



# COMPUTER SOFTWARE PRODUCT END ITEMS

## OPTIMIZATION TOOLS

---

Submitted to:

SPAWAR Systems Center San Diego  
53560 Hull Street, Code 2206  
San Diego, California 92152-5001

Attention:

Mr. Mark Tukeman, Code 2712  
(619) 553-1510

In fulfillment of the requirements for:

Contract No. N66001-02-D-0039  
*Development of Port Related Transportation Technologies  
to Advance Military Responsiveness to National Needs*

Task Order No. 0011  
*Computational Fluid Dynamic (CFD) Design Tool Development and Validation*

CDRL Data Item No. A001BA

Security Classification: Unclassified

Prepared and Submitted by:

Center for the Commercial Deployment of Transportation Technologies  
California State University, Long Beach Foundation  
6300 State University Drive, Suite 220 • Long Beach, CA 90815 • 562.985.7394

Approved for public release: distribution is unlimited

**September 30, 2003**

## *Computer Software Product End Items*

Computational Fluid Dynamics (CFD) Tool Development

### *Deliverable 2*

### *Optimization Tool Development Based on Neural Networks*

P/E Number: 2.20

Task Order Number: 11

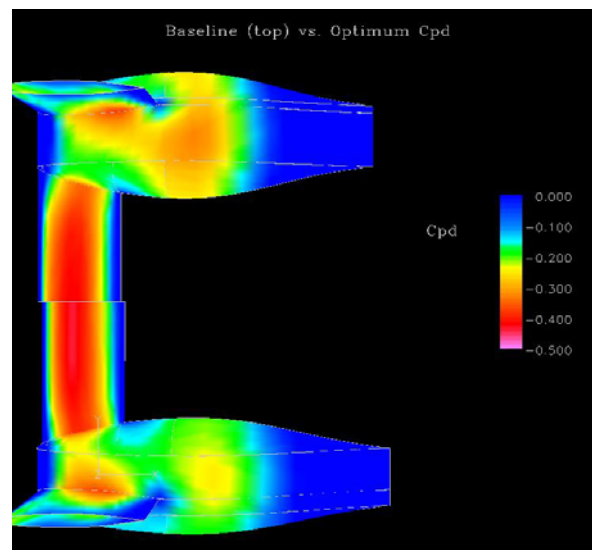
POC

Hamid Hefazi, Professor & Chair

Mechanical and Aerospace Engineering Department

California State University, Long Beach

[hefazi@csulb.edu](mailto:hefazi@csulb.edu)



## Table of Contents

<b>Introduction</b> .....	<b>3</b>
<b>1. Optimization method based on Neural Networks</b> .....	<b>3</b>
1.1. Overall approach.....	3
1.2. Neural Networks (NN) overview.....	5
1.3. Training set (TS) and validation set (VS).....	6
1.4. Non-dimensionalization and error definitions.....	7
1.5. Improvements.....	7
1.6. Integration of the NN into the optimization process.....	13
<b>2. Software user’s manual</b> .....	<b>14</b>
2.1. Input File.....	15
2.2. Output File.....	16
<b>3. Summary/Conclusion</b> .....	<b>21</b>
<b>4. Glossary</b> .....	<b>22</b>
<b>5. Acknowledgments</b> .....	<b>22</b>
<b>6. References</b> .....	<b>23</b>
<b>7. Appendix A</b> .....	<b>25-76</b>

## Introduction

This report presents the development of optimization methods based on Neural Networks. The method, along with its theoretical foundation, is briefly described in Sect. 1. Section 2 presents the actual product resulting from the research work and includes a description of how the method is used, including a presentation of input and output data. The computer code developed is also presented. Validation of the method is presented in the accompanying software test report (STR). Further application of the method to a three dimensional configuration is presented in reports on deliverable 3 of this task.

### 1. Optimization method based on Neural Networks

#### 1.1. Overall approach

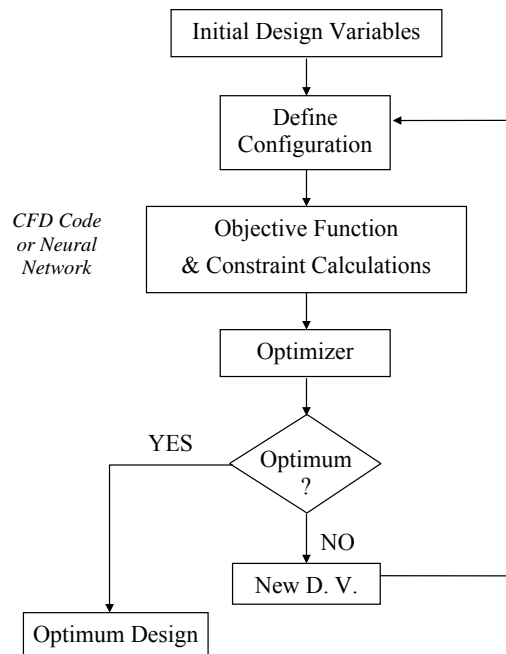
The three main components of a numerical optimization method for aerodynamic and/or hydrodynamic optimization are, (1) the representation of a configuration by a set of parameters called design variables (D.V.),  $\mathbf{x}$ , (2) the optimization method, and (3) the evaluation of the objective function ( $f$ ), which, in CFD, is the aero- or hydrodynamic performance for that given configuration.

The general optimization process is illustrated in Fig. 1. An initial set of design variables, which might represent the configuration designed by experienced engineers, is supplied to the optimizer. Then, for this design, the objective function,  $f$ , is evaluated and the constraints,  $g_i$ , are analyzed to check whether they are violated or not. If the optimum is not reached, these values are fed back to the optimizer which modifies the D.V.'s. The process is repeated until convergence.

In the classical case, the objective function calculation requires the use of a CFD code that solves the equations of fluid dynamics and evaluates the aero- or hydrodynamic performance needed to perform the optimization. This calculation is usually very computer intensive – and drives the cost of the optimization process – with one flow solution taking from several minutes to several hours or days in the most complex problems, even using massive parallel computations.

In the approach using neural networks (NN), the process is divided in two parts: (1) generation of a training set and training the neural network, and (2) optimization of the

configuration, like in Fig. 1, with the NN instead of the CFD code. The Network is trained with a relatively small set of data, which can be a mix of experimental data and/or data generated with the CFD code. Then, this network is coupled with a global optimization method without the drastic increase in CPU time since the NN allows for a very fast evaluation of the aerodynamic performance. All other elements of the optimization approach, such as configuration representation and selection of the numerical optimization algorithm are currently independent of the use of Neural Networks to predict the aero/hydrodynamic properties.



**Fig. 1** Flowchart of the numerical optimization. In the NN approach, the CFD code is replaced by a NN evaluator which has been trained *a priori*

Instead of having the cost buried in the function evaluations during the optimization, and therefore not easily known a priori, the approach using Neural Networks (NN) moves the cost to the generation of the training set. By controlling the size of the training set, not only the cost can be known a priori, but also reduced. Knowing the cost of the optimization offers advantages from a programmatic stand point because it allows managers to know exactly when they will get a solution to the problem at hand and also what the cost will be. In addition, the algorithm allows for external data, such as previous suitable computations or experimental data, to be included in the training set if desired.

1.2. Neural Networks (NN) overview

The ability of the NN to accurately predict the hydrodynamic properties of the configuration over a large design space is key to the overall approach. For this purpose, fully forward connected networks, a.k.a. Ordered Neural Networks (ONN), have been used and improved. Ref. 1 describes in detail the topology of such neural networks for their application to function evaluation. The ONN is trained using Cascade-Correlation. Instead of just adjusting the weights in a network of fixed topology, Cascade-Correlation begins with a minimal network, then automatically trains and adds new hidden units one-by-one, creating a structure of the type ONN. It was first introduced by Fahlman and Lebiere [2] and is described in detail in Ref. 1.

This architecture has several advantages over other algorithms:

- It learns very quickly
- The network determines its own size and topology
- It retains the structure it has build even if the training set changes
- It requires no back-propagation of error signals through the connections of the network.

A synopsis of the Cascade-Correlation as well as the equations governing the neural network have been described in detail in the FY01 report [1] and will not be repeated here. The resulting neural network can be described by the following two matrices [1].

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} v_{11} & \dots & v_{1,n+1+h} \\ \vdots & & \vdots \\ v_{m1} & \dots & v_{m,n+1+h} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ +1 \\ z_1 \\ \vdots \\ z_h \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ \vdots \\ u_h \end{bmatrix} = \sigma \left( \begin{bmatrix} w^1_1 & \dots & w^1_{n+1} & 0 & \dots & 0 \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & \ddots & & \vdots \\ w^h_1 & \dots & \dots & \dots & w^h_{n+h} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ +1 \\ z_1 \\ \vdots \\ z_h \end{bmatrix} \right)$$

where  $x_l$  ( $l \in \{1, \dots, n\}$ ) are the inputs to the neural network,  $y_i$  ( $i \in \{1, \dots, m\}$ ) the outputs,  $z_k$  ( $k \in \{1, \dots, h\}$ ) the intermediate states, or the output to the  $h^{\text{th}}$  hidden unit.  $\mathbf{W}=[w^h_j]$  are the weights between inputs and each hidden unit saved after each unit has been added to the network (at iteration  $h$ ),  $\mathbf{V}=[v_{ij}]$ , the weights between inputs, plus all hidden units and outputs. The output to each hidden unit, which only depends on the inputs to the network and the outputs from the previous hidden units, can be calculated by recurrence. The vector containing all inputs and intermediate outputs is also called  $\mathbf{Z}$ ,

$$\mathbf{Z} = [x_1 \quad \dots \quad x_n \quad +1 \quad z_1 \quad \dots \quad z_h]^T.$$

While the work of Fahlman and Lebiere was geared towards pattern recognition, substantial modifications and improvements have been made over the past few years for the application to function evaluation. The method has been successfully applied to a model problem [1]. As part of the current work, the NN model has been further improved to address several limitations. These improvements are described below.

Prior discussing the improvements made to the NN of [1], it is necessary to introduce the definitions of training and validation sets, as well as for the error terms which will be analyzed in the next sections.

### 1.3. Training set (TS) and validation set (VS)

A training set (TS) corresponds to a set of known data points (design variables and their associated values, such as objective function(s) and constraints) used to train the NN, i.e. the network attempts to achieve an output which matches the input (training set).

A validation set (VS) is a set which, unlike the TS, is *not* used for training per say, but rather is used for *stopping* the training. The purpose if the VS is to avoid over-fitting which can occur with Cascade Correlation. [1]

When data sets are not available a priori, both types of sets are generated using Latin Hypercube [3]. The process for generation is the same as that of Fig. 1, except that the optimizer cycles through a Latin Hypercube. The size of these sets is decided by the user and is problem-dependent. Typically, the VS is much smaller than the TS, e.g. several hundred for the VS vs. several thousands for the TS.

#### 1.4. Non-dimensionalization and error definitions

Inputs are normalized between 0 and 1. Because some output values may be far from the average, outputs are normalized such that the *mean of the output of each training set is 1*. For example, for a dimensional 2-element output training set {2, 4}, the normalized training set would be {2/3, 4/3}.

The maximum of the error is,

$$E_{\max} = \sqrt{\max_{p \in \{1, \dots, P_{\max}\}} \|\mathbf{y}_p - \mathbf{t}_p\|^2}$$

where  $\mathbf{y}_p$  is the output vector from the NN,  $\mathbf{t}_p$  the target vector, and  $p$  is the index for each element of the training or validation set (which contains  $P_{\max}$  elements).

The squared error, which is used during the learning algorithm, is given by

$$E = \frac{1}{2} \sum_{p=1}^{P_{\max}} \|\mathbf{y}_p - \mathbf{t}_p\|^2$$

$(E_{\max})^2$  is typically 1/2 to 1/3 of the squared error at the end of training. This means that at least one point of the training set is not well approximated and accounts for 1/2 to 1/3 of the total error. This problem is inherent to the cascade correlation algorithm as only one neuron is trained at a time with only the connections to and from this hidden unit allowed to vary.

#### 1.5. Improvements

##### *1.5.1. Reorganization of the code*

NN code as left at end of FY01 was in its development stage. It was only meant to be used by the developer. The flowchart of code has been reorganized and unnecessary subroutines have been removed. The code was made more general, it was modified to be to have all the different training options read in the input file, also it allowed to read any database file created with Isight [4] or any other program as long as the variables are written in columns format.

The input file currently allows for choosing the number of inputs, number of outputs, training, validation and generalization set sizes, the names and location of those databases and which columns to read from those databases, which stopping criterion to use (stopping criteria GL, PQ, UP as defined in FY01[1]), which Correlation Formula to use to train the candidate hidden units, the number of candidate units to try before installing a

permanent hidden unit on the NN (ncand), and the number of NN's to build before choosing the best (ntrys).

The outputs of the code were also modified. The weights are written out for each of the NN's build. The best NN is saved in a separate file. Those files are formatted as a C program that serves as the NN Evaluator for the function being approximated, so that it can be used in the optimization without any user intervention. The process of training the NN to approximate any function has been made 100% automatic.

### *1.5.2. Weight initialization*

In any nonlinear optimization problem, the initialization of the parameters has an important influence on the ability of the training program to converge and the speed of that convergence. The training of weights in NNs can be viewed as a nonlinear optimization problem in which the goal is to find a set of network weights that minimizes a cost function. The cost function which in our case is either the error on the training set or the correlation function describes a surface in the weight space. Training algorithms can be viewed as methods to find the minimum of this surface. The complexity of the search is governed by the nature of this surface. Error surfaces for multilayer NNs have typically many flat regions where learning is slow and long narrow “canyons” that are flat in one direction and steep in the other directions. This makes it very difficult to search the surface efficiently using gradient based routines. In addition the cost function is characterized by a large number of local minima with values in the vicinity of the best global minimum. The efficiency of the search method depends much on the initial weight distribution. The simplest category among the weight initialization methods is random weight initialization. It is commonly known that if all the weights of a NN are initialized with a zero, they cannot change to any other value during training if some simple training algorithms are used. Random initialization has been proposed to avoid this undesired situation and its ability to break the symmetry. The program as written in FY01 had random weight initialization but no systematic study had been conducted on the weight range to use for each weight search optimization. Very little research has been reported on weight initialization in the literature [5, 6].

During FY02, studies on weight initialization were conducted on the two-dimensional model problem to determine the best range for initialization and optimization. In the Cascade-Correlation algorithm, there are three separate weight optimization problems to

investigate. The first one is the squared error minimization between input and output units before any hidden unit is added to the network. The second one is the maximization of the Correlation formula when candidate hidden unit training is performed. And the third one is the error minimization each time a new hidden unit (with the largest Correlation amongst the candidates) has been added to the network.

1. In FY01, for the first optimization problem, the weights between the inputs and the outputs  $v_{ij}$  were initialized randomly such that for each output  $l$  ( $y_l$ ),

$$\|v_l\| = \sum_{j=1}^{n+1} v_{lj}^2 \leq 4 * v_{\max} \text{ for a given real } v_{\max} > 0 \text{ where } y_l = [v_{l1} \quad \dots \quad v_{l,n+1}] \begin{bmatrix} z_1 \\ \dots \\ z_{n+1} \end{bmatrix} \text{ (see Ref. 1}$$

for full equations).

Also the weights were constrained to remain between  $-v_{\max}$  and  $+v_{\max}$  during optimization, so that  $|v_{ij}| \leq v_{\max}$ . The value of  $v_{\max}$  was fixed at 10. Values from 0.5 to 100 were investigated, none leading to a satisfactory result. The study showed that it gave better results to initialize each weight  $v_{ij}$  randomly between  $[-0.5, +0.5]$  and to keep the minimum and maximum values during optimization between -10 and +10.

2. For the second optimization problem consisting in finding the best candidate hidden unit by maximizing the Correlation formula, our study showed that the initialization used previously leads to good results and does not require any change. For this weight initialization, the norm of the vector  $\mathbf{Z}$ ,  $\|\mathbf{Z}_p\|$ , is calculated for each Training Set point  $p$  (also called pattern). And the weights for the new candidate unit  $w_j$  are initialized so that  $\|w\| = \sum_{j=1}^{n+h+1} w_j^2 \leq 4 * \max_{p=1, \dots, P_{\max}} (\|\mathbf{Z}_p\|)$ . Once

optimized those weights are saved in the weight matrix  $\mathbf{W}$ , on row  $h$ , where  $h$  corresponds the  $h^{\text{th}}$  hidden unit added on to the network. Also during the optimization, the weights values are limited between -10 and +10. Lower values lead to very deep networks, and higher values lead to severe overfitting of the data.

3. The third optimization consists in minimizing again the squared error  $E$  with the new hidden unit ( $h^{\text{th}}$  hidden unit) added to the network. The study showed that the weights  $v_{ij}$  found at the previous step (unit  $h-1$ ) are already close to the optimal value with this new hidden unit added to the network. For this weight

initialization problem, it is then best to use as initial weights the ones found at the previous iteration (weights between the inputs and previous hidden units to hidden unit  $h-1$ ) and to initialize at zero the new weights between the inputs and previous hidden unit to hidden unit  $h$ . This method leads to the fastest search for the optimum as well as the smallest overfitting. And again the weights are allowed to vary only between  $-10$  and  $+10$  during optimization.

### *1.5.3. Modification of optimization routine*

The weight optimization is performed with the DOT optimization software. [7] First order search gradient methods do not perform well because of the complicated weight surface. The Sequential Quadratic Programming (SQP) optimization routine from DOT had previously been chosen. This is a second order method that performs best for constrained optimization problems. Indeed, a constraint was imposed on the maximum norm of the weights during optimization.

However, with the modifications implemented in the weight initialization and range, this constraint became obsolete and was thus removed. A new optimization routine, the Broydon-Fletcher-Goldfarb-Shanno (BFGS) method [7] from DOT was chosen to replace SQP. The BFGS method is a second order method for unconstrained optimization problems. It is faster than SQP.

