White Rose Research Online
eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk
Universities of Leeds, Sheffield and York

Deposited via The University of Sheffield.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/84955/

Monograph:
Hatzikos, V.E. and Owens, D.H. (2003) Genetic Algorithms in Iterative Learning Control. Research Report. ACSE Research Report 844 . Department of Automatic Control and Systems Engineering

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

UNIVERSITY OF LEEDS
University of Sheffield
UNIVERSITY of York

M0093223SH

GENETIC ALGORITHMS

IN

ITERATIVE LEARNING CONTROL

Appendix: An Introduction to the Genetic Algorithm Toolbox in Matlab

V.E. Hatzikos **D.H. Owens**

Department of Automatic Control and Systems Engineering

University of Sheffield

Sheffield, S1 3JD

UK

Research Report No. 844

July 2003

Genetic Algorithms in Iterative Learning Control-
Appendix: An Introduction to the Genetic Algorithm Toolbox in Matlab

V. E. Hatzikos*, D. H. Owens*

*Automatic Control and Systems Engineering Department
The University of Sheffield
Mappin Street, Sheffield, S1 3JD, United Kingdom

Abstract

Recently it was explored by the authors whether or not a Genetic Algorithm (GA) based approach can be used in the context of norm-optimal Iterative Learning Control (ILC). It turnes out the answer is positive for both linear and nonlinear plant models. The strength of this method is that it can cope with nonlinearities and hard constraints in the problem definition whereas most of the existing algorithms would fail. Simulations are used to illustrate the performance of this new approach, and give good results in terms of convergence speed and tracking of the reference signal regardless the nature of the plant under investigation. It is in fact shown in this paper that under suitable assumptions, the proposed GA-ILC approach will result in monotonic exponential convergence, which is a very strong property of an ILC algorithm. The proposed approach also involves the design of a low-pass FIR filter which successfully smoothes the *noisy* input signal naturally generated by the GA method.

1 Introduction

Over the last two decades, intelligent control (IC) methods have been heavily investigated by the researchers in many engineering problems with varying degrees of success. Some of the most known IC theories are Neural Networks, Genetic Algorithms, Fuzzy control and Learning control. The key characteristic that makes these techniques substantial different from most analytical control methods is the fact that they use human, animal or biologically motivated procedures to develop or implement a controller for a dynamical system. This significant diversity offers them numerous advantages over traditional methods in terms of applicability on different classes of dynamical systems. Furthermore, the idea of combining two or more IC methods for implementing more advanced algorithms has been introduced by many scientists with really significant results. In this paper the Iterative Learning Control (ILC) technique will be discussed.

Iterative learning control is a technique to control systems operating in a repetitive mode with the additional requirement that a specified output trajectory $r(t)$ in a finite time interval $[0, T]$ is to be followed with high precision. Motivated by human learning, the fundamental idea of ILC is to use information from previous executions of the task in order to improve performance from trial to trial in the sense that the tracking error is sequentially reduced [4]. Typical examples of systems that work in this kind of repetitive mode are robotic manipulators in manufacturing industry and chemical batch processes. It can be in fact stated that the repetitive processes comprise a very large group of industrial processes, ranging from robotics and semiconductors to steels and chemical process industries.

Most of the ILC laws construct the input into the plant on a given trial from the input used during previous trials plus an additive incremental, which is typically a function of the past observed output error, that is, the difference between the achieved output $y(t)$ and the specified output trajectory $r(t)$. According to the value of the error the dynamical system is said to 'learn' by remembering the effectiveness of previously tried inputs and using information on their success or failure to construct new trial control input functions. Hence in contrast with adaptive schemes, ILC does not attempt to identify the plant but changes only the control input. Furthermore, this adaptation or updating takes place after each trial and not after each time step as in adaptive control.

The goal of minimising the tracking error during each trial can be easily formulated into a suitable optimisation problem. In fact, over the years many researchers have proposed optimisation based ILC algorithms providing good convergence properties [10], [6], [2]. A more detailed description of these algorithms will be given in the next section. However, most of the algorithms with guaranteed convergence properties work only for linear plants. This is a severe limitation because the dynamics of repetitive systems can be highly non-linear. For this reason it is necessary to derive a new class of ILC algorithms that are able to cope with nonlinearities. Furthermore, in practise process variables are subject to constraints that are set by safety considerations or physical limitations. Hence there is a real need for algorithms that can handle these hard constraints in a straightforward manner.

