METHODS FOR FEATURES SELECTION:
GENETIC ALGORITHMS AND TIME CONSUMING OPTIMISATION

By David Giner Viñuelas.

Escola Tècnica Superior d’Enginyeria

Universitat Rovira i Virgili

Tarragona, Catalonia, Spain
Report made by David Giner Viñuelas
Automàtica i Electrònica Industrial
Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili, Tarragona, Spain.
# INDEX

**Chapter 1.- Introduction and Background** .................................................................5  
1.1 Introduction to Electronic Noses .................................................................5  
1.2 Project Aims ................................................................................................8  
1.3 Electronic Noses Structure .................................................................9  
1.4 Initial Data .................................................................................................10  
1.5 Methods for Feature Selection .........................................................12  
   1.5.1 Deterministic Methods .................................................................13  
      1.5.1.1 Forward Selection .......................................................13  
      1.5.1.2 Backward Elimination .................................................14  
      1.5.1.3 Stepwise Selection .......................................................14  
      1.5.1.4 Correlated PCR ..........................................................15  
   1.5.2 Stochastic Methods ....................................................................15  
      1.5.2.1 Simulated Annealing ...................................................15  
      1.5.2.2 Genetic Algorithms .....................................................17  

**Chapter 2.- Genetic Algorithms** .................................................................18  
2.1 Definition ..................................................................................................18  
2.2 Function .....................................................................................................18  
2.3 Genetic Algorithms Constitution ..............................................................19  
2.4 Genetic Algorithm Architecture ...............................................................21  
   2.4.1.- Simple Architecture ..................................................................21  
   2.4.2.- Chromosome Representation ....................................................22  
   2.4.3.- Selection Function ....................................................................23  
   2.4.4.- Genetic Operators .....................................................................25  
2.4.5.- Creation of the Initial Population .......................................................28  
2.4.6.- Termination Criteria ..................................................................29  
2.5 Constraints .................................................................................................30  
   2.5.1.- Searching Domain ...................................................................30  
   2.5.2.- Penalty Scheme ........................................................................30  
   2.5.3.- Mappings ...................................................................................31  
2.6 Parallel Architecture ..................................................................................31
Chapter 3.- Data Analysis ........................................................................33

3.1 Initial Assumptions and Internal Processes ........................................33
3.2 Backward Elimination ........................................................................35
3.3 Forward Selection ...............................................................................36
3.4 Stepwise Selection ..............................................................................37
3.5 Genetic Algorithms for Features Selection .........................................38
  3.5.1.-Representation and Operators ...................................................38
  3.5.2.-Parameter Settings ....................................................................38
  3.5.3.-Selection Mechanism ................................................................39
  3.5.4.-Objective Function and Fitness Evaluation ..............................39
  3.5.5.-Termination Criteria ................................................................40
  3.5.6.-Program Flow ............................................................................40
3.6 Genetic Algorithm Execution ................................................................40
3.7 Computation Time Optimisation in GA ..............................................42
  3.7.1.-Advance the Termination Criteria .............................................43
  3.7.2.-Decrease the Members in the Population ..................................44
  3.7.3.-Adaptive Population ................................................................45
  3.7.4.-Collinear and Noisy Features ....................................................47
  3.7.5.-Hybrid GA - Backward Elimination .........................................49
3.8 Discussions and Conclusions ................................................................52

Chapter 4.- Graphical User Interface ......................................................58

4.1 Introduction ........................................................................................58
4.2 Commands ..........................................................................................58
4.3 Interface Design ..................................................................................64

Chapter 5.- Further Work ........................................................................67

Chapter 6.- References ............................................................................68

Appendix ....................................................................................................70
.m Files .....................................................................................................71
1.1 Introduction to Electronic Noses

The use and study of electronic noses has increased during the last years both in academia and industry. The fact of being able to analyse only the desired parameter, getting results in real time, simplifying the conditioning of the samples, analysed in an automated way without involving expert human noses and the interested support from food and cosmetic industry has propitiated the research and development of techniques to get better accuracy, robustness and low computation time when analysing odours [1].

If we compare the olfaction sense to the ear and eye ones, the human nose is much more complicated at least in primary reactions to an external stimulus. Whereas a video camera can catch the image very similar to that seen by the eye, in artificial olfaction the situation is quite different. Although several interesting developments have been already made, their performance is far from our olfactory sense.

It is still even unclear how all the system works, but some aspects have been investigated [2]. This is a simple graphic how our olfactory system works.

![Figure 1.1](image_url)

There are a large number of olfactory receptor cells (more than 10 million) but with a limited amount of selectivity classes (≈ 10-100). The odour produces a pattern of signals to the olfactory cortex via the mitral cells (≈ 10000). The brain interprets the signal pattern as a specific odour.

In order to use the information in the receptor signal, it has to be processed in a suitable way. Electrical signals are transferred from the receptors to the olfactory bulb through axons and dendrites. The signals then reach simple signal processing cells called neurons. A neuron has (in general) many inputs but only one output,
which can either be excited or not. There will be a signal on the output if there is enough excitation on its inputs, with different importance (weight) being attached to the different inputs. These weights can be changed in a learning process, making it possible for us to learn to better recognise odours we are often exposed to. In the olfactory bulb there are many neurons, together forming a whole network. This network processes the information and then transfers the processed data to the olfactory cortex. This is where the final processing is made, by another network of neurons, and also where the communication with the rest of the brain takes place. The brain can then use this new information together with stored knowledge and tell us, e.g., to run away from or approach the odour source.

When defining an electronic nose a very similar behaviour is considered. An electronic nose can be considered as a modular system comprising a set of active materials which detect the odour, associated sensors which transducer the chemical quantity into electrical signals, followed by appropriate signal conditioning and processing to classify known odours or identify unknown odours [3].

Based on that definition we can deduce how an electronic nose works. An array of chemical sensors, usually between 5 and 100, change their physical properties according the concentration of some chemical species. These changes will be translated into electrical or optical signal which is recorded into a computer. A suitable data processor method will get quantitative and qualitative information from the chemical environment.

![Diagram of an electronic nose](image)

**Figure 1.2**

Here a schematic comparison between human and electronic noses in order to have a clearer picture of similitude and differences:
**Table 1.1**

Nowadays the electronic noses are in an early development stage. Anyway there is commercial equipment such as FOX 2000, AromaScan, Nordic Sensor which can be used in different applications such as [4]:

**Electronic noses for environmental monitoring**

The high amount of waste, including nuclear and chemical wastes, made start exploring the technologies required to perform environmental restoration and waste management in a cost effective manner. This effort includes the development of portable, inexpensive systems capable of identifying in real time the contaminant in the field.

There exists environmental applications of electronic noses analysing fuel mixtures [5], detecting oil leaks [6], and testing ground water for odours. The potential applications include identification of toxic wastes, air quality monitoring and monitoring factories emissions.

**Electronic noses for medicine**

An electronic nose has applicability as a diagnostic tool. An electronic nose can examine odours from the body (e.g., breath, wounds, body fluids, etc.) and identify possible problems. Odours in the breath can be indicative of gastrointestinal problems, infections, diabetes, and liver problems.
Infected wounds and tissues emit distinctive odours that can be detected by an electronic nose. Odours coming from body fluids can indicate liver and bladder problems.

Electronic noses for the food industry

Currently, the biggest market for electronic noses is the food industry. Applications of electronic noses in the food industry include quality assessment in food production, inspection of food quality by odour, control of food cooking processes, inspection of fish, monitoring the fermentation process, checking rancidity of mayonnaise, verifying if orange juice is natural, monitoring food and beverage odours, grading whisky, inspection of beverage containers, checking plastic wrap for containment of onion odour, and automated flavour control to name a few. In some instances electronic noses can be used to increase or replace panels of human experts. In other cases, electronic noses can be used to reduce the amount of analytical chemistry that is performed in food production especially when qualitative results will do.

1.2 Project Aims

The high amount of data extracted by an electronic nose makes a difficult task to work with. Because samples measured by gas sensor arrays are usually described by tens or even hundreds of variables that are not mutually independent or affected by noise, not all of them are needed and we could get rid of them. Investigators have put attention to different strategies for feature selection in gas analysis in order to keep the same information obtained by the electronic nose but reducing the initial dataset carrying direct and relevant information. Several variable selection methods have been reported as useful. The deterministic methods such as forward or backward selection can make a good selection with a relatively few operations but can be trapped in a local optimum of the search space.

On the other hand stochastic methods like the genetic algorithms have been also found to be a useful method for variable selection since they are more likely to find a global optimum. A GA is a problem solving method that uses genetic rules such as reproduction, gene crossover a mutation to build pseudo-organisms that are then selected, based on a fitness criterion, to survive and pass information on to the next generation [7].
The main disadvantage of this method is the large computation time dependable of the number of features to work with. The aim of this work will be to decrease the computation time to run a GA modifying some of its parameters and analyse how the deterministic methods can contribute using their main advantages to improve the final result in combination with the GA.

Different combinations will be tested and conclusions extracted.

1.3 Electronic Noses Structure

In order to get a better understanding why we are going to run a Genetic Algorithm in a electronic nose data application, we will briefly introduce how a electronic nose works.

As we described in a previous chapter, an electronic nose is based on three main parts:
- Sample preparation and measurement (chemical)
- Signal preparation and features extraction (electronics)
- Pattern recognition and visualization (software)

In this project we will be focused on the pattern recognition part since this is where the Genetic Algorithm is applied, but in this chapter we will also introduce the other parts to have an overall idea about how the complete system works. More detailed explanations will be given in the pattern recognition in next chapters.

The electronic noses strongly depend on chemical sensor features which are built on. Since decades ago, investigators have been developing different kind of sensors based on different material and working principles. The common characteristic is that they have partially overlapped sensitivities since each sensor reacts to a wide spectrum of gases and vapours.

The sensors used in an electronic nose can be either mass transducers (such as Quartz microbalanz or QMB) or chemoresistors (based on metal-oxides or conducting polymers). Some arrays comprise both types of sensors.

Semiconductor sensors work at high temperatures (100-600 °C) and measure the variation of electrical conductivity on the surface layer depending of the concentration of absorbed oxygen. The more amount of “volatile molecules” the less amount of absorbed oxygen [8].

Another organical material is the conducting polymers which have high sensitivity working in ambient temperatures (20-60 °C) [9].
Currently extensive research is being carried out on the exploitation of metallo-porphyrins as coating material for QMB. The main feature of such sensors is the dependence of the sensing properties (selectivity and sensitivity) on the nature of the constituents of the porphyrin. This flexibility makes this class of compounds of interest for electronic nose applications [10].

1.4 Initial Data

The system is built on an array of 12 commercially available metal oxide gas sensors from Figaro Engineering Inc. (TGS type sensors 825(x2), 826, 822 (x2), 800 (x2), 882 (x2), 2160, 2611 and 2620). The sensors are kept in a temperature controlled chamber (30°C) and the R.H. humidity is set to 10%. The amount of volatile compounds needed to create the desired concentration in the sensor chamber is introduced in the liquid phase by using high-precision liquid chromatography syringes. A fan is used to clean the chamber between measurements. More details on the experimental set up can be found elsewhere [10].

The vapours measured were acetone, ammonia and o-xylene at 50, 100, 200 and 400 p.p.m. and their binary mixtures (measuring all the possible combinations of two vapours, given the four concentrations used). This gives a total of 3 measurements of single vapours and 21 measurements of binary mixtures. Since each measurement was replicated 4 times, there are a total of 96 measurements in the database. The number of outputs was set to 6 corresponding to 3 single vapours and 3 binary mixtures,
ignoring their concentration. The measurements were gathered during 5 consecutive
days and randomised (i.e. they were performed in a disordered way).

The measurement procedure was as follows: data acquisition started one minute
before the injection of the volatile compounds. This allowed for the baseline
resistance of the sensors to be acquired. The sampling rate was set to 10 samples/s.
Acquisition ended after 10 minutes and the sensor chamber was flushed with synthetic
air. Next figure shows a typical response of a TGS sensor o-xylene and some of the
features extracted from the response. The complete list of features extracted from the
response of the 12 sensors in the array is:

- Initial (or baseline) conductance $G_i$. Conductance of a sensor in the presence
  of air.
- Final conductance, $G_f$. Conductance of a sensor at the end of the acquisition
  period (600 s).
- Conductance change, $AG = G_f - G_i$.
- Normalized conductance change, $AG_n = (G_f - G_i)/G_i$.
- Maximum conductance, $G_{max}$. Maximum value of the sensor conductance
  (see figure below).
- Maximum conductance change, $AG_{max} = G_{max} - G_i$.
- Normalized maximum conductance change, $AG_{max_n} = (G_{max} - G_i)/G_i$.
- Time when the maximum conductance is reached, $t_{max}$.
- Rise time of the conductance caused by injection of volatile compounds,
  defined between 10% and 90% of the maximum conductance change, $t_{10-90}$.
- Rise time of the conductance caused by injection of volatile compounds,
  defined between 30% and 60% of the maximum conductance change, $t_{30-60}$. 


Given those 10 features were extracted from the response of each sensor and there were 12 sensors within the array, each measurement was described by 120 features, the data matrix had 96 rows (i.e. measurements) and 120 columns (i.e. variables).

1.5 Methods for Feature Selection

Sometimes a large number of independent variables, $X_i$, are available for a given modelling problem, and not all of these predictor variables may contribute equally well to the explanation of the predicted variable $Y$. Some of the independent variables may not contribute at all to the model. Thus we have to select from these variables to obtain a model which contains as little variables as possible while still being the "best" model. In principle, all possible combinations of independent variables should be tried for calculating a suitable model. This could turn out to be a formidable task, even if high performance computers are available. Besides the practicability of this approach, there are also several theoretical considerations which should be taken into account:

- Trying all possible combinations may lead to chance correlations.
- The contribution of a single variable to the explanation of $Y$ may not easily be assessed if only a small number of observations is available.
- A simple criterion, like the goodness of fit, $r^2$, may lead to wrong conclusions if the number of selected variables approaches the number of observations.
• For more complicated models (e.g. artificial neural networks) the calculation of a single model may be so time-consuming that it is practically impossible to find the "best" combination of independent variables.

• The selection of combinations is guided by the available data; thus the resulting final selection reflects the "best" model for the given data set, and not the "best" subset for the population.

• Some of the selection methods are specifically tailored to linear (regression) models; they are unusable with non-linear methods such as neural networks.

Depending on the type of model being used, there are several strategies to (partially) solve the problem.

Methods of minimisation can be largely classified into two groups, deterministic and stochastic. Deterministic methods have the strength of being extremely fast, but have the weakness of being liable to be caught in a local minimum fairly easily. On the other hand, a stochastic method is far less likely to be trapped in a local minimum, but it can be shown that no stochastic method has a probability of one to converge to the global minimum in a finite number of steps.

1.5.1 Deterministic Methods

Several deterministic methods have been shown as useful such as forward selection, backward elimination, stepwise selection, analysis of weights resulting from MLR, leaps-and-bounds regression. Here it is a brief explanation of the main ones.

