|\frac{(X^*-X_0)_i}{UB_i-LB_i}\right|\right)$, where LB_i and UB_i are the lower bound and upper bound of the output variable $\left(M_2, \frac{\rho_2}{\rho_1}, \frac{p_2}{p_1}, \sigma\right)$, respectively ($LB_i = (0.25,1,1,0)$ and $UB_i = (3.5,12,12,90°)$). The residue $r = \max_i(|f_i(X^*)|)$ is computed using Eqs. (1)--(4) and the numerical solver output flag is f. The selected *nleqslv* approach for the considered point is a while the number of function evaluations, Jacobian matrix calls, and iterations are stored in n_f, n_j, and n_{it}, respectively. The CPU time and the analysis start time are stored in Δt and t_s, respectively. Information about the different attempted *nleqslv* approaches and their results is stored in the array A with the following format,

a	n_f	n_j	n_{it}	f	$\widehat{M_1}$	$(M_2)^*$	$\left(\frac{\rho_2}{\rho_1}\right)^*$	$\left(\frac{p_2}{p_1}\right)^*$	σ^*	$\hat{\delta}$	t_s

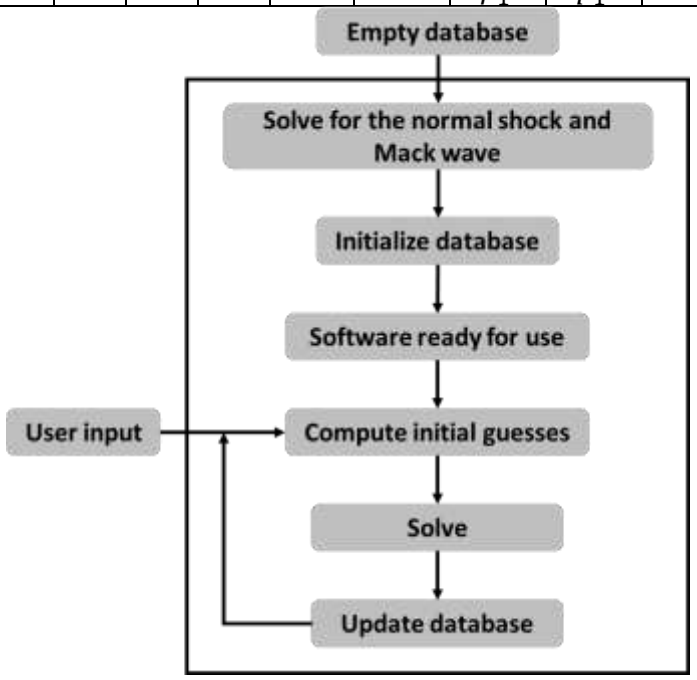


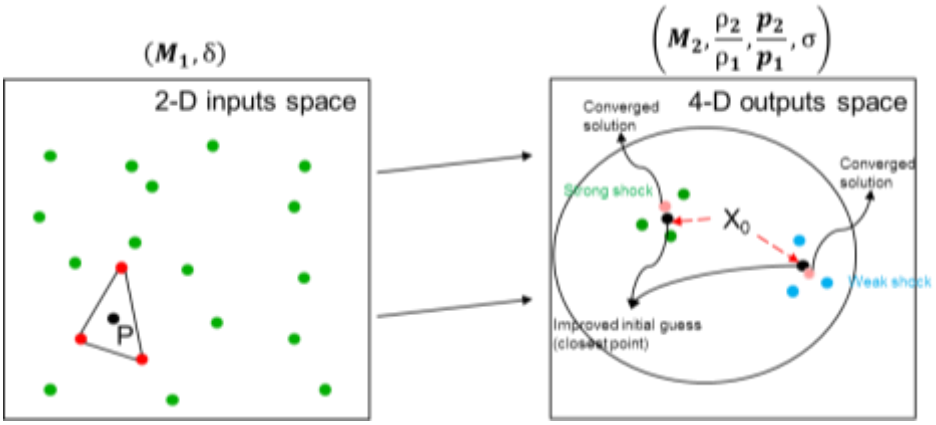
Figure 5. Proposed self-learning concept

Three methods are employed for collecting and processing information from the database: The Delaunay triangulation, cubic interpolation, and artificial neural network, and their performance is compared. The proposed self-learning concept is shown in Figure 5. It should be noted that the database initialization step is mainly used to form an initial comparison dataset for the different *nleqslv* approaches that will be employed. In the case where no comparisons are required (e.g., only one *nleqslv* approach is used), the initialization database can be, then, limited to a very low number of points, depending on the employed machine learning technique. For example, with the Delaunay triangulation, only three points are needed, whereas 10 points are needed for the cubic interpolations and the artificial neural network methods.

1. Delaunay Triangulation

The Fortran 2003 VT Delaunay triangulation code by Chang et al. [10] has been wrapped in Python and employed in this work. Unlike the traditional Delaunay triangulation, this new Delaunay triangulation interpolation can be applied to high dimensional problems. For this method, the starting database is required to have a minimum of three points. The proposed algorithm, described in Figure 6, is as follows.