Recently, the possibility of using evolutionary based strategies in solving optimisation problems in ILC was investigated in [12]. In the proposed idea a preliminary study suggested that Genetic Algorithms (GAs) can be used into an optimisation based ILC scheme in order to overcome the limitations analysed above. This approach appeared to be computationally realistic. Furthermore, a significant point is that while GAs has been shown to be able to solve a variety of different optimisation problems in engineering, to our knowledge the idea to use the GAs to implement optimality based ILC algorithm has not been reported before. In addition, according to [8], the advantages of a GA in such environments are free-

dom from the need to possess an explicit model of behaviour and the intrinsically parallel search diversity, which in redundant environments has the additional effect of distinguishing the important from the irrelevant, thus further simplifying the problem. Also, GAs are not limited by typical control problem attributes such as ill-behaved objective functions, the existence of constraints, and variations in the nature of control variables.

In this paper, we propose a Genetic Algorithm based optimisation method for Iterative Learning Control (GA-ILC). Our main contribution is the fact that the proposed GA-ILC framework has the ability to successfully produce optimal solutions for various control systems under mild assumptions on the plant model. The obtained results indicate that the proposed scheme performs satisfactory for any class of dynamical systems (i.e linear or nonlinear, continuous or discrete, minimum or nonminimum) whereas the more traditional optimisation based algorithms work only for specific classes of linear systems. Also the proposed algorithm result in monotonic exponential convergence of the cost function which is a very strong property for ILC systems. Furthermore, the proposed procedure includes also the design of a low-pass FIR filter which successfully smoothes any 'noisy' part of the selected input signal.

The rest of the paper is organised as follows: in the next section we have the problem definition. In section 3 a small historical review of some of the most interesting optimisation based ILC methods is provided In section 4 convergence properties for nonlinear optimal ILC applications are explored. In section 5 the proposed $GA-ILC$ framework is described and in section 6 GA implementation is provided. In section 7 the filtering process is provided. Section 8 includes the simulation examples followed by conclusions and recommendation for further work.

2 ILC Problem definition

Consider the following possibly non-linear discrete-time dynamical system defined over finite time interval, $t \in [0, T_s, 2T_s \dots, T_f]$:

$$\begin{aligned} x(t+Ts) &= f(x(t), u(t), t) \\ y(t) &= g(x(t), u(t), t) \end{aligned} \tag{1}$$

with a suitable initial condition $x(0) = x_0$. In addition, a reference signal $r(t)$ is specified and the control objective is to design an learning algorithm that will drive the output variable $y(t)$ to track this reference signal as closely as possible by manipulating the input variable $u(t)$. The special feature of the problem is that when the system (1) has reached the final time point $t = T_f$, the state of the system is reset back to x_0, and after the resetting

the system is supposed to follow the same reference signal $r(t)$ again. This repetitive nature of the problem opens up possibilities for modifying iteratively the input function $u(t)$ so that as the number of repetitions or trials increases, the system learns the input function that gives perfect tracking. To be more precise, the idea is to find a control law

$$u_{k+1} = f(u_k, u_{k-1}, \dots, u_{k-r}, e_{k+1}, e_k, \dots, e_{k-s}) \tag{2}$$

so that

$$\lim_{k\to\infty} ||e_k|| \to 0 \quad \text{and} \quad \lim_{k\to\infty} ||u_k - u^*|| \to 0 \tag{3}$$

where

$$y_k = [y_k(0), y_k(T_s), y_k(2T_s), \dots, y_k(T_f)]^T \tag{4}$$

$$u_k = [u_k(0), u_k(T_s), u_k(2T_s), \dots, u_k(T_f)]^T \tag{5}$$

$$e_k = [r(0) - y_k(0), r(T_s) - y_k(T_s), r(2T_s) - y_k(2T_s), \dots, r(T_f) - y_k(T_f)]^T \tag{6}$$

and u^* is the input function that gives perfect tracking (i.e. we are assuming the reference signal belongs to the range of the plant). Note that if the original plant model is a linear time-invariant model

$$\begin{aligned} x(t + T_s) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned} \tag{7}$$

it can be represented equivalently with a matrix equation $y_k = G_e u_k$, where

$$G_e = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ CB & 0 & 0 & \dots & 0 \\ CAB & CB & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{T_l-1}B & CA^{T_l-2}B & \dots & \dots & 0 \end{bmatrix}, \tag{8}$$

where $T_l = T_f/T_s$. This equivalent representation can typically simplify considerably the convergence analysis of ILC algorithms. In the next section we will describe in details the proposed Genetic Algorithm based method for Iterative Learning control systems (GA-ILC).

3 Optimisation based ILC algorithms

The ILC theory was introduced independently by several researchers in the beginning of the 1980s. Most important of all was [4] who defined the principles that underlie 'learning control'. After that the number of published papers has increased significantly. Some of

the most popular methodologies include the design of D-type learning update laws (time derivative of the error signal) as [4], [16] while others implemented PID-type compensation strategies as [5]. Model-based algorithms have also been implemented by [13], [3] offering good performance results regarding good tracking of the reference signal.

However, algorithms based on optimality have been proved to be