1.5.1.1 Forward Selection

This procedure starts by selecting the feature which gives the lowest value for the objective function. Subsequent features are selected so that at each step, the objective function value is the lowest. This procedure continues until the new lowest objective function value is higher than the previous optimum. Some researchers or applications also include a stop if next loop is not improving the result in a percentage.
1.5.1.2 Backward Elimination

Initially, this method calculates the objective function value for the complete data set. It then calculates the influence of each feature on the model by calculating the objective function value for the model without that feature. The feature that gives the lowest objective value, i.e. the one that has least influence in the model, is removed. This procedure continues until the new lowest objective function value is higher than that of the previous optimum. Like in the previous method, it can also be include a stop if the percentage of improvement has not been fulfilled.

1.5.1.3 Stepwise Selection

This algorithm is performed in either the forwards or the backwards modes to find the subset of features that best minimises an objective function.

This addresses the situation where variables are added or removed early in the process and we want to change our mind about them later. At each stage a variable may be added or removed and there are several variations on exactly how this is done. Stepwise procedures are relatively cheap computationally but they do have some drawbacks.

1. Because of the one-at-a-time nature of adding/dropping variables, it's possible to miss the optimal model.

2. The p-values used should not be treated too literally. There is so much multiple testing occurring that the validity is dubious. The removal of less significant predictors tends to increase the significance of the remaining predictors. This effect leads one to overstate the importance of the remaining predictors.

3. The procedures are not directly linked to final objectives of prediction or explanation and so may not really help to solve the problem of interest. With any variable selection method, it is important to keep in mind that model selection cannot be divorced from the underlying purpose of the investigation.

Variable selection tends to amplify the statistical significance of the variables that stay in the model. Variables that are dropped can still be correlated with the response. It would be wrong to say these variables are unrelated to the response, it's just that they provide no additional explanatory effect beyond those variables already included in the model.

4. Stepwise variable selection tends to pick models that are smaller than desirable for prediction purposes.
1.5.1.4 Correlated PCR [12]

This method is a correlation-based variant of the classical variance-based PCR that has been developed to select subsets of PC’s ordered according to their correlation with the dependent variable (y). The motivation behind this method is to use only those PC’s that are related to the parameter being studied (dependent variable). Since in many data sets some PC’s with high variance are in fact not related to the dependent variable but to others sources of variations, e.g., particle size (scattering effects) and instrument noise, a high eigenvalue does not necessary imply a high relevant information content. Therefore, this method can include low-variance PC’s in the regression model, for they are informative for prediction purposes.

1.5.2 Stochastic Methods

1.5.2.1 Simulated Annealing [13]

Simulated annealing (SA) is a generic probabilistic meta-algorithm for the global optimisation problem, namely locating a good approximation to the global optimum of a given function in a large search space.

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

In the simulated annealing (SA) method, each point $s$ of the search space is compared to a state of some physical system, and the function $E(s)$ to be minimised is interpreted as the internal energy of the system in that state. Therefore the goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy.

At each step, the SA heuristic considers some neighbours of the current state $s$, and probabilistically decides between moving the system to state $s'$ or staying put in state $s$. The probabilities are chosen so that the system ultimately tends to move to
states of lower energy. Typically this step is repeated until the system reaches a state which is good enough for the application, or until a given computation budget has been exhausted.

The probability of making the transition to the new state $s'$ is a function $P(dE, T)$ of the energy difference $dE = E(s') - E(s)$ between the two states, and of a global time-varying parameter $T$ called the temperature.

One essential feature of the SA method is that the transition probability $P$ is defined to be nonzero when $dE$ is positive, meaning that the system may move to the new state even when it is worse (has a higher energy) than the current one. It is this feature that prevents the method from becoming stuck in a local minimum, a state whose energy is far from being minimum, but is still less than that of any neighbour.

### Convergence to optimum

It can be shown that, for any given finite problem, the probability that the simulated annealing algorithm terminates with the global optimal solution approaches 1 as the annealing schedule is extended. This theoretical result is, however, not particularly helpful, since the annealing time required to ensure a significant probability of success will usually exceed the time required for a complete search of the solution space.

### Transition probabilities

The transition probability function $P$ is not as critical as the neighbourhood graph, provided that it follows the general requirements of the SA method stated before. Since the probabilities depend on the temperature $T$, in practice the same probability function is used for all problems, and the annealing schedule is adjusted accordingly.

### The "classical" formula

In the original formulation of the method by Kirkpatrick et. al, the transition probability $P(dE, T)$ was defined as 1 if $dE < 0$ (i.e., downhill moves were always performed); otherwise, the probability would be $e^{dE/T}$. This formula comes from the Metropolis-Hastings algorithm, used here to generate samples from the Maxwell-
Boltzmann distribution governing the distribution of energies of molecules in a gas. Other transition rules can be used, also.

1.5.2.2 Genetic Algorithms
Since this is the method will be based on this work, next chapter will be completely dedicated to it.
CHAPTER 2. GENETIC ALGORITHMS

2.1. Definition

Genetic algorithm (GA) optimisers are robust stochastic search methods modelled on the principles and concepts of natural selection and evolution, where stronger individuals are likely to be the winners in a competing environment [14].

2.2. Function

GA optimisers are particularly effective when the aim is to find and approximate global maximum or minimum in a high-dimension, multimodal function domain in a near-optimal manner [15].

Algorithms for function optimisation are generally limited to convex regular functions, although, GAs have been used to solve difficult problems with objective functions that do not possess properties such as continuity, differentiability, etc. These algorithms maintain and manipulate a family or populations of solutions and implement a “survival of the fittest “strategy in the search for better solutions. This provides both implicit and explicit parallelism that allows the exploitation of several promising areas of the solution space at the same time. The implicit parallelism arises from the manipulation of a population of points.

In general, the fittest individuals of any population tend to reproduce and survive to the next population, thus improving successive generations. However, inferior individuals can, by chance, survive and also reproduce.
2.3. Genetic algorithms constitution

First of all we introduce some GA vocabulary.

**GA terminology [15]:**

- **Gene**    Coded optimisation parameter
- **Chromosome**   A trial solution vector (string) consisting of genes
- **Generation** Successively created population (GA generations)
- **Population** Set of trial solutions
- **Parent** Member of the current generation
- **Child** Member of the next generation
- **Fitness** A number assigned to an individual representing measure of goodness

*Figure 2.1 GA terminology*

**Genes and Chromosomes:**

As in natural evolution, the gene is the basic building block in the GA optimisation. Generally genes are a coded representation of individual optimisation parameters. Within the GA paradigm, a string of genes is called a chromosome. A specific realisation of a chromosome can be decoded into a set of parameters representing a trial solution. Chromosomes can be coded as binary strings, strings of real numbers, or combinations of both.

**Populations and Generations:**

In GA based optimisations a set of trial solutions encoded in the form of chromosomes is assembled as a population. The population is the source that the GA optimiser utilises to search for the optimum solution. The iterations in GA optimisation are called generations. Reproduction, consisting of selection and recombination/mutation, continues until a new generation is created to replace the original generation. Ideally, highly fit individuals, or, more precisely, highly fit characteristics, produce more copies of themselves in the subsequent generation, resulting in a general drift of the population as a whole toward an optimal solution.
point. The process can be terminated in several ways: threshold on the best individual (i.e., the process stops when an individual has an error less than some amount $E$), number of generations exceeds a preselected value, or some other appropriate criteria.

Parents:

Following an initialisation process in which a population is created, pairs of individuals are selected (with replacement) from the population in a probabilistic manner weighted by their relative fitness and designated as parents. In a typical selection scheme, modelled as a weighted roulette wheel, each individual in the population is assigned a space on the roulette wheel proportional to the individual's relative fitness. The wheel is spun each time a parent is required. Individuals with the largest spaces on the wheel have the greatest chance of being selected and, therefore, the greatest probability of passing on their characteristics to the next generation.

Children:

Offspring, or children, are then generated from the selected pair of parents by application of simple stochastic operators. The principal operators are crossover and mutation. Crossover involves the random selection of a crossover site(s) and the combining of the two parents' genetic information. Specifically, in single-point crossover, child 1 receives the chromosomal sub-string that precedes the cross-site in parent 1 and the substring following the cross-site in parent 2. Child 2 gets the remaining genetic information not given to child 1. The two children produced share the characteristics of the parents as a result of this recombination operator. Other recombination operators are sometimes used but crossover is the most important. Recombination (e.g., crossover) and selection are the principal ways in which evolution occurs in a GA optimisation. Different kinds of crossover and mutation will be explained in section 2.3.4.

Fitness:

The objective function defining the optimisation goal, called a fitness function in the GA paradigm, is a mean of assigning a value to each individual in the population. The fitness function is usually the link between the physical problem and
the GA optimisation process. The fitness function assigns to an individual a number representing the “goodness” of the trial solution represented by that individual.

### 2.4. Genetic Algorithm Architecture

#### 2.4.1. Simple Architecture

1. **[Start]** Generate random population of $n$ chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness $f(x)$ of each chromosome $x$ in the population
3. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
4. **[New population]** Create a new population until the new population is complete
   1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
   2. **[Crossover]** Cross over the parents to form a new offspring (children). If no crossover was performed, the offspring are an exact copy of the parents.

![Simple GA Diagram]

Figure 2.2 *Simple GA*
3. **[Mutation]** Using a mutation probability mutate new offspring at each locus (position in chromosome).

4. **[Accepting]** Place new offspring in a new population

5. **[Replace]** Use new generated population for a further run of algorithm

As you can see, the outline of Basic GA is very simple. There are many things that can be implemented differently according to the problems to solve.

The use of a genetic algorithm requires the determinations on the next five issues.

### 2.4.2. Chromosome representation

A chromosome representation is needed to describe each individual in the population of interest. Each one is made up from a sequence of genes from a certain alphabet. An alphabet could consist of binary digits (0 and 1), floating point numbers, integers, symbols, matrices, etc.

The first problem found is to choose a binary or float representation. Many researchers are still investigating which are the advantages or disadvantages of each one and some results are found at the moment.

The binary representation [16] traditionally used in genetic algorithms has some drawbacks when applied to multidimensional, high-precision numerical problems. For instance, for 100 variables with domains in the range \([-500,500]\) where a precision of six digits after the decimal point is required, the length of the binary solution vector is 3000. This, in turn, generates a search space of about \(10^{1000}\). For such problems genetic algorithms perform poorly.

The binary alphabet offers the maximum number of schemata per bit of information of any coding and consequently the bit string, representation of solutions, has dominated genetic algorithm research.

According to some experiments [17] carried on by some researchers, it indicates that the floating point representation is faster, more consistent from run to run, and provides a higher precision, especially with large domains where binary representation would require prohibitively long representation. At the same time, its performance can be enhanced by special operators to achieve high accuracy (even higher than the one of binary representation). In addition, the floating point representation, being intuitively closer to the problem space, is easier to use for
designing other operators incorporating the problem specific knowledge. These conclusions are in accordance with the reasons of the users of genetic-evolutionary techniques who prefer floating point representation.

The chromosome should in some way contain information about the solution that it represents, for instance a number. The mostly used way of encoding is using a binary string. The chromosome then could look like this:

<table>
<thead>
<tr>
<th>Chromosome 1</th>
<th>1101100100110110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 2</td>
<td>1101111000011110</td>
</tr>
</tbody>
</table>

Table 2.1

The coding chromosome representation may vary according to the nature of the problem itself. In general, the bit string encoding is the most classic method used by GA researchers due to its simplicity and traceability. GA is the most successful when the encoding used is as close to the problem space as possible [18]. Nevertheless, a Gray encoding can be used when we are working with a close population because it ensures that any pair of adjacent points in the problem space differ by only a single bit in the representation space. A logarithmic representation is useful when one is working with a large space and search is conducted without the need for excessive string lengths and maintaining reasonable accuracy.

2.4.3. Selection function

The selection of individuals to produce successive generations plays an extremely important role in a genetic algorithm. A probabilistic selection is performed based upon the individual’s fitness such that the best individuals have increased chance of being selected.

There are several possible schemes for the selection process:

2.4.3.1. Roulette wheel

This is the simplest selection scheme, also called stochastic sampling with replacement. This is a stochastic algorithm that involves the following technique:

The individuals are mapped into contiguous segments of a line, such that each individual's segment is equal in length to its fitness. A random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained (called mating population). In this example the probability of each individual is calculated according to its fitness value:
For selecting the mating population the appropriate number of uniformly distributed random numbers (uniform by distributed between 0.0 and 1.0) is independently generated.

For instance, let us consider a sample of 6 random numbers:

0.81, 0.32, 0.96, 0.01, 0.65, 0.42.

The figure below shows the selection process of the individuals for the previous example and together with the above sample trials.

After selection the mating population consists of the individuals:

1, 2, 3, 5, 6, 9.

2.4.3.2. Tournament [17]

In a tournament selection, a subpopulation of N individuals is chosen at the random from the population. The chromosomes of this subpopulation then compete on the basis of their fitness. The chromosome in the subpopulation with the highest fitness wins the tournament and becomes the selected chromosome. All of the
subpopulation members are then placed back into the general population and the process is repeated.

### 2.4.3.3. Elitist models

When creating a new population by crossover and mutation, there is a high probability that the best chromosome will be lost.

Elitism is the name of the method, which first copies the best chromosome (or a few best chromosomes) to the new population. The rest is done in classical way. Elitism can very rapidly increase the performance of GA, because it prevents the GA losing the best solution found in the previous generation.

### 2.4.3.4. Ranking methods

The previous selection method (elitist) will have problems when the fitness of the chromosome varies very greatly. For example, if the best chromosome fitness is 90% of the entire roulette wheel, the other chromosomes will have very little chance of being selected.

One way to overcome this problem is rank selection, this first ranks the population and then each chromosome receives fitness from this ranking. The worst will have fitness $1$, second worst $2$ etc. and the best will have fitness $N$ (number of chromosomes in population).

Using this approach can help the avoidance of premature convergence and speed up the search when the population approaches convergence [18].

### 2.4.3.5. Steady-state selection

This is not a particular method for selecting parents. The main idea of this selection method is that a large proportion of chromosomes should survive to the next generation.

GA then works in the following way. In every generation a few (good - with high fitness) chromosomes are selected for creating a new offspring. Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to the new generation.

### 2.4.4. Genetic Operators

Genetic operators provide the basic search mechanism for the GA. The operators are used to create new solutions based on existing solutions in the population. There are two basic types of operators: crossover and mutation. Crossover
takes two individuals and produces two new individuals, while mutation alters one individual to produce a single new solution. The application of these two basics types of operators and their derivatives depends on the chromosome representation used.

### 2.4.4.1.-Crossover

Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way how to do this is to choose randomly a crossover point and everything before this point is copied from the first parent and then everything after a crossover point copies from the second parent.