1. Read input $P = \left(\widehat{M}_1, \hat{\delta}\right)$.
2. Search for Delaunay triangle in the database input space containing P by means of the Delaunay triangulation code [10]. The obtained triangle vertices have weight values w_i (generated also by the Delaunay triangulation code [10]) in addition to the analysis history of these points from the database.
3. Use a convex combination to compute the initial guess based on the results of the vertices of the selected triangle. In other words, our initial guess $\mathrm{X}_0 = \left((M_2)_0, \left(\frac{\rho_2}{\rho_1}\right)_0, \left(\frac{p_2}{p_1}\right)_0, \sigma_0\right)$ is computed through the equation $(X_0)_i = \sum_{j=1}^{j=n} w_j \left(\mathrm{X}^{*(j)}\right)_i$, where $\left(\mathrm{X}^{*(j)}\right)_i$ is the solution component i ($1 \leq i \leq 4$) of the triangle's vertex $\mathrm{X}^{*(j)}$, and n is the number of vertices that have a solution ($1 \leq n \leq 3$). For example, in Figure 6, the three neighboring points that form the obtained Delaunay triangle of the point $\left(\widehat{M}_1, \hat{\delta}\right)$ are shown in red in Figure 6-a while their weak shock solution $(X^*)^{(w)}$ and strong shock solution $(X^*)^{(s)}$ are shown in Figure 6-b in blue and green, respectively.
4. Figure 6-b also shows how the obtained initial guess (colored black), in both oblique shock cases, is close to the converged solution (shown in pink).
5. Select also the best *nleqslv* approach to use based on the stored information of the same vertices.



(a) Determining the Delaunay triangle (b) Determining the weak and strong solutions

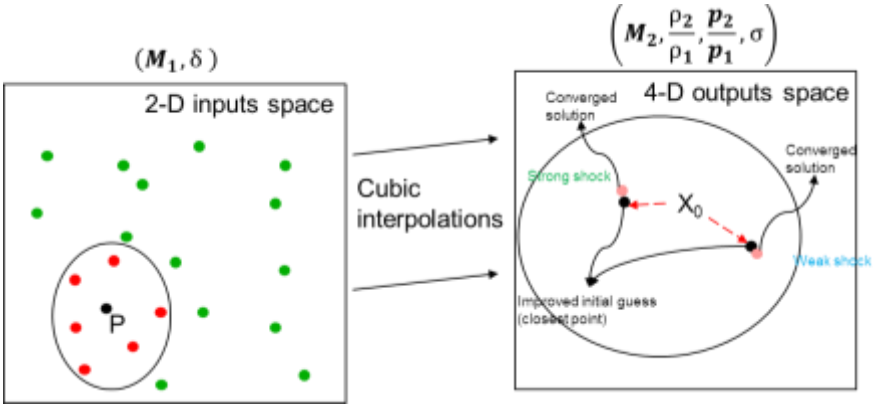
Figure 6. Learning approach based on Delaunay triangulation

2. Cubic Interpolation

As its name states, this method creates a surrogate function using cubic interpolation[1] on a subset of points surrounding the point of interest. The proposed algorithm, described in Figure 7, is as follows:

1. Read input $P = (\hat{M}_1, \hat{\delta})$.
2. Search for the nearest surrounding points in the database input space, using the Euclidian distance, as shown in Figure 7-a. The minimum number of points is defined by the number of unknowns required to construct the cubic interpolation. In our case, we select ten points. Note that polynomial interpolation in dimension greater than one is ill-posed, meaning that in the general case, ten distinct points may not determine the ten cubic coefficients.
3. Apply piecewise cubic interpolation to approximate the four objective functions. Compute the initial guesses for both shocks. Figure 7-b shows how the computed initial guess (black dot) is very close to the solution (pink dot) for both shock cases.
4. Select also the best *nleqslv* approach to use based on the information stored for the same vertices.

[1] Note that the interpolation in dimensions greater than 1 is ill-posed, meaning that 10 distinct points need not determine the 10 coefficients of a cubic polynomial in two variables.



(a) Determining the set of nearest points (b) Determining the weak and strong solutions