### *1.5.4. New Correlation functions*

The subject of this section is to study a number of objective functions for training new candidate hidden units in constructive algorithms for multilayer feedforward networks. The aim is to study objectives functions the computation of which and the corresponding weight updates can be done in  $O(P_{max})$  time, where  $P_{max}$  is the size of the TS, like for the Correlation formula. Tin-Yau Kwok and Dit-Yan Yeung have extensively studied constructive algorithms for feedforward NNs [8,9,10] and have derived a class of objective functions based on the convergence properties of the NN to approximate any function  $f$ . They have applied it successfully to simple functions of 2 variables using a constructive network architecture where all the hidden units are added in a single hidden layer. The purpose of this study is to apply these new objective functions on the 2D Model problem (8 variables) using our algorithm where units are added in layers, making a multilayer network with one hidden unit per layer. Currently the Correlation Formula as

introduced by Fahlman and Lebiere, is used to train the new candidate units and is denoted  $S_{CASCOR}$ . Kwok and Yeung have introduced 3 additional “Correlation type formulas” denoted  $S_1$ ,  $S_2$  and  $S_3$ . In Reference 10, they also claim that the square root of these functions should even give a better result during optimization because the slope of the objective function during learning is consequently improved. These new correlations functions are also function of the output at the candidate hidden unit for pattern  $p$ ,  $z_{o,p}$ , and the residual error  $E_p = |y_p - t_p|$  for pattern  $p$ . The bar “ $\bar{\phantom{x}}$ ” above a variable denotes the average over the training set. [1] Also, the gradient with respect to the weights,  $\mathbf{W}$ , must be calculated since it must be supplied to the optimizer during training. Following are the equations for each of the new correlation formulas and its gradient. They were coded in the program and tested on the 2D Model Problem.

$$S_{CASCOR} = \left| \sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)(E_p - \bar{E}) \right|$$

$$\frac{\partial S_{CASCOR}}{\partial w_l} = \left[ \text{sign} \left( \sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)(E_p - \bar{E}) \right) \left( \sum_{p=1}^{P_{\max}} (E_p - \bar{E}) \left( \frac{\partial z_{o,p}}{\partial w_l} - \frac{\partial \bar{z}_o}{\partial w_l} \right) \right) \right]$$

where  $\frac{\partial z_{o,p}}{\partial w_l} = \sigma' \left( \sum_{i=1}^{n+h} w_i z_{ip} \right) z_{lp}$

and  $\frac{\partial \bar{z}_o}{\partial w_l} = \frac{1}{P_{\max}} \sum_{k=1}^{P_{\max}} \frac{\partial z_{o,k}}{\partial w_l}$

$$S_1 = \frac{\left( \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right)^2}{\sum_{p=1}^{P_{\max}} z_{o,p}^2}$$

$$\frac{\partial S_1}{\partial w_l} = \frac{2 \sum_{p=1}^{P_{\max}} E_p z_{o,p}}{\left( \sum_{p=1}^{P_{\max}} z_{o,p}^2 \right)^2} \left[ \left( \sum_{p=1}^{P_{\max}} z_{o,p}^2 \right) \left( \sum_{p=1}^{P_{\max}} E_p \frac{\partial z_{o,p}}{\partial w_l} \right) - \left( \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right) \left( \sum_{p=1}^{P_{\max}} z_{o,p} \frac{\partial z_{o,p}}{\partial w_l} \right) \right]$$

$$\sqrt{S_1} = \frac{\left| \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right|}{\sqrt{\sum_{p=1}^{P_{\max}} z_{o,p}^2}}$$

$$\frac{\partial \sqrt{S_1}}{\partial w_l} = \frac{\text{sign} \left( \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right)}{\left( \sum_{p=1}^{P_{\max}} z_{o,p}^2 \right)^{3/2}} \left[ \left( \sum_{p=1}^{P_{\max}} z_{o,p}^2 \right) \left( \sum_{p=1}^{P_{\max}} E_p \frac{\partial z_{o,p}}{\partial w_l} \right) - \left( \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right) \left( \sum_{p=1}^{P_{\max}} z_{o,p} \frac{\partial z_{o,p}}{\partial w_l} \right) \right]$$

$$S_2 = \left( \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right)^2$$

$$\frac{\partial S_2}{\partial w_l} = 2 \left( \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right) \left( \sum_{p=1}^{P_{\max}} E_p \frac{\partial z_{o,p}}{\partial w_l} \right)$$

$$\sqrt{S_2} = \left| \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right|$$

$$\frac{\partial \sqrt{S_2}}{\partial w_l} = \text{sign} \left( \sum_{p=1}^{P_{\max}} E_p z_{o,p} \right) \left( \sum_{p=1}^{P_{\max}} E_p \frac{\partial z_{o,p}}{\partial w_l} \right)$$

$$S_3 = \frac{\left( \sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)(E_p - \bar{E}) \right)^2}{\sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)^2}$$

$$\frac{\partial S_3}{\partial w_l} = \frac{2 \left( \sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)(E_p - \bar{E}) \right)}{\left( \sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)^2 \right)^2} \left[ \left( \sum_{p=1}^{P_{\max}} \left( \frac{\partial z_{o,p}}{\partial w_l} - \frac{\partial \bar{z}_o}{\partial w_l} \right) (E_p - \bar{E}) \right) \left( \sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)^2 \right) - \left( \sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)(E_p - \bar{E}) \right) \left( \sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o) \left( \frac{\partial z_{o,p}}{\partial w_l} - \frac{\partial \bar{z}_o}{\partial w_l} \right) \right) \right]$$

$$\sqrt{S_3} = \frac{\sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)(E_p - \bar{E})}{\sqrt{\sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)^2}}$$

$$\frac{\partial \sqrt{S_3}}{\partial w_l} = \frac{\text{sign}\left(\sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)(E_p - \bar{E})\right)}{\left(\sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)^2\right)^{3/2}} \left[ \left(\sum_{p=1}^{P_{\max}} \left(\frac{\partial z_{o,p}}{\partial w_l} - \frac{\partial \bar{z}_o}{\partial w_l}\right)(E_p - \bar{E})\right) \left(\sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)^2\right) - \left(\sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o)(E_p - \bar{E})\right) \left(\sum_{p=1}^{P_{\max}} (z_{o,p} - \bar{z}_o) \left(\frac{\partial z_{o,p}}{\partial w_l} - \frac{\partial \bar{z}_o}{\partial w_l}\right)\right) \right]$$

Following are the results applied to the 5 objective functions (Drag, Lift, Surface, Cpmin, Total Lift) of the 2D Model Problem using a TS of 500 points and a VS of 100 points.[1] The best result out of ten tries was chosen and is summarized in table 1. The squared error on the VS divided by the size of the VS, (E/p)val, differs approximatively by 20% between the best and worst result. The best result is highlighted in red while the worst result is in green. It seems like the error obtained with the squared root of S<sub>1</sub>, S<sub>2</sub>, and S<sub>3</sub> is a little bit smaller than for the other functions, and the time (Time) used to train the networks is about half of what is needed for S<sub>CASCOR</sub>, and S<sub>2</sub>. S<sub>1</sub> seems to be the fastest to compute but except for Cpmin leads to a relatively bad error. Conclusions are hard to draw from these results. More extensive testing should be done. However, it seems like  $\sqrt{S_1}$  or  $\sqrt{S_2}$  are good compromise between speed and accuracy.

**Table 1. Results for different Correlation Formula applied to 2d Model Problem**

Formula	Drag			Lift			Surface			Cpmin			Total Lift		
	Time (min)	#HU	(E/p)val	Time (min)	#HU	(E/p)val	Time (min)	#HU	(E/p)val	Time (min)	#HU	(E/p)val	Time (min)	#HU	(E/p)val
S <sub>CASCOR</sub>	404	1	3.34E-05	403	6	1.81E-05	415	4	6.66E-12	403	11	2.51E-05	405	8	1.93E-05
S <sub>1</sub>	162	6	3.12E-05	149	46	1.97E-05	23	52	6.83E-12	109	45	1.28E-05	139	12	1.79E-05
S <sub>2</sub>	429	6	3.11E-05	411	8	1.85E-05	23	59	6.71E-12	370	7	2.46E-05	403	15	1.87E-05
S <sub>3</sub>	283	11	3.22E-05	296	47	1.86E-05	23	15	6.76E-12	206	70	2.07E-05	278	21	1.91E-05
$\sqrt{S_1}$	168	26	2.80E-05	167	15	1.97E-05	27	7	6.72E-12	192	17	2.01E-05	162	17	1.71E-05
$\sqrt{S_2}$	216	4	2.97E-05	213	11	1.95E-05	151	15	5.88E-12	202	3	2.32E-05	220	7	1.88E-05
$\sqrt{S_3}$	193	36	2.92E-05	207	27	1.81E-05	64	35	6.83E-12	170	29	2.39E-05	211	25	1.90E-05

### 1.6. Integration of the NN into the optimization process

As discussed in Sect. 1, the use of the NN approach encompasses three steps,

1. Generation of the TS & VS
2. NN training to obtain a NN “evaluator(s)”

### 3. Optimization as in Fig. 1 with the NN evaluator(s)

Step 1 is done as described in Sect. 2.1.3, using the same approach as the classical one, but one in which the “optimizer” tool is a Latin Hypercube sampling program. For all present applications, the TS is generated using the design/optimization tool, iSIGHT [4]. Step 1 is done for the VS and for the TS. The outputs are 2 datasets which are used in Step 2 for training the network.

The training program is a C++ software in which the Cascade Correlation algorithm with improvements discussed in Sect. 1.5. The outcome of the training is a NN in the form of an executable in which the proper number of hidden units and corresponding weights – found during training – have been implemented. This approach of generating such executable for each network was selected for efficiency during the optimization process. The other approach would have consisted in printing a list of weights and hidden unit information. This information would have to be read at each function call during the optimization process, thus requiring substantial amount of computer input/output (I/O) time when compared with the short CPU time needed for the actual computation. Also printed is error information (w.r.t. TS and VS) for the user to see what kind of accuracy should be expected during optimization.

The thirist step is the same as that of the classical problem, but instead of linking the optimization tool to the CFD code, the tool is linked to the executable generated in Step 2.

## 2. Software user’s manual

The C++ program called NN\_OPT takes one input file (input.dat) and writes 3 types of output files: Unit 6 (main output), weights\_tryxx.data (weights for each try of the NN), and best\_weights.data (weights for the best NN found).

The files weights\_tryxx.data and best weight.data are formatted as C++ subroutines themselves. The subroutine comprising the weights for the NN is to be used as main subroutine (main.C) for the Evaluator program and is compiled with the some of the subroutines from NN\_OPT to create the Evaluator program.

The next section input describes the input file (input.dat) to the NN\_OPT program and section 2.2 describes the various output files. A listing of the program is given in Appendix A.

2.1. Input file

The file input.dat is the main input to the program, the parameters described here allow for choosing the various options available for training the NN as well as specifying the Training, Validation and the eventual Generalization Sets.

DataBase name for the Training Set	Gives path and file name for TS. TS must be in iSIGHT database format (extension .db) or similar.
DataBase name for the Validation Set	Gives path and file name for VS. VS must be in iSIGHT database format (extension .db) or similar.
DataBase name for the Generalization Set	Gives path and file name for GS, 0 if none.
ninputs	Number of inputs for the NN or number of Design variables for optimization
noutputs	Number of outputs for the NN or number of objective functions
TSSize	Size of TS (also called Pmax in eqs.)
VSsize	Size of VS
GSsize	Size of GS
ntrys	Number of identical NNs to construct, the best out of those ntrys NN is then chosen
ncandidates	Number of candidate hidden units to be tried on NN before adding a permanent HU. Must be less than 10.
Correlation formula	Number between 0 and 6 for respectively SCasCor, S1, S2, S3, Sqrt_S1, Sqrt_S2, Sqrt_S3. Determines which formula is to be used for training the candidate HUs.
StopCrit	Value between 0 and 12 for NONE (no stopping criterion), GL1, GL2, GL3, GL5, PQ005, PQ0075, PQ01, PQ02, PQ03, UP1, UP2, UP3. Determines the stopping criterion to be used to stop training. If a null value is chosen, training will continue until the maximum number of hidden units hmax=70 is attained.
DBInputCol	Array of size ninputs containing the number of the columns to be read in the database file as inputs

DBOutputCol	array of size noutputs containing the number of the columns to be read in the database file as outputs
-------------	--

Sample input file

```

****File input.dat for DBs created for 2D-Model problem (8 inputs)*****
#inputs: cols 3 to 10 // outputs drag=col 12, lift= col 13, Surface=col 14, Cpmin=col 15, Total lift (buoyant+dynamic)=col 16
#DataBase name for the Training Set (MAX 80 CHARACTERS) Include path if not in current directory
/disk6/mu/schmitz/Neural_net/CCDoTT_02/TS/Optim33.db
#DataBase name for the Validation Set
/disk6/mu/schmitz/Neural_net/CCDoTT_02/TS/Optim31.db
#DataBase name for the Generalization Set (RM: if none enter 0)
/disk6/mu/schmitz/Neural_net/CCDoTT_02/TS/Optim35.db
#InputArray
#ninputs noutputs TSsize VSsize GSsize ntrys ncandidates Correlation formula( 0 to 6 for
SCasCor,S1,S2,S3,Sqrt_S1,Sqrt_S2,Sqrt_S3)
8 1 486 97 1955 10 7 0
# Stopping Criterion ( 0 to 12, StopCrit={NONE,GL1,GL2,GL3,GL5,PQ005,PQ0075,PQ01,PQ02,PQ03,UP1,UP2,UP3})
0
#DBInputCol: input columns to be read from DB (ninputs)
3 4 5 6 7 8 9 10
#DBOutputCol: output columns to be read from DB (noutputs)
13
    
```

2.2. Output files

2.2.1. *Main output file (unit 6)*

The following describes the information printed in the main output file. This file contains the information on the non-dimensionalization, the squared error  $E$ , the maximum error  $E_{max}$  for each of the ntrys as the well as a summary of the results obtained for the best network found (best out of ntrys). A sample output file follows

The first section of the output file repeats the values read in the input data file. The next section shows the minimum ([MinInput[j]]) and maximum (MaxInput[j]) values found in the Training Set for each input, and the average output (Meantarget[j]).

The following section prints the non-dimensionalized Training Set where inputs have been rescaled between 0 and 1 and outputs have been divided by their average. There are ninputs+noutputs columns and TSsize rows.

Next, is the same information for the non-dimensionalized Validation and Generalization (if it exists) Sets.

Following are the results printed for each try, or each network generated during training.

Printed in columns are respectively, the number of hidden units #HU, the squared error on the training set divided by the number of training set points  $(E/P)_{tr}$ , the maximum Error on the training set  $(EMax)_{tr}$ , the squared error on the VS divided by the number of VS points  $(E/p)_{val}$ , the maximum Error on the VS  $(EMax)_{val}$ , the squared error on the GS divided by the number of GS points  $(E/p)_{gen}$ , the maximum Error on the VS  $(EMax)_{gen}$ , the value of the stopping criterion  $GL_{GL[h]}$ , the value of PQ  $PQ[h]$ , and the value of UPs  $UPs[h]$ .

At the end of each try, a little summary gives the number of hidden units leading to the minimum validation error  $(E/p)_{val}$ , and the time required to complete the training. This is repeated for each network build (ntrys time). At the end of training for all networks, the time required to train the ntrys networks as well as the best network and its corresponding error on the validation set  $(E/p)_{val}$  are printed.

Sample output file

```

Output file generated by NN_OPT_1

Reading input data
#****File input.dat for DBs created for 2D-Model problem (8 inputs)*****
#inputs: cols 3 to 10 // outputs drag=col 12,lift= col 13,Surface=col 14,Cpmin=col 15, Total lift
(buoyant+dynamic)=col 16
Number of Inputs=8 Number of Outputs=1
DataBase file for Training Set: /disk6/mu/schmitz/Neural_net/CCDoTT_02/TS/Optim33.db size=486
DataBase file for Validation Set: /disk6/mu/schmitz/Neural_net/CCDoTT_02/TS/Optim31.db size=97
DataBase file for Generalization Set: /disk6/mu/schmitz/Neural_net/CCDoTT_02/TS/Optim35.db
size=1955
Number of tries to be completed before choosing the best Network: 10
Number of candidate units to try before adding new HU to network (<10 Opt, >10 Max Cov method): 7
Correlation formula 0 to 6 (SCasCor,S1,S2,S3,Sqrt_S1,Sqrt_S2,Sqrt_S3)0
Stopping Criterion(0 to 12,
StopCrit={NONE, GL>1, GL>2, GL>3, GL>5, PQ>0.05, PQ>0.075, PQ>0.1, PQ>0.2, PQ>0.3, UP1, UP2, UP3}): 0
Columns to be read in DB files for input: 3 4 5 6 7 8
9 10
Columns to be read in DB files for output 13
Finished reading input data
-----
- Training Set -
Reading database file: /disk6/mu/schmitz/Neural_net/CCDoTT_02/TS/Optim33.db

*****Non-Dimensionalization*****
For the training set
MinInput[j] = 0 MaxInput[j] = 0.02
MinInput[j] = -0.004 MaxInput[j] = 0.02
MinInput[j] = -0.004 MaxInput[j] = 0.02
MinInput[j] = 0 MaxInput[j] = 0.01
MinInput[j] = -0.02 MaxInput[j] = 0
MinInput[j] = -0.004 MaxInput[j] = 0.02
MinInput[j] = -0.004 MaxInput[j] = 0.02
MinInput[j] = 0 MaxInput[j] = 0.01
MeanTarget[j] = 0.159774
-----
New values for inputs and targets
Inputs are rescaled between [0,1], and outputs are divided by the mean
Following Columns are inputs(j,i) for j=1,n and targets(l,i) for l=1,m with i=1,486
0 0 0.781667 0.270542 0.6794 0.63525 0.607083 0.150312 0.3387
0.807389
1 0.002004 0.473 0.509042 0.2966 0.96192 0.336708 0.97 0.04008
0.920674
2 0.004008 0.975833 0.689583 0.2585 0.1325 0.256542 0.805833 0.1683
0.926307
3 0.00601 0.94 0.965833 0.3768 0.77155 0.060125 0.567167 0.3367
0.952594
4 0.008015 0.392833 0.254542 0.8898 0.99198 0.721667 0.35675 0.1463
0.925055
5 0.01002 0.282583 0.7175 0.7254 0.2705 0.905833 0.6275 0.8958
1.45956
6 0.012025 0.262542 0.93375 0.3507 0.2445 0.91375 0.695417 0.06613
1.0809

```

CCDoTT FY 02 – DI-MCCR-80700 – Deliverable 2

7	0.01403	0.821667	0.36475	0.5792	0.032	0.585417	0.452958	0.5411
1.01643								
8	0.01603	0.76375	0.12025	0.3567	0.3105	0.346708	0.491	0.6673
0.861215								
9	0.018035	0.372792	0.967917	0.7475	0.7595	0.711667	0.336708	0.1403
1.05461								
...								
...								
485	1	1	1	1	0.93385	1	1	1
1.93711								

-----  
 - Validation Set -  
 Reading database file: /disk6/mu/schmitz/Neural\_net/CCDoTT\_02/TS/Optim31.db  
 -----

New values for inputs and targets  
 Inputs are rescaled between [0,1], and outputs are divided by the mean  
 Following Columns are inputs(j,i) for j=1,n and targets(l,i) for l=1,m with i=1,97

0	0	0.414083	0	0.3434	0.3435	0.88875	0.807917	0.1313
0.888128								
1	0.0101	0.727083	0.1313	0.6969	0.8585	0.535292	0.63625	0.7878
1.17603								
2	0.0202	0.979583	0.717083	0.8585	0.5656	0.787917	0.565583	0.8282
1.47208								
3	0.0303	0.1717	0.656667	0.505	0.6161	0.7575	0.979583	0.5454
1.33876								
4	0.0404	0.1515	0.424208	0.2323	0.0705	0.0404167	0.101	1
0.722269								
5	0.0505	0.0909167	0.929167	0.2424	0.8787	0.575708	0.89875	0.3838
1.16414								
...								
...								
96	0.596	1	1	1	0.6161	1	1	1
1.91896								

-----  
 - Generalization Set -  
 Reading database file: /disk6/mu/schmitz/Neural\_net/CCDoTT\_02/TS/Optim35.db  
 -----

New values for inputs and targets  
 Inputs are rescaled between [0,1], and outputs are divided by the mean  
 Following Columns are inputs(j,i) for j=1,n and targets(l,i) for l=1,m with i=1,1955