Crossover can then be presented look like this ( | is the crossover point):

<table>
<thead>
<tr>
<th>Chromosome 1</th>
<th>01011</th>
<th>00100110110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 2</td>
<td>11011</td>
<td>11000011110</td>
</tr>
<tr>
<td>Offspring 1</td>
<td>01011</td>
<td>11000011110</td>
</tr>
<tr>
<td>Offspring 2</td>
<td>11011</td>
<td>00100110110</td>
</tr>
</tbody>
</table>

Table 2.3

Crossovers can be rather complicated to handle and very depend on encoding of chromosome.

There are other ways to make crossover, for example we can choose more than one crossover points or a specific crossover made for a specific problem can improve performance of the genetic algorithm.

#### Different kinds of crossover:

**Single point crossover** - one crossover point is selected, the binary string from the beginning of the chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent

\[
11001011 + 11011111 = 11001111
\]
Two point crossover - two crossover points are selected, the binary strings from the beginning of the chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent.

\[
\begin{array}{c|c|c}
\text{Parent A} & \text{Parent B} & \text{Offspring} \\
\hline
11001001 & 11011110 & = 11011111 \\
\end{array}
\]

Uniform crossover - bits are randomly copied from the first or the second parent.

\[
\begin{array}{c|c|c}
\text{Parent A} & \text{Parent B} & \text{Offspring} \\
\hline
11001011 & 11011101 & = 11001101 \\
\end{array}
\]

Arithmetic crossover - some arithmetic operations are performed to make a new offspring.

In binary representation it could be and AND or OR operation:

\[
11001011 + 11011111 = 11001001 \text{ (AND)}
\]

Heuristic crossover - some arithmetic operations are performed to make a new offspring in the direction of the better parent.

We can generate new children like this:

\[-X \text{ is a random number between 0 and 1} \]

\[
\text{children} = X \times (\text{best parent} - \text{worst parent}) + \text{best parent}
\]
2.4.4.2. Mutation

Mutation is used to prevent from falling all solutions in population into a local optimum of the solved problem. Mutation changes randomly the new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. Mutation can then be represented as follow:

<table>
<thead>
<tr>
<th>Original offspring 1</th>
<th>1101111000011110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original offspring 2</td>
<td>1101100100110110</td>
</tr>
<tr>
<td>Mutated offspring 1</td>
<td>1100111000011110</td>
</tr>
<tr>
<td>Mutated offspring 2</td>
<td>1101101100110110</td>
</tr>
</tbody>
</table>

Table 2.4

The mutation depends on the encoding as well as the crossover. For example when we are encoding permutations, mutation could be exchanging two genes.

The common way to apply mutation is to invert one or more genes on a specified probability.

**Bit inversion** - selected bits are inverted

11001001 => 10001001

There are many other kinds of crossover and mutation and these can be changed according to the characteristics of the problem to improve the behaviour of the Genetic Algorithm and obtain better results.

2.4.5. Creation of the initial population

The GA must be provided with an initial population. The most commonly used method is to randomly generate solutions for the entire population. But existing solutions can be improved and the beginning population can be seeded with potentially good solutions, with the remainder of the population being solutions randomly generated.
Population size is an important parameter, which determines the amount of the search space that is scanned during the run [18]. It may seem intuitive to use the largest population possible, but it is not always the best idea because populations that are too large can cause problems. They move slowly in response to good points they find, and they have many children in each generation, requiring many simulations. Often they will be more consistent in the solutions they find but they are not always the best answers. The reason is that a large population can overpower the genetic material from an above-average individual that resides on a narrow spike in the search space. The largest population with its gene pool diversity overpowers these individuals by sheer numbers, and generally the largest hills will be exploited. If large hills do not contain the best answers, the answer will often be suboptimal. It takes many more simulations to obtain good answers with large simulations. It may make sense, then, to tackle a problem with several runs of a small-population GA rather than with just one run of a large-population GA, as both will be of the same cost in terms of simulation time.

On the other hand, populations that are too small generally do not have enough diversity for the GA to search much of the space. They converge quickly to suboptimal points. It is important, then, to choose a good population size for the problem.

**2.4.6. Termination criteria**

The GA moves from generation to generation selecting and reproducing parents until a termination criterion is met. The most frequently used stopping criterion is a specified maximum number of generations. Another termination strategy involves population convergence criteria. In general, GAs will force most of the entire population to converge to a single solution. When the sum of the deviations among individuals becomes smaller than a specified threshold, the algorithm can be stopped. The algorithm can also terminate due to the lack of improvement in the best solution over a specified number of generations.
2.5. Constraints

In the resolution of some problems, not all the chromosomes in the population are suitable to be a solution, since the problem specified to fulfil previous requirements that we call them "constraints".

Working with constraints is a difficult obstacle to overcome. It depends on the problem to be resolved, and can make difficult the convergence to the optimal solution. Various methods can be used in this area [19]:

2.5.1. Searching domain

It is possible to embed the condition of constraints in the system by confining the searching space of a chromosome. This approach guarantees that all chromosomes are valid and that the constraint will not be violated. A typical example is to limit the searching domain of the coefficients of a digital lattice filter design, to the range of –1 to +1 whose pole locations will be confined within the unit circle for stability. This method of solving the constraint problem requires no additional computation power, and all chromosomes created are regarded as potential solutions to the problem.

2.5.2. Penalty scheme

Another approach for handling constraints is to set up a penalty scheme for invalid chromosomes such that they became low performers. The constraint problem is then transformed to an unconstrained condition by associating the penalty with all the constraint violations. This can be done by including a penalty to adjust the optimised objective function.

Consider this equation as the original objective function to be optimised:

\[ f(x_1, x_2, \ldots, x_n) \]

Including a penalty scheme within the objective function, it becomes:

\[ f(x_1, x_2, \ldots, x_n) + \delta \sum_{i=1}^{m} \ell_i \]

\[ , \text{where } m \text{ is the total number of constraints, } \delta \text{ is a penalty coefficient which is negative for maximisation and positive for minimisation problems, and } \ell_i \text{ is a penalty related to the i-th constraint (i=1,2,\ldots,m).} \]

The penalty scheme has two distinct characteristics:
- some vital information may be thrown away; and
- a small violation of a constraint may qualify if it produces a large payoff in other areas.
However, an appropriate penalty function is not easy to define and will affect the efficiency of the genetic search [20].

If one incorporates a high penalty into the evaluation routine and the domain is one in which production of an individual violating the constraint is likely, one runs the risk of creating a genetic algorithm that spends most of its time evaluating illegal individuals [20]. Furthermore, when a legal individual is found, it can drive the others out and the population converges without finding better individuals, since the likely paths to other legal individuals require the production of illegal individuals as intermediate unlikely that such intermediate structures will be produced. If one imposes moderate penalties, the system may generate individuals that violate the constraint but are rated better than those that do not because the rest of the evaluation function can be satisfied better by accepting the moderate constraint penalty than by avoiding it.

At the best, the technique based on penalty functions seems to work reasonably well for narrow classes of problems and for a few constraints.

### 2.5.3. Mappings

Another approach focuses on the use of special representation mappings (decoders) which guarantee (or at least increase the probability of) the generation of feasible solution, and the application of special repair algorithms to correct any infeasible solutions. However, decoders are frequently computationally intensive to run, not all the constraints can be tailored to the particular application.

### 2.6. Parallel architecture

GAs could work in parallel architecture and three main systems can be used [21]:

- **The Global Parallel** [22] treats the population as a single unit and assigns different individuals to different processors. In this simplest form, the approach uses one machine or processor to control one part of the program, and a series of other processors to control the other part.

- **The Island Model** where several isolated subpopulations evolve in parallel, periodically exchanging by migration their best individuals with the neighbouring subpopulations.
- The *Fine-Grained Model* (or neighbourhood model) where a single population evolves, each individual of which is placed in a cell of a planar grid; selection and crossover are applied only between neighbouring individuals on the grid (according to a pre-defined neighbouring structure).
CHAPTER 3.- DATA ANALYSIS

3.1 Initial assumptions and internal processes

In order to give an objective result after the techniques applied, we have considered to use the same set of dataset and neural network model for training and validation in every single different technique. This assumption will be based on the following initial considerations:

As explained in chapter 1, the data base is based on 12 sensors with 10 features extracted from the sensor response which means 120 different features. The total number of measurements is 96, therefore the data base is a matrix of 96 rows (measurements) and 120 columns (features). This data base will be split in two different matrices (selection and validation matrix, respectively). In order to keep independent the selection and validation data sets, any normalisation in the selection set will be based exclusively on the data from this set.

Although the measurements are randomised, we split the data in even rows (validation set) and odd rows (selection set). The "target" will be a classification in 6 different groups corresponding to 3 single vapours and 3 binary mixtures and ignoring the concentration.

The selection matrix will be normalixed using a mean centre, so every data in the matrix is subtracted the mean value within its column. The validation set will be also normalised using the mean values of each column computed by the selection matrix.

The classifier will be built using a Fuzzy ARTMAP network. This is based on a supervised analysis which the net learns from input and output data. The reason for using this method is based on its main characteristics:

- Fast learning with a low computation time.
- No need to have large samples for training. This is interesting when the set of samples is difficult to get. Take into account that in some other techniques is a must to have at least the same number of samples as features in the data set.
- Keep learning new categories without forgetting previous learning.
- In comparison with other neural networks, fuzzy artmap calculates automatically the number neurons in hidden layer.
- Learns 100% of the training set.

- It is a method open to be internally analysed in order to extract data from the internal process and how they have influenced in the categorisation of the results. This last point is the main reason why we have chosen this method. The high amount of set of features which leads to the same solution made us think it was needed another factor to distinguish which set of variables was better than another with exactly the same effectiveness after applying the validation set.

Internally the Fuzzy Artmap classifier uses a parameter called "wija". This parameter says the number of internal categories the net has created during the training phase. Ideally this parameter should have the same number of real categories, but actually the net needs to create many more. Therefore, if we have two sets of solutions with same result (effectiveness in the validation set), we can use the "wija" parameter to know which one had less difficulties in creating the net.

During the selection process we will use a crossvalidation in groups of 6 measurements instead of using the "leave-one-out" approach. This is a way to dramatically reduce the computation time while keeping accuracy and effectiveness high.

The initial data to start comparing results is the rate of correct classifications for the validation set considering 100% of the features (120). In this case the success rate is **85.4167%**.

Every technique used to decrease the number of features will be compared to that rate. Take into account that the aim of this project is not to improve the result but compare the consequences to reduce the number of features when creating the model and the time needed to find out the right features.
3.2 Backward elimination

The flow to run this program is as follow:

![Diagram](image_url)

**Figure 3.1**

<table>
<thead>
<tr>
<th>Initial variables</th>
<th>Features chosen</th>
<th>Success Rate</th>
<th>Computation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>120 (All of them)</td>
<td>1 2 5 6 9 10 11 12 13 14 15 16 17 18 22 23 26 27 29 30 35 37 42 52 54 57 58 59 62 85 90 99 100 101 117</td>
<td>91.66%</td>
<td>5 hours 37 minutes</td>
</tr>
</tbody>
</table>

The features selected are reduced to 35 improving the classification in 6% of success. Even if the amount of features is approximately 1/4 of the initial variables, the reduction is far away from what we consider should be the outcome of a good variable selection technique without reducing the classification success rate. As we commented in chapter 1, the main disadvantage of this technique is to fall in a local optimum, if we consider optimum the minimum in the classification error during the crossvalidation process. Taking into account the result using other techniques, we can assure that the program is stuck in a local minimum. We will also see that the time required to run the program is high, but we will be able to use Backward Elimination and take profit of their performance integrated with the genetic algorithm.
3.3 Forward Selection

The flow to run this program is as follow:

Initial variables = 1
Features chosen = 18 53 56 66 72 84 88 89
Success Rate = 70.83 %
Computation time = 32 minutes

The result shows clearly that the algorithm has been trapped in a local optimum of the search space. The Stepwise Selection method will be in this case very useful to skip at least this local minimum but like the rest of the determinisitic methods it can be trapped in another one. We will not consider this method as the best one in computation time since the success rate is to low to be comparable to the rest of the methods.
3.4 Stepwise selection

The stepwise selection executes sequentially backward elimination and forward selection during a number of loops selected. There is no reason to keep running the program as soon as the selected variables can not be increased or decreased using forward selection or backward elimination respectively, so we will consider the computation time until the variables selected can not be "improved".

**Initial variables** = 120

1.-Backward elimination

1     2     5     6     9    10    11    12    13    14    15    16    17    18    22    23    26    27
29    30    35    37    42    52    54    57    58    59    62    85    90    99   100   101   117

**Success Rate** = 91.66%

2.-Forward selection

1     2     5     6     9    10    11    12    13    14    15    16    17    18    22    23    26    27
29    30    35    37    42    49    52    54    57    58    59    62    64    85    90    99   100
101   105   112   116  117

**Success Rate** = 91.66%

3.-Backward elimination

1     6     9    10    11    13    14    16    17    18    22    23    26    29    30    35    37    42
49    52    54    57    59    62    85    90   100   101   105   112   116

**Success Rate** = 91.66%

4.-Forward selection:

1     2     6     9    10    11    13    14    16    17    18    22    23    26    29    30    35    37
42    49    52    54    57    59    62    85    90   100   101   105   112   116

**Success Rate** = 91.66%

**Total computation time** = 6 hours 4 minutes

The features selected are reduced to 32 without improving the classification success rate compared to the *Backward Elimination*. Although in this case the *Stepwise Selection* is not reducing drastically the set of features selected for the model (compared to the *Backward Elimination*), it was able to skip the local optimum and be stuck in another one reducing the number of features in 3 compared to the *Backward Elimination*.

The computation time will obviously be longer than in *Backward Elimination*. 
3.5 Genetic Algorithms for features selection

Before introducing the results obtained we will explain briefly how the genetic algorithm has been programmed.

3.5.1. Representation and Operators

In this section we will explain how we have represented each chromosome for encoding candidate solutions to be manipulated by the genetic algorithm.

Each individual in the population represents a candidate solution to the feature subset selection problem. Let be the total number of features available to choose from to represent the patterns to be classified (120 in this case). The individual (chromosome) is represented by a binary vector of dimension 120. If a bit is a 1, it means that the corresponding feature is selected, otherwise the feature is not selected. This is the simplest and most direct representation scheme. The initial population will be randomly selected considering all the features (120) with a percentage of features initially in the population at 50%.

Since we are representing a chromosome through a binary string, the operators mutation and crossover operate in the following way: Mutation operates on a single string and generally changes a bit at random. Thus, a string 11010 may, as a consequence of random mutation gets changed to 11110.

Crossover on two parent strings produces two offsprings. With a randomly chosen crossover position 4, the two strings 01101 and 11000 yield the offspring 01100 and 11001 as a result of crossover. In our application will be used a 2 point crossover.

3.5.2. Parameter Settings

Our experiments used the following parameter settings:

- Population size: Maximum 256
- Number of generation: i.e. 200
- Crossover: Two point
- Probability of mutation: 0.005

The parameter settings were based on results of several preliminary runs. They are comparable to the typical values reported in paper [24]. Later on these parameters will be modified in order to achieve lower computation time.
3.5.3.-Selection Mechanism

The selection mechanism is responsible for selecting the parent chromosome from the population and forming the mating pool. The selection mechanism emulates the survival of the fittest mechanism in nature. It is expected that a fitter chromosome receives a higher number of offsprings and thus has a higher chance of surviving on the subsequent evolution while the weaker chromosomes will eventually die.