Figure 7. Learning approach based on cubic interpolations

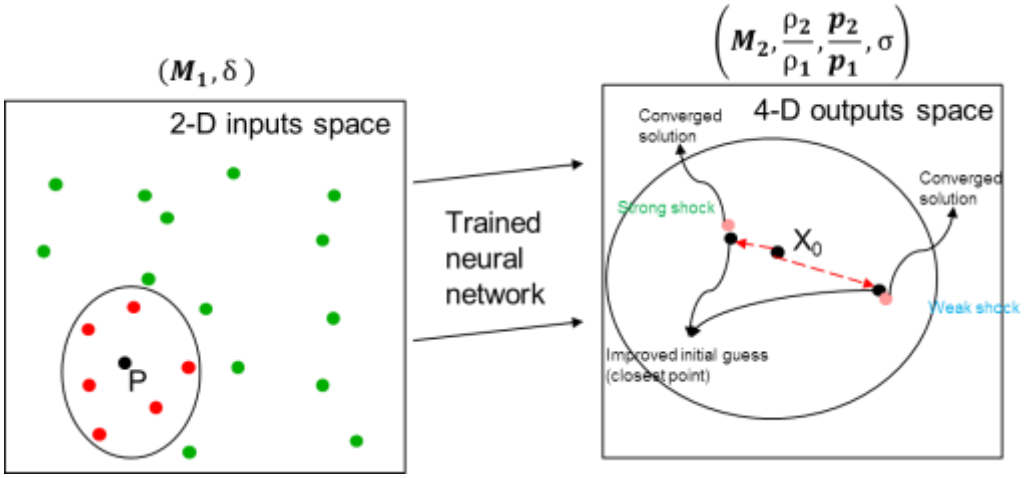
3. Artificial Neural Network

Artificial neural networks (ANN) based on the logistic sigmoid activation function are also employed in this framework. The MLPRegressor of the Sklearn library [2] is employed using LBFGS solver and a constant learning rate. This regressor implements a multi-layer perceptron (MLP) that is trained using back propagation. The method trains a small neural network using a subset of points surrounding the point of interest to find an appropriate initial guess. Note that this approach requires several points already existing in the database (for example: Ten points computed with continuation method, Delaunay triangulation, or cubic interpolations). A traditional neural network is trained using a training dataset to generate the ANN's final weights. Once this training dataset is updated, the neural network has to be regenerated from scratch to account for the changes, which is computationally expensive, especially if it has to be repeated every time a new point is added to the database. The proposed neural network implementation, however, trains a one-time use network at a local region of interest and generates weights that are tied to it. This process takes less computational time due to the small size of the locally selected training set and allows for continuous growth of the global input database. The proposed algorithm, described in Figure 8, is as follows

1. Read input $P = (\hat{M}_1, \hat{\delta})$.
2. Search for the ten nearest points in the database input space using the Euclidian distance (Figure 8-a).
3. Train a neural network for each of the four objective functions $\left(M_2, \frac{\rho_2}{\rho_1}, \frac{p_2}{p_1}, \sigma\right)$ using the information from the ten points. Compute the initial guesses for both

shocks. Figure 8-b shows how the computed initial guess (black dot) is very close to the solution (pink dot) for both the shock cases. The number of hidden layers can be based on a parametric study. However, in this work, we have chosen 5 layers.

4. Select also the best approach to use based on the information stored for the same vertices.



(a) Determining the set of nearest points (b) Determining the weak and strong solutions

Figure 8. Learning approach based on artificial neural networks

4. <u>Initializing the Database and Collecting Information on the Different Solvers:</u>

It is required to have a minimum number of solution points using a regular method, the continuation method, for example, before starting to use the self-learning approach. It is also possible to use a few points from the Mach wave and normal shock curves, since they are already available. As shown in Figure 9, the five points selected in this case are shown in black for the Mach wave and normal shock solutions and in red for the continuation method. The set of three points using a continuation method are selected quasi-randomly, as described below.

To collect some preliminary information about the different *nleqslv* approaches (Approaches 1, 7, and 11 defined above), the fresh database D is initialized using the following algorithm:

1. Statistical sampling: create a set of 20 points $\Gamma = \{(M_1, \delta)\}$ using the Latin hypercube ($1.0 \leq M_1 \leq 3.5$ and $0 \leq \delta \leq 45°$). Sort the points along ascending δ.

2. Solve for the normal shock and Mach wave: Add two points from the normal shock/Mach wave curves to the empty database, the solutions X for $(M_1, \delta) = (1,0)$ and $(3.5,0)$.
   Calculate the solution for following three quasi-randomly selected points using the continuation method and add them to the database D:

$$\widehat{M}_1 = \max_{P\in\Gamma}(M_1), \qquad \hat{\delta} = (\max_{P\in\Gamma}(\delta) + \operatorname{median}_{P\in\Gamma}(\delta))/2,$$
$$\widehat{M}_1 = \max_{P\in\Gamma}(M_1), \qquad \hat{\delta} = (\min_{P\in\Gamma}(\delta) + \operatorname{median}_{P\in\Gamma}(\delta))/2,$$
$$\widehat{M}_1 = \operatorname{median}_{P\in\Gamma}(M_1), \qquad \hat{\delta} = (\min_{P\in\Gamma}(\delta) + \operatorname{median}_{P\in\Gamma}(\delta))/2.$$

   Once these five initial points have been added to the fresh database D, the oblique shock solution is computed for all the points in Γ as described in the remaining steps below. A typical sample set Γ is shown in Figure 9. Note that only the points that have an oblique shock solution are shown in the figure.

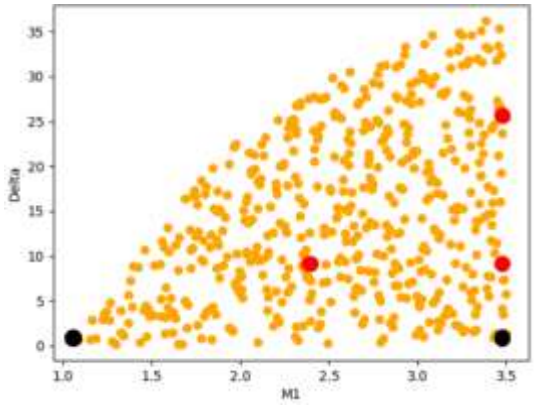


Figure 9. Initialization points (Mach wave and normal shock in black, continuation method in red) and feasible solutions space (orange points). A small number of points is recommended for this problem (n<20). The high number of points shown in the picture is only for a better visualization.