0	0	0.699167	0.735	0.818	0.288	0.974167	0.695417	0.6969
1.4896								
1	0.0005005	0.530458	0.869583	0.4518	0.6882	0.488917	0.265208	0.7725
1.08027								
2	0.001001	0.219167	0.786667	0.9496	0.147	0.28275	0.613333	0.858
1.34565								
3	0.002002	0.0390417	0.83125	0.9581	0.53555	0.220667	0.219167	0.4077
0.956975								
4	0.0025025	0.923333	0.877083	0.3067	0.55455	0.716667	0.257208	0.791
1.12221								
5	0.003003	0.133112	0.627083	0.4122	0.9044	0.146121	0.362292	0.4973
0.806137								
...								
...								
1954	0.985	1	1	1	1	1	1	1
1.93523								

\*\*\*\*\*Try number 1\*\*\*\*\*  
 E/Pmax After Optimization- no HU: 2.87728e-05

..... #HU	(E/Pmax)tr	GL[h]	(EMax)tr	UPs[h]	(E/p)val	(EMax)val	(E/p)gen
(EMax)gen			PQ[h]				
..... 1	2.14338e-05		0.0225970		2.73458e-05	0.0203295	2.34744e-05
0.0247901	0.00000	0.00000			0		
..... 2	1.87312e-05		0.0193056		2.31156e-05	0.0161325	2.07863e-05
0.0271225	0.00000	0.00000			0		
..... 3	1.78384e-05		0.0192007		2.34628e-05	0.0181199	1.99869e-05
0.0255502	1.50186	0.00000			0		
..... 4	1.63920e-05		0.0173375		2.29731e-05	0.0196657	2.03174e-05
0.0240866	0.00000	0.00000			0		
..... 5	1.54422e-05		0.0154955		2.19982e-05	0.0202779	2.15115e-05
0.0264148	0.00000	0.00000			0		
..... 6	1.48063e-05		0.0156305		2.30018e-05	0.0199623	2.18623e-05
0.0277272	4.56216	0.00000			0		
..... 7	1.42903e-05		0.0165586		2.33096e-05	0.0211070	2.18068e-05
0.0263231	5.96160	0.00000			0		
..... 8	1.36861e-05		0.0164403		2.43941e-05	0.0234750	2.25663e-05
0.0246117	10.8913	0.00000			0		
..... 9	1.29327e-05		0.0163794		2.40383e-05	0.0227942	2.29819e-05
0.0233866	9.27435	0.00000			0		
..... 10	1.24346e-05		0.0168899		2.46604e-05	0.0245917	2.37044e-05
0.0246065	12.1019	0.125883			1		
..... 70	0						
..... 70	1.62640e-08		0.00116031		3.21549e-05	0.0188888	3.72486e-05
0.0286285	46.1709	0.120517			1		

For this try, the optimum result was found for h=5 hidden units  
 Best mean squared error Eval(tmin)= 2.19982e-05  
 Calculation Time : 39 min 33 sec  
 Saving weight file in weights/weights\_try\_XX.data

```

*****Try number 2*****
E/Pmax After Optimization- no HU: 2.75729e-05

..... #HU (E/Pmax)tr      (EMax)tr      (E/p)val      (EMax)val      (E/p)gen
(EMax)gen
..... 1      2.15808e-05      0.0236005      2.78362e-05      0.0207328      2.38600e-05
0.0235745      0.00000      0.00000      0
..... 2      1.86741e-05      0.0200202      2.29628e-05      0.0165717      2.14191e-05
0.0262292      0.00000      0.00000      0
..... 3      1.72266e-05      0.0181260      2.31206e-05      0.0184325      2.19550e-05
0.0281701      0.686941      0.00000      0
..... 4      1.62708e-05      0.0188199      2.45549e-05      0.0204431      2.23136e-05
0.0290525      6.93328      0.00000      0
..... 5      1.52709e-05      0.0168774      2.36230e-05      0.0226784      2.26148e-05
0.0308795      2.87477      0.0173261      0
...
...
..... 70      1.20347e-08      0.000582638      2.90318e-05      0.0240310      3.37846e-05
0.0281474      32.7195      0.0803459      0

For this try, the optimum result was found for h=10 hidden units
Best mean squared error Eval(tmin)= 2.18746e-05
Calculation Time      : 41 min 23 sec
Saving weight file in weights/weights_try_XX.data
*****Try number 3*****
...
...
*****Try number 10*****
E/Pmax After Optimization- no HU: 2.75994e-05

..... #HU (E/Pmax)tr      (EMax)tr      (E/p)val      (EMax)val      (E/p)gen
(EMax)gen
..... 1      2.17477e-05      0.0240674      2.83740e-05      0.0211502      2.42877e-05
0.0234197      0.00000      0.00000      0
..... 2      1.91375e-05      0.0197835      2.39364e-05      0.0161788      2.20001e-05
0.0215431      0.00000      0.00000      0
..... 3      1.75603e-05      0.0177309      2.31084e-05      0.0161258      2.21745e-05
0.0238413      0.00000      0.00000      0
..... 4      1.61779e-05      0.0175574      2.27406e-05      0.0175684      2.13327e-05
0.0237191      0.00000      0.00000      0
..... 5      1.52258e-05      0.0179309      2.21718e-05      0.0191528      2.16977e-05
0.0252443      0.00000      0.00000      0
...
...
..... 70      0      0.000641451      3.84180e-05      0.0227461      3.63721e-05
0.0286027      76.9552      0.191996      4

For this try, the optimum result was found for h=8 hidden units
Best mean squarred error Eval(tmin)= 2.17106e-05
Calculation Time      : 41 min 30 sec
Saving weight file in weights/weights_try_XX.data
Calculation Time for the 10 trys      : 403 min 28 sec
Best Result for try 3 ,Best Error = 1.80512e-05
Best Weight file found in ./weights/best_weights.data

```

### 2.2.2. Weight data file (weights\_try\_XX.data and best\_weights.data)

For each network constructed or each try, the weights corresponding to the network with the minimum validation error are written in weights\_try\_XX.data where XX is the try number. This file is formatted as a C++ program and serves as the main subroutine for the evaluator code. The file best\_weights.data corresponds to the network that leads to the best error on the VS out of ntrys. For example, if try #3 is the best network, then best\_weights.data and weights\_try\_03.data will be the same file. This was done so that the user does not need to look at the main output file to find which weights data file he must use in his optimization. This subroutine along with the other subroutines of NN\_OPT can be compiled together to create the Evaluator program.

The input-to-output weights are saved in a vector,  $v_{ij}$ , and the inputs-to-hidden-unit weights are saved in a vector,  $w_{ij}$ , as shown in the sample file below.

The Evaluator, once compiled, is capable of reading an input file `dv.data` which contains the design variables to evaluate and prints out the output to the neural network.

Sample weights data file

```
#include "NN.H"
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
// #include <time.h>
/*
Evaluator (NN1_Evaluator)
Calculation of the NN output, an input vector being given
The code looks for a dv.data file in the current folder, this
file contains the values for the DV.
PS: INPUTS-OUTPUTS ARE SUPPOSED DIMENSIONALIZED */

int main()
{
    int i;
    int j;
    double aux1,aux2;
    NN* Network;
    // Neural Network Generated by NN_OPT_1
    // Best Mean squared Error = 1.8051229873897787e-05 for try number 3
    const int nl=8;
    const int ml=1;
    const int hl=6;
    // MinInput[j] & MaxInput[j] for j=1,n
    double MinInput[nl]= {0,-0.0040000000000000001,-0.004000000000000001,0,-0.02,-0.004000000000000001,-
0.004000000000000001,0};
    double MaxInput[nl]= {0.02,0.02,0.02,0.01,0,0.02,0.02,0.01};
    // MeanTarget[j] for j=1,m
    double MeanTarget[ml] = {0.15977427983539072};
    double
    vij[ml*(nl+hl+1)]= {0.015197799563602054,0.044234941402215365,0.27661594929751776,0.35892377081308724,0.0216745815317897
46,0.3011856596903785,0.48005152831000325,0.41559781790208583,0.029300117654549498,0.012083128865297495,-
0.013724105693491207,0.0078422151426914104,-0.004879984951905279,-0.0079812813099029607,0.0071903665759384255};
    double wij[hl*(nl+hl)]= {-2.5017233664398217,1.0007690473069926,10,9.9996810708647335,-
3.9647692188346242,3.181563425189561,7.461124200978726,4.3759185013226043,-9.5280675762031937,0,0,0,0,0
,-1.2296481836682758,-0.92714831420241406,0.71530049716709532,-
4.9163493487291614,0.9351576156639283,10.000000000000002,9.999999999999998,-0.61244854544229643,-6.8082331523650286,-
4.4994620624464083,0,0,0,0,0
,0.41724101942901221,9.2306495680340515,9.999999999999998,-7.2243788491917487,-2.3917482950017299,-
1.519109174412453,-6.7083807330304115,-10,0.76958075246808444,-2.7406873223440886,6.6223610459803686,0,0,0
,-2.7699168418671598,-8.7531223547793289,-10,-3.0671639002846365,-6.7088894373450803,0.95062257041599973,-
0.057724732766451514,-6.130119888750019,9.9594558786709459,0.53282912087593892,-
2.5132985392836931,5.8242826351654085,0,0
,-3.9132198149245006,-10,3.7479900279768974,2.3262756905540694,1.7388272906472357,-0.30192372522725019,-10,-
5.3865853741610037,8.101515110646794,-2.4836087713652395,-5.7395568560866108,-2.5402040417081748,10,0
,-5.3791262995428157,-5.9257961829767929,-7.7132244648789126,-10,3.138554652860047,-
2.2372199315006829,9.851949979517741,1.9807299463564656,9.2346039778454827,5.2067879600985005,-
4.6473794506033066,2.0739813459741572,9.3691825720179907,-1.2405487687207595
};
    cout.setf(ios::fixed);
    char buffer[1000];
    Vector x(nl);
    Vector y(ml);
    cout<<"    output= ";
    ifstream fin("dv.data");
    fin.getline(buffer,1000);// this throws away the comment line;
    for(i=0;i<nl;i++)
    {
        fin>>aux1;
        //non-dimensionalize the inputs
        aux2=(aux1-MinInput[i])/(MaxInput[i]- MinInput[i]);
        x.initialize_element(i,aux2);
    }
    fin.ignore(80,'\n'); //this throws away the endl character
    Network= new NN(ml, nl, hl,vij,wij);
    y=Network->Evaluate3(x);
    for(j=0;j<ml;j++) cout<<y.element(j)*MeanTarget[j]<<endl;
    cout<<"    MinInput[j]= {"<<MinInput[0];
    for(j=1;j<nl;j++) cout<<"<<MinInput[j];
    cout<<"<<endl;
    cout<<"    MaxInput[j]= {"<<MaxInput[0];
    for(j=1;j<nl;j++) cout<<"<<MaxInput[j];
    cout<<"<<endl;
    cout<<"    MeanTarget= "<<MeanTarget[0];
```

```

for(j=1; j<m1; j++) cout<<"      "<<MeanTarget[j];
cout<<endl;
delete Network;
}

```

### 3. Summary/Conclusion

An optimization method based on neural networks has been developed for application to hull design. This work is an extension of previous CCDoTT work. Substantial improvements have been made to the method and its applicability has been extended to three dimensional complex configurations such as Pacific Marine underwater hulls forms. The method is set up for hydrodynamic shape optimization with the objective of minimizing drag and or optimizing lift to drag ratios of two and three dimensional configurations. Validation of this method is reported in the accompanying Software Test Report [11]. Its application to a complex three dimensional configuration is reported in reports on deliverable 3 of this task [12] and [13].

### 4. Glossary

#### Symbols

$n$ : number of inputs

$m$ : number of outputs

$h$ : number of hidden units

$k$ : training length strip

$P_{max}$ : number of points in training set

$\mathbf{x}_p$ :  $p^{\text{th}}$  vector input of the training set

$\mathbf{t}_p=f(\mathbf{x}_p)$ :  $p^{\text{th}}$  vector target

$\mathbf{y}_p$ :  $p^{\text{th}}$  vector output from network

$\mathbf{W}=[w^h_j]$ , the weights between inputs and each hidden units

$\frac{\partial}{\partial x}$  Partial Derivatives

$C_p = \frac{P - P_{atm}}{q_\infty} = 1 - \left(\frac{V}{V_\infty}\right)^2 - \frac{2}{F_r^2} \left(\frac{z}{L}\right)$ : pressure coefficient

$C_{pmin}$  Minimum Pressure Coefficient

**Subscripts/Superscripts**

i: input number (i=1,n)

j: output number (j=1,m)

p: point number in Training Set ( p=1,Pmax)

tr: Training

va: Validation

gen: Generalization

min: minimum

max: maximum

**Acronyms**

CFD: Computational Fluid Dynamics

NN Neural Networks

D.V Design Variables

CPU Central Processing Unit

ONN Ordered Neural Networks

TS Training Set

VS Validation Set

HU Hidden Unit

SQP Sequential Quadratic Programming

GA: Genetic Algorithm

GS: Generalization Set

**Acknowledgement:**

The CCDoTT project is a collaborative effort involving several faculty and students at California State University, Long Beach. Staffs primarily responsible for this portion of

the work are Dr. Eric Besnard, Associate Professor and Adeline Schmitz, Research Associate in the Mechanical and Aerospace Engineering Department.

## References

1. Hamid Hefazi, et al. “CFD Desing Tool Development and Validation Part I” CCDoTT FY01 Task 2.14 report , Center for the Commercial Deployment of Transportation Technologies, Long Beach, CA, Januray 2003. Available on line at [www.ccdot.org](http://www.ccdot.org)
2. S.E. Fahlman and C. Lebiere. “The Cascade-Correlation Learning Architecture,” Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA, 1990.
3. M. L. Stein, *Large Sample Properties of Simulations using Latin Hypercube Sampling*, Technometrics, 29 (2): 143-151, 1987.
4. Engineous Software Inc., iSIGHT software, <http://www.engineous.com/>
5. Mikko Lehtokangas, “Fast Initialization for Cascade-Correlation Learning,” IEEE Transactions on Neural Networks, Vol. 10, n° 2, March 1999.
6. Jani J.T. Lahnajärvi et al., “Evaluation of Constructive Neural Networks with Cascaded Architectures,” *Neurocomputing* 48, pp 573-607, 2002.
7. Vanderplaats, Miura & Associates, Inc., "DOT Users Manual, Version 4.10," VMA Engineering, ©1994.
8. TY. Kwok and DY. Yeung “Theoretical Analysis of constructive Neural Networks,” Technical Report HKUST-CS-93-12, Hong Kong University of Science and Technology, 1993.
9. TY. Kwok and DY. Yeung “Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems,” IEEE transactions on Neural Networks, Vol. 8, Issue 3, pp 630-645, May 1997.
10. TY. Kwok and DY. Yeung “Objective Function for Training New Hidden Units in Constructive Neural Networks,” IEEE transactions on Neural Networks, Vol. 8, Issue 5, pp 1131-1148, Sep 1997.
11. Hamid Hefazi, et al. “Software Test Report, Optimization Tool Development Based on Neural Networks, Application, Validation, Deliverable 2 DI-IPSC 81440 A” report Center for the Commercial Deployment of Transportation Technologies, Long Beach, CA, Sept. 2003.
12. Hamid Hefazi *et al.*, “Computer Software Product End Items, CAD-Based CFD tool Devlopment and Application, Deliverable 3, DI-MCCR-80700”, report Center for the Commercial Deployment of Transportation Technologies, Long Beach, CA, Sept. 2003.
13. Hamid Hefazi *et al.*, “Software Test Report, CAD-Based CFD tool Development and Application, Deliverable 3, DI-IPSC-81440A” report Center for the

Commercial Deployment of Transportation Technologies, Long Beach, CA, Sept. 2003.

## Appendix A: Source code for NN\_OPT

Following is the source code written in C++ for NN\_OPT. This code is linked with DOT software [7] subroutines ddot1.f, ddot2.f, ddot3.f, ddot4.f, ddot5.f, ddot6.f written in Fortran 77. These subroutines are property of VMA Engineering, and are not listed here.

### DataSet.C

```
// File DataSet.C
#include "DataSet.H"

/* Class implementation of Datasets, can be Training, Validation and
   Generalization Sets
   used to manage a Set
   Class used bu the class NN    */

/*   Constructor from 2 existing Matrix    */

DataSet::DataSet(Matrix &x,Matrix &y)
{
    inputs=&x;
    targets=&y;
}

/*   Default Constructor    */

DataSet::DataSet()
{
    inputs=new Matrix();
    targets=new Matrix();
}

/*   Copy Constructor    */

DataSet::DataSet(DataSet & tr)
{
    inputs=&(tr.getInputs());
    targets=&(tr.getTargets());
}

/*   Reading of DB File generated by iSIGHT (Database File)
   the num indicates the parameter to read
   This method is for a one target Set    */

void DataSet :: ReadFileDB(char *name,int num)
{
    int i,j;
    int k;
    double aux;
    char buffer[256];

    ifstream fin(name);
    fin.getline(buffer,256);
```

```

fin.getline(buffer,256);
for(j=0;j<inputs->nb_columns();j++)
{
    fin>>aux;
    fin>>aux;
    for(k=0;k<inputs->nb_rows();k++)
    {
        fin>>aux;
//        cout<<" d "<<aux;
        inputs->initialize_element(k,j,aux);
    }
    i=0;
//    cout<<endl;
    while(i<num) { fin>>buffer ; /*cout<<" s "<<buffer; */ i++;
}

    i++;

//    cout<<endl;
    fin>>aux;
//    cout.setf(ios::showpoint);
    cout<<"v="<<aux<<endl;
    targets->initialize_element(0,j,aux);
    while(i<18) { fin>>buffer ;/* cout<<" s "<<buffer; */ i++;
}

//    cout<<endl;
}
fin.close();
//    for(j=0;j<200;j++)
//    {
//        for(k=0;k<8;k++)
//            cout<<inputs.element(k,j)<<" ";
//        cout<<endl;
//    }
}
//Function added Feb25 03
/*    Reading of DB File generated by iSIGHT (Database File)
    this method allows for specifying the columns to read in the
Database
    for the inputs and outputs. Assumed there are n inputs and m
outputs
    PS: assumed that DBInputCol[n-1]<DBoutputCol[0] :the last column
for input is before the first output column
*/

void DataSet :: ReadFileDB(char *name,int DBInputCol[],int
DBOutputCol[])
{
    int i,j;
    int k,l;
    int il;
    double aux;
    char buffer[5000];

    ifstream fin(name,ios::nocreate);
    if (!fin)
    {