In this work we are using a ranking selection. Basically the algorithm will breed the best half of population and replace worst half.

We will also keep the best chromosome within the population (elitism) in order to prevent the GA losing the best solution found in the previous generation.

3.5.4.-Objective Function and Fitness Evaluation

The fitness evaluation is a mechanism used to determine the confidence level of the optimised solutions to the problem.

Usually, there is a fitness value associated with each chromosome, e.g., in a minimisation problem, a lower fitness value means that the chromosome or solution is more optimised to the problem while a higher value of fitness indicates a less optimised chromosome.

Our problem consists of optimising (minimising) the classification error after a continuous 6-block crossvalidation using the selection set of measurements.

We are validating with 48 measurements, which means that we can get 49 different results or fitness (0-100 % classification error). The low amount of different levels of fitness makes the GA have less accuracy and therefore efficiency since there are many different combinations with the same fitness. One way to solve that is to rank the chromosomes, selecting those with fewer features in their genes in case of having the same fitness value. After applying this second condition we saw there were still too many combinations (chromosomes) with same fitness and number of features selected. It is here when the third factor will distinguish between these two or more chromosomes. As we explained before, the parameter "wija" will give us internal data when the Fuzzy ARTMAP classifier creates the net. Then, we will choose the one with the "wija" parameter lower since it had fewer difficulties in creating the net.

3.5.5.- Termination criteria
Initially the termination criteria will be based in a maximum number of generations. The genetic algorithm will be able to stop before when the maximum percentage of duplicated population overpass 80%.

### 3.5.6. Program Flow

As we previously said the features selected by the selection measurements will be validated with the validation measurements.

![Flowchart](image)

#### Figure 3.3

### 3.6 Genetic Algorithm execution

Initially we will run the GA with maximum population possible (256) and a large number of iterations (250).

**Set up GA**

- Population = 256
- Maximum generation = 250
- Mutation = 0.005
- Two point crossover

**Features chosen** = 40 64 66 92 98

**Success Rate** = 85.4167%

**Computation time** = 39 hours
The GA coupled to a Fuzzy ARTMAP classifier is applied to select between the 120 initial variables stopping when the maximum generation is reached. See evolution of the GA in the following graphic:

![Figure 3.4](image)

During the first generations the winner chromosome is varying practically at every generation. It is also in the first generations when the population goes down exponentially in the number of variables within the population since we are selecting those chromosomes that have fewer variables if they have better fitness.

The genetic algorithm selects 5 out of 120 features. The rate of correct classifications for the validation set is 85.4167% which means that taking only the 4.2% of the initial features we are keeping the same correct classifications considering all the features.

Since we have already developed the deterministic techniques for variable selection, we can apply them over the result obtained by the GA and check out if an improvement in the success rate in classification is possible.

**STEPWISE SELECTION**

1.- Backward elimination

**Features chosen** = 40  64  66  92  98

There is no variable improving the selection model.

**Success Rate** =85.4167%
2.- Forward selection

Features chosen = 40 58 64 66 85 87 92 98
Success Rate = 93.75%

The backward elimination is not able to get a better result, which indicates that the GA is reaching a much optimised solution. On the other hand the forward selection is able to find 3 variables more which improve the success classification rate in the selection set. The improvement is also reflected with the validation set and the success rate is raised up to 93.75%, the best result obtained compared with any other technique. This demonstrates that the genetic algorithm had not reached the global minimum yet, but the huge computation time until generation 250 (39 hours) makes impracticable to leave the GA running until getting the global minimum. In this case we can consider the Stepwise selection a good method to improve the selected features after the GA and it will be used in next tests in cascade.

3.7 Computation time optimisation in GA

Different authors have addressed their investigations in decreasing the time to run the GA. Here you can see some examples applicable to our problem resolution:

- When applying genetic algorithms to the selection of principal components, the author try to choose the principal components without taking into account their "eigenvalue". That means that the PC's with lower eigenvalue can be necessary if they keep a big correlation with the independent variable. In order to decrease the computation time, a maximum number of PC's can be chosen since probably the rest only bring "noise". In general all the investigators try to put a limit on the maximum variables that will be in the final model, independently of the GA application.

- To put a limit in the GA objective function [12]. The author makes an estimation in which value the "fitness" should be. Every solution over passing this value will not be considered to be inside the GA.

- To modify the termination criteria. In this specific application [12] the GA evaluates the variance in the fitness for the best 20% of population members. If
there is no change in variance beyond a small tolerance level over 10 generations the computations are stopped.

- Genetic algorithms in cascade. Initially the variables or features that not include information are eliminated and then another GA with the variables left is applied. In some papers the last step is to apply a Stepwise selection instead of a GA.
- To run GA’s in parallel. Different GA are executed and interchanging chromosomes among them.
- To put a maximum number of chromosomes in the population. This alternative is very subjective to the application, number of variables and/or characteristics. We can also modify the crossover and mutation factors to get a quicker convergence, but the effectiveness is also very subjective to the application.
- To favour the chromosomes with less variables (genes). It is not only one of the goals of the problem but it is also helping to get a quicker execution.

Analysing the evolution of the GA already executed we can get some ideas how to apply some of the points mentioned above.

3.7.1.- Advance the termination criteria

The termination criteria is in this case too long since the features chosen in iteration 134 will not change anymore. On the other hand the result obtained by the GA was improved applying a stepwise with the GA final result. That means that the result obtained by the GA was not the optimum yet.

See the evolution of the GA and iteration number where the features selected have changed:

| ITER 50: 23 | 40 | 51 | 57 | 58 | 66 | 75 | 88 | 90 | 97 | 98 | 101 | 110 | 114 | 117 |
| ITER 52: 23 | 40 | 51 | 58 | 66 | 75 | 88 | 90 | 97 | 98 | 99 | 101 | 102 | 114 |
| ITER 55: 24 | 40 | 51 | 57 | 58 | 66 | 75 | 85 | 88 | 90 | 92 | 98 | 99 |
| ITER 58: 14 | 27 | 40 | 54 | 66 | 71 | 74 | 85 | 88 | 89 | 90 | 92 | 98 | 99 |
| ITER 59: 24 | 32 | 40 | 54 | 66 | 71 | 74 | 85 | 88 | 89 | 92 | 98 | 99 |
| ITER 60: 24 | 40 | 51 | 57 | 58 | 75 | 85 | 88 | 90 | 92 | 98 | 99 |
| ITER 61: 24 | 40 | 57 | 58 | 75 | 85 | 88 | 90 | 92 | 98 | 99 |
| ITER 63: 23 | 58 | 66 | 85 | 88 | 90 | 98 | 99 | 100 | 101 | 114 |
| ITER 64: 23 | 40 | 51 | 58 | 66 | 75 | 88 | 90 | 92 | 98 |
| ITER 67: 23 | 40 | 58 | 66 | 85 | 87 | 88 | 89 | 92 | 98 |
| ITER 68: 23 | 40 | 51 | 58 | 66 | 88 | 89 | 92 | 98 |
| ITER 77: 40 | 51 | 58 | 66 | 88 | 89 | 92 | 98 |
| ITER 81: 23 | 40 | 58 | 66 | 88 | 89 | 92 | 98 |
| ITER 88: 40 | 51 | 57 | 88 | 89 | 92 | 98 |
ITER 92: 40  54  57  88  92  98  99
ITER 95: 23  40  58  66  88  98  99
ITER 99: 40  54  57  92  98  99
ITER 126: 40  57  61  88  98  99
ITER 128: 23  40  64  66  92  98
ITER 132: 40  60  64  66  92  98
ITER 133: 40  64  66  92  98
ITER 134-250 : No changes

Getting data from generation 50, we can see how the best chromosome has changed from generation to generation. At generation 50, the success rate using the validation set was 87.5%. This is even better than the result after the generation 250 but there are still too many features to create the net.

The main idea we get from this analysis is that most of the final features selected are already included in this chromosome at this generation. Applying a Backward Elimination to this chromosome, we can improve the model and number of features selected.

**Backward elimination from ITER 50**

Features chosen = 23  40  57  58  66  88  90  97  98  101  110  114  117
Success Rate = 93.75 %
Total Computation time (GA “gen. 50” + stepwise) = 7 hours 50 minutes

Even that we only reduce the features selected in two, we get a good success rate in the validation set. This test demonstrates that we can always stop the GA in a medium state, when the algorithm is already oriented to the global minimum, even if it has not arrived yet.

**3.7.2.- Decrease the members in the population**

This is a direct way to decrease proportionately the computation time since there are less possible solutions to cross-validate. Since we are working with a large member in the population (256) we can consider to reduce it to 48, thus the computation time goes down to 5.33 times less.

**Set up GA**
Population = 48
Maximum generation = 100
Mutation = 0.02
All variables can be chosen
Genetic Algorithms and Time Consuming Optimisation

Features chosen = 10 18 25 65 81 90 92 98
Success Rate = 85.4167%
Computation time = 3 hours

Stepwise Selection

1.- Backward elimination

Features chosen = 10 25 65 81 90 92 98
Success Rate = 89.5833%

2.- Forward selection

Features chosen = 10 21 25 61 65 81 83 89 90 92 98 116
Success Rate = 75%

3.- Backward elimination and local minimum

Features chosen = 10 21 61 65 81 83 89 90 92 98 116
Success Rate = 79.1667%

Total Computation time = 3 hours 30 min

The success rate is quite good after the GA, but after applying a Stepwise selection the success rate decreases to a lower value compared with the remaining methods. The fact to work with a large range of variables makes important to have a good representation in the population to get into the right fitness solution space. In the last test the backward elimination is stuck in the wrong solution space, so we understand the significance to have a good representation in the first generations and we will not consider to decrease the members of the population when the GA has just started, but it looks interesting when the number of genes in the chromosome is not so large, for instance when the search space is already focussed.

Next step can be to decrease the number of members in the population after a stated generation. We will call it "adaptive" population.

3.7.3.- Adaptive population

We will start working with the maximum members in the population (256) and we will decrease it to 64 after generation 30.
Set up GA  
Population1 = 256 (until generation 30)  
Population2 = 64 (from generation 30)  
Maximum generation = 200  
Mutation = 0.005

**Features chosen** = 18 44 61 65 71 88 98  
**Success Rate** = 87.5%

After applying a Backward Elimination the chosen variables does not change.

Taking the variables chosen in generation 50:

**Features chosen (gen. 50) =** 3 18 19 36 44 53 58 59 61 71 74 84 88 90 102 105 116 117  
**Success Rate** = 93.7500%

STEPWISE

1.- Backward elimination  
3 18 36 44 53 58 59 61 71 84 90 98 102 105 116 117  
**Success Rate** = 97.92%

2.- Forward selection  
3 9 18 36 44 53 58 59 61 71 84 88 90 98 102 105 116 117  
**Success Rate** = 95.83%

3.- Backward elimination  
3 9 18 36 44 53 58 59 61 71 84 90 98 102 105 116 117  
**Success Rate** = 95.83%

4.- Forward selection  
3 9 18 36 44 53 58 59 61 71 84 88 90 98 102 105 116 117  
**Success Rate** = 95.83%

This last test summarises all the techniques we have used so far to reduce computation time. The resulting success rate is the highest got so far. The time used to run GA + Stepwise is 5 1/2 hours and the total selected features 18.  
This is approximately the same time the Backward Elimination needed to find the optimum solution, but using half of the features that the Backward Elimination needed.
3.7.4 Collinear and noisy features

The GA itself is supposed to get rid of those noisy features, which actually do not carry informative data or they provide no additional explanatory effect beyond those variables already included in the model. However different papers [26] have stated that the GA does not prevent from selecting meaningless variables and the solution found should be investigated carefully.

The idea is to eliminate first the collinear and noisy features before running the GA, thus guide the GA to the features that initially contain real information. The dimension of search space will be in this way decreased as well as the computation time to run the GA.

We will start removing the noise features. The procedure is as follow: Initially it is known the category every measurement belongs to, so we will be able to calculate for a specific category and for each one of the variables the average and the standard deviation of that variable and thus how repeatable is this value inside the same category (low standard deviation). Separately if we calculate the standard deviation among the averages we get the standard deviation between categories. The higher value means the higher difference between categories. We will use a factor $F = \frac{\text{Std Dev Intra-Cat}}{\text{Std Dev Inter-Cat}}$ which will define which features are liable to contain noise (the lower factor the higher possibility to contain noise).

The previous result will not avoid removing the collinear variables, in fact two features collinear should be chosen at the same time. In order to get rid of the collinear variables, we will normalise the array of remaining features and we will calculate the scalar product for all the combinations. The ones with values close to 1 are liable to be collinear, thus the ones closer to 0 are mutually independent and favourable to be chosen.

After applying these methods:

1-Features chosen = 12 16 21 24 26 27 28 33 34 36 37 38 42 45 46 48 52 57 60 62 63 64 69 70 72 73 78 81 84 88 101 102 104 108 113 120

Success rate = 89.58 %
We run the GA (notice that now we do not need a large amount of members in the population since the search space is smaller). On the other hand we will have to increase the mutation index, to avoid reaching too quick the maximum number of duplicated members in the population and try to explore all the search areas.

**Stepwise Selection**

Features chosen = 3 16 21 22 32 33 34 38 42 51 52 58 62 66 70 85 88 102

Success Rate = 93.75%

Total computation time = 1 hour 20 minutes

**Genetic Algorithm + Adaptive Population + Stepwise**

Starting with the features left after applying the “Scanning”, we will run a Genetic Algorithm with “adaptive” population size and a Stepwise Selection. It is clear that now the search space is not so large and since we have reduced the number of initial features to approximately 1/4, we will do the same with the population size.

**Set up GA**

Population size = 64
Population1 size = 64 (until generation 30)
Population2 = 16 (from generation 30)
Maximum generations = 50
Mutation Rate = 0.01

Features chosen = 12 26 28 45 46 62 73 78 88 102 104 108
Success Rate = 93.75%

Total Computation time = 2 hours 40 minutes

We have got a method mixing all the consumption time reduction which can get a good success rate, time consumption and number, but is not the best in any of the targets. Anyway this summarises all the methods we have used so far and is the method to be used when the user wants to have a good compromise between the computational time, success rate and number of features selected.
3.7.5 Hybrid GA - Backward elimination

The idea to integrate a Backward Elimination inside the GA is extracted from a previous investigation [26]. The method consists to choose a random chromosome inside the population in a given frequency chosen by the user and apply a backward elimination. The original chromosome is only replaced if the result obtained after applying the Backward Elimination is better than the original. It is out of question to apply the Backward Elimination in the early generations since the computation time will be exponentially increasing, but it could help when the number of features in the chromosome is lower.

The initial variables are the ones obtained after eliminating collinear and noisy features.