3. Consider a new point $P = (\widehat{M}_1, \hat{\delta})$ in Γ. The remaining steps of the algorithm will be applied for both oblique shocks (starting with the weak shock case).
   a. Use the Delaunay Triangulation to find the enclosing triangle (

   b. Figure **6**-a). The objective of this step is to know the number of solutions for this case by looking into the neighboring points in the inputs space, and then compute an appropriate initial guess.
      i. If no solution exists at all the triangle vertices: This input point $(\widehat{M}_1, \hat{\delta})$ has no physical solution either, based on the assumption that the set of all solutions to the governing equations (1)--(4) is

a convex set. If it is not the case, the continuation method can be applied here. The user has to specify which option to use before running the software.

ii. If at least one of the vertices has a solution: Apply one of the two interpolation techniques (Delaunay or cubic interpolation) to compute the initial guess $\mathrm{X}_0 = \left((M_2)_0, \left(\frac{\rho_2}{\rho_1}\right)_0, \left(\frac{p_2}{p_1}\right)_0, \sigma_0\right)$.

c. Attempt the solution with the three approaches (1, 7, and 11) and determine the one that has the lowest number of function evaluations.

d. For the weak shock case, if the algorithm by chance converges to a strong shock solution, or does not converge at all, save the result and continue to Step 3-d. Otherwise, go to Step 3-e.
For the strong shock case, if the algorithm does not converge, go to Step 3-d, otherwise, go to Step 3-f.

e. Determine the solution using the continuation method

f. Go to Step 3-a and redo these steps using the strong shocks sub database.

g. After getting both weak and strong shocks, save the result into the database and go to Step 3-a with the next point *P*.

Note that Latin hypercube statistical sampling is performed. However, any other method that serves the same purpose can also be used. It is important to have a set of points that covers the input space evenly. While quasi random methods can be a good fit for this case, purely random methods can provide poor distribution of uncorrelated random points, which can form clusters in some areas and leave large gaps in other areas. The Latin hypercube is employed in this work in the database initialization and self-learning method testing. Note also that the continuation method in Step 3-d will only be used if there is not enough information in the region surrounding the point of interest such that the initial guess predicted by the algorithm fails to converge to the desired solution.

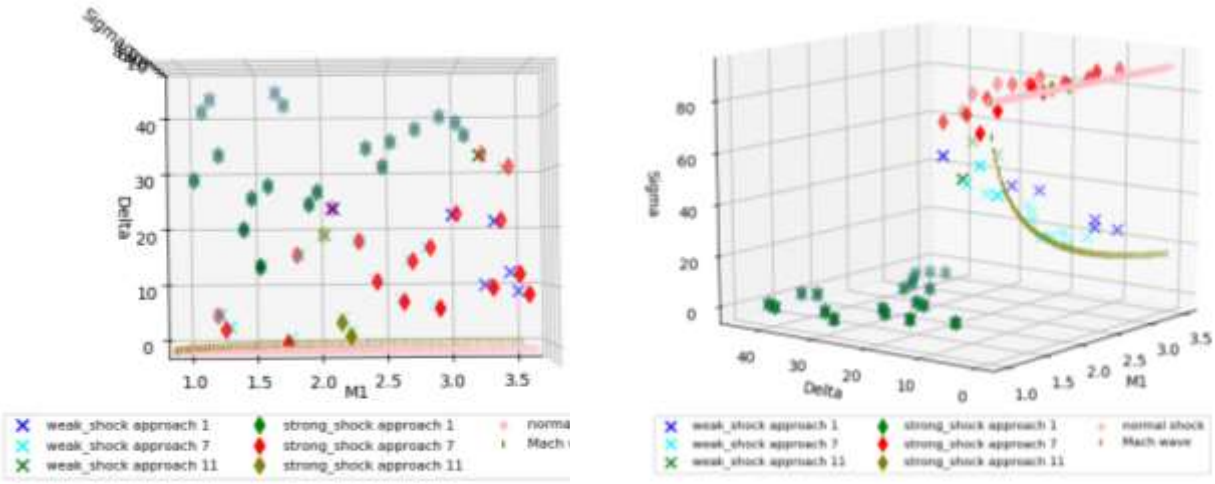
5. The Learning Approach Algorithm

After obtaining an initialized database with some preliminary comparison of the three *nleqslv* approaches (1, 7, and 11), the following self-learning algorithm is applied to any input point *P*.

1. Read input $P = \left(\widehat{M}_1, \hat{\delta}\right)$.
2. The remaining steps of the algorithm will be applied individually for both oblique shocks (starting with the weak shock case):

a. Use the Delaunay Triangulation to find the enclosing triangle Figure 6-a). The objective of this step (a- and b-) is to determine the number of solutions and the best *nleqslv* approach to use for this case by investigating the neighboring points in the input space, and then compute an appropriate initial guess.
    i. If no solution exists at all the vertices: This input point $(\widehat{M}_1, \hat{\delta})$ has no physical solution either, based on the assumption that the set of all solutions to the governing equations (1)--(4) is a convex set. If it is not the case, the continuation method can be applied here. The user has to specify which option to use before running the software.
    ii. If at least one of the vertices has a solution: Apply one of the three learning techniques (Delaunay, cubic interpolation, or ANN) to compute the initial guess $(\mathrm{X})_0 = \left((M_2)_0, \left(\frac{\rho_2}{\rho_1}\right)_0, \left(\frac{p_2}{p_1}\right)_0, \sigma_0\right)$.
b. Select also the best *nleqslv* approach to use based on the information of the same vertices.
c. For the weak shock case, if the algorithm converged to the strong shock, or did not converge, save the result and continue to Step 2-d. Otherwise, go to Step 2-e.
   For the strong shock case, if the algorithm did not converge, go to Step 2-d, otherwise, go to Step 2-f.
d. Determine the solution using the continuation method
e. Go to Step 2-a and redo these steps using the strong shock solutions sub database.
f. After getting both weak and strong shocks, insert the result into its position in the database such that it remains always sorted along ascending δ. Keeping the database sorted enables the use of the bisection method and results, therefore, in much faster search operations.