```

```

        cout<<"Error file "<<name<<" not found/n";
        exit(0);
    }
    fin.getline(buffer,5000);
    fin.getline(buffer,5000);
    il=inputs->nb_rows()-1;
//    cout<<"DBCols\t"<<DBInputCol[il]<<"\t"<<DBOutputCol[0]<<endl;
    cout<<"Reading database file:\t"<<name<<endl;
//    cout.setf(ios::showpoint);
//    cout.setf(ios::fixed);
    if (DBInputCol[il]>=DBOutputCol[0])
    {
        cout<<"*****Error in read DBase
file*****\n";
        cout<<"The Design Variables columns (input columns) are
written after the Objective (target column)\n";
        exit(0);
    }
    for(j=0;j<inputs->nb_columns();j++)
    {
        i=1;
        for(k=0;k<inputs->nb_rows();k++)
        {
            while(i<DBInputCol[k]) { fin>>buffer ; /*cout<<"
s="<<buffer;*/ i++; }
            i++;
            fin>>aux;//cout<<" input= "<<aux;
            inputs->initialize_element(k,j,aux);
        }
        for(l=0;l<targets->nb_rows();l++)
        {
            while(i<DBOutputCol[l]) { fin>>buffer ; /*cout<<"
s="<<buffer; */ i++; }
            i++;
            fin>>aux;//cout<<" output = "<<aux<<endl;
            targets->initialize_element(l,j,aux);
        }
        //while(i<25){ fin>>buffer ; cout<<" s"<<i<<"="<<buffer;
i++; }
        fin.ignore(5000,'\n');//ignores the rest of the line (up to
5000 characters) until it finds a EOL character
    }
    fin.close();
}
//endadd feb25 03
/*    Reading of a DB File generated by iSIGHT (Database File)
    3 parameters are read : CpMin, Lift and Drag
    This method is for a three targets Set    */

void DataSet :: ReadFileDB(char *name)
{
    int i,j;
    int k;
    double aux;
    char buffer[256];

```

```

ifstream fin(name);
fin.getline(buffer,256);
fin.getline(buffer,256);
for(j=0;j<inputs->nb_columns();j++)
{
    fin>>aux;
    fin>>aux;
    for(k=0;k<inputs->nb_rows();k++)
    {
        fin>>aux;
        inputs->initialize_element(k,j,aux);
    }
    fin>>buffer ;
    fin>>aux ;
    targets->initialize_element(1,j,aux);
    fin>>buffer ;
    fin>>buffer ;
    fin>>aux ;
    targets->initialize_element(2,j,aux);
    fin>>aux ;
    targets->initialize_element(0,j,aux);
    i=6;
    while(i<18) { fin>>buffer ;/* cout<<" s "<<buffer; */ i++;
}
//    cout<<endl;
}
fin.close();
}

/*    Constructor using the Latin hypercubes Method
iSight is no longer used
the bound matrix contains the bounds for the DV
This method needs to be changed is the CFD code is changed */

DataSet :: DataSet(Matrix &x, Matrix &y,Matrix &bound)
{
    int i,j;
    char buffer[2550];
    char aux1[8000];
    char aux2[8000];
    double surf;
    double temp,cp,cpmin,lift,drag,yy;

    ofstream *fout;
    ifstream *fin;

    inputs=&x;
    targets=&y;
    cout<<"Generating a DataSet of Size : "<<inputs->nb_columns()<<"\n";
    GenerateInputs(bound);
    for(i=0;i<inputs->nb_columns();i++)
    {
        cpmin=10;

```

```

//      system("rm dv.data");
      fout=new ofstream("dv.data");
      for(j=0;j<inputs->nb_rows();j++)
        (*fout)<<inputs->element(j,i)<<"\n";
      fout->close();
      cout<<"Shape Generation\n";
      system("ShapeGenerator > ShapeOut");
      delete fout;
      fin=new ifstream("ShapeOut");
      fin->getline(buffer,255);
      fin->getline(buffer,255);
      fin->getline(buffer,255);
      fin->getline(buffer,255);
      fin->getline(buffer,255);

      (*fin)>>aux1;
      (*fin)>>aux1;
      (*fin)>>surf;
      fin->close();
      delete fin;
      cout<<"CFD Calculation : Hlift\n";
      cout.flush();
      system("hlift < hlift.temp > hlift.out");
      fin=new ifstream("hlift.out");
      strcpy(aux1,"jk");
      strcpy(aux2,"jk");
      while((strcmp(aux2,"N")!=0)|| (strcmp(aux1,"X")!=0))
      {
        strcpy(aux2,aux1);
        (*fin)>>aux1;
      }
      fin->getline(buffer,255);
      for(j=0;j<100;j++)
      {
        (*fin)>>temp;
        (*fin)>>temp;
        (*fin)>>yy;
        (*fin)>>temp;
        (*fin)>>temp;
        (*fin)>>cp;
        fin->getline(buffer,255);
//      cout<<j<<" "<<yy<<" "<<cp<<endl;
        cp=cp-2/2.632/2.632*yy;
        if(cp<cpmin) cpmin=cp;
      }
      while(strcmp(aux1,"CL")!=0)
        (*fin)>>aux1;
      fin->getline(buffer,255);
      fin->getline(buffer,255);
      (*fin)>>aux1;
      (*fin)>>lift;
      (*fin)>>drag;
      fin->close();
      delete fin;
      lift=lift/2*1025*20.6*20.6*6.1959+surf*9.81*1025;
      cout<<i<<" Lift = "<<lift<<" Drag = "<<drag<<" CPMin =
"<<cpmin<<endl;

```

```

        targets->initialize_element(0,i,lift);
    }
}
//cad added training set for the test function: (n inputs, 1 output)
test function f(x)=SUMa(1/(1+100*(x-xa)**2)

DataSet :: DataSet(Matrix &x, Matrix &y, Matrix &bound, Matrix &xa)
{
/* xa(amax,n) location of maxima
   x(n,Pmax) inputs, y(m,Pmax) targets*/
   int i,p,a;
   double f_of_x,dist;

   inputs=&x;
   targets=&y;
   cout<< "in TRAININGSET inputs"<<inputs<<"\n";
   cout<< " targets"<<targets<<"\n";
   cout<< "Test Function - number of inputs: " <<inputs->
>nb_rows()<<"\n"; /*n*/
       cout<< " number of maximas: " <<xa.nb_columns()<<"\n"; /*amax*/
       cout<<"Generating a DataSet of Size : " <<inputs->
>nb_columns()<<"\n"; /*Pmax*/
       cout<<" inputs(i,p) for i=1,n and target " <<"\n";
       GenerateInputs(bound);
       for(p=0;p<inputs->nb_columns();p++)
       {
           f_of_x=0.;
           for ( a = 0; a <xa.nb_columns() ; a++)
           {
               dist=0.0;
               for ( i = 0; i <inputs->nb_rows() ; i++) dist +=
pow((inputs->element(i,p)-xa.element(i,a)),2);
               f_of_x += 1/(1+100*dist);
           }
           targets->initialize_element(0,p,f_of_x);
           cout<<" p= ";cout.width(5);cout<<p;
           for ( i = 0; i <inputs->nb_rows() ; i++)
{cout.width(11);cout<<inputs->element(i,p)<<" ";}
           cout<<" " << targets->element(0,p)<<"\n";
       }
}

/* Latin HyperCubes Method */

void DataSet :: GenerateInputs(Matrix &b)
{
   int i,j;
   double value;

   Matrix Permu(2,inputs->nb_columns());;

   cout<<"Latin Hypercubes Method\n";
   for(j=0;j<inputs->nb_rows();j++)
   {
       GeneratePermu(&Permu);
       for(i=0;i<inputs->nb_columns();i++)

```

```

        {
            value=(Permu.element(0,i)-random()/(pow(2,31)-
1))/inputs->nb_columns();
            value=value*(b.element(j,1)-
b.element(j,0))+b.element(j,0);
            inputs->initialize_element(j,i,value);
            //cout<<"j,i,inputs->element(j,i)=""<<j<<"    "<<i<<"
"<<inputs->element(j,i)<<"\n";
        }
    }
}

void DataSet :: GeneratePermu(Matrix *M)
{
    int n=M->nb_columns();
    int i,j;
    double sav;

    for(i=0;i<n;i++)
    {
        M->initialize_element(0,i,i+1);
        M->initialize_element(1,i,random()/(pow(2,31)-1));
    }
    for(i=0;i<n;i++)
        for(j=i+1;j<n;j++)
            if(M->element(1,j)>M->element(1,i))
            {
                sav=M->element(0,j);
                M->initialize_element(0,j,M->element(0,i));
                M->initialize_element(0,i,sav);
                sav=M->element(1,j);
                M->initialize_element(1,j,M->element(1,i));
                M->initialize_element(1,i,sav);
            }
}

void DataSet :: CalculMinMaxMean(Vector &MinInput,Vector
&MaxInput,Vector &MeanTarget)
{
    int i,j;
    double temp;

    // cout<< "in CALCULMINMAXMEAN inputs ""<<inputs<<"\n";
    // cout<< " targets ""<<targets<<"\n";

    int n=inputs->nb_rows();
    int m=targets->nb_rows();
    int Pmax=targets->nb_columns();
    // MinInput=new Vector(n);
    // MaxInput=new Vector(n);
    // MeanTarget=new Vector(m);

    for(i=0;i<n;i++){MinInput.initialize_element(i,+1e99);MaxInput.in
italize_element(i,-1e99);}
    for(i=0;i<m;i++) MeanTarget.initialize_element(i,0);
    for(i=0;i<Pmax;i++)

```

```

    {
        for(j=0;j<n;j++)
        {
            if (MinInput.element(j)>inputs->element(j,i))
MinInput.initialize_element(j,inputs->element(j,i));
            if (MaxInput.element(j)<inputs->element(j,i))
MaxInput.initialize_element(j,inputs->element(j,i));
        }
        for(j=0;j<m;j++)
        {
            temp=MeanTarget.element(j)+targets->element(j,i);
            MeanTarget.initialize_element(j,temp);
        }
    }
    for(j=0;j<m;j++) MeanTarget.initialize_element(j,temp/Pmax);
    cout<<"\n\n*****Non-
Dimensionalization*****"
    *"<<endl;
    cout<<"For the training set "<<"\n";
    for(j=0;j<n;j++)
    {
        cout<<"\t\tMinInput[j] =";
        cout.width(11); cout<<MinInput.element(j)<<" MaxInput[j]
= ";
        cout.width(11); cout<<MaxInput.element(j)<<"\n";
    }
    for(j=0;j<m;j++)
    {
        cout<<"\t\tMeanTarget[j] =";
        cout.width(11); cout<<MeanTarget.element(j)<<"\n";
    }
}
void DataSet :: NDimensionalize(Vector MinInput,Vector MaxInput,Vector
MeanTarget)
{
    int i;
    int j;
    double temp = 0.0;
    int n=inputs->nb_rows();
    int m=targets->nb_rows();
    int Pmax=targets->nb_columns();
    cout<<"-----\n";

    cout<<" New values for inputs and targets\n";
    cout<<" Inputs are rescaled between [0,1], and outputs are
divided by the mean\n";
    cout<<" Following Columns are inputs(j,i) for j=1,n and
targets(1,i) for l=1,m with i=1,"<<Pmax<<"\n";
    // cout<<"in N-DIMENSIONALIZATION inputs\t"<<inputs<<"\n";
    // cout<<" targets\t"<<targets<<"\n";
    for(i=0;i<Pmax;i++)
    {
        cout.width(5);cout<<i<<" ";
        for(j=0;j<n;j++)
        {
            temp =(inputs->element(j,i)-
MinInput.element(j))/(MaxInput.element(j)- MinInput.element(j));

```

```

        inputs->initialize_element(j,i,temp);
        cout.width(11);cout<<inputs->element(j,i)<<"  ";
    }

    for(j=0;j<m;j++)
    {
        temp = targets->element(j,i)/MeanTarget.element(j);
        targets->initialize_element(j,i,temp);
        cout.width(11);cout<<targets->element(j,i);
    }
    cout<<"\n";
}
}

```

## **DataSet.H**

```

//File DataSet.H
/*      - Copyright by Adeline Schmitz, 2003- All rights Reserved-
      */
#ifndef DataSet_H
#define DataSet_H
#include <iostream.h>
#include <math.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include "Matrix.H"
#include <string.h>
/* Class implementation of Datasets, can be Training, Validation and
Generalization Sets*/
class DataSet
{
    public:
        DataSet(Matrix &inputs,Matrix &targets);
        DataSet(Matrix &x, Matrix &y,Matrix &bound);
        DataSet(Matrix &x, Matrix &y, Matrix &bound, Matrix &xa);
        DataSet(DataSet &);
        DataSet();
        void setInputs(Matrix &in) {inputs=&in;}
        void setTargets(Matrix &ta) {targets=&ta;}
        Matrix &getInputs() {return *inputs;}
        Matrix &getTargets() {return *targets;}
        void ReadFileDB(char *name,int num);
//Added Feb25 03
        void ReadFileDB(char *name,int DBInputCol[],int
DBOutputCol[]);
//endAdd
        void ReadFileDB(char *name);
        //void ReadFileDOE(char *name,int num);
        //void Add(DataSet &tr);
        void CalculMinMaxMean(Vector &MinInput,Vector
&MaxInput,Vector &MeanTarget);
        void NDimensionalize(Vector MinInput,Vector MaxInput,Vector
MeanTarget);
    private:

```

```

    Matrix *inputs;
    Matrix *targets;
    void GenerateInputs(Matrix &b);
    void GeneratePermu(Matrix *m);

};

#endif

```

## **Main.C**

```

// Main subroutine for NN_OPT main.C
/* - Copyright by Adeline Schmitz, 2003- All rights Reserved-
   */
#include "NN.H"
#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <sys/types.h>
#include <time.h>
//
void ReadInputData(char *DBforTS,char *DBforVS,char *DBforGS, int
InputArray[],int &Method, int DBInputCol[],int DBOutputCol[])
// This function reads a input.dat file
//-----
-
// DBforTS : stores the DataBase name for the Training Set (MAX 79
CHARACTERS)
// DBforVS : stores the DataBase name for Validation Set,
// DBforGS is for Generalization Set
// InputArray[]:0- number of inputs
//          1- number of outputs
//          2- Training Set Size
//          3- VS size
//          4- GS size
//          5- nbt nb of trys (how many networks do we want to build)
//          6- ncandidate: How many candidate neurons do we want to try
when maximizing the covariance- 1 to 10 - uses optimization for each
neuron
//
//          more than 10 uses max covariance method ( random initialization
of candidate, and pick the one with best Covariance)
//
//          ( For best results use at least 100)
//          7-Correlation formula to use for building the NN 0 to 6
(SCasCor,S1,S2,S3,Sqrt_S1,Sqrt_S2,Sqrt_S3)
// Method: Which criterion should be used to stop training
//          0 to 12      enum StopCrit
{NONE,GL1,GL2,GL3,GL5,PQ005,PQ0075,PQ01,PQ02,PQ03,UP1,UP2,UP3}; //used
for stopping criteria
//          defined in NN_Constructor_SC5
// DBInputCol: array stores the columns to be read in DataBase for the
inputs (Design Variables) size=nb inputs<50

```

```
// DBOutputCol: array stores the columns to be read in DataBase for the
outputs ( Objective function(s) ) size=nb outputs<10
//-----
-----
-
{
    int i;
    char buffer[256],title1[256],title2[256];
    ifstream fin("input.dat",ios::nocreate);
    if (!fin)
    {
        cout<< "Error, file input.dat does not exist in
directory\n";
        exit(0);
    }
    fin.getline(title1,256); //LINES 1 to 2 ARE title LINES
    fin.getline(title2,256);
    fin.getline(buffer,256);//line 3 is comment
// read database name for Training set
    fin>>DBforTS;// this will read everything up to the next blank
    fin.ignore(80,'\n'); //this throws away the endline character
// read database name for Validation set
    fin.getline(buffer,256);// this throws away the comment line
    fin>>DBforVS;
    fin.ignore(80,'\n');
// read database name for Generalization set
    fin.getline(buffer,256);
    fin>>DBforGS;
    fin.ignore(80,'\n');
//read input array
    fin.getline(buffer,256);
    fin.getline(buffer,256);
    for (i=0;i<8;i++)
    {
        fin>>InputArray[i];
    }
    fin.ignore(256,'\n');
//read method
    fin.getline(buffer,256);
    fin>>Method;
    fin.ignore(256,'\n');
//read DBInputCol
    fin.getline(buffer,256);
    if (InputArray[0]>50)
    {
        cout<<"Error ninputs exceeds limit (<50), ninputs=
"<<InputArray[0];
        exit(0);
    }
    for (i=0;i<InputArray[0];i++)
    {
        fin>>DBInputCol[i];
    }
    fin.ignore(256,'\n');
//read DBOutputCol
    fin.getline(buffer,256);
    if (InputArray[1]>50)
```

```

    {
        cout<<"Error noutputs exceeds limit (<50), noutputs="
"<<InputArray[1];
        exit(0);
    }
    for (i=0;i<InputArray[1];i++)
    {
        fin>>DBOutputCol[i];
    }
    fin.ignore(256, '\n');
// to be continued.....
    fin.close();
    cout<<"\tOutput file generated by NN_OPT_1\n";
    cout<<"\tReading input data\n";
    cout<<title1<<endl;
    cout<<title2<<endl;
    if (*DBforGS=='0')
    {
        InputArray[4]=0;
        cout<<"\tRem: no Generalization set, GsSize set to 0\n";
    }
    cout<<"\tNumber of Inputs="<<InputArray[0]<<" \tNumber of
Outputs="<<InputArray[1]<<endl;
    cout<<"\tDataBase file for Training Set:      \t"<<DBforTS<<"\t
size="<<InputArray[2]<<endl;
    cout<<"\tDataBase file for Validation Set:    \t"<<DBforVS<<"\t
size="<<InputArray[3]<<endl;
    cout<<"\tDataBase file for Generalization Set:\t"<<DBforGS<<"\t
size="<<InputArray[4]<<endl;
    cout<<"\tNumber of tries to be completed before choosing the best
Network:\t"<<InputArray[5]<<endl;
    if (InputArray[6]>10) cout<<"warning, will use max covariance
method, currently weight initialization not appropriate"<<endl;
    cout<<"\tNumber of candidate units to try before adding new HU to
network (<10 Opt, >10 Max Cov method):\t"<<InputArray[6]<<endl;
    cout<<"\tCorrelation formula 0 to 6
(SCasCor,S1,S2,S3,Sqrt_S1,Sqrt_S2,Sqrt_S3)"<< InputArray[7]<<endl;

    cout<<"\tStopping Criterion(0 to 12,
StopCrit={NONE, GL>1, GL>2, GL>3, GL>5, PQ>0.05, PQ>0.075, PQ>0.1, PQ>0.2, PQ>0.
3, UP1, UP2, UP3}):\t"<<Method<<endl;
    cout<<"\tColumns to be read in DB files for input:\t";
    for (i=0;i<InputArray[0];i++)cout<<DBInputCol[i]<<"\t";
    cout<<endl;
    cout<<"\tColumns to be read in DB files for output\t";
    for (i=0;i<InputArray[1];i++)cout<<DBOutputCol[i]<<"\t";
    cout<<endl;
    cout<<"\tFinished reading input data\n";
}
void CreateNNet(char *DBforTS,char *DBforVS,char *DBforGS, int
InputArray[],int Method,int DBInputCol[],int DBOutputCol[])
{
// This functions generates nbt Networks and
// Keeps the best Network based on one of the stopping criterion
(see NN_Constructor_5 for which criterion)
// Size of the different TS :
// Size          1    2    3    4    5