Features chosen = 2 24 27 57 73 88 90 97 100 102 104 108

Success rate = 83.33%

Total computation time = 6 hours

Genetic Algorithm Evolution

Figure 3.5
The fact to include a backward elimination in random chromosomes could help to get a faster approach to the global optimum since we try to decrease the number of features in the selected chromosome. However, results demonstrate that it is not worth since the computation charge increase too much with no significant improvement in the success rate, but still keeping the same one obtained when all the features are considered.

As a final test we wanted to accelerate as much as possible the convergence to the global optimum based on the idea of the hybrid algorithm. Instead of applying a Backward Elimination to a random chromosome, we have applied a Backward to the best four chromosomes in the population for each iteration. To avoid increasing too much the computation time, we have start running the backwards from iteration 30 on.

Additionally, another technique not used so far is to modify the selection function. As explained in Chapter 2, the roulette-wheel method is the one that allows a quicker approach to the global optimum since the best chromosomes will be the ones with more probabilities to have offspring. Considering that the best chromosomes will be also the ones that we have applied a Backward Elimination, the number of features in the model will also decrease quicker.

**Figure 3.6**
The evolution of the Hybrid Genetic Algorithm shows that the fact to use the Roulette-Wheel selection do not help for a faster approach to the global optimum. The reason is that we have used the classification error after the crossvalidation as a fitness to assign probabilities in the Roulette-Wheel, and we have ignored the number of features in the chromosome and the “wija” parameter. That means that all the chromosomes with the same classification error will have the same probability to be chosen, thus we are not favouring the chromosomes with less variables and this is reflected in the graphic “Evolution of number of variables” where the number of variables is very high compared to the GA’s run previously. On the other hand the fact to use a Backward Elimination to the four best chromosomes decreases dramatically the average of number of features on the chromosomes, but the time consumption to run in each generation is very high. The features chosen after running the Hybrid are = 16 38 45 60 63 88 101 , lower number of features. Anyway the “Stepwise” run after the GA will increase up to similar values obtained in rest of test. The number of features finally obtained is 11 [ 2 4 16 38 45 63 88 96 98 101 119 ]. The success rate after runing the GA and also after running the Stepwise is 89.58% but the computation time has increased up to 8 hours. Sumarizing; the Roulette-Wheel selection does no help in this case to a faster aproximation to the global optimum, but including other factors like the number of variables in the chromosome when calculating the probabilities during the GA selection process and together with the succes rate, the evolution of the GA could be different and profitable for the time consuming. The fact to apply Backward Elimination to the best chromosomes can also be profitable for a faster execution, but taking into account that we have more chances to be trapped in a local optimum since the deterministic method takes an important role inside the GA.
3.8 Discussions and Conclusions

We have developed different programs to run deterministic and stochastic strategies for features selection. In the following table we can resume the results obtained for every single test:

<table>
<thead>
<tr>
<th>Technique</th>
<th>1.- Pop.</th>
<th>1.- Gen.</th>
<th>2.- Pop.</th>
<th>2.- Gen.</th>
<th>Number of variables</th>
<th>Success Rate %</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>120 ( All )</td>
<td>83.33</td>
<td></td>
</tr>
<tr>
<td>Backward Elimin.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>35</td>
<td>89.58</td>
<td>5h 37 min</td>
</tr>
<tr>
<td>Forward Selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>70.83</td>
<td>32 min</td>
</tr>
<tr>
<td>Stepwise Selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td>91.66</td>
<td>6h 4 min</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>256</td>
<td>250</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>85.41</td>
<td>≈39 h</td>
</tr>
<tr>
<td>GA + Stepwise</td>
<td>256</td>
<td>250</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>93.75</td>
<td>≈39 h</td>
</tr>
<tr>
<td>GA + Stepwise</td>
<td>256</td>
<td>50</td>
<td>-</td>
<td>-</td>
<td>13</td>
<td>85.41</td>
<td>7 h 50 min</td>
</tr>
<tr>
<td>GA + Stepwise</td>
<td>48</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>11</td>
<td>79.17</td>
<td>3 h</td>
</tr>
<tr>
<td>GA adap. + Stepw.</td>
<td>256</td>
<td>30</td>
<td>64</td>
<td>20</td>
<td>18</td>
<td>95.83</td>
<td>5 h 32 min</td>
</tr>
<tr>
<td>Scanning + Stepw.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>18</td>
<td>93.75</td>
<td>1 h 20 min</td>
</tr>
<tr>
<td>Scan+GA+Stepw.</td>
<td>64</td>
<td>30</td>
<td>16</td>
<td>20</td>
<td>12</td>
<td>93.75</td>
<td>2 h 40 min</td>
</tr>
<tr>
<td>Scan+Hybrid+Stw.</td>
<td>64</td>
<td>30</td>
<td>16</td>
<td>20</td>
<td>12</td>
<td>83.33</td>
<td>≈6 h</td>
</tr>
</tbody>
</table>

Table 3.1

All the methods have been found to be useful for variable selection since we obtain equal or better classification results than when considering all the features available are used. The way to distinguish which ones can work better for feature selection is to consider as target the computation time, the number of variables selected and the success rate after the validation.

As first step we have implemented the deterministic methods which are characterised by their relatively low computation time. As previously known, the deterministic methods (backward elimination, forward selection and stepwise selection) can be trapped in a local optimum, which explains that the number of features selected is still high in the Backward and Stepwise (35, 32 respectively) and low in the Forward. The success improves to rates up to 91.66% which demonstrates the efficiency to delete those features that actually do not contribute to a good selection and probably only bringing disturbances or noise.
The fact that these methods are good in getting an optimum when the search space is getting more narrow made us think that we could include them in cascade when some other method is applied.

Second step was to develop a Genetic Algorithm. It is astonishing that the number of features finally selected can be even decreased down to 5 variables. Considering that we have been validating with a dataset not included in the training set, we must deal carefully with this low number which could not represent all the input data set and to have deleted features needed for a good success rate with the validation set. Nevertheless the success rate improves compared to the one considering all the features and only using the 4.166 % of the initial variables. On the other hand the computation time increases drastically and next methods have been addressed to decrease it.

The first combination GA plus Stepwise in cascade give us an improvement in the success rate (93.75%) and only increasing the number of features in three. The index (number of features-success rate) is probably the best one obtained but it is still pending to decrease the computation time since we have added more including the Stepwise in cascade.

The most logical and quicker way to decrease the computation time is to modify the GA parameters number of chromosomes in the population and maximum generations. The number of chromosomes in the population is a factor which has been in discussion by many investigators. The correct number should be the one that keeps a good representation of the search space. Since we started with the maximum quantity possible, we decreased the population to 48 members. The result obtained was not successful since the success rate went down to 79.17% and another solution should be applied. Investigating the curves obtained after running the GA showed that it is during the first iterations when the number of the features in the chromosome is decreasing drastically (from 120 up to 15 in generation 50) which means that a good representation is necessary at the beginning. But as soon as the number of genes in the chromosome is reduced together with the search space, the number of chromosomes could be also decreased. The method called “adaptive population” reduces the number of chromosomes when the GA overpasses a specific iteration.

The second GA parameter “maximum number of generations” affects proportionally the computation time. The result obtained after running the GA until generation 250 showed that a long GA run decreased the number of features to a very
low levels, even too much after seeing that the Stepwise applied in cascade increase the number of features instead of decreasing, in other words, the Backward Elimination can not find initially any better solution but the forward selection is able to. The main idea is to stop the number of generations in an early stage when the evolution of the GA seems to have found the way to a global optimum, and keep the Stepwise to decrease the number of features though Backward Elimination. Applying both methods at the same time we got the best success rate of 95.83% with the validation set, keeping a computation time equal to the Backward Elimination method and decreasing the number of features to 18, a good average between the one obtained running the GA alone and any of the deterministic methods.

Many investigators have remarked that the Genetic Algorithms can not avoid selecting variables that actually do not carry informative data. Their studies have demonstrated that including random variables in the variable set, those could be also selected.

Next step of our investigation was to get rid of those variables that presumably contained noise or collinealities with other variables. Using statistics and defining some factors to keep only the most informative and representative variables we got rid of 84 variables, keeping the rest (36) to create the model. Applying the Stepwise in cascade like in the rest of the methods, we got a success rate of 93.75%, 18 features in the population and the lower computation time of 1 hour and 20 minutes. It is out of doubt that is the best method to decrease the computation time, but it is risky to define the index which will get rid of the noisy variables. Next step was to see the results when a GA is applied starting with the features left after applying the scanning.

Finally we wanted to investigate in methods based on hybrids using GA’s and the Backward Elimination integrated. Even if we know that the fact to include a backward elimination means a computation charge in the training phase, the idea to include a backward elimination in random chromosomes could help to get a faster approach to the global optimum, since we try to decrease the number of features in the selected chromosome. The results demonstrate that it is not worth since the GA itself is already ranking and selecting the chromosomes with smaller number of features and, as main drawback, the computation time is increased with no significant improvement in the success rate.

In conclusion, we have demonstrated that several methods can be used for feature selection. Taking profit of the main advantages of each one, we have been able
to develop a Genetic Algorithm that integrated to deterministic methods and modifying its initial parameters can get excellent results in terms of effectiveness, time and number of features finally selected.

In order to prove the analysis done in the previous studies, we have tested the same data set but in this case classifying in groups of 24 because this is the number of categories corresponding to the combinations between 2 vapors at 4 different concentrations.

1.-Considering all the features (120)
Success rate = 64.58 %

2.-Backward Elimination
Features chosen = 1 2 3 4 5 6 8 9 10 11 12 13 14 15 16 17 18 21 22 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 42 43 44 45 46 50 55 57 60 62 67 72 85 89 90 93 94 95 100 102 112 119
Success Rate = 62.5 %
Computation time = 3 hours 45 minutes

3.-Genetic Algorithm
Features chosen = 23 24 27 43 47 50 60 63 79 86 102
Success Rate = 77.0833 %
Computation time = 39 hours

Figure 3.7
4.- Genetic Algorithm with adaptive population and Stepwise in cascade

Features chosen = 5 10 17 20 21 22 24 28 31 32 40 43 45 50 51 56 59 60 61 62 66 68 73 80 85 86 92 98 100 102
114

Stepwise

5 10 17 20 21 22 24 28 31 32 40 43 45 50 51 56 60 65 85 86 89 90 92 100 102 104 111 116 119

Success Rate = 72.92%

5.- Deleting noisy and collinear variables including a Stepwise in cascade

Features chosen = 24 25 26 27 28 30 33 34 36 45 48 60 63 64 66 69 72 84 88 90 99 102 104 108 113 116 120

Success rate = 81.25 %
Stepwise
2 23 24 25 26 27 28 30 33 45 56 60 64 72 85 88 89
90 92 97 102 116
Success Rate = 81.25%
Total computation time = 1 hour 10 minutes

Summarizing these are the results that confirm the effectiveness of the feature selection methods. In this case the scanning together with a Stepwise in cascade can get a better success rate, but taking into account that we have modified the index defining which variables are noisy or collinears.

<table>
<thead>
<tr>
<th>Technique</th>
<th>1-POP</th>
<th>1.-GEN</th>
<th>2-POP</th>
<th>2.-GEN</th>
<th>Number of variables</th>
<th>Result % Success</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>120 ( All )</td>
<td>64.58</td>
<td></td>
</tr>
<tr>
<td>Backward</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57</td>
<td>62.5</td>
<td>3h 45 min</td>
</tr>
<tr>
<td>GA</td>
<td>256</td>
<td>250</td>
<td>-</td>
<td>-</td>
<td>11</td>
<td>77.08</td>
<td>39 h</td>
</tr>
<tr>
<td>GA adap. + Stepw.</td>
<td>256</td>
<td>30</td>
<td>64</td>
<td>20</td>
<td>29</td>
<td>72.92</td>
<td>5 h 30 min</td>
</tr>
<tr>
<td>Scanning + Stepw.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>22</td>
<td>81.25</td>
<td>1 h 10 min</td>
</tr>
</tbody>
</table>

Table 3.2
4.1 Introduction

To simplify the process to run, see the results, compare techniques, move data and initialise the programs, we decided to create an easy user interface which will allow people not familiar with Matlab language to run the different programs and see the results. In order to create the interface got the help from GUIDE, the MATLAB® Graphical User Interface development environment [27].

This Matlab toolbox provides a set of tools for creating graphical user interfaces (GUIs). These tools greatly simplify the process of designing and building GUIs. We can use the GUIDE tools to:

- Lay out the GUI
- Using the GUIDE Layout Editor, you can lay out a GUI easily by clicking and dragging GUI components -- such as panels, buttons, text fields, sliders, menus, and so on -- into the layout area.
- Program the GUI thorough the .m file that GUIDE will generate automatically. Using the M-file editor, you can add code to the callbacks to perform the functions you want them to.

4.2 Commands

In this chapter we will explain briefly the different commands we have used to develop our interface.

Initially we will open a new graphical interface to be programmed. When we open a GUI in GUIDE, it is displayed in the Layout Editor, which is the control panel for all of the GUIDE tools. You can lay out your GUI by dragging components, such as push buttons, pop-up menus, or axes, from the component palette, at the left side of the Layout Editor, into the layout area. For example, if we drag a push button into the layout area, it appears as in the following figure.
Figure 4.1

The component palette at the left of the Layout Editor contains the components that you can add to your GUI. Elements we have used for our application:

**Push Button**

Push buttons generate an action when clicked, for instance to run an .m file. When you click a push button, it appears depressed; when you release the mouse, the button appears raised and its callback executes.

**Radio Button**

Radio buttons are similar to check boxes, but are typically mutually exclusive within a group of related radio buttons. That is, you can select only one button at any given
time. To activate a radio button, we have to click the mouse button on the object. The display indicates the state of the button.

**Check Box**

Check boxes generate an action when checked and indicate their state as checked or not checked. Check boxes are useful when providing the user with a number of independent choices that set a mode, for example, displaying a toolbar or generating callback function prototypes.

**Edit Text**

Edit text controls are fields that enable users to enter or modify text strings. That means that we can read and write characters within the field.

**Static Text**

Static text controls display lines of text. Static text is typically used to label other controls, provide directions to the user, or indicate values associated with a slider. Users cannot change static text interactively and there is no way to invoke the callback routine associated with it.
Slider

Sliders accept numeric input within a specific range by enabling the user to move a sliding bar, which is called a slider or thumb. Users move the slider by pressing the mouse button and dragging the slider, by clicking in the trough, or by clicking an arrow. The location of the slider indicates a percentage of the specified range.

Pop-Up Menu

Even that this command has not been used in our application, it is interesting in the case we used check-boxes which were not activated at the same time. Pop-up menus open to display a list of choices when users click the arrow.

After selecting these components for our GUI and placing them in the layout area, we need to set their properties and program their callbacks. You can also use the Layout Editor to set basic properties of the GUI components.