Note that the continuation method (Step 2-d) will only be used if there is not enough information in the region surrounding the point of interest such that the algorithm fails to converge from the initial guess to the desired solution. Note also that the minimum length of the database depends on the employed method (Delaunay, cubic interpolation, or ANN).



Top view Side view

Figure 10. Initialized database with the continuation method. Comparison of the 3 *nleqslv* approaches.

VII. Implementation and Evaluation

1. Database Initialization using Continuation Method

In this case, the fresh database is initialized using only the continuation method. In other words, every point is simulated independently, and the database is not storing the results. The obtained results (shock angle σ^*) are shown in

Figure **10**. The 3 *nleqslv* approaches are also compared here and different colors are used to represent the best approach obtained for each point. Results show that Approaches 1 (Newton with cubic line search) and 7 (Broyden with cubic line search) are better than 11 (Broyden with Powell dogleg) when initial guesses are far from the solution. Moreover, while the maximum number of function calls reached 296 in this case for the feasible points (

Figure **11**), it did not exceed 56 for the self-learning case (Figure 13) as will be shown later. In other words, disabling the learning feature leads the number of function evaluations and consequently the CPU time to rise significantly.

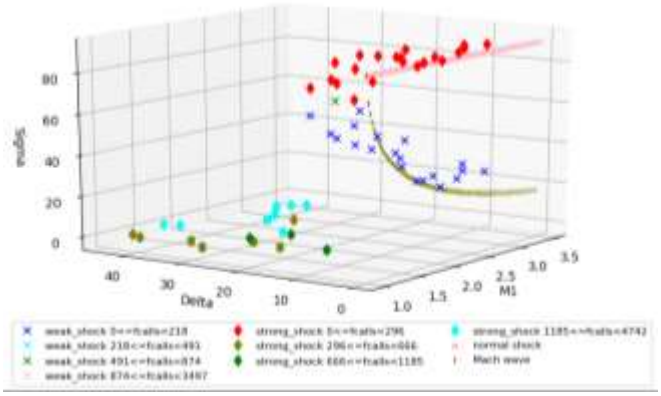
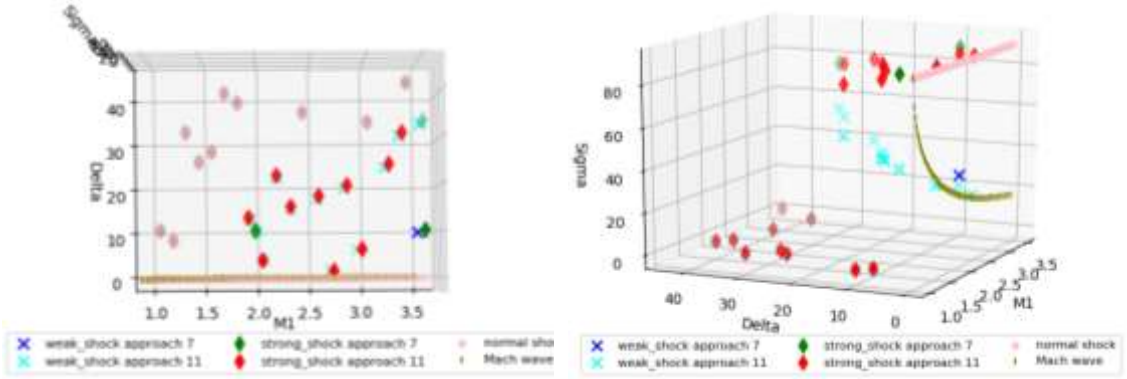


Figure 11. Initialized database with the continuation method. Comparison of the number of function calls.