```

```

// Design Space 2 : 101 198 501 991 NOT USED
// Design Space 3 : 97 195 486 980 1955
// parameters : drag=1 (col 12),lift=2,Surface=3,Cpmin=4, Total lift
(buoyant+dynamic)=5 (col 16)
// added sept02 for best Error
    double Error=1.e99; // current error found on this network
    double Ebest=1.e99;// Best error found on this network
//end add
//this still need to add a case where GS doesn't exist)
    time_t t0;
    time_t t1;
    time_t t2;
    char buffer[256];
    int ninputs=InputArray[0]; //ninputs<nmax
    int noutputs=InputArray[1]; //noutputs<nmax
    int TSsize=InputArray[2];
    int VSsize=InputArray[3];
    int GSsize=InputArray[4];
    int nbt=InputArray[5];
    int ncand=InputArray[6];
    int index_corr=InputArray[7];
    int i;
    int best_try=0;
    int eps[10]={1,2,3,4,5,6,7,8,9,10};
    NN* Network;
//Create TS and VS sets
    Matrix TSinp(ninputs,TSsize);
    Matrix TStarg(noutputs,TSsize);
    Matrix VSinp(ninputs,VSsize);
    Matrix VStarg(noutputs,VSsize);
//Initialize training and validation sets &Non dimensionalize all
training and validation sets
    DataSet TrSet(TSinp,TStarg);
    DataSet ValSet(VSinp,VStarg);
    cout<<"-----\n";
-----\n";
    cout<<"-\tTraining Set\t-\n";
    TrSet.ReadFileDB(DBforTS,DBInputCol,DBOutputCol);
//
    Vector MinInput(ninputs);
    Vector MaxInput(ninputs);
    Vector MeanTarget(noutputs);
    TrSet.CalculMinMaxMean(MinInput,MaxInput,MeanTarget);
    TrSet.NDimensionalize(MinInput,MaxInput,MeanTarget);
    cout<<"-----\n";
-----\n";
    cout<<"-\tValidation Set\t-\n";
    ValSet.ReadFileDB(DBforVS,DBInputCol,DBOutputCol);
    ValSet.NDimensionalize(MinInput,MaxInput,MeanTarget);
    if ((DBforGS=="0")||(GSsize==0)) GSsize=1;
    Matrix GSinp(ninputs,GSsize);
    Matrix GStarg(noutputs,GSsize);
    DataSet GenSet(GSinp,GStarg);
    if (GSsize>1)
    {
        cout<<"-----\n";
-----\n";

```

```

        cout<<"-\tGeneralization Set\t-\n";
        GenSet.ReadFileDB(DBforGS,DBInputCol,DBOutputCol);
        GenSet.NDimensionalize(MinInput,MaxInput,MeanTarget);
    }
//call NN_constructor 5 to build the network
time(&t0);
for(i=0;i<nbt;i++)
{
    cout<<"*****Try number
"<<i+1<<"*****"<<endl;
    time(&t1);
    Network=new
NN(ncand,TrSet,ValSet,GenSet,Method,Error,index_corr);
    //july 03 Network=new
NN(ncand,TrSet,ValSet,GenSet,Method,Error);
    time(&t2);
    cout<<"Calculation Time      : "<<(t2-t1)/60<<" min
"<<(t2-t1)-60*((t2-t1)/60)<<" sec\n";
    if (Error<Ebest)
    {
        Ebest = Error;
        sprintf(buffer,"./weights/best_weights.data");
        Network-
>WriteFile(buffer,MinInput,MaxInput,MeanTarget, Error,(i+1));
        best_try = i+1;
    }
    cout<< "Saving weight file in
weights/weights_try_XX.data\n";

    sprintf(buffer,"./weights/weights_try_%i.data",eps[i]);
    Network-
>WriteFile(buffer,MinInput,MaxInput,MeanTarget, Error,(i+1));
    delete Network;
}
    cout<<"Calculation Time for the "<<nbt<<" trys      : "<<(t2-
t0)/60<<" min "<<(t2-t0)-60*((t2-t0)/60)<<" sec\n";
    cout<<"Best Result for try "<<best_try<<" ,Best Error =
"<<Ebest<<"\n";
    //cout<<"Best Weight file found in
./weights/best_weights_S"<<cout.width(1)<<index_corr<<".data\n";
    cout<<"Best Weight file found in ./weights/best_weights.data\n";
}

void main()
{
    const int maxsize=80;
    const int maxinput=50;
    char DBforTS[maxsize];
    char DBforVS[maxsize];
    char DBforGS[maxsize];
    int InputArray[8];
    int Method=0;
    int DBInputCol[maxinput];
    int DBOutputCol[maxinput];
    ReadInputData(DBforTS,DBforVS,DBforGS,InputArray,Method,DBInputCo
l,DBOutputCol);

```

```
CreateNNNet(DBforTS,DBforVS,DBforGS,InputArray,Method,DBInputCol,DBOutputCol);  
}
```

## **Matrix.C**

```
//File Matrix.C  
/* - Copyright by Adeline Schmitz, 2003- All rights Reserved-  
*/  
// Class implementation of a matrix  
#include <iostream.h>  
#include "Matrix.H"  
  
Matrix :: Matrix()  
{  
}  
  
Matrix :: Matrix(int r, int c)  
{  
    max_rows = r+20 ;  
    max_columns = c+20;  
    rows = r;  
    columns = c;  
    matrix_elements = new double[max_rows*max_columns];  
    // cout <<"in Matrix.C"<<matrix_elements<<endl;  
}  
  
Matrix :: ~Matrix()  
{  
    delete [] matrix_elements;  
    // matrix_elements=NULL;  
}  
  
//added default copy  
Matrix :: Matrix(const Matrix & rhs)  
{  
    max_rows = rhs.max_nb_rows();  
    max_columns = rhs.max_nb_columns();  
    rows = rhs.nb_rows();  
    columns = rhs.nb_columns();  
    matrix_elements = new double[max_rows*max_columns];  
    for (int zz = 0; zz < rows; zz++)  
        for (int yy = 0; yy < columns; yy++)  
matrix_elements[yy*rows + zz] = rhs.element(zz,yy);  
    //cout<< "making a copy in M "<<matrix_elements[0]<<endl;  
}  
  
void Matrix :: initialize_element(int i, int j, double d)  
{  
    if (((i >= rows) || (i < 0)) || ((j >= columns) || (j < 0)))  
    {
```

```

        cout << "ERROR3: this element matrix is not defined i,j =
"<<i<<" , "<<j<<endl;
    }
    else
    {
        matrix_elements[j*rows + i] = d;
    }
}

double Matrix :: element(int i, int j) const
{
    if (((i >= rows) || (i < 0)) || ((j >= columns) || (j < 0)))
    {
        cout << "ERROR4: this element matrix is not defined i,j
="<<i<<" , "<<j<<endl;
        return 0;
    }
    else
    {
        /*
        cout <<"matrix_elements"<<matrix_elements<<endl;
        cout <<"matrix_elements"<<j*rows + i<<endl;
        cout <<"matrix_elements"<<matrix_elements[j*rows +
i]<<endl;
        */
        return matrix_elements[j*rows + i];
    }
}

int Matrix :: nb_rows () const
{
    return rows;
}

int Matrix :: nb_columns () const
{
    return columns;
}

int Matrix :: max_nb_rows () const
{
    return max_rows;
}

int Matrix :: max_nb_columns () const
{
    return max_columns;
}

Vector Matrix :: vectorization() const
{
    Vector v(columns*rows);
    for (int j = 0; j < columns; j++)
    {
        for (int i = 0; i < rows; i++)
            v.initialize_element(j*rows +
i,matrix_elements[j*rows + i]);
    }
}

```

```

    }
    return v;
}

Vector Matrix :: vertical_average() const
{
    Vector v(rows);
    for (int d = 0; d < rows; d++)
    {
        double f = 0;
        for (int g = 0; g < columns; g++)
        {
            f = f + matrix_elements[g*rows + d];
        }
        v.initialize_element(d,f/columns);
    }
    return v;
}

void Matrix :: equal(Matrix m)
{
    if ((m.nb_columns() != columns) || (m.nb_rows() != rows))
        cout << "ERROR: the matrices don't match \n";
    else
    {
        for (int u = 0; u < rows; u++)
        {
            for (int w = 0; w < columns; w++)
                matrix_elements[w*rows + u] = m.element(u,w);
        }
    }
}

//adds one line
void Matrix :: lin()
{
    if (rows+1 >= max_rows)
    {
        //cout<<" before increasing max_rows= "<<max_rows<<endl, rows
        ="<< rows<<endl;
        mtemp = new double[rows*columns];
        for (int i = 0; i < rows*columns; i++)
            mtemp[i]=matrix_elements[i];
        delete [] matrix_elements;
        matrix_elements=0;
        //max_rows = 2*(rows++);
        max_rows = (rows++)+20;
        matrix_elements = new double[(max_rows)*(max_columns)];
        // need to move all values
        for (int jj = columns - 1; jj >= 0; jj--)
        {
            matrix_elements[jj*rows +rows - 1] = 0;
            for( int ii = rows - 2; ii >= 0; ii--)
            {
                matrix_elements[jj*rows + ii] =mtemp[jj*(rows -
1) + ii];
            }
        }
    }
}

```

```

        }
    }
    delete [] mtemp;
    mtemp=0;
    //cout<<"increasing max_rows "<<max_rows<<endl;
}
else
{
    //cout<<"before increasing rows ="<< rows << ",max_rows="
<<max_rows<<endl;
    rows++;
    for (int jj = columns - 1; jj > 0; jj--)
    {
        matrix_elements[jj*rows +rows - 1] = 0;
        for( int ii = rows - 2; ii >= 0; ii--)
        {
            matrix_elements[jj*rows + ii] =
matrix_elements[jj*(rows - 1) + ii];
        }
        matrix_elements[rows - 1] = 0;
        //cout<<"increasing rows= "<< rows << ",max_rows ="
<<max_rows<<endl;
    }
}
//adds one column
void Matrix :: col()

{
    if (columns+1 >= max_columns)
    {
        //cout<<"before increasing max_columns "<<max_columns
<<" ,columns = "<<columns<<endl;
        mtemp = new double[rows*columns];
        for (int i = 0; i < rows*columns; i++)
            mtemp[i]=matrix_elements[i];
        delete [] matrix_elements;
        matrix_elements=0;
        max_columns = (columns+1)+20;
        matrix_elements = new double[max_rows*max_columns];
        for (i = 0; i < rows*columns; i++)
            matrix_elements[i]=mtemp[i];
        delete [] mtemp;
        mtemp=0;
        //cout<<"increasing max_columns "<<max_columns <<" ,columns
= "<<columns<<endl;
    }
    //cout<<"before increasing columns "<<columns <<" ,max_columns =
"<<max_columns<<endl;
    columns++;
    for (int de = 0; de < rows; de++)
        matrix_elements[(columns - 1)*rows + de] = 0;
    //cout<<"increasing columns "<<columns <<" ,max_columns =
"<<max_columns<<endl;
}

ostream& operator<< ( ostream& theStream, Matrix & a)

```

```
{
    int i,j;

    for(i=0;i<a.nb_rows();i++)
    {
        for(j=0;j<a.nb_columns();j++)
        {
            theStream.width(11);
            theStream<<a.element(i,j)<<"\t";
        }
        theStream<<endl;
    }
    return theStream;
}
```

## **Matrix.H**

```
//File Matrix.H
/* - Copyright by Adeline Schmitz, 2003- All rights Reserved-
   */
// Class declaration of a matrix
#ifndef Matrix_H
#define Matrix_H
#include <iostream.h>
#include "Vector.H"

//les elements d'une matrice nxm vont de (0,0),(0,1)...jusqu'a (n-1,m-1)

class Matrix
{
public:

    Matrix (); //default constructor
    Matrix (int, int); // constructor (horizontal dim, vertical dim)
    ~Matrix (); // destructor
    Matrix (const Matrix & rhs); //copy constructor
    void initialize_element (int, int, double);
    double element(int, int) const;
    int nb_rows () const;
    int nb_columns () const;
    int max_nb_rows () const;
    int max_nb_columns () const;
    Vector vectorization() const;
    Vector vertical_average() const;
    void equal(Matrix);
    void lin(); //adds one line (row)
    void col(); //adds one column
    friend ostream& operator<< (ostream& theStream, Matrix & a);

//private:
    int max_rows; //max numb of rows ( allocated space)
    int max_columns; //max numb of columns ( allocated space)
```

```

int rows; // horizontal dimension == nb of rows
int columns; // vertical dimension == nb of columns
double *matrix_elements; // pointer to array
double *mtemp; //temporary array for copy

};
#endif

```

## NN.C

```

//File NN.C
#include "NN.H"
/*   Class NN Creation of a Neural Network
    Most important class
    This class contains many private methods that come from the
original
    source but that have been modified in order to fit into the class

    The Constructor for this class is in the NN_Constructor5.C Files
*/
/*   - Copyright by Adeline Schmitz, 2003- All rights Reserved-
*/

/*   Calculation of the output vector of an hidden unit that may be
added */

void NN :: unit_output(Vector &weights_to_freeze,Vector &unit_out)
{
    // weigts_to_freeze = w(j)
    // inputs = z(jp) = inputs + outputs of former hidden units
(depend on the pattern p)

    int Pmax=inputs->nb_columns();
    int nh=inputs->nb_rows();//n+h
    int p;

    for (p = 0; p < Pmax; p++)
    {
        double r = 0;
        for (int j = 0; j < nh; j++)
            r = r + weights_to_freeze.element(j)*inputs-
>element(j,p);
        unit_out.initialize_element(p, sigmoid(r));
    }
}

/*   Sigmoid Function */

double NN :: sigmoid(double x)
{
    double s;
    s=1 / (1 + exp(-x));
    return s;
}

```

```

/* Calculation of the squarred error between the output of
the NN and the targets given by the TS */

double NN :: squarred_error(Vector &weights)
{
    // weigts = v(ij) (a matrix coded in vector form)
    // inputs = z(jp) = inputs of NN + outputs of hidden units
    (depend on the pattern p)
    // targets = t(ip) (depend on the pattern p)
    double r = 0, ri = 0;
    // cout<<"i<<p<< output(i,p)<<targets(i,p) in squarred error"
    <<"\n";

    for (int p = 0; p < targets->nb_columns(); p++)
    {
        for (int i = 0; i < targets->nb_rows(); i++)
        {
            ri=0;
            for (int j = 0; j < inputs->nb_rows(); j++)
                ri = ri + weights.element(j*targets-
>nb_rows()+i)*inputs->element(j,p);
            //cout<<i<<" " <<p<<" " << ri<<" " <<targets-
>element(i,p)<<"\n";
            ri = ri - targets->element(i,p);
            ri = ri*ri;
            r = r + ri;
        }
    }
    return r/2;
}

/* Calculation of the Error gradient, that will be used by the local
optimiser
when we will add a hidden unit */

Vector NN :: squarred_error_gradient(Vector &weights)
{
    // weigts = v(ij) (a matrix coded in vector form)
    // inputs = z(jp) = inputs of NN + outputs of hidden units
    (depend on the pattern p)
    // targets = t(ip) (depend on the pattern p)
    double r,s;
    Vector m(weights.dimension());

    for (int k = 0; k < targets->nb_rows(); k++)
    {
        for (int l = 0; l < inputs->nb_rows(); l++)
        {
            r = 0;
            for (int p = 0; p < targets->nb_columns(); p++)
            {
                s = 0;
                for (int j = 0; j < inputs->nb_rows(); j++)
                    s = s + weights.element(j*targets-
>nb_rows()+k)*inputs->element(j,p);
            }
        }
    }
}

```

```

                r = r + (s - targets->element(k,p))*inputs-
>element(l,p);
            }
            m.initialize_element(1*targets->nb_rows()+k,r);
        }
    }
    return m;
}

//Subroutine Added July 03 for looking at different correlation
functions to optimize input-to-hidden weigths
/* Calculation of the correlation between the hidden unit output
and the NN output error */

double NN::correlation(Vector &Zo, Matrix &outputs,int index_corr)
{
    // unit_outp = z(p) = output of the hidden unit =Zo(p)
    // outputs = y(ip) = global outputs of the NN
    // targets = t(ip) = targets
    // index_corr correlation index (0=Scascor, 1=S1, 2=S2, 3=S3,
4=Sqrt_S1,5=Sqrt_S2,6=Sqrt_S3)
    int i,p;
    int m=outputs.nb_rows();
    int Pmax=outputs.nb_columns();
    Matrix E(m, Pmax); //residual_error = E(ip) = y(ip) - t(ip)
    Vector EAvg(m); // Average Resi error E(i)=(1/Pmax)sump(E(i,p))
    double ZoAvg = 0; // average unit output ZoAvg=(1/Pmax)
sump(Zo(p))
    double SUM_Zop_square = 0; // sump(Zo(p)**2)
    double SUM_Zop_ZoAvg_square = 0;//sump((Zo(p)-ZoAvg)**2)
    Vector SUM_Eip_Zop(m); //sump(Eip*Zo(p))
    Vector SUM_Eip_Zop_Avg(m);//sump((Eip-EAvg)*(Zo(p)-ZoAvg))
    double sum_E=0;
    double sum_EZ=0;
    double sum_EZAVG=0;
    double SCasCor = 0;
    double S1 = 0;
    double S2 = 0;
    double S3 = 0;
    double Sqrt_S1 = 0;
    double Sqrt_S2 = 0;
    double Sqrt_S3 = 0;
    for (i = 0; i < m; i++)
    {
        sum_E=0;
        sum_EZ=0;
        for (p = 0; p < Pmax; p++)
        {
            E.initialize_element(i, p, outputs.element(i,p) -
targets->element(i,p));
            sum_E += outputs.element(i,p) - targets-
>element(i,p);
            sum_EZ += E.element(i,p)*Zo.element(p);
        }
        EAvg.initialize_element(i,sum_E/Pmax);
        SUM_Eip_Zop.initialize_element(i,sum_EZ);
        //cout<< SUM_Eip_Zop.element(i)<<endl;

```

```

}
for ( p = 0; p < Pmax; p++)
{
    ZoAvg          += Zo.element(p);
    SUM_Zop_square += Zo.element(p)*Zo.element(p);
}
ZoAvg = ZoAvg / Pmax;
for (p = 0; p < Pmax; p++)
{
    SUM_Zop_ZoAvg_square += pow((Zo.element(p)-ZoAvg),2);
}
for (i = 0; i < m; i++)
{
    sum_EZAVG=0;
    for (p = 0; p < Pmax; p++)
    {
        sum_EZAVG += (Zo.element(p) - ZoAvg)*(E.element(i,p)
- EAvg.element(i));
    }
    SUM_Eip_Zop_Avg.initialize_element(i,sum_EZAVG);
    //cout<< SUM_Eip_Zop_Avg.element(i)<<endl;
}
for (i = 0; i < m; i++)
{
    SCasCor += fabs(SUM_Eip_Zop_Avg.element(i));
    S1      += pow(SUM_Eip_Zop.element(i),2)/SUM_Zop_square;
    S2      += pow(SUM_Eip_Zop.element(i),2);
    S3      +=
pow(SUM_Eip_Zop_Avg.element(i),2)/SUM_Zop_ZoAvg_square;
    Sqrt_S1 +=
fabs(SUM_Eip_Zop.element(i))/sqrt(SUM_Zop_square);
    Sqrt_S2 += fabs(SUM_Eip_Zop.element(i));
    Sqrt_S3  +=
fabs(SUM_Eip_Zop_Avg.element(i))/sqrt(SUM_Zop_ZoAvg_square);
    /*cout<<" SCasCor = " <<SCasCor<<endl;
    cout<<" S1      = " << S1 <<endl;
    cout<<" S2      = " << S2<<endl;
    cout<<" S3      = " << S3<<endl;
    cout<<" Sqrt_S1 = " << Sqrt_S1<<endl ;
    cout<<" Sqrt_S2 = " << Sqrt_S2 <<endl;
    cout<<" Sqrt_S3 = " << Sqrt_S3 <<endl; */
}
if (index_corr==1) return S1;
else if (index_corr==2) return S2;
else if (index_corr==3) return S3;
else if (index_corr==4) return Sqrt_S1;
else if (index_corr==5) return Sqrt_S2;
else if (index_corr==6) return Sqrt_S3;
else return SCasCor;
}
/* Calculation of the correlation gradient, used by the DOT local
optimizer
This is the new version added July 03 for trying different
formula S1,S2,S3 besides SCascor*/