![Property Inspector]

Figure 4.2
We will need to program some of these properties before running the interface. The main one are:

**Tag**

The Tag property is an identifier for the component. GUIDE assigns a value to the Tag property of every component you insert in your layout (e.g., pushbutton1), but it is important to assign a name that it will be easily identified by the programmer when using it. This will avoid mixing calls when a high amount of commands have been used. GUIDE uses the Tag property to:

- Construct the name of the generated callback (e.g., close_button_Callback) when you run or save the GUI
- Set corresponding callback properties to point to the callback.
- Add a field to the handles structure containing the handle of the object (e.g., handles.close_button)

**String**

The String property contains text for the component. For buttons, check boxes, list boxes, edit text, and static text, the String text is displayed on or next to the component. For an edit text, the String property contains a list of strings that is displayed in the text box. When a user edits the text, the String property is updated.

**Value**

The Value property contains a numerical value for the component, which must lie in the range specified by the Max and Min properties. For radio buttons and check boxes, Max and Min are set to 1 and 0, respectively, by default. The Value property is set to 1 when the radio button or check box is selected and 0 when it is cleared. When a user drags a slider, the Value is set to a number between Max and Min corresponding to the location of the slider button.
Callback

The Callback property specifies the callback that is executed in the GUI M-file when a user activates the component.

Components use callbacks to do their work. A callback is a function that executes when a user performs a specific action such as clicking a push button, selecting a menu item, or pressing a keyboard key, or when a component is created or deleted. Each component and menu item has properties that specify its callbacks. When you create a GUI, you must program the callbacks you need to control operation of the GUI. A component can have many callback properties, but the most common one is the Callback property. The code you provide for the Callback property performs the primary work of the component. It executes, for example, when a user presses a push button, moves a slider, or selects a menu item.

Until now we have seen how to create or run commands separately. But one of the purposes of our application is to combine different features selection programs, which means sharing data between action "callbacks". For that reason will be necessary to create variables which will contain values visible for all the callbacks.

When you run a GUI, the M-file creates a handles structure that contains all the data for GUI objects, such as controls, menus, and axes. The handles structure is passed as an input to each callback. You can use the handles structure to

- Share data between callbacks
- Access GUI data

Sharing Data

To store data that is contained in a variable X, set a field of the handles structure equal to X and then save the handles structure with the guidata function:

- handles.current_data = X;
- guidata(hObject,handles)
- ....
You can retrieve the data in any other callback with the command:

- \( X = \text{handles.current
data}; \)

### 4.3 Interface design

Our interface will be able to run different features selection techniques, share the variables obtained between them, apply different time reduction techniques, show the results and validate the set of features finally obtained. Initially we have split the interface in the deterministic methods (forward selection, backward elimination and stepwise selection) and Genetic Algorithms (Stochastic Method).

When running any of the deterministic methods, we have to point out which set of data to start working with. For that reason we have created three radio buttons which are incompatible between them:

"All Initial Features" contains initially all the 120 features in the model, "None Initial Features contains no features which means an array of zero elements, and finally "Features Selected" will contain only the features that we have obtained by running one of the other features selection techniques.

Since they are incompatible between them, we have to avoid that the user can choose more than one option at the same time. The "handles" will help us to read/write to all the radio buttons affected when one of them has been selected. When one of the radio buttons is pushed, its callback function will set to "0" the "value" for the rest of radio buttons affected. The function "set" will write over them:

\[
\text{set(handles.radiobuttonX,'Value',0)}
\]

It was also possible to create a "pop-up menu" which automatically avoids this problem.

Notice that when running a Backward Elimination or Stepwise, the radio button "None Initial Features" can not be selected. On the contrary the radio button
"All Initial Features" can not be selected when Forward Selection is going to be ran. "Features Selected" radio button has been selected to be the as defect option.

Looking at the Genetic Algorithms execution a more complicated interface has been done. First of all we need to initialize the GA parameters.

With this window we will be able to set the initial GA parameters such as population size, maximum generations with the GA running, the mutation rate and the crossover type.

Notice that the minimum and maximum levels to be used in the "slides" are stated on both sides of the slides by mean of static text. Just beneath the sliders there is an editable text within the value chosen by the slide that means that as the slide is moving the editable text is updated with the value.

When choosing the Population Size only values multiple of four are considered. The program itself avoids than any other value is written. In the same way as the radio buttons used before, the crossover options are incompatible and the interface will make sure that this statement is accomplished.

Before running the GA we need to consider if one of the techniques to reduce the running time is going to be used. In this case all three techniques are compatible thus they can be chosen at the same time. After these considerations the GA can be ran pushing "RUN GA".

The results (features selected to create the model) will be displayed on the editable text automatically as the program (Backward, Forward, Stepwise or GA) has
ended up. We needed to take into account that the editable text is supposed to be a string, thus the numerical value must be converted:

\[
set(handles.editX,'String',num2str(resultatvariables))
\]

The result (features selected) obtained is saved in order to be used in cascade by another feature selection technique.

Finally and with the features obtained we can validate with the validation data set and show the result (Success Rate on the screen):

![Figure 4.6](image)

The complete "interface" is shown in the following graphic:

![Figure 4.7](image)
CHAPTER 5.- FURTHER WORK

Regarding the computation time optimisation it is pending to run parallel algorithms. The idea is to split the Genetic Algorithm program in different processors. As explained in Chapter 2, different topologies can be used, for instance to assign different individuals to different computers or processors or to split the population in different subpopulations and interchange periodically the best individuals to the different subpopulations. It is quite easy to predict that the computation time will decrease proportionally with the number of processors that have been used.

Different techniques modifying the GA parameters have been applied to decrease the time but we could also modified the way to deal with the data internally in the GA. We considered the fitness the success rate after a crossvalidation, but some other investigators have been modifying the fitness function in order to prioritize the chromosomes with less features, delete noisy features and avoid overfitting problems. A deep analysis can be done in order to know if same results are obtained.

The numbers of measurements (96) are not enough to include stop functions on the way other investigators have done. In case to get more measurements for the training set, we can have more accurate fitness value and apply also these techniques. For instance to evaluate the variance in fitness for the best percentage of population members. If there is no change in variance beyond a small tolerance level over a generation number the computations are stopped.

We observed that actually the quickest and one of the methods with best success rate is the one we get rid of noisy and collinear variables. We must handle carefully with this method since the result depends of the index we have put to take only the informative data. Deeper investigations can be done in order to get a method which can get good results on the success rate point of view running with different measurements.
CHAPTER 6.- REFERENCES

[1] Apunts Sistemes de Percepció – Automàtica i electrònica industrial-
[3] University of Warwick/School of engineering/Sensor Research Laboratory/Electronic nose. www.eng.warwick.ac.uk


APPENDIX

Software:
The genetic algorithm for variable selection has been inspired on standard routines from the PLS_Toolbox developed by Eigenvector Research Inc. The programs have been run using a PC AMD Athlon(tm) 2400 XP+, 2.01 GHz, 512 RAM.

Matlab User Guide:
In a PC, from any Matlab directory, type “projecteGA” to start the guide user interface.
This interface has been developed to run the deterministic methods (Backward Elimination, Forward Selection and Stepwise Selection) and the Genetic Algorithm.
The initial variables to run the deterministic methods can be selected using the radio buttons on the left. Take into account that it is the first time the user is going to run the program, there will not be any variables selected on the screen.
Once we have already run one of the methods, “variables selected” can be used to run another method.
When running a Genetic Algorithm the initial variables will be all of them. GA parameters and functions can be modified using the radio buttons and sliders on the screen.
When whatever method has been run and stopped, the variables selected will be automatically printed on the screen.
The user can push Run Validation to know the success rate for these features selected.
### EXECUTE BACKWARD ELIMINATION

```matlab
function [resultatvariables]= executa_backwardeliminationGUI(varselect)
%close all
%clear all
clc
matinput=varselect;
%Carreguem la matriu de dades . En aquest cas treballarem amb la matriu "xgascomp1"
%que està formada per 96 mesures X 120 parametres. Com a matriu de targets tenim "yprueba"
%classificarem les 96 mesures en 6 categories (3 gasos simples i 3 mescles binaries
%independent de la concentracions.
load C:\matlabR12\work\datososcar.mat
[mostres,variables]=size(xgascomp1)
%Per a l'elecció de les variables treballarem amb la meitat de les mostres. Una vegada
%escolides les variables testearem amb l’ altre meitat
%Els imparrells son per al conjunt de prediccio i els parells per la validacio
for i=1:48
indexpar(i)=2*i;
indeximpar(i)=(2*i-1);
end
%Matriu de prediccio per escolhir les variables
[datos,meandatos] = mncn(delsamps(xgascomp1,[indexpar]));
target= (delsamps(yprueba,[indexpar]))/max(yprueba);
x=datos;
y=target;
matriuvar=zeros(1,120);
[n_fil,n_var]=size(matinput);
for i=1:n_var
  matriuvar(matinput(i))=1;
end
[variablesselection]=backward_elimination(x,y,6,matriuvar);
%Retornem les variables escollides
resultatvariables=find(variablesselection);

function [variables_sel]=backward_elimination(x,y,grups,array_var)
%eliminem les variables a utilitzar
clear data_in;
clear data_out;
clear n_samples;
clear n_variables;
```
clear split;
clear array_varselec;
clear bestindex;
clear acabar;
clear inds;
clear xx;
clear yy;
clear trobat;
clear variables_sel;

%inicialitzem variables
data_in=x;
[n_samples,n_variables]=size(x);
data_out=y;
split=grups;
array_varselec=array_var;
bestindex=0;
acabar=0;

%Partim del valor de "fitnesses" del conjunt de variables incials que li passem al %programa. El programa
%Backward elimination s'enarrega de disminuir aquest "fitnesses" (error obtingut de %la validacio creuada i el wija)
inds = find(array_varselec); %troba l’ index de totes les variables que son 1.
xx = data_in(:,inds); %crea la matriu amb les variables escollides
yy = data_out;
[bestpressant,bestpressant2]=crossvalidationfuzartmap(xx,yy,split)

while (acabar==0), %si no hem trobat cap variable que disminueixi el valor del error %de la validacio creuada del cicle anterior, llavors parem el prog.
trobat=0; %si trobat=1 significa que hi ha una variable que eliminant-la %millora l’error
bestindex=0; %guardem la posicio d’ aquesta variable a la matriu
for (i=1:n_variables),
    if (array_varselec(i)==1) %si aquesta variable encara continua al model, es %escollida per
        array_varselec(i)=0; %averiguar la seva influencia al model
        inds = find(array_varselec); %troba l’ index de totes les variables que son 1.
        xx = data_in(:,inds); %crea la matriu amb les variables escollides
        yy = data_out;
        [press1,press2]=crossvalidationfuzartmap(xx,yy,split);
        array_varselec(i)=1;
        if (press1<bestpressant)
            bestpressant=press1;
            bestpressant2=press2;
            bestindex=i;
            trobat=1; %el següent cicle del model
    end
end
if (press1==bestpressant)    % si el error resultant d'eliminar dues variables es % igual, llavors considerem
if (press2<=bestpressant2) % eliminar la que tingui el valor de "wija" menor % amb les variables que % que hem entrenat la red.
    bestpressant=press1;       % que hem entrenat la red.
    bestpressant2=press2;
    bestindex=i;
    trobat=1;
end
end
end

clear inds;
clear xx;
clear yy;
end

if (trobat==1)  
    array_varselec(bestindex)=0;
    bestindex ;
else
    variables_sel=array_varselec;
    acabar=1;
end
end

clear data_in;
clear data_out;
clear n_samples;
clear n_variables;
clear split;
clear array_varselec;
clear bestindex;
clear acabar;
clear inds;
clear xx;
clear yy;
clear trobat;
EXECUTE FORWARD SELECTION

```matlab
function [resultatvariables] = executa_forwardselectionGUI(varestlect)
%close all
%clear all
cle
matinput=varestlect;

%Carreguem la matriu de dades. En aquest cas treballarem amb la matriu
"xgascomp1"
%que esta formada per 96 mesures X 120 parametres. Com a matriu de targets tenim
"yprueba"
%classificarem les 96 mesures en 6 categories (3 gasos simples i 3 mescles binaries
%independent de la concentracions.

load C:\matlabR12\work\datososcar.mat
[mostres,variables]=size(xgascomp1);

%Per a l'eleccio de les variables treballarem amb la meitat de les mostres. Una
vegada
%escollides les variables testejarem amb l' altre meitat
%Els imparells son per al conjunt de prediccio i els parells per la validacio

for i=1:48
    indexpar(i)=2*i;
    indeximpar(i)=(2*i-1);
end

%Matriu de prediccion per escollir les variables
[datos,meandatos] = mncn(delsamps(xgascomp1,[indexpar]));
target= (delsamps(yprueba,[indexpar]))/max(yprueba);

x=datos;
y=target;

matriuvar=zeros(1,120);
[n_fil,n_var]=size(matinput);

for i=1:n_var
    matriuvar(matinput(i))=1;
end
```

74
[variablesselection]=forward_selection(x,y,6,matriuvar);

% Amb les variables seleccionades i el mateix conjunt de prediccio entrenem el model
%i i testejem amb el conjunt de validacio
resultatvariables=find(variablesselection);
FUNCTION FORWARD SELECTION

function [variables_sel]=forward_selection(x,y,grups,array_var)

% eliminem les variables a utilitzar
clear data_in;
clear data_out;
clear n_samples;
clear n_variables;
clear split;
clear array_varselec;
clear bestindex;
clear acabar;
clear inds;
clear xx;
clear yy;
clear trobat;
clear variables_sel;

% inicialitzem variables
data_in=x;
[n_samples,n_variables]=size(x);
data_out=y;
split=grups;
array_varselec=array_var;
bestindex=0;
acabar=0;

% Partim del valor de "fitnesses" del conjunt de variables incials que li passem al
% programa.
% El programa Forward selection s'encarrega de disminuir aquest "fitnesses" ( error
% obtingut de la validacio creuada i el wija)
% Si partim de cero, considerem la primera variable del model com la variable a
% superar
if (sum(array_varselec)==0)
    array_varselect(1)=1;
end

inds = find(array_varselec); % troba l’ index de totes les variables que son 1.
xx = data_in(:,inds); % crea la matriu amb les variables escollides
yy = data_out;
while (acabar==0),  %si no hem trobat cap variable que disminueixi el valor del
error
    %de la validació creada del cicle anterior, llavors parem el
programa
    trobat=0;  %si trobat=1 significa que hi ha una variable que eliminant-la
millora
        %l’ error
    bestindex=0;  %guardem la posicio d’ aquesta variable a la matriu

    for (i=1:n_variables),
        if (array_varselect(i)==0)           %averiguem la seva influencia en el model
            array_varselect(i)=1;
            inds = find(array_varselect);  %troba l’ index de totes les variables que son 1.
            xx = data_in(:,inds);             %crea la matriu amb les variables escollides
            yy = data_out;
            [press1,press2]=crossvalidationfuzartmap(xx,yy,split);
            array_varselect(i)=0;
            if (press1<bestpressant)      %si la influencia d’ aquest variable es mes gran
                bestpressant=press1;       %ja que ens ha donat un error menor,
                bestpressant2=press2;     %llavors queda "momentaneament" escollida per a
                %continuar el següent cicle del model
                bestindex=i;
                trobat=1;
            end
            if (press1==bestpressant)     %si el error resultant d’eliminar dues variables es
                if (press2<=bestpressant2) %iguales, llavors considerem
                    % escollir la que tingui el valor de "wija" mes
                    bestpressant=press1;
                    bestpressant2=press2;
                    bestindex=i;
                    trobat=1;
                end
            end
        end
    clear inds;
    clear xx;
    clear yy;
end

if (trobat==1)
    array_varselect(bestindex)=1;
    bestindex
else
    variables_sel=array_varselect;
    acabar=1;
end
EXECUTE STEPWISE SELECTION

function [resultatvariables] = executa_stepwiseGUI(varselect)