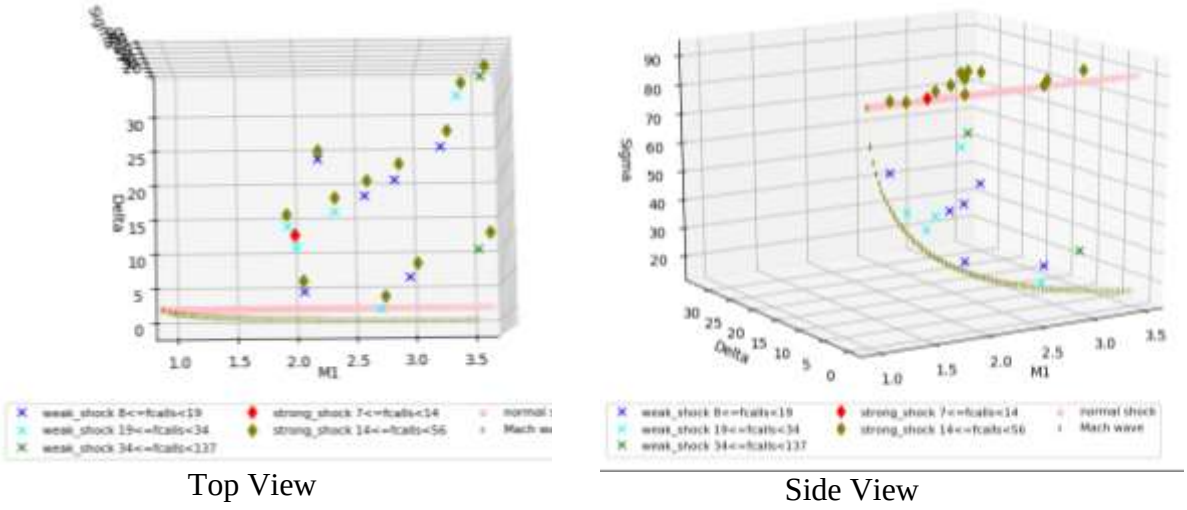
Top View Side View

Figure 12. Initialized database with Delaunay triangulation. Comparison of the 3 *nleqslv* approaches.

2. Database Initialization using Delaunay Triangulation

In this case, starting from a fresh database, we start by initializing it as described in the previous section. First, three points are simulated using the continuation method and two more points are imported from the Mach wave and normal shock curves. Then, another twenty points are simulated using the self-learning algorithm based on the Delaunay triangulation. The obtained results (shock angle σ^*) are shown in Figure 12. The points that did not have a solution are assigned with zero response functions $(M_2, \frac{\rho_2}{\rho_1}, \frac{p_2}{p_1}, \sigma)$. The Approaches 1, 7, and 11 are compared and different colors are used to represent the best approach obtained for each point. The figure shows that Approach

11 (Broyden with Powell dogleg) is dominant since the initial guess is always close to the solution due to the self-learning method. Figure 13 shows the number of function evaluations for each point. Most of the weak shock solutions required less than 34 function calls (except for the continuation method points where it reached 137 function calls) while the strong shock solutions required less than 56 function calls. Therefore, employing the proposed learning feature reduces the number of function evaluations (and consequently the CPU time) significantly.



Top View Side View

Figure 13. Initialized database with Delaunay triangulation. Comparison of the number of function calls.

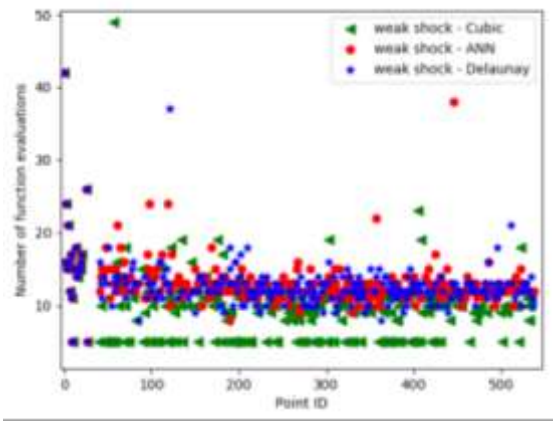
3. Comparison of Delaunay Triangulation, Cubic Interpolation, and ANN

After initializing a fresh database with 40 points, we perform the test case with another statistical sampling of another 500 points using each of the three methods (Delaunay triangulation, cubic interpolation, and artificial neural networks). These test points are shuffled to simulate a more realistic case of user input points for the software over time. About half of these points have an oblique shock solution.

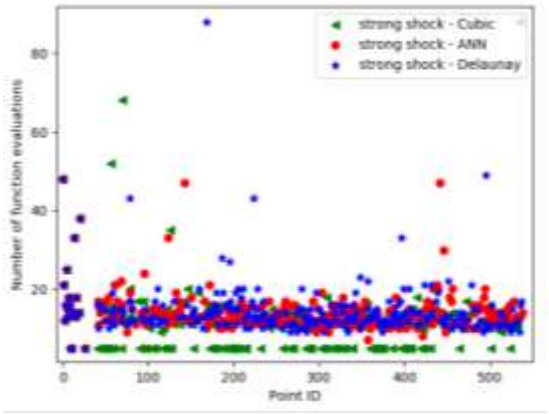
Figure 14 shows a comparison of the evolution of the number of function evaluations as the database grows for the three methods. In this figure, all the points are shown (feasible and non-feasible points). It can be seen clearly that while the Delaunay triangulation and ANN have similar trends, the cubic interpolation gave a better performance. The implication of this result is that the shock solutions are very smooth functions of (M_1, δ), and hence
the higher order cubic interpolation is more accurate than the first order Delaunay and ANN interpolation.

The software performance can also be assessed by evaluating the distance between the predicted initial guess and the computed solution $dist = \max_i \left(\left| \frac{(X^* - X_0)_i}{UB_i - LB_i} \right| \right)$, where X_0 is the initial guess, X^* is the obtained solution, and LB_i and UB_i are the lower bound

and upper bound of the output variable i $\left(M_2, \frac{\rho_2}{\rho_1}, \frac{p_2}{p_1}, \sigma\right)$, respectively ($LB_i = \{0.25,1,1,0\}$ and $UB_i = \{3.5,12,12,90°\}$). Figure 15 shows also that the three methods have similar trend and that the cubic interpolation method has the least variability.