```

```

Vector NN :: correlation_gradient(Vector &weights_to_freeze,Matrix
&outputs, int index_corr)
{
    // weigts_to_freeze = w(j)
    // unit_outp = z(p) = output of the hidden unit =Zo(p)
    // inputs = z(j,p) = inputs + outputs of former hidden units
    (depend on the pattern p)
    // outputs = y(i,p) = global outputs of the NN
    // targets = t(i,p) = targets
    // index_corr correlation index (0=Scascor, 1=S1, 2=S2, 3=S3,
    4=Sqrt_S1,5=Sqrt_S2,6=Sqrt_S3)

    double r,sum,sum1,sum2,i,p,l;
    int m=outputs.nb_rows();
    int nh=inputs->nb_rows(); //n+h
    int Pmax=outputs.nb_columns();
    Vector Zo(Pmax);
    Vector sign_EZ(m);
    Vector sign_EZAVG(m);
    Matrix E(m, Pmax);//Residual Error E(i,p) = yip-tip
    Vector EAvg(m);//Average Resi error E(i)=(1/p)sump(E(i,p))
    double ZoAvg = 0;
    double SUM_Zop_square = 0; // sump(Zo(p)**2)
    double SUM_Zop_ZoAvg_square = 0;//sump((Zo(p)-ZoAvg)**2)
    Vector SUM_Eip_Zop(m); //sump(Eip*Zo(p))
    Vector SUM_Eip_Zop_Avg(m);//sump((Eip-EAvg)*(Zo(p)-ZoAvg))
    double sum_E=0;
    double sum_EZ=0;
    double sum_EZAVG=0;
    Matrix dZop_dwl(nh,Pmax); //dZo(p)/dw(l) = sigm'(sumi(w(i)
z(i,p))) . z(l,p)
    Vector dZop_dwl_Avg(nh); // 1/Pmax sumk (dZo(k)/dw(l))
    Matrix SUM_Eip_dZop_dwl(m,n+h);// sump E(i,p).dZo(p)/dw(l)
    Matrix SUM_Eip_dZop_dwl_Avg(m,n+h);//sump (E(i,p)-
EAvg(i)).(dZo(p)/dw(l)- dZoAvg/dw(l))
    Vector SUM_Zop_dZop_dwl(n+h);// sump Zo(p).dZo(p)/dw(l)
    Vector SUM_Zop_dZop_dwl_Avg(n+h);// sump (Zo(p)-
ZoAvg).(dZo(p)/dw(l)-dZoAvg/dw(l))
    double sum_dSCasCor = 0;
    double sum_dS1 = 0;
    double sum_dS2 = 0;
    double sum_dS3 = 0;
    double sum_dSqrt_S1 = 0;
    double sum_dSqrt_S2 = 0;
    double sum_dSqrt_S3 = 0;
    Vector dSCasCor_dwl(nh); //gradient vector for Cascade Correl
formula
    Vector dS1_dwl(nh) ; //gradient vector for S1
    Vector dS2_dwl(nh) ; //gradient vector for S2
    Vector dS3_dwl(nh) ; //gradient vector for S3
    Vector dSqrt_S1_dwl(nh); //gradient vector for S1^.5
    Vector dSqrt_S2_dwl(nh); //gradient vector for S2^.5
    Vector dSqrt_S3_dwl(nh); //gradient vector for S3^.5

    unit_output(weights_to_freeze,Zo);
    //Begin same calculation as correlation formula
    for (i = 0; i < m; i++)

```

```

    {
        sum_E=0;
        sum_EZ=0;
        for (p = 0; p < Pmax; p++)
        {
            E.initialize_element(i, p, outputs.element(i,p) -
targets->element(i,p));
            sum_E += outputs.element(i,p) - targets-
>element(i,p);
            sum_EZ      += E.element(i,p)*Zo.element(p);
        }
        EAvg.initialize_element(i,sum_E/Pmax);
        SUM_Eip_Zop.initialize_element(i,sum_EZ);
        if ( r >= 0)
            sign_EZ.initialize_element(i,1);
        else
            sign_EZ.initialize_element(i,-1);

        //cout<< "SUM_Eip_Zop.element(i)="<<
SUM_Eip_Zop.element(i)<<endl;
    }
    for ( p = 0; p < Pmax; p++)
    {
        ZoAvg      += Zo.element(p);
        SUM_Zop_square      += Zo.element(p)*Zo.element(p);
    }
    ZoAvg = ZoAvg / Pmax;
    for (p = 0; p < Pmax; p++)
    {
        SUM_Zop_ZoAvg_square      += pow((Zo.element(p)-ZoAvg),2);
    }
    for (i = 0; i < m; i++)
    {
        sum_EZAVG=0;
        for (p = 0; p < Pmax; p++)
        {
            sum_EZAVG += (Zo.element(p) - ZoAvg)*(E.element(i,p)
- EAvg.element(i));
        }
        SUM_Eip_Zop_Avg.initialize_element(i,sum_EZAVG);
        if ( sum_EZAVG >= 0)
            sign_EZAVG.initialize_element(i,1);
        else
            sign_EZAVG.initialize_element(i,-1);

        //cout<< "SUM_Eip_Zop_Avg.element(i)="<<
SUM_Eip_Zop_Avg.element(i)<<endl;
    }
    //

    //calculation of partial derivative of Zo(p) with respect to wl
    // dZop/dwl = sigm'(sumi(w(i) z(i,p))) . z(l,p)
    //Attention: Only if sigm(x) = 1/(1+e^-x), sigm'(x) = sigm(x)(1-
sigm(x)) if change the sigmoid then need to change it's derivative
    r = 0;
    for (l = 0; l < nh; l++)
    {

```

```

sum=0;
for (p = 0; p < Pmax; p++)
{
    r      = Zo.element(p)*(1 - Zo.element(p)); //this
would need to be changed if not using sigmoid
    dZop_dwl.initialize_element(1,p,(r*inputs-
>element(1,p)));
    sum    += dZop_dwl.element(1,p);
}
dZop_dwl_Avg.initialize_element(1,sum/Pmax);
}

for (i = 0; i < m; i++)
{
    for (l = 0; l < nh; l++)
    {
        sum1 = 0;
        sum2 = 0;
        for (p = 0; p < Pmax; p++)
        {
            sum1 += E.element(i,p)*dZop_dwl.element(1,p) ;
            sum2 += (E.element(i,p) -
EAvg.element(i))*(dZop_dwl.element(1,p) - dZop_dwl_Avg.element(1));
        }
        SUM_Eip_dZop_dwl.initialize_element(i,l,sum1);
        SUM_Eip_dZop_dwl_Avg.initialize_element(i,l,sum2);
    }
}

for (l = 0; l < nh; l++)
{
    sum1 = 0;
    sum2 = 0;
    for (p = 0; p < Pmax; p++)
    {
        sum1 += Zo.element(p)*dZop_dwl.element(1,p) ;
        sum2 += (Zo.element(p) -
ZoAvg)*(dZop_dwl.element(1,p) - dZop_dwl_Avg.element(1));
    }
    SUM_Zop_dZop_dwl.initialize_element(l,sum1);
    SUM_Zop_dZop_dwl_Avg.initialize_element(l,sum2);
}

// Calculation of the different derivatives
for (l = 0; l <nh; l++)
{
    sum_dSCasCor      = 0;
    sum_dS1            = 0;
    sum_dS2            = 0;
    sum_dS3            = 0;
    sum_dSqrt_S1      = 0;
    sum_dSqrt_S2      = 0;
    sum_dSqrt_S3      = 0;
    for (i = 0; i < m; i++)
    {
        sum_dSCasCor      +=
sign_EZAVG.element(i)*SUM_Eip_dZop_dwl_Avg.element(i,l);
        sum_dS1            +=
2*SUM_Eip_Zop.element(i)/pow(SUM_Zop_square,2)*(SUM_Zop_square*SUM_Eip_

```

---

```

dZop_dwl.element(i,1)-
SUM_Eip_Zop.element(i)*SUM_Zop_dZop_dwl.element(1));
    sum_dS2          +=
2*SUM_Eip_Zop.element(i)*SUM_Eip_dZop_dwl.element(i,1);
    sum_dS3          +=
2*SUM_Eip_Zop_Avg.element(i)/pow(SUM_Zop_ZoAvg_square,2)*(SUM_Zop_ZoAvg
_square*SUM_Eip_dZop_dwl_Avg.element(i,1)-
SUM_Eip_Zop_Avg.element(i)*SUM_Zop_dZop_dwl_Avg.element(1));
    sum_dSqrt_S1     +=
sign_EZ.element(i)/pow(SUM_Zop_square,1.5)*(SUM_Zop_square*SUM_Eip_dZop
_dwl.element(i,1)-SUM_Eip_Zop.element(i)*SUM_Zop_dZop_dwl.element(1));
    sum_dSqrt_S2     +=
sign_EZ.element(i)*SUM_Eip_dZop_dwl.element(i,1);
    sum_dSqrt_S3     +=
sign_EZAVG.element(i)/pow(SUM_Zop_ZoAvg_square,1.5)*
(SUM_Zop_ZoAvg_square*SUM_Eip_dZop_dwl_Avg.element(i,1)-
SUM_Eip_Zop_Avg.element(i)*SUM_Zop_dZop_dwl_Avg.element(1));
    }
    dSCasCor_dwl.initialize_element(1,sum_dSCasCor);
    dS1_dwl.initialize_element(1,sum_dS1);
    dS2_dwl.initialize_element(1,sum_dS2);
    dS3_dwl.initialize_element(1, sum_dS3);
    dSqrt_S1_dwl.initialize_element(1,sum_dSqrt_S1);
    dSqrt_S2_dwl.initialize_element(1,sum_dSqrt_S2);
    dSqrt_S3_dwl.initialize_element(1,sum_dSqrt_S3);
    //cout<<" In gradient Correlation l = "<<l<<endl;
    //cout<<" gradient SCasCor = "
<<dSCasCor_dwl.element(1)<<endl;
    /*cout<<" gradient S1      = "<< dS1_dwl.element(1)<<endl;
    cout<<" gradient S2      = "<< dS2_dwl.element(1)<<endl;
    cout<<" gradient S3      = "<< dS3_dwl.element(1)<<endl;
    cout<<" gradient Sqrt_S1 = "<<
dSqrt_S1_dwl.element(1)<<endl ;
    cout<<" gradient Sqrt_S2 = "<<
dSqrt_S2_dwl.element(1)<<endl;
    cout<<" gradient Sqrt_S3 = "<<
dSqrt_S3_dwl.element(1)<<endl; */

    }
    if (index_corr==1) return dS1_dwl;
    else if (index_corr==2) return dS2_dwl;
    else if (index_corr==3) return dS3_dwl;
    else if (index_corr==4) return dSqrt_S1_dwl;
    else if (index_corr==5) return dSqrt_S2_dwl;
    else if (index_corr==6) return dSqrt_S3_dwl;
    else return dSCasCor_dwl;
}

// Calculates the L2 norm of the largest input vector: max over
{p=1,Pmax} of ||zp||
double NN :: wmaxi()
{
    int nh=inputs->nb_rows();//n+h+1
    int Pmax=inputs->nb_columns();

    double wmax = 0;
    double w = 0;

```

---

```

for (int p = 0; p < Pmax; p++)
{
    w = 0;
    for (int j = 0; j < nh ; j++)
        w = inputs->element(j,p)*inputs->element(j,p) + w;
    if (w > wmax)
        wmax = w;
}
return sqrt(wmax);
}

/*    Link between Fortran and C++ */

extern "C" void dot510_(int &ndv,int &ncon,int &ncola,int &method,int
&nrvk,int &nriwk,
                    int &nrb,int &ngmax,double *xl,double *xu);
extern "C" void dot_(int &info,int &method,int &iprint,int &ndv,int
&ncon, double *x,
                    double *xl, double *xu, double &obj, int &minmax,
double *g,
                    double *rprm,int *iprm, double *wk,int &nrvk,int
*iwk,int &nriwk);

/* Optimization of the weight of the NN
July 03 Added index_corr to optimize with different correlation
formulas */

void NN::optim(int method, int minmax, int iwrite, double g_max, Vector
&x, Vector &xl, Vector &xu, Matrix &outputs, int index_corr)
{
    int ncon=0, iprint=0, ncola=0;
//    int ncon=1, iprint=0, ncola=0;
// 06/03 Found that constraint can never be violated for gmax=vmax^2*h
and x[i]<vmax. So it has been removed
    int nrwk,nriwk,nrb,ngmax,iprm[20];
    double rprm[20];
    Vector gradientobj(x.dimension());
    Vector unit_out(outputs.nb_columns());

    // DOT arrays & their sizes
    // ncon = number of constraints (1 in our case)
    // g(ncon) = array of constraints ( here is is just a double)
    // iprint = print flag =0 --> no output, =5 writes bunch of
things
    // method = 0,1,2,3 optim method to be used (3=SQP)
    // x(ndv) = array of design variables
    // xl(ndv),xu (ndv) = lower and upper bounds for d.v's
    // obj= value of objective function
    // minmax = 0,-1 for a min
    //          = 1 for a max
    // rprm(20) = array of double control parameters ( can be
overridden)
    // iprm(20) = array of int control parameters ( can be overridden)
    // wk(nrvk) = array used by dot to store internal double var.(
also stores gradients)

```

---

```

//iwk(nriwk) = array used by dot to store internal int var.
//int ncon=1, iprint=5, ncola=0;

//g_max=1000;
// determine dimensions of wk and iwk (ndv,ncon,xl,xu must be
provided)
// ncola is the default value unless non-zero value is input
// ncola = min(10*ndv,ncon)

dot510_(x.dim,ncon,ncola,method,nrwk,nriwk,nrb,ngmax,
        &xl.elements[0],&xu.elements[0]);

// Allocate memory for wk and iwk
double *wk = new double[nrwk];
int *iwk = new int[nriwk];
// double *g = new double[ncon];
double *g = new double[1];
// 06/03 Found that constraint can never be violated for gmax=vmax^2*h
and x[i]<vmax. So it has been removed
// initialize rprm anf iprm
for (int i = 0; i<20; i++)
{
    rprm[i] = 0.;
    iprm[i]=0;
}
// Override all parameters that are needed for this optimization
//iprint=5 for start
// default values for iprm:
//          iprm(1)= igrad = 0 --> change to one to specify
that gradients have to be provided
//          iprm(3)= itmax = 100 number of iterations
allowed during optim
//          iprm(5)= iwrite file number for printed output
iprm[0]=1;// ==iprm(1) in fortran
//This avoids writing in unit 5 (standard input) and unit 6
(standard output) by mistake
if (iwrite>6)iprm[4]=iwrite;//iprm(5) in fortran
//Ready to optimize
int info = 0;
double obj=0.;

// Optimize until info=0
do
{
    dot_(info,method,iprint,x.dim,ncon,&x.elements[0],&xl.elements[0]
,
&xu.elements[0],obj,minmax,g,&rprm[0],&iprm[0],&wk[0],nrwk,&iwk[0],nriwk);
// When info is 1 calculate objective function and constraint
//(*outputs).equal(SAVE);

    if (info==1)
    {
        //Objective function calculation

```

---

```

correlation) // (minmax=0,-1--> squarred_Error, minmax=1 -->
if (minmax==1)
{
    unit_output(x,unit_out);
    obj = correlation(unit_out,outputs,index_corr);
    //cout<<"correl ="<<obj<<endl;
    //obj = real_correlation(unit_out,outputs,targets);
}
if ((minmax==0)|| (minmax==-1)){ obj=squarred_error(x);}

}
// When info is 2 calculate gradients (obj+g)
if (info==2)
{
    if (minmax==1)
gradientobj.equal(correlation_gradient(x,outputs,index_corr));
    if ((minmax==0)|| (minmax==-1))
gradientobj.equal(squarred_error_gradient(x));
    //copy grad of correlation to first ndv elts of wk
    //cout << " gradient="<< gradientobj<<endl;
    for (i = 0; i<x.dim; i++) wk[i] =
gradientobj.element(i);
}
} while (info != 0);
delete [] wk;
delete [] iwk;
delete [] g;
}
/* Filling of the output array with the outputs of the NN for each
element
of the TS */

void NN :: NNoutputs (Matrix &outputs)
{
    // weigts = v(ij)
    // inputs = z(jp) = inputs + outputs of former hidden units
(depend on the pattern p)
    int mm=outputs.nb_rows();
    int Pmax=inputs->nb_columns();
    int n_h = inputs->nb_rows(); //n+h+1
    int i,j,p;
    double r;

    for (i = 0; i < mm; i++)
    {
        for (p = 0; p < Pmax; p++)
        {
            r = 0;
            for (j = 0; j < n_h; j++)
                r = r + weights->element(j*mm + i)*inputs-
>element(j,p);
            outputs.initialize_element(i,p,r);
        }
    }
}

```

```

/*    Calculation of the Max Squarred Error on the TS */

double NN :: max_sq_error(int &psave,Matrix &outputs)
{
    // weights = v(ij) (a matrix coded in vector form)
    // inputs = z(jp) = inputs of NN + outputs of hidden units
    (depend on the pattern p)
    // targets = t(ip) (depend on the pattern p)
    // psave saves the point set which has the largest error
    int Pmax=outputs.nb_columns(); //number of training points
    int m=outputs.nb_rows(); //number of ouputs
    double error,error_element;
    double max_error=0;
    for (int p = 0; p < Pmax; p++)
    {
        error=0;
        for (int i = 0; i < m; i++)
        {
            error_element=(outputs.element(i,p) - targets-
>element(i,p));
            error+=error_element*error_element;
        }
        if (error>=max_error)
        {
            max_error=error;
            psave=p;
        }
    }
    return max_error;
}

/*    After creating a NN, this method will write a file on disk
containing all the
information needed to rebuild the NN, without any calculation.
This one works with the nondimensionalization introduced in
DataSet.C
IT is formatted so that it can be pasted and cut into the main.C
from Evaluator SEPT 02 */

void NN :: WriteFile(char *name,Vector &MinInput,Vector
&MaxInput,Vector &MeanTarget, double Error, int ntry)
{
    int i,j;
    ofstream fout(name);
    //fout<<fixed;
    fout<< setprecision(20);
    fout<<"#include \"NN.H\"\n";
    fout<<"#include <iostream.h>\n";
    fout<<"#include <fstream.h>\n";
    fout<<"#include <stdlib.h>\n";
    fout<<"//#include <time.h>\n";
    fout<<"/*    Evaluator (NN1_Evaluator)\n";
    fout<<"    Calculation of the NN output, an input vector
being given\n";
    fout<<"    The code looks for a dv.data file in the current
folder, this\n";
}