%close all
%clear all
cle

matinput=varselect;
%Carreguem la matriu de dades . En aquest cas treballarem amb la matriu "xgascomp1"
%que esta formada per 96 mesures X 120 parametres. Com a matriu de targets tenim %"yprueba"
%classifiedes les 96 mesures en 6 categories (3 gases simples i 3 mescles binaries
%independent de la concentracions.

load C:\matlabR12\work\datososcar.mat
[mostres,variables]=size(xgascomp1);

%Per a l'eleccio de les variables treballarem amb la meitat de les mostres. Una vegada
%escollides les variables testejarem amb l' altre meitat
%Elis imparells son per al conjunt de predicció i els parells per la validacio

for i=1:48
    indexpar(i)=2*i;
    indeximpar(i)=(2*i-1);
end

%Matriu de validacio
[testdatos,meantestdatos] = mncn(delsamps(xgascomp1,[indeximpar]));
testtarget=(delsamps(yprueba,[indeximpar]))/max(yprueba);

%Matriu de predicció per escollir les variables
[datos,meandatos] = mncn(delsamps(xgascomp1,[indexpar]));
target= (delsamps(yprueba,[indexpar]))/max(yprueba);
x=datos;
y=target;

%Creem una array binaria on les variables incials tenem valor '1'
array_varselec=zeros(1,120);
[n_fil,n_var]=size(matinput);

for i=1:n_var
    array_varselec(matinput(i))=1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=1:4
    [variablesselection]=backward_elimination(x,y,6,array_varselec);
    array_varselec2=variablesselection;
    [variablesselection]=forward_selection(x,y,6,array_varselec2);
    array_varselec=variablesselection;
end

%Retornem les variables seleccionades
resultatvariables=find(array_varselec);

GENETIC ALGORITHM

function [resultatvariables] GA_GUI(sliderpopsize_value, slidermaxgen_value, slidermutationrate_value ,crs,crd,SCA,AP,HYGA)
clc

%SETUP del GA
nopop=sliderpopsize_value        %poblacion
maxgen=slidermaxgen_value;    %iteraciones
mut=slidermutationrate_value;   %mutacion
window=1;                                  %todas la variables pueden ser elegidas
converge=80;                               %tanto por ciento de poblacion doblada
begfrac=80/100;
sp=6;                                           %grupos de "sp" para la validacion cruzada
executascaning=SCA;
executaadappop=AP;
executahybrid=HYGA;

if crs==1
    cross=1;
else
    cross=2;
end;                                               %crossover cr=1 single o cr=2 doble

%%%%%%%%
Carreguem la matriu de dades. En aquest cas treballarem amb la matriu "xgascomp1" que està formada per 96 mesures X 120 paràmetres. Com a matriu de targets tenim "yprueba". Classificarem les 96 mesures en 6 categories (3 gases simples i 3 mescles binaries independent de la concentracions.

```matlab
load C:\matlabR12\work\datososcar.mat
[mostres,variables]=size(xgascomp1);

% Per a l’elecció de les variables treballarem amb la meitat de les mostres. Una vegada escollides les variables testejarem amb l’ altre meitat. Els imparells son per al conjunt de predicció i els parells per la validació.

for i=1:48
    indexpar(i)=2*i;
    indeximpar(i)=(2*i-1);
end

%Matriu de predicció per escollir les variables
%Realitzem un Scanning en el cas que ho haguem escollit al interface.

if executascanning==1
    datos = delsamps(xgascomp1,[indexpar]);
    target= delsamps(yprueba,[indexpar]);
    x=datos;
    y=target;
    matriuvariables_1=[1:120]; %matriu de variables que sobreviuen
    num_cat=6; %nombre de les categories que pot clasificar
    mxcat=zeros(1,num_cat); %nombre de mostres x categoria
    auxvar=zeros(num_cat,120); %matriu auxiliar per operar els valors per categoria
    midintra=zeros(num_cat,120); %matriu de mitges intra-classe
    desintra=zeros(num_cat,120); %matriu de desviacio estandar intra-classe

    %%% Calcul mitja intra-classe %%%
    for i=1:48
        auxvar(target(i),:)=auxvar(target(i),:)+datos(i,:);
        mxcat(target(i))=mxcat(target(i))+1;
    end

    for i=1:num_cat
        midintra(i,:)=auxvar(i,:)/mxcat(i);
    end

    %%% Calcul desviacio estandar intra-classe %%%
    for i=1:48
        midintra(i,:)=auxvar(i,:)/mxcat(i);
    end
```

```matlab
auxvar=zeros(num_cat,120);
```
aux = (datos(i,:) - midintra(target(i,:)));  
auxvar(target(i,:)) = auxvar(target(i,:)) + dot(aux, aux, 120);
end  
clear aux;

for i = 1:num_cat  
desintra(i,:) = sqrt(auxvar(i,:)/(mxcat(i)-1));
end

%%%Calcul mitja inter-classe %%%%%%%

midinter = zeros(1,120);
desinter = zeros(1,120);

for i = 1:num_cat  
    midinter = midinter + midintra(i,:);
end
midinter = midinter / num_cat;

%%%Calcul desviacio estandard inter-classe %%%%%%

for i = 1:num_cat  
    aux = midintra(i,:) - midinter;  
    desinter = desinter + dot(aux, aux, 120);
end  
clear aux;

desinter = sqrt(desinter / (num_cat-1));
index = zeros(num_cat, 120);

for i = 1:num_cat  
    for j = 1:120  
        index(i,j) = desintra(i,j) / desinter(j);
    end
end

varsoroll = zeros(1,120);

for i = 1:120  
    for j = 1:num_cat  
        if index(j,i) < 0.1  
            varsoroll(i) = 1;
        end
    end
end

find(varsoroll);
matriuvariables_2 = delsamps(matriuvariables_1',[find(varsoroll)]');

%%% COLINEALITATS %%%%
x = datos(:, matriuvariables_2);
[mmostres, nvariables] = size(x);
vecnorma = zeros(1, nvariables);  % vector que guarda la norma de cada variable

% Calcula la norma de cada variable i matriu de resultats normalitzada

for i = 1:nvariables
    vecnorma(i) = sqrt((x(:, i))' * x(:, i));
x1(:, i) = x(:, i) / vecnorma(i);
end

matrixcol = zeros(nvariables, nvariables);  % matriu amb els índex de colinealitats entre variables

for i = 1:nvariables
    for j = i:nvariables
        matrixcol(i, j) = (x1(:, i))' * x1(:, j);
    end
end

%%% Seleccionem les variables mes ortogonals entre si %%%%  

for i = 1:nvariables
    for j = i:nvariables
        if matrixcol(i, j) < 0.57
            matrixcol(i, j) = 1;
        else
            matrixcol(i, j) = 0;
        end
    end
end

varorto = zeros(1, nvariables);
for i = 1:nvariables
    [num, varaux] = find(matrixcol(i,:));
    sum(num);
    for j = 1:(sum(num))
        varorto(varaux(j)) = 1;
        varorto(i) = 1;
    end
end

matriuvariables_3 = matriuvariables_2(find(varorto))
x = datos(:, matriuvariables_3);
[x, meandatos] = mncn(x);
y = y / max(y);

else
    [datos, meandatos] = mncn(delsamps(xgascomp1, [indexpar]));
target = (delsamps(yprueba, [indexpar])) / max(yprueba);
x = datos;
y = target;
matriuvariables_3=[1:120];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% EXECUCIO ALGORISME GENETIC %%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Inicialment hem inicialitzat els parametres de GA

[m,n] = size(x);
fig = figure;
gcount = 1;
specsplit = ceil(n/window);
%Check to see that nopop is divisible by 4
dp1 = nopop/4;
if ceil(dp1) ~= dp1
    nopop = ceil(dp1)*4;
    disp('Population size not divisible by 4')
    s = sprintf('Resizing to a population of %g',nopop);
    disp(s)
end
%Generate initial population
pop = rand(nopop,specsplit);
for i = 1:nopop
    for j = 1:specsplit
        if pop(i,j) < begfrac
            pop(i,j) = 1;
        else
            pop(i,j) = 0;
        end
    end
    if sum(pop(i,:))<0.5
        colm = round(rand(1)*specsplit);
        if colm <0.5
            colm = 1;
        end
        pop(i,colm) = 1;
    end
end

%Set limit on number of duplicates in population
maxdups = ceil(nopop*converge/100); %Redondea
%Iterate until dup > maxdups
dat = [x y];
dups = 0;
cavterms = zeros(1,maxgen);
cavfit = zeros(1,maxgen);
cbfit = zeros(1,maxgen);
resultatsfitness=zeros(1,maxgen);  
resultatpop=zeros(maxgen,n);  
resultatsfitness2=zeros(1,maxgen);  
resultatsumbestpop=zeros(1,maxgen);

% Main Loop

while dups < maxdups
    gcount
drawnow

% Shuffle data and form calibration and test sets
s = sprintf('At generation %g the number of duplicates is %g',gcount,dups);
disp(s)
avterms = mean(sum(pop'));
cavterms(gcount) = avterms;
s = sprintf('The average number of terms is %g',avterms);
disp(s)
dups = 0;
%array on guardem el fitness de cada cromosoma
fit = zeros(1,nopop);
fit2 = zeros(1,nopop);
% Test each model in population
drawnow

% Per a tots els cromosomes de la poblacio mirem si esta repetit. Comparem un
% cromosoma "i" amb tots els anteriors. Si es igual (dif=0) assignem el fitness que
% ja ha tingut el seu identic, si no es aixi calculem el seu fitness.
for i = 1:nopop
    drawnow
% Check to see that model isn't a repeat
    dflag = 0;
    if i > 1
        for ii = 1:i-1
            dif = sum(abs(pop(i,:) - pop(ii,:)));
            if dif == 0
                dflag = 1;
                fit(:,i) = fit(:,ii);
                fit2(:,i) = fit2(:,ii);
            end
        end
    end
    if dflag == 1;
        dups = dups + 1;
    else
        % Select the proper columns for use in modeling
        inds = find(pop(i,:)*window);  % troba l' index de totes les variables que son 1.
        [smi,sni] = size(inds);
if inds(1) <= n
    ninds = [inds(1)-window+1:inds(1)];
else
    ninds = [inds(1)-window+1:n];
end
for aaa = 2:sni
    if inds(aaa) <= n
        ninds = [ninds [inds(aaa)-window+1:inds(aaa)]];  
    else
        ninds = [ninds [inds(aaa)-window+1:n]];  
    end
end
xx = dat(:,ninds); % crea la matriu amb les variables escollides
[mxx,nxx] = size(xx);
yy = dat(:,n+1);
[press,press2] = crossvalidationfuzartmap(xx,yy,sp);
fit(i)= press;
fit2(i)=press2;
end
end
drawnow
mfit = fit;
mfit2= fit2;

% Ordenem la poblacio primer per el seu fitness, despres per el numero de variables
% en el cromosoma i finalment per el valor del parametre wija
% d'aquesta forma prioritzem els cromosomes amb el mateix fitness pero menor
% nombre de variables i wija menor

[mfit2,ind]=sort(mfit2);
mfit=mfit(ind);
pop=pop(ind,:);

for i=1:nopop
    orden(i)=sum(pop(i,:));
end

[orden,ind]=sort(orden);

pop=pop(ind,:);
mfit=mfit(ind);
mfit2=mfit2(ind);

[mfit,ind]=sort(mfit);
mfit2=mfit2(ind);
pop=pop(ind,:);
The best fitness is %g',mfit(1));
disp(s)
cbfit(gcount) = mfit(1);
s = sprintf('The average fitness is %g',mean(mfit));
disp(s)
cavfit(gcount) = mean(mfit);

figure(fig)
subplot(2,2,1)
sumpop = sum(pop');
plot(sumpop,mfit,'og'), mnfit = min(mfit); mxfit = max(mfit);
dfit = mxfit - mnfit; if dfit == 0, dfit=1; end
axis([min(sumpop)-1 max(sumpop)+1 mnfit-dfit/10 mxfit+dfit/10])
if window > 1
    xlabel('Number of Windows')
s = sprintf('Fitness vs. # of Windows at Generation %g',gcount);
else
    xlabel('Number of Variables')
s = sprintf('Fitness vs. # of Variables at Generation %g',gcount);
end
title(s)
ylabel('Fitness')
    set(gca,'FontSize',9)
    set(get(gca,'Ylabel'),'FontSize',9)
    set(get(gca,'Title'),'FontSize',9)
    set(get(gca,'Xlabel'),'FontSize',9)
subplot(2,2,2)
plot(1:gcount,cavfit(1:gcount),1:gcount,cbfit(1:gcount))
xlabel('Generation')
ylabel('Average and Best Fitness')
title('Evolution of Average and Best Fitness')
    set(gca,'FontSize',9)
    set(get(gca,'Ylabel'),'FontSize',9)
    set(get(gca,'Title'),'FontSize',9)
    set(get(gca,'Xlabel'),'FontSize',9)
subplot(2,2,3)
plot(cavterms(1:gcount))
xlabel('Generation')
ylabel('Average and Best Fitness')
title('Evolution of Average and Best Fitness')
    set(gca,'FontSize',9)
    set(get(gca,'Ylabel'),'FontSize',9)
    set(get(gca,'Title'),'FontSize',9)
    set(get(gca,'Xlabel'),'FontSize',9)
subplot(2,2,4)
plot(cavterms(1:gcount))
xlabel('Generation')
ylabel('Average and Best Fitness')
title('Evolution of Average and Best Fitness')
    set(gca,'FontSize',9)
    set(get(gca,'Ylabel'),'FontSize',9)
    set(get(gca,'Title'),'FontSize',9)
    set(get(gca,'Xlabel'),'FontSize',9)
end
set(get(gca,'Ylabel'),'FontSize',9)
set(get(gca,'Title'),'FontSize',9)
set(get(gca,'Xlabel'),'FontSize',9)
subplot(2,2,4)
bar(sum(pop))
if window > 1
    xlabel('Window Number')
ylabel('Models Including Window')
s = sprintf('Models with Window at Generation %g',gcount);
else
    xlabel('Variable Number')
ylabel('Models Including Variable')
s = sprintf('Models with Variable at Generation %g',gcount);
end
title(s)
axis([0 ceil(n/window)+1 0 nopop+2])
set(gca,'FontSize',9)
set(get(gca,'Ylabel'),'FontSize',9)
set(get(gca,'Title'),'FontSize',9)
set(get(gca,'Xlabel'),'FontSize',9)
drawnow
% Check to see if maxgen has been met
if gcount >= maxgen
    dups = maxdups;
end
%guardem el millor -> Elistisme
bestpop=pop(1,:);
% If Ranking Selection Breed best half of population and replace worst half
pop(1:nopop/2,:) = shuffle(pop(1:nopop/2,:));
pop((nopop/2)+1:nopop,:) = pop(1:nopop/2,:);