(a) Weak shock solutions

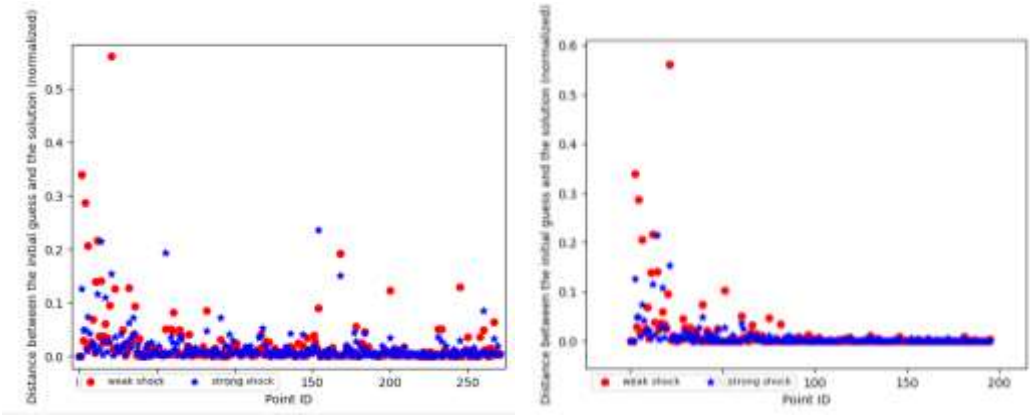


(b) Strong shock solutions

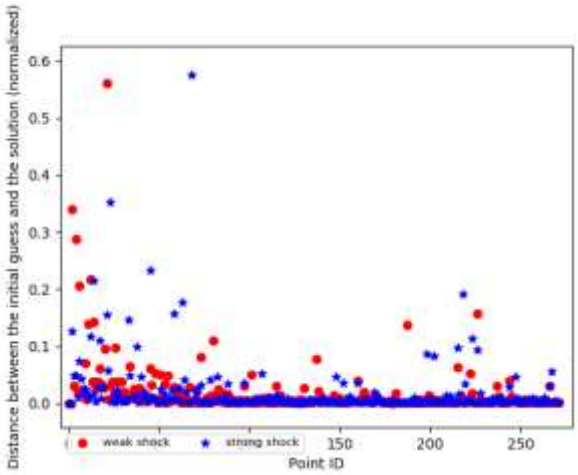
\

Figure 14. Comparison of the software performance for the three methods as the database grows.

Figure 16 shows the evolution of the CPU time for the three methods as the 500 points are being solved. The points that have a CPU time over 0.6s are non feasible points. Even though about half of the 500 points do not have a solution, only a few of them were attempted using the continuation method and they required a high CPU time, while the others were found faster to have no solution due to the self-learning process. Note also here that the cubic interpolation cases have a better CPU time and clearer pattern while the Delaunay triangulation and ANN behave similarly. It can also be noted that there is a slight increase in the CPU time for solving a given problem, which can be explained as an outcome of a growing database. The bigger database makes search operations slightly more time expensive. However, this increase would not even be noticeable for other problems that require much more time for function evaluations, such as computational fluid dynamics or multi disciplinary problems.



(a) Delaunay triangulation (b) Cubic



(c) Neural Network

Figure 15. Improved precision in initial guess predictions as the database grows.

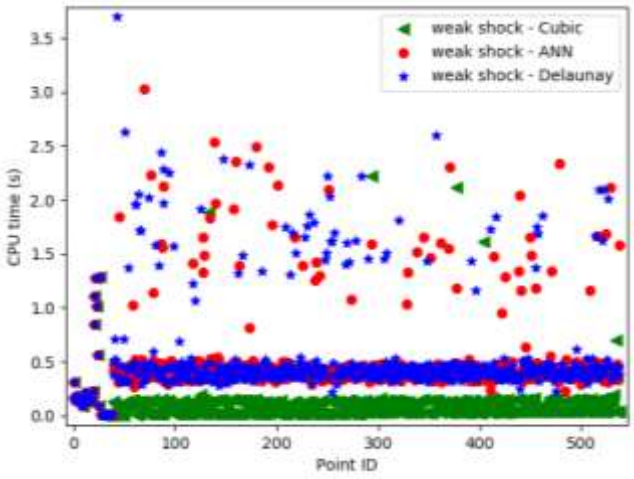


Figure 16. CPU time comparison for weak shock solutions with the three methods

VIII. Conclusion

A new approach for computationally solving engineering problems using the emerging area of machine learning has been proposed and tested to significantly reduce the computational time as the software solves an increasing number of similar problems. As it solves a large number of problems, the software develops an ability to make effective decisions towards improving its performance when solving complex/nonlinear problems. The method consists of employing the previously collected analysis information when solving a set of similar problems to make better decisions (e.g., which numerical method to employ and which initial guess to use for solving a set of nonlinear equations) and ensure faster convergence. One key idea of the proposed self-learning concept consists of constructing local representations of a function that help determine a good approximation for the expected solution, and use it as the initial guess. To efficiently construct the local surrogate functions, three different approaches (Delaunay triangulation, cubic interpolation, and artificial neural network) have been implemented and compared. An implementation of the new high-dimensional Delaunay triangulation in a machine learning application was demonstrated in this effort. Moreover, unlike the traditional use of the neural network, the concept of locally trained neural network with moving weights was presented in this work as a way to allow a continuous growth of the constructed database used in the training. Different numerical techniques are compared in each region of the input space to help collect some useful information that serves in determining which numerical solver to consider when solving a new problem.