```

```

        fout<<"          file contains the values for the DV.\n";
        fout<<"          PS: INPUTS-OUTPUTS ARE SUPPOSED
DIMENSIONALIZED */\n";
        fout<<"\n";
        fout<<"\n";
        fout<<"int main()\n";
        fout<<"{\n";
        fout<<"          int i;\n";
        fout<<"          int j;\n";
        fout<<"          double aux1,aux2;\n";
        fout<<"          NN* Network;\n";
        fout<<"//\tNeural Network Generated by NN_OPT_1 \n";
        fout<<"//\tBest Mean squarred Error = "<<Error<<" for try number
"<<ntry<<endl;
        fout<<"\tconst int n1="<<n<<";\n";
        fout<<"\tconst int m1="<<m<<";\n";
        fout<<"\tconst int h1="<<h<<";\n";
        fout<<"//\tMinInput[j] & MaxInput[j] for j=1,n \n";
        fout<<"\tdouble MinInput[n1]= {"<<MinInput.element(0);
for(j=1;j<n;j++) fout<<","<<MinInput.element(j);
        fout<<"};\n";
        fout<<"\tdouble MaxInput[n1]= {"<<MaxInput.element(0);
for(j=1;j<n;j++) fout<<","<<MaxInput.element(j);
        fout<<"};\n";
        fout<<"//\tMeanTarget[j] for j=1,m \n";
        fout<<"\tdouble MeanTarget[m1] = {"<<MeanTarget.element(0);
for(j=1;j<m;j++) fout<<","<<MeanTarget.element(j)<<"\n";
        fout<<"};\n";
        fout<<"\tdouble vij[m1*(n1+h1+1)]={"<<weights->element(0);
for (i=1;i<m*(n+h+1);i++) fout<<","<<weights->element(i);
        fout<<"};\n";
        fout<<"\tdouble wij[h1*(n1+h1)]={"<<weights_frozen->element(0,0);
for(j=1;j<n+h;j++) fout<<","<<weights_frozen->element(0,j);
        fout<<"\n\t";
        for (i=1;i<h;i++)
        {
            for(j=0;j<n+h;j++)
            {
                fout<<","<<weights_frozen->element(i,j);
            }
            fout<<"\n\t";
        }
        fout<<"};\n";
        fout<<"          cout.setf(ios::fixed);\n";
        fout<<"          char buffer[1000];\n";
        fout<<"          Vector x(n1);\n";
        fout<<"          Vector y(m1);\n";
        fout<<"          cout<<\"\toutput= \";\n";
        fout<<"          ifstream fin(\"dv.data\");\n";
        fout<<"          fin.getline(buffer,1000);// this throws away the
comment line;\n";
        fout<<"          for(i=0;i<n1;i++)\n";
        fout<<"          {\n";
        fout<<"              fin>>aux1;\n";
        fout<<"              //non-dimensionalize the inputs\n";
        fout<<"              aux2=(aux1-MinInput[i])/(MaxInput[i]-
MinInput[i]);\n";

```

```

        fout<<"           x.initialize_element(i,aux2);\n";
        fout<<"           }\n";
        fout<<"           fin.ignore(80, '\\\n'); //this throws away the
endl ine character\n";
        fout<<"           Network= new NN(m1, n1, h1,vij,wij);\n";
        fout<<"           y=Network->Evaluate3(x);\n";
        fout<<"           for(j=0;j<m1;j++)
cout<<y.element(j)*MeanTarget[j]<<endl;\n";
        fout<<"           cout<<"\tMinInput[j]= {"<<MinInput[0];\n";
        fout<<"           for(j=1;j<n1;j++) cout<<"\,"<<MinInput[j];\n";
        fout<<"           cout<<"}\n";
        fout<<"           cout<<"\tMaxInput[j]= {"<<MaxInput[0];\n";
        fout<<"           for(j=1;j<n1;j++) cout<<"\,"<<MaxInput[j];\n";
        fout<<"           cout<<"}\n";
        fout<<"           cout<<"\tMeanTarget= {"<<MeanTarget[0];\n";
        fout<<"           for(j=1;j<m1;j++) cout<<"
\n"<<MeanTarget[j];\n";
        fout<<"           cout<<endl;\n";
        fout<<"           delete Network;\n";
        fout<<"}\n";

        fout.close();
    }

//    New Calculation of the output vector for an input vector that is
already non-dimensionalized.

Vector NN :: Evaluate3(Vector &X)
{
    int h = (*weights_frozen).nb_rows();
    Vector Y(m);
    int k,j,i;
    Vector Xin(n+h+1);
    Vector u(h);
    double r = 0;
    //    for(j=0;j<n;j++) Xin.initialize_element(j,((X.element(j)-
MinDV[j])/(MaxDV[j]-MinDV[j])));
    for(j=0;j<n;j++) Xin.initialize_element(j,X.element(j));
    Xin.initialize_element(n,1);//bias
    for(k=0;k<h;k++)
    {
        r=0;
        for(j=0;j<n+k+1;j++)
            r=r+(*weights_frozen).element(k,j)*Xin.element(j);
        u.initialize_element(k,sigmoid(r));
        Xin.initialize_element(n+k+1,u.element(k));
    }
    for (i=0;i<m;i++)
    {
        r= 0;
        for (j = 0; j < n+h+1; j++)
            r=r+(*weights).element(j*m+i)*Xin.element(j);
        // rem:weights is written in vector form instead of matrix (m,n+h+1)
        for the optimization.
        // Weights are saved in columns [[v11 v12 ...vm1][ v21
v22..vm2]...[vm1...vm,n+h+1]]

```

---

```

        Y.initialize_element(i,r);
// This returns a nondimensional output ( it should later be multiplied
by the mean.....
    }
    return Y;
}

/*    Calculation of the Design Variables Mean
and the targets mean for a Training Set, in order to normalize
the
TS later */

void NN :: CalculMean(DataSet &tr)
{
    int i,j;

    inputs=new Matrix(tr.getInputs().nb_rows()+1, //Increases the
size of inputs by 1 to add the bias in n+1 position
    tr.getInputs().nb_columns());
    targets=new Matrix(tr.getTargets().nb_rows(), // targets stays
the same size.
    tr.getTargets().nb_columns());

    n=inputs->nb_rows()-1;
    m=targets->nb_rows();
    Pmax=targets->nb_columns();

    MeanDV=0;
    for(i=0;i<m;i++) MeanOut[i]=0;

    for(i=0;i<Pmax;i++)
    {
        for(j=0;j<n;j++)
            MeanDV=MeanDV+pow(tr.getInputs().element(j,i),2);
        for(j=0;j<m;j++)

            MeanOut[j]=MeanOut[j]+pow(tr.getTargets().element(j,i),2);
    }
    MeanDV=sqrt(MeanDV/n/Pmax);
    cout<<"For the training set :MeanDV ="<<MeanDV<<"\n";
    for(j=0;j<m;j++)
    {
        MeanOut[j]=sqrt(MeanOut[j]/Pmax);
        cout<<"                MeanOut[j]
="<<MeanOut[j]<<"\n";
    }

}

/*    Normalization of the TS, using the calculation of the mean made
before    */

void NN :: NormalizeTS(DataSet &tr)
{
    int i;
    int j;

```

---

```

        cout<< " Normalization of training set, new values for inputs and
targets "<<"\n";
        cout<< " inputs(j,i) for j=1,n      and targets(l,i) for l=1,m
"<<"\n";
        for(i=0;i<Pmax;i++)
        {
            cout.width(5);cout<<i<<" ";
            for(j=0;j<n;j++)
            {
                inputs-
>initialize_element(j,i,tr.getInputs().element(j,i)/MeanDV);
                cout.width(11);cout<<inputs->element(j,i)<<" ";
            }

            for(j=0;j<m;j++)
            {
                targets-
>initialize_element(j,i,tr.getTargets().element(j,i)/MeanOut[j]);
                cout.width(11);cout<<targets->element(j,i);
            }
            cout<<"\n";
        }
// the bias is in position n+1 in the equations ( n in the input
matrix) and is worth +1
        for(i=0;i<Pmax;i++)
            inputs->initialize_element(n,i,1);

    }
    /*  Calculation of the Max error on another set
        Used by the stopping criteria Loss of generalization and node
        Patience on
        Testing Set*/

double NN :: EvaluateTS( DataSet &tr,double &EmaxVal)
{
    Matrix inp(tr.getInputs());
    Matrix tar(tr.getTargets());
    int s=inp.nb_rows();
    int npoints=inp.nb_columns() ;
    int m = tar.nb_rows();
    Vector x(s);
    Vector y(m);
    double *E,*Emax,*Eplus;
    double *ratio,*rmax;
    int i;
    int j;
    int jsave=0;
    int k;
    double error;

    E=new double[m];
    Eplus=new double[m];
    Emax=new double[m];
    ratio=new double[m];
    rmax=new double[m];

    for(k=0;k<m;k++) { E[k]=rmax[k]=Emax[k]=ratio[k]=0;}

```

```

    EmaxVal=0.0;
    for(j=0;j<npoints;j++)
    {
        for(k=0;k<s;k++)
            x.initialize_element(k,inp.element(k,j));
        //changed because evaluate(x) normalize the results (not good)   we
        need to nondimensionalize them instead   y=Evaluate(x);
        //june 25 removed non-dim   y=Evaluate2(x);
        y=Evaluate3(x);
        for(k=0;k<m;k++)
        {
            Eplus[k]=pow(tar.element(k,j)-y.element(k),2);
            E[k]=E[k]+Eplus[k];
            if (Eplus[k]>Emax[k])
            {
                Emax[k]=Eplus[k];
                jsave = j;// rem this only works for 1 output
just usefull for printout under here
            }
        }
    }

    error=0;
    for(i=0;i<m;i++)
    {
        // E=1/2(sum from p=0,npoints (sum from i=0,m (output-target)^2))
        error=error+E[i];
        if (EmaxVal<Emax[i]) EmaxVal=Emax[i];
    }
    delete [] E;
    delete [] Eplus;
    delete [] Emax;
    delete [] rmax;
    delete [] ratio;
    // return error;
    return error/2;
}
//CAD ADDED JUNE2 TO PRINT TARGET AND OUTPUT FROM NETWORK AT ANY TIME.
function is same as EvaluateTS
// But it only works if the Nondimensionalized Training set is
used!!!!

void NN :: PrintOutTS( DataSet &tr)
{
    Matrix inp(tr.getInputs());
    Matrix tar(tr.getTargets());
    int s=inp.nb_rows();
    int npoints=inp.nb_columns() ;
    int m = tar.nb_rows();
    Vector x(s);
    Vector y(m);
    double *E,*Emax,*Eplus;
    double *ratio,*rmax;
    int i;
    int j,jsave;
    jsave=0;
    int k;

```



```

    {
// E=1/2(sum from p=0,npoints (sum from i=0,m (output-target)^2))
        error=error+E[i];
        cout<<" E/Pmax ="<<error/2/npoints;
        cout<<" (Emax)validation at
js=";cout.width(11);cout<<sqrt(Emax[i]);
        cout<<" | js=";cout.width(3);cout<<jsave<<"|\n";
    }

    delete [] E;
    delete [] Eplus;
    delete [] Emax;
    delete [] rmax;
    delete [] ratio;
}
//CAD JUNE 02 TO PRINT TARGETS AND OUTPUT FROM NETWORK AT ANY TIME/

/* Function to calculate early stopping criterium GL(t) or loss of
generalization*/
double NN :: CalculateGL ( double Eopt[], double Eva, int t)
{
    double GL;
    if (Eopt[t-1]>Eva)
        Eopt[t]=Eva;
    else
        Eopt[t]=Eopt[t-1];
    GL = 100*(Eva/Eopt[t]-1.0);
// cout <<" in calculateGL
Eopt[t],Eva,GL"<<Eopt[t]<<"\t"<<Eva<<"\t"<<GL<<"\n";
    return GL;
}
/* Function to calculate the PQ criteria or quotient of generalization
loss (GL) and progress Pk*/

double NN :: CalculatePQ ( double Etr[], double GL, int t, int k)
{
    double min=1.e99;
    double sum=0.0;
    double Pk,PQ;
    int i;
    for (i=t-k+1; i<t+1;i++)
    {
        sum = sum+Etr[i];
        if (min>Etr[i]) min=Etr[i];
    }
    Pk = 1000*(sum/k/min-1.0);
    PQ = GL/Pk;
    return PQ;
}
/* Function to calculate UPs stop when generalization error increase in
s successive strips*/
int NN :: CalculateUPs(double Eva[], int t, int k, int UPs_1)
{
    int UPs;
    if (Eva[t]>Eva[t-k])
        UPs=UPs_1+1;
}

```

```

        else
            UPs=0;
        return UPs;
    }
    /* Copy Constructor added and modified september 02 - removed from
    NN_Constructor_SC1.C */

NN :: NN( NN &network)
{
    m=network.getM();
    n=network.getN();
    h=network.getH();
    inputs=new Matrix(n+1,1); //not used but it is deleted in the
destructor so it must remain.
    targets=new Matrix(m,1); //not used but it is deleted in the
destructor so it must remain.
    weights=new Vector(network.getWeights());
    weights_frozen= new Matrix(network.getWf());
}

/* Destructor */

NN :: ~NN()
{
    delete inputs;
    delete targets;
    delete weights_frozen;
    delete weights;
}

```

## NN.H

```

// FILE nn.H
#ifndef NN_H
#define NN_H
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <math.h>
#include "Matrix.H"
#include "DataSet.H"
/* Class NN Creation of a Neural Network */
/* - Copyright by Adeline Schmitz, 2003- All rights Reserved-
*/

class NN
{
public:
    //NN(int nb,DataSet &tr,DataSet &ts);
    NN(int nb,DataSet &TrSet,DataSet &ValSet,DataSet
&GenSet,int Method, double &Error,int index_corr);
    NN( NN &);

```

---

```

NN( char *);
~NN();
void setWeights(Vector *w) { weights=w;}
void setMeanDV(double m) { MeanDV=m; }
void setMeanOut(double m,int i) { MeanOut[i]=m; }
double getMeanDV() { return MeanDV; }
double getMeanOut(int i) { return MeanOut[i]; }
double getWeightFrozen(int i,int j) { return
weights_frozen->element(i,j); }
double getWeight(int i) { return weights->element(i); }
int getM() {return m;}
int getN() {return n;}
int getH() {return h;}
Vector getWeights() const {return (*weights);}
Matrix getWf() const {return (*weights_frozen);}
void WriteFile(char *,Vector &MinInput,Vector
&MaxInput,Vector &MeanTarget, double Error, int ntry);
Vector Evaluate3(Vector &x);
double EvaluateTS(DataSet &tr, double &EmaxVal);//july 02
added EmaxVal to print it all in the NN_constructor5
void PrintOutTS( DataSet &tr);
double CalculateGL ( double Eopt[], double Eva, int t);
double CalculatePQ ( double Etr[], double GL, int t, int
k);

int CalculateUPs(double Eva[], int t,int k, int UPs_1);
NN(int m, int n, int h, double vij[], double wij[]);

private:
int n , m , Pmax, ncand;
Matrix *weights_frozen; //wij
Vector *weights;//v[ij]
Matrix *weights_frozen_opt; // wij_opt
Vector *weights_opt;//v[ij]_opt
Matrix *inputs;
Matrix *targets;
int h ; //number of hidden units
double MeanDV;
double *MeanOut;
double *MinDV;
double *MinOut;
double *MaxDV;
double *MaxOut;
void NDimensionalizeTS(DataSet &);
void CalculMinMax(DataSet &);
void NormalizeTS(DataSet &);
void CalculMean(DataSet &);
double sigmoid(double x);
void unit_output(Vector &weights_to_freeze,Vector &u_opt);
double squarred_error(Vector &weights);
Vector squarred_error_gradient(Vector &weights);
double correlation(Vector &unit_outp,Matrix &outputs, int
index_corr); //added july 03
Vector correlation_gradient(Vector
&weights_to_freeze,Matrix &outputs, int index_corr);//added july 03
double wmaxi();

```

---

```

        void optim(int method, int minmax, int iwrite, double
g_sq_max, Vector &x, Vector &xl, Vector &xu,Matrix &outputs, int
index_corr);//added july 03
        void NNoutputs (Matrix &outputs);
        double max_sq_error(int &psave,Matrix &outputs);
};

#endif

```

### **NN Constructor SC5.C**

```

// File NN_Constructor_SC5.C
/*    - Copyright by Adeline Schmitz, 2003- All rights Reserved-
    */
#include "NN.H"

/*    Constructors for the new Stopping Criteria        */

/*    Basic Constructor this constructor will create a Neural Network
and stop when best network has been found
    it saves the network and the best error found (Error)*/
NN :: NN(int nb,DataSet &TrSet,DataSet &ValSet,DataSet &GenSet,int SC,
double &Error, int index_corr)

//NN :: NN(int nb,TrainingSet &tr,TrainingSet &tstest1,TrainingSet
&tstest2, double &Error)
{
// added for testing random init
    const int h_max=70; //max number of hidden units to install on
Network
    int minmax,method,iwrite;// parameters used in optimizer
(Optimizer is Vanderplaats DOC/DOT software)
    // method = 0,1,2,3 optim method to be used (3=SQP)
    // minmax = 0,-1 for a min
    //          = 1 for a max
    // iwrite file number for printed output
    int i,j,e,p;
    int nValidation=ValSet.getTargets().nb_columns();//nb of
validation points
    int nGen=GenSet.getTargets().nb_columns();//nb of generalization
points
    int psave[h_max+1];// saves the point # in TS which shows the
largest error (E_max) (0<psave<Pmax-1)
    double wmax=0;// max(norm(xp))=wmax ----- norm of weight array-
related to weight initialization
    // double vmax=10;//max value when initializing weights also max
value for norm of weights in optimization
    double vmx=10;//max value when initializing weights also max
value for norm of weights in optimization
    double    init_value=0.5;// to initialize weights between -
0.5,+0.5
    double Cmax = 0;//variable used in max Covariance calculation
    double C = 0;// Covariance or Correlation Calculation
    double E[h_max+1];//Squarred Error on TS
    double E_max[h_max+1];//Maximum Squarred Error