% If Roulette Wheel Selection
%pop=roulettemain(oldpop,mfit)

for i = 1:nopop/4
    for j = 1:cross
        %Select twist point at random
tp = ceil(rand(1)*(specsplit-1));
%Twist pairs and replace
    p1 = (nopop/2)+(i*2)-1;
p2 = (nopop/2)+(i*2);
p1rep = [pop(p1,1:tp) pop(p2,tp+1:specsplit)];
p2rep = [pop(p2,1:tp) pop(p1,tp+1:specsplit)];
    pop(p1,:) = p1rep;
    pop(p2,:) = p2rep;
    end
end

%Mutate the population if dups < maxdups
if dups < maxdups
[mi, mj] = find(rand(nopop, specs) < mut);
[ms, ns] = size(mi);
for i = 1:ms
    if pop(mi(i), mj(i)) == 1
        pop(mi(i), mj(i)) = 0;
    else
        pop(mi(i), mj(i)) = 1;
    end
end

%elitisme
pop(1,:) = bestpop;
%guardem en una array el valor de fitness del millor cromosoma
resultatsfitness(gcount) = mfit(1);
%guardem en una matriu el millor cromosoma en cada generacio
resultatpop(gcount,:) = bestpop;
%guardem en una array el valor de wija del millor cromosoma
resultatsfitness2(gcount) = mfit2(1);
%guardem el nombre de variables del millor cromosoma
resultatsumbestpop(gcount) = sum(bestpop);
gcount = gcount + 1;

% % % % % % % % % % % % % % % % HYBRID % % % % % % % % % % % % % % % % %
% apliquem un backward elimination en una frequencia i cromosoma aleatori

if (gcount > 30) & (executahybrid == 1)
    executahybrid
    [mi] = find(rand(1, nopop) < 0.05)
    [ms, ns] = size(mi);
    for i = 1:ns
        [variablesselection] = backward_elimination(x, y, 6, pop(mi(i),:));
        ninds = find(variablesselection);
        xx = dat(:, ninds); % crea la matriu amb les variables escollides
        [mxx, nxx] = size(xx);
        yy = dat(:, n+1);
        [press, press2] = crossvalidationfuzartmap(xx, yy, sp);
    if (press < fit(mi(i)))
        press = fit(mi(i));
        fit(mi(i)) = press;
        fit2(mi(i)) = press2;
        pop(mi(i), :) = variablesselection;
    end
    if (press == fit(mi(i)))
        if (press2 <= fit2(mi(i)))

87
Genetic Algorithms and Time Consuming Optimisation

```
press
fit(mi(i))
fit(mi(i))=press;
fit2(mi(i))=press2;
pop(mi(i),:)=variablesselection;
end
end
end

%%%%%%%%%%%%%%%% ADAPTIVE POPULATION %%%%%%%%%%%%%%%%%
if (gcount==30) & (executaadappop==1)
executaadappop
nopop=nopop/4;
maxdups=ceil(nopop*converge/100);
mfit=mfit(1,nopop);
mfit2=mfit2(1,nopop);
for j=1:nopop
    pop_aux(j,:)=pop(j,:);
end
    pop=pop_aux;
clear orden;
end
% End of Main Loop

%Retornem les variables escollides
matriuvariables_4=matriuvariables_3(find(pop(1,:)));
matriuvar=zeros(1,120);
[n_fil,n_var]=size(matriuvariables_4);
for i=1:n_var
    matriuvar(matriuvariables_4(i))=1;
end
resultatvariables = find(matriuvar)

CROSSVALIDATION

function [fitness,fitness2] = crossvalidationfuzartmap(xx,yy,split)
% Inicializar valores de red PNN
clear ptrain;
clear ttrain;
clear ptest;
clear ttest;
clear s;
clear m;
clear i;
clear j;
```
clear acierto;
datin=xx;
datout=yy;
dimens=size(xx);
mesuresentrenament=dimens(1);
mostres=split;
punter=1;
acert=0;
error=0;
um_wijas=0;

%Creem els inputs per entrenar i testear la xarxa neuronal
while (punter<=mesuresentrenament)
i=1;
j=1;
m=1;
for i=1:mesuresentrenament
if (i<punter)||(i>=(punter+mostres))
ptrain(j,:)=datin(i,:);
ttrain(j,:)=datout(i,:);
j=j+1;
else
ptest(m,:)=datin(i,:);
ttest(m,:)=datout(i,:);
m=m+1;
end;
end;

%Red Fuzzy Artmap
[viga,wija,wijb,wab]=fzmaptrnok(0,1,1,0.001,ptrain,ttrain);

%Testejem un per un per tal de poder guardar correctaments el resultats dins d'una
%matriu
%guardem el valor de la variables wijas
[v_wija col_wija]=size(wija);
um_wijas=num_wijas+v_wija;

[a,ok,nok,nsnc,rateok,viga,vija,vigb,wijb]=fzmaptstok(0,1,1,0.001,wija,wijb,wab,ptest,ttest);

acert=acert+ok;
punter=punter+mostres;
end;

error=100*(mesuresentrenament-acert)/mesuresentrenament;
fitness=error;
fitness2=num_wijas;
clear error;
clear num_wijas;
clear datin;
clear datout;
clear acert;
clear ptrain;
clear ttrain;
clear ptest;
clear ttest;
clear mostres;
clear punter;
clear dimens;
clear mesuresentrenament;

**ROULETTE-WHEEL SELECTION**

```matlab
function [newPop] = roulettemain(oldPop,fitness)
% roulette is the traditional selection function with the probability of
% surviving equal to the fitness of i / sum of the fitness of all individuals
% newPop    - the new population selected
% oldPop    - the current population

[numSols,nvar] = size(oldPop);
totalFit = sum(fitness);
prob=fitness/totalFit;
for i=1 : numSols
    prob(i) = (1/prob(i));
end

totalProb=sum(prob(:));
prob=prob(:)/totalProb;
prob=cumsum(prob);
rNums=sort(rand(numSols,1))    %Generate random numbers

%Select individuals from the oldPop to the newPop
fitIn=1;newIn=1;
while newIn<=numSols
    if(rNums(newIn)<prob(fitIn))
        newPop(newIn,:) = oldPop(fitIn,:);
        newIn = newIn+1;
    else
        fitIn = fitIn + 1;
    end
end

**VALIDATION TEST**

```matlab
function [resultattest]= Test_validacio_FA_GUI(varselect)

clc
```
matinput=varselect;

%Carreguem la matriu de dades. En aquest cas treballarem amb la matriu
%xgascomp1
%que esta formada per 96 mesures X 120 parametres. Com a matriu de targets tenim
%yprueba
%classificarem les 96 mesures en 6 categories (3 gasos simples i 3 mescles binaries
%independent de la concentracions.

load C:\matlabR12\work\datososcar.mat
[mostres,variables]=size(xgascomp1);
matriuvar=zeros(1,120);

[n_fil,n_var]=size(matinput);

for i=1:n_var
    matriuvar(matinput(i))=1;
end

inds=find(matriuvar);
xx = xgascomp1(:,inds);         %crea la matriu amb les variables escollides
[mxx,nxx] = size(xx);
yy = yprueba;
press = validationfuzzyartmap(xx,yy);
resultattest=press;

VALIDATION FUZZY ARTMAP

function [acierto] = validationfuzzyartmap(xx,yy)

clear dat1;
clear target1;
clear viga;
clear wij1;
clear wijb;
clear wab;
clear a;
clear ok;
clear nok;
clear nsnc;
clear rateok;

%inicialitzem
dat1=xx;
target1=yy;

%Creem les matrius entrenament i validacio

for i=1:48
indexpar(i)=2*i;
indeximpar(i)=(2*i-1);
end

%Matriu de prediccio per escollir les variables
[datos1,meandatos1]=mncn(delsamps(dat1,[indexpar]));
predictarget1=delsamps(target1,[indexpar])/max(target1);

%matriu de validacio
testdatos1=delsamps(dat1,[indeximpar]);
testtarget1=delsamps(target1,[indeximpar])/max(target1);

[m1,n1]=size(testdatos1);
for i=1:m1
    for j=1:n1
        testdatos1(i,j)=testdatos1(i,j)-meandatos1(j);
    end
end

[viga,wija,wijb,wab]=fzmaptnok(0,1,1,0.001,datos1,predictarget1);
[a,ok,nok,nsnc,rateok,viga,wija,vigb,wijb]=fzmaptstok(0,1,1,0.001,wija,wijb,wab,testdatos1,testtarget1);
acierto=rateok;
clear dat1;
clear target1;
clear testtarget1;
clear predictarget1;
clear viga;
clear wija;
clear wijb;
clear wab;
clear a;
clear ok;
clear nok;
clear nsnc;
clear rateok;

INTERFACE

function varargout = projecteGA(varargin)
    % PROJECTEGA Application M-file for projecteGA.fig
    % FIG = PROJECTEGA launch projecteGA GUI.
    % PROJECTEGA('callback_name', ...) invoke the named callback.
    % Last Modified by GUIDE v2.0 06-Jul-2005 22:45:18
    if nargin == 0  % LAUNCH GUI
fig = openfig(mfilename,'reuse');
% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(fig);
guidata(fig, handles);
if nargout > 0
    varargout{1} = fig;
end
elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK
    try
    [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
    catch
    disp(lasterr);
    end
end

% --------------------------------------------------------------------
function varargout = radiobutton1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.radiobutton1.
set(handles.rb_allfeat,'Value',1)
set(handles.rb_nonefeat,'Value',0)
set(handles.rb_selfeat,'Value',0)
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = radiobutton2_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.radiobutton2.
set(handles.rb_allfeat,'Value',0)
set(handles.rb_nonefeat,'Value',1)
set(handles.rb_selfeat,'Value',0)
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = radiobutton3_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.radiobutton3.
set(handles.rb_allfeat,'Value',0)
set(handles.rb_nonefeat,'Value',0)
set(handles.rb_selfeat,'Value',1)
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton2.
set(handles.edit3,'String','EXECUTANT');
going(handles.rb_allfeat,'Value');
going(handles.rb_nonefeat,'Value');
going(handles.rb_selfeat,'Value');
if get(handles.rb_allfeat,'Value')==1
    resultatvariables=executa_forwardselectionGUI([1:120]);
end
if get(handles.rb_nonefeat,'Value')==1
resultatvariables=executa_forwardselectionGUI([]);
end
if get(handles.rb_selfeat,'Value')==1
    load('C:\MATLABR12\work\variablesvalidacio.mat')
    resultatvariables=executa_forwardselectionGUI(resultatvariables);
end
save('C:\MATLABR12\work\variablesvalidacio.mat');
set(handles.edit3,'String',num2str(resultatvariables));
guidata(h, handles);

% --------------------------------------------------------------------
function varargout = pushbutton3_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton3.
set(handles.edit3,'String','EXECUTANT');
get(handles.rb_allfeat,'Value');
get(handles.rb_nonefeat,'Value');
get(handles.rb_selfeat,'Value');
if get(handles.rb_allfeat,'Value')==1
    resultatvariables=executa_stepwiseGUI([1:120]);
end
if get(handles.rb_nonefeat,'Value')==1
    resultatvariables=executa_stepwiseGUI([]);
end
if  get(handles.rb_selfeat,'Value')==1
    load('C:\MATLABR12\work\variablesvalidacio.mat')
    resultatvariables=executa_stepwiseGUI(resultatvariables);
end
save('C:\MATLABR12\work\variablesvalidacio.mat');
set(handles.edit3,'String',...
    num2str(resultatvariables));
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton1.
set(handles.edit3,'String','EXECUTANT');
get(handles.rb_allfeat,'Value');
get(handles.rb_nonefeat,'Value');
get(handles.rb_selfeat,'Value');
if get(handles.rb_allfeat,'Value')==1
    resultatvariables=executa_backwardeliminationGUI([1:120]);
end
if get(handles.rb_nonefeat,'Value')==1
    resultatvariables=executa_backwardeliminationGUI([]);
end
if  get(handles.rb_selfeat,'Value')==1
    load('C:\MATLABR12\work\variablesvalidacio.mat')
    resultatvariables=executa_backwardeliminationGUI(resultatvariables);
end
save('C:\MATLABR12\work\variablesvalidacio.mat');
set(handles.edit3,'String',...
num2str(resultatvariables));
guidata(h, handles);

function varargout = edit3_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.edit3.
guidata(h, handles);

function varargout = pushbutton6_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton6.
load('C:\MATLABR12\work\variablesvalidacio.mat')
varsel=Test_validacio_FA_GUI(resultatvariables);
save('C:\MATLABR12\work\variablesvalidacio.mat');
set(handles.edit4,'String','
num2str(varsel));
guidata(h, handles);

% ------------------------------  GA  ------------------------------------
function varargout = popsize_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.popsize.
divisible = 4*round(get(handles.popsize,'Value')/4);
set(handles.et_popsize,'String','
num2str(divisible));
guidata(h, handles);

function varargout = maxgen_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.maxgen.
mg=round(get(handles.maxgen,'Value'));
set(handles.et_maxgen,'String','
num2str(mg));
guidata(h, handles);

function varargout = mutationrate_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.mutationrate.
mr=round(1000*get(handles.mutationrate,'Value'))/1000;
set(handles.et_mutationrate,'String','
num2str(mr));
guidata(h, handles);

function varargout = Single_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Single.
set(handles.Single,'Value',1);
set(handles.Double,'Value',0);
guidata(h, handles);

function varargout = Double_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Double.
Genetic Algorithms and Time Consuming Optimisation

set(handles.Single,'Value',0);
set(handles.Double,'Value',1);
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = et_popsize_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.et_popsize.
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = et_maxgen_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.et_maxgen.
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = et_mutationrate_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.et_mutationrate.
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = scanning_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.scanning.
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = adappop_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.adappop.
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = hybrid_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.hybrid.
guidata(h, handles);
% --------------------------------------------------------------------
function varargout = runGA_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.runGA.
SCA= get(handles.scanning,'Value');
AP= get(handles.adappop,'Value');
HYGA= get(handles.hybrid,'Value');
divisible  = 4*round(get(handles.popsize,'Value')/4);
mg=round(get(handles.maxgen,'Value'));
mr=round(1000*get(handles.mutationrate,'Value'))/1000;
cri=get(handles.Single,'Value');
ccr=get(handles.Double,'Value');
resultatvariables=GA_GUI(divisible,mg,mr,cri,ccr,SCA,AP,HYGA);
save('C:\MATLABR12\work\variablesvalidacio.mat');
set(handles.edit3,'String',num2str(resultatvariables));
guidata(h, handles);