The oblique shock solution of a supersonic flow over a turn or wedge was considered as a test case. The problem consists of a set of four coupled nonlinear equations with six variables (two user-preselected and four unknowns). The nonlinear solver *nleqslv* has been employed to implement and to compare different numerical techniques and line search methods. Results show that Broyden method

with the Powell dogleg method performs better when the initial guess is close to the solution; the Newton method with cubic line search and the Broyden method with cubic line search are better when using initial guesses that are somewhat far from the solution. A self-learning algorithm has been proposed such that it finds the number of solutions *a priori*, makes a proper decision on which *nleqslv* method to use, and comes up with an appropriate initial guess for the new problem. The approach has been shown to be very efficient, especially since it reduces the difficulty of identifying appropriate initial guesses when solving the problem numerically. It has been shown that the proposed self-learning approach reduces the number of function evaluations significantly as compared to a reference method, the continuation approach. For example, in the presented case study, while the maximum number of function calls reached 296 using the continuation method, it did not exceed 56 with the self-learning approach. The self-learning method was also shown to have an improved performance as the database grows. Furthermore, it has been shown that the cubic interpolations method performs better than the Delaunay triangulation and artificial neural network for this problem.

Acknowledgments: This effort was funded by NASA through the National Institute of Aerospace with Joseph H. Morrison as the project manager. The authors would like to thank Tyler Chang, for providing his VT Delaunay triangulation code, and Thomas Lux, both of Virginia Tech's Department of Computer Science, for help in wrapping it in PYTHON.

References

[1] Y. S. Ong, P. B. Nair, A. J. Keane and K. C. Wong, *Surrogate-Assisted Evolutionary Optimization Frameworks for High-Fidelity Engineering Design Problems*, in Knowledge Incorporation in Evolutionary Computation, edited by Jin, Y., New York, Springer–Verlag, 2005, 307–331.
https://doi.org/10.1007/978-3-540-44511-1_15

[2] T. Shao, S. Krishnamurty and G. Wilmes, Preference-Based Surrogate Modeling in Engineering Design, *AIAA Journal*, **45** (2007), no. 11, 2688-2701.
https://doi.org/10.2514/1.27777

[3] A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon and M. W. Trosset, A Rigorous Framework for Optimization of Expensive Functions by Surrogates, *Structural Optimization*, **17** (1999), no. 1, 1–13.
https://doi.org/10.1007/bf01197708

[4] P. R. Caixeta Jr. and F. D. Marques, Multiobjective Optimization of an Aircraft Wing Design with Respect to Structural and Aeroelastic Characteristics using Neural Network Metamodel, *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, **40** (2018), no. 1, 1-11.
https://doi.org/10.1007/s40430-017-0958-7

[5] Y. LeCun, Y. Bengio and G. Hinton, Deep Learning, *Nature*, **521** (2015), 436–444. https://doi.org/10.1038/nature14539

[6] G. Wang and S. Shan, Review of Metamodeling Techniques in Support of Engineering Design Optimization, *Journal of Mechanical Design*, **129** (2007), no. 4, 370–380. https://doi.org/10.1115/1.2429697

[7] R. Laddaga, *Self-Adaptive Software*, DARPA BAA., 1997, 98-12.

[8] M. Salehie and L. Tahvildari, Self-Adaptive Software: Landscape and Research Challenges, *ACM Transactions on Autonomous and Adaptive Systems*, **4** (2009), no. 2, 1-42. https://doi.org/10.1145/1516533.1516538

[9] A. H. Shapiro, *The Dynamics and Thermodynamics of Compressible Fluid Flow*, vol. I, New York: Ronald Press Co., 1953-54.

[10] T. H. Chang, L. T. Watson, T. C. H. Lux, A. R. Butt, K. W. Cameron and Y. Hong, Algorithm 1012: DELAUNAYSPARSE: interpolation via a Sparse Subset of the Delaunay Triangulation in Medium to High Dimensions, *ACM Transactions on Mathematical Software*, **46** (2020), no. 4, 1–20. https://doi.org/10.1145/3422818

[11] R. P. Lippmann, An Introduction to Computing with Neural Nets, *IEEE Acoustic Speech Signal Processing Magazine*, **4** (1987), 4-22, https://doi.org/10.1109/massp.1987.1165576

[12] B. Hasselman, Package "nleqslv", https://cran.rproject.org/web/packages/nleqslv/index.html, 2017.

[13] J. J. Dennis and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Siam, 1996. https://doi.org/10.1137/1.9781611971200

[14] M. Powell, *A Hybrid Method for Nonlinear Algebraic Equations*, Numerical Methods for Nonlinear Algebraic Equations. Gordon and Breach, 1970.

[15] M. Powell, *A Fortran Subroutine for Solving Systems Nonlinear Equations*, Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz (Ed.), Gordon & Breach, 1970.

Received: January 29, 2022; Published: February 25, 2022