```

```

// For calculation of early stopping criteria. Defined as Prechelt
"early stopping but when"
    int UPs[h_max+1]; //UP criterion (see Prechelt "early stopping but
when")
    int h_mod; // modulo of h - to determine whether GL, PQ and UP
criterion need to be calculated or not (every h_period)
    int h_period = 5; //training strip length
    int h_opt=h_max; // nb of hidden units associated with best
result so far in training (optimal result=Min Error = E_opt)
    double Eva[h_max+1]; //Squarred Error on VS ( Validation Error)
    double Egen[h_max+1]; //Squarred Error on GS
    double Eopt[h_max+1]; //Optimal error found so far on VS
    double GL[h_max+1]; //GL criterion (see Prechelt "early stopping
but when")
    double PQ[h_max+1]; //PQ criterion (see Prechelt "early stopping
but when")
    double EmaxVal=0.0; //maximum validation error
    enum StopCrit
{NONE, GL1, GL2, GL3, GL5, PQ005, PQ0075, PQ01, PQ02, PQ03, UP1, UP2, UP3}; //used
for stopping criteria
    bool Continue=true; //used for stopping criteria
// added july 03
    //int index_corr=6;
//endadd
    if ((StopCrit(SC)<NONE) || (StopCrit(SC)>UP3))
    {
        cout<< "error StopCriterion beyond range permitted, Stop
Criterion= "<<StopCrit(SC)<<endl;
        exit(0);
    }
//
// m, Pmax and n are calculated during the CalculMinMaxMean, they are
private to DataSet so they are only known in the DataSet class
// The following variables are private in the NN class constructor,
they must be defined.
    m=TrSet.getTargets().nb_rows(); //number of outputs
    n=TrSet.getInputs().nb_rows(); //number of inputs
    Pmax=TrSet.getInputs().nb_columns(); //number of points in
Training Set
    h=0; //nb hidden units
    ncand=nb; //nb of candidate units to try on network before adding
the best one
//
    Matrix outputs(m, Pmax); //y[ip]
    Vector unit_outp(Pmax); //Zi[p]
    weights_frozen=new Matrix(1,n+1); //w[ij]
    weights=new Vector(m*(n + 1)); //v[ij]
    weights_frozen_opt=new Matrix(1,n); //to save wij with optimal
result so far
    weights_opt=new Vector(m*(n + 1)); //to save v[ij] with optimal
result so far
    Vector weights_lower(m*(n + 1));
    Vector weights_upper(m*(n + 1));
    Vector weights_to_freeze(n+1); // Wj: Vector weights that will be
copied into the matrix wij each time a hidden unit is added
    Vector weights_to_freeze2(n+1);
    Vector w_to_freeze_lower(n+1);

```

```

    Vector w_to_freeze_upper(n+1);
// Since Inputs and Targets are private in both NN and DataSet class
they must be copied here in the NN class
// Inputs in the NN class have one extra row for bias compared to the
Inputs in the DataSet class
// n is the number of inputs, the bias is added in position n+1 ( n in
the matrix)
// So,from now on the inputs matrix will have an extra row
    inputs=new Matrix(TrSet.getInputs().nb_rows()+1,
        TrSet.getInputs().nb_columns()); //Increases the size of
inputs by 1 to add the bias in n+1 position
    targets=new Matrix(TrSet.getTargets().nb_rows(),
        TrSet.getTargets().nb_columns()); // targets stays the same
size.
    for(i=0;i<Pmax;i++)
    {
        for(j=0;j<n;j++)
        {
            inputs-
>initialize_element(j,i,TrSet.getInputs().element(j,i));
        }
        inputs->initialize_element(n,i,1);//bias is in position n+1
in the equations ( n in the input matrix) and is equal to 1
        for(j=0;j<m;j++)
        {
            targets-
>initialize_element(j,i,TrSet.getTargets().element(j,i));
        }
    }

//Initialization of Error Arrays
    for (i=0 ; i<h_max+1 ; i++)
    {
        Eva[i]=1e99; Egen[i]=0;
        Eopt[i]=1e99;GL[i]=0;
        PQ[i]=0;UPs[i]=0;
    }
//Initialization of weight arrays
    for (i=0 ; i<weights->dimension() ; i++)
    {
        weights_lower.initialize_element(i,-vmax);
        weights_upper.initialize_element(i,vmax);
    }
    for (i=0 ; i<weights_to_freeze.dimension() ; i++)
    {
        w_to_freeze_lower.initialize_element(i,-vmax);
        w_to_freeze_upper.initialize_element(i,vmax);
    }

//initialization of v[ij] between -init_value, and +init_value
    weights->random_init_value(init_value);//test_init
    NNoutputs(outputs);
//Optimization of weights by minimizing E calls Vanderplaats
optimization software DOC
//SUBROUTINE
DOT( INFO, METHOD, IPRINT, NDV, NCON, X, XL, XU, OBJ, MINMAX, G, RPRM, IPRM, WK, NRWK,
IWK, NRIWK)

```

```

minmax=-1;
method=1;
iwrite=7;
optim(method,minmax,iwrite,vmax*vmax,*weights,weights_lower,weights_upper,outputs,index_corr);
// july03
optim(method,minmax,iwrite,vmax*vmax,*weights,weights_lower,weights_upper,outputs);
iwrite++;
NNoutputs(outputs);
E_max[0]=max_sq_error(psave[0],outputs);
E[0] = squarred_error(*weights)/Pmax;
cout <<"E/Pmax After Optimization- no HU: "<<E[0]<<endl<<endl;
cout <<" ..... #HU (E/Pmax)tr (EMax)tr
(E/p)val (EMax)val (E/p)gen (EMax)gen
GL[h] PQ[h] UPs[h] \n";
//OPTIMIZATION of E gives new v[ij]=weights
do
{
//old_error=error;
h++;// we add a hidden unit

//rewriting weights_frozen w[hj] with a new size
if (h != 1)
{
weights_frozen->lin();
weights_frozen->col();
weights_to_freeze.plus(1);//add an element
weights_to_freeze2.plus(1);
w_to_freeze_lower.plus(1);
w_to_freeze_upper.plus(1);
}
//initialize the extra element at position n+h-1.
w_to_freeze_lower.initialize_element(n+h-1,-vmax);
w_to_freeze_upper.initialize_element(n+h-1,vmax);
wmax = wmaxi();// calcul de max(norm(xp))=wmax
Cmax = 0;
C = 0;
// Optimize for the new hidden unit starting from several
//randomly selected initial weights_to_freeze (several
candidates hidden unit)
// keep the one that gives the max correlation Cmax
cout.flush();
for (e = 0; e < ncand; e++)
{
weights_to_freeze2.random_initialization(wmax);
if(ncand<10)
{
cout<<".";
cout.flush();
// optimization of weights_to_freeze2 by
maximizing the Correlation
minmax=1;

optim(method,minmax,iwrite,vmax*vmax*h,weights_to_freeze2,w_to_freeze_lower,w_to_freeze_upper,outputs,index_corr);

```

```

// july 03
optim(method,minmax,iwrite,vmax*vmax*h,weights_to_freeze2,w_to_freeze_1
ower,w_to_freeze_upper,outputs);

        iwrite++;
    }
    unit_output(weights_to_freeze2,unit_outp);
    C = correlation(unit_outp,outputs,index_corr);
    //july 03 C = correlation(unit_outp,outputs);
//added for testing july 03
    //cout<<"Correlation = "<<C<<endl;
    //C = correlation(unit_outp,outputs,0);
    //Vector gradientobj(n+h);

//gradientobj.equal(correlation_gradient(weights_to_freeze2,outputs));

//gradientobj.equal(correlation_gradient(weights_to_freeze2,outputs,0));
// endadd
//keep the new candidate hidden unit if better than
previously found
    if (C > Cmax)
    {
        Cmax = C;
        weights_to_freeze.equal(weights_to_freeze2);
    }
//exit(0);
// Once the best candidate has been chosen, save and
"freeze" the weights_to_freeze
    for ( j = 0; j < weights_frozen->nb_columns(); j++)
        weights_frozen->initialize_element(h-1, j,
weights_to_freeze.element(j));
//rewriting inputs with a new hidden unit
    unit_output(weights_to_freeze,unit_outp);
    inputs->lin();
    for (p = 0; p < Pmax; p++)
        inputs-
>initialize_element(n+h,p,unit_outp.element(p));
//initialization of added column v[ij]=weights
// to be forgotten for now / vmax =
vmax*(1+1/(1+n+h));
    weights->plus(m);
    weights_lower.plus(m);
    weights_upper.plus(m);
    for (i=0;i<m;i++)
    {
        weights->initialize_element(i+m*(n+h),0);
        weights_lower.initialize_element(i+m*(n+h),-vmax);
        weights_upper.initialize_element(i+m*(n+h),vmax);
    }
    NNoutputs(outputs);
//Optimization of weights by minimizing E
    minmax=-1,method=1;

```

```

    optim(method,minmax,iwrite,vmax*vmax*h,*weights,weights_lower,wei
ghts_upper,outputs,index_corr);
        //july 03
    optim(method,minmax,iwrite,vmax*vmax*h,*weights,weights_lower,weights_u
pper,outputs);
        iwrite++;
        NNoutputs(outputs);
        E_max[h]=max_sq_error(psave[h],outputs);
        E[h] = squarred_error(*weights)/Pmax;
//Writes everything in main output file
    cout.setf(ios::showpoint);
    cout.width(5);cout<<h<<"\t";
    cout.width(11);cout<<E[h]<<"\t";
    cout.width(11);cout<<sqrt(E_max[h])<<"\t";
    Eva[h]=EvaluateTS(ValSet,EmaxVal)/nValidation;
    cout.width(11);cout<<Eva[h]<<"\t";
    cout.width(11);cout<<sqrt(EmaxVal)<<"\t";
//added to avoid calculating error on Generalization set when not
present.
        Egen[h] = 0.;
        EmaxVal = 0.0;
        if (nGen>1)Egen[h]=EvaluateTS(GenSet,EmaxVal)/nGen;
        cout.width(11);cout<<Egen[h]<<"\t";
        cout.width(11);cout<<sqrt(EmaxVal)<<"\t";
// Added sept 02 to save best result so far.
        if (Eopt[h-1]>Eva[h])
        {
            delete weights_frozen_opt;
            delete weights_opt;
            weights_opt=new Vector(m*(n+h+1));
            weights_frozen_opt= new Matrix(h,n+h);
            weights_opt->equal(*weights);
            weights_frozen_opt->equal(*weights_frozen);
            h_opt= h;
        }
//Added july 02 for early stopping criteria
// Test done every h_period hidden units have been added to the
Network.
        GL[h]=CalculateGL ( Eopt, Eva[h], h);
        h_mod= h % h_period;
        if (h<h_period)
        {
            cout.width(11);cout<<GL[h]<<"\t";
            cout.width(11);cout<<PQ[h]<<"\t";
            cout.width(11);cout<<UPs[h]<<"\n";
        }
        if (h_mod == 0)
        {
            PQ[h]=CalculatePQ ( E, GL[h], h, h_period);
            UPs[h]=CalculateUPs( Eva, h, h_period, UPs[h-1]);
            cout.width(11);cout<<GL[h]<<"\t";
            cout.width(11);cout<<PQ[h]<<"\t";
            cout.width(11);cout<<UPs[h]<<"\n";
        }
        else if (h>h_period)
        {

```

```

        PQ[h]=PQ[h-1];
        UPs[h]=UPs[h-1];
        cout.width(11);cout<<GL[h]<<"\t";
        cout.width(11);cout<<PQ[h]<<"\t";
        cout.width(11);cout<<UPs[h]<<"\n";
    }
    //Check if stopping criterion became true
    if (StopCrit(SC)==GL1) {if (GL[h]>1.) Continue=false;}
    else if (StopCrit(SC)==GL2) {if (GL[h]>2.) Continue=false;}
    else if (StopCrit(SC)==GL3) {if (GL[h]>3.) Continue=false;}
    else if (StopCrit(SC)==GL5) {if (GL[h]>5.) Continue=false;}
    else if (StopCrit(SC)==PQ005) {if (PQ[h]>0.05)
Continue=false;}
        else if (StopCrit(SC)==PQ0075) {if (PQ[h]>0.075)
Continue=false;}
        else if (StopCrit(SC)==PQ01) {if (PQ[h]>0.1)
Continue=false;}
        else if (StopCrit(SC)==PQ02) {if (PQ[h]>0.2)
Continue=false;}
        else if (StopCrit(SC)==PQ03) {if (PQ[h]>0.3)
Continue=false;}
        else if (StopCrit(SC)==UP1) {if (UPs[h]>=1) Continue=false;}
        else if (StopCrit(SC)==UP2) {if (UPs[h]>=2) Continue=false;}
        else if (StopCrit(SC)==UP3) {if (UPs[h]>=3) Continue=false;}
        else if (StopCrit(SC)>UP3){cout<<"error stopping criterion
out of range\n";exit(0);}
    }
    while((Continue)&&(h<h_max));
    h = h_opt;
    //This saves the NN with the number of Hu leading to the optimal error
    found up to the stopping criterion
    delete weights_frozen;
    delete weights;
    weights=new Vector(m*(n+h+1));
    weights_frozen= new Matrix(h,n+h);
    weights->equal(*weights_opt);
    weights_frozen->equal(*weights_frozen_opt);
    Error = Eopt[h];
    cout<< "\nFor this try, the optimum result was found for h="<<h<<
" hidden units\n";
    cout << "Best mean squarred error Eval(tmin)= "<<Error<<endl;
    // cout<< "Results follow\n\n";
    // PrintOutTS(ValSet);
}
/* NN Constructor from two arrays vij and wij , this has been added
sept 02 for Evaluating a NN for Optimization ( used only by Evaluator)
*/

NN :: NN(int m1, int n1, int h1, double vij[], double wij[])
{
    int i,j;
    m=m1;
    n=n1;
    h=h1;
    inputs=new Matrix(n+1,1); //not used but it is deleted in the
destructor so it must remain.

```

```

    targets=new Matrix(m,1); //not used but it is deleted in the
destructor so it must remain.
    weights=new Vector(m*(n+h+1));
    weights_frozen= new Matrix(h,n+h);
    for(i=0;i<m*(n+h+1);i++)
    {
        weights->initialize_element(i,vij[i]);
    }
    for(i=0;i<h;i++)
    {
        for(j=0;j<n+h;j++)
        {
            weights_frozen-
>initialize_element(i,j,wij[i*(n+h)+j]);
        }
    }
}

```

### **Vector.C**

```

//File Vector.C
/*    - Copyright by Adeline Schmitz, 2003- All rights Reserved-
    */
//    Class implementation for a Vector
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include "Vector.H"

Vector :: Vector()
{
}

Vector :: Vector (int l)
{
    max_dim=l+20;
    dim = l;
    elements = new double[max_dim];
//    cout <<"in Vector.C"<<elements<<endl;
}

Vector :: ~Vector()
{
    delete [] elements;
//    elements=NULL;
}

//added default copy
Vector :: Vector(const Vector & rhs)
{
    max_dim = rhs.max_dimension();
    dim = rhs.dimension();
    elements = new double[max_dim];
    for (int zz = 0; zz < dim; zz++)
    {

```

---

```

        elements[zz]=rhs.element(zz);
//      cout<< "making a copy in V "<<rhs.element(0)<<endl;
    }
}
void Vector :: initialize_element (int i, double k)
{
    if ((i >= dim) || (i < 0))
    {
        cout << "ERROR1: this vector element is not defined \n";
    }
    else
    {
        elements[i] = k;
    }
}
// Initialize randomly so that the initialized vector has a norm less
// than 4*max_norm
void Vector :: random_initialization (double max_norm)
{
    double p = 0;
    double v ;
    for (int d = 0; d < dim; d++)
    {
        // this will generate numbers between {-max_norm,+max_norm}
//      elements[d] = max_norm*2*(drand48()-0.5);
        elements[d] = 2*(drand48()-0.5);
        p = elements[d]*elements[d] + p;
//      cout<<"elements[d] = "<<elements[d]<<endl;
    }
    p = sqrt(p); //calculates the norm of p
    v = 4*drand48();// number between 0 and 4
    //v = 0.5 +drand48()/2.;
    v = max_norm*v;// number between 0 and 4*max_norm
    for (int dd = 0; dd < dim; dd++)
        elements[dd] = elements[dd]*v/p;// rescales the vector so
that its norm is between 0 and 4*max_norm
}

// Initialize randomly between [-value,+value]
void Vector :: random_init_value(double value)
{
    for (int d = 0; d < dim; d++)
    {
        // this will generate numbers between {-value,+value}
        elements[d] = 2*value*(drand48()-0.5);
    }
}

void Vector :: plus(int p)
{
    if (dim + p > max_dim)
    {
//      cout<<"before Increased max dimension of Vector max_dim =
"<< max_dim<<endl;
        temp = new double[dim];
        for (int d=0; d < dim; d++) temp[d]=elements[d];
    }
}

```

---

```

        delete [] elements;
        elements = 0;
        //max_dim=2*(dim+p);
        max_dim=(dim+p)+20;
        elements = new double[max_dim];
        for (d=0; d < dim; d++)
            elements[d]=temp[d];
        delete [] temp;
        temp = 0;
//        cout<<" Increased max dimension of Vector"<<endl;
//        cout<<" new max_dim = "<< max_dim<< " , new dim =
"<<dim+p<<endl;
    }
    dim = dim + p;
//    cout<<" increased size, old dim = "<<(dim-p)<< " new dim =
"<<dim<<" max_dim = "<<max_dim<<endl;
}

double Vector :: element(int i) const
{
    if ((i >= dim) || (i < 0))
    {
        cout << "ERROR2: this element vector is not defined
="<<i<<endl;
        return 0;
    }
    else
    {
        return elements[i];
    }
}

int Vector :: dimension() const
{
    return dim;
}
int Vector :: max_dimension() const
{
    return max_dim;
}

void Vector :: equal(Vector v)
{
    if (v.dimension() != dim)
        cout << "ERROR: the vectors don't match \n";
    else
    {
        for (int j = 0; j < dim; j++)
            elements[j] = v.element(j);
    }
}

ostream& operator<< ( ostream& theStream, Vector & vect)
{
    int i;
    for(i=0;i<vect.dimension();i++)
    {

```

```

        theStream.width(11);
        theStream<<vect.element(i)<<"\t";
    }
    theStream<<endl;
    return theStream;
}

```

## **Vector.H**

```

//File Vector.H
/*    - Copyright by Adeline Schmitz, 2003- All rights Reserved-
    */
//    Class implementation for a Vector
#ifndef Vector_H
#define Vector_H
#include <iostream.h>

class Vector
{
public:

    Vector (); // default constructor
    Vector (int); // constructor with dimension of the Vector
    ~Vector (); // destructor
    Vector (const Vector & rhs); // copy constructor
    void initialize_element (int, double); // initializes the ith
element of the Vector with a value (double)
    void random_initialization(double); // initializes the vector
randomly with its norm is between 0 and 4*max_norm
    void random_init_value(double value); // initializes the vector
randomly between [-value,+value]
    double element(int) const; // returns the ith element
    int dimension() const; // returns the Vector dimension
    int max_dimension() const; // returns the max dim of the Vector
(allocated max space)
    void equal(Vector); // equals the vector to another given vector;
    void plus(int); // increases the dim of the vector by the value of
int
    friend ostream& operator<< (ostream& theStream, Vector & vect);

//private:
    int max_dim; // max dim of Vector (space allocated)
    int dim; // dimension of Vector
    double *elements; // pointer to array of elements
    double *temp; // used to make a copy
};
#endif

```

## **Makefile for NN\_OPT**

```

OBJSC= main.o          DataSet.o          NN.o \
        NN_Constructor_SC5.o  Matrix.o  Vector.o

```

OBJSF= ddot1.o      ddot2.o      ddot3.o      ddot4.o      ddot5.o  
          ddot6.o

C\_COMPILER=CC -g  
F\_COMPILER=f77 -n32 -O2 -g

COMMAND= ./NN\_OPT

\$(COMMAND): \$(OBJSC) \$(OBJSF)  
          \$(C\_COMPILER) -o \$(COMMAND) \$(OBJSC) \$(OBJSF) -lm -lftn

\$(OBJSF):  
.f.o:  
          \$(F\_COMPILER) -c \$<

\$(OBJSC): \*.H  
.C.o:  
          \$(C\_COMPILER) -c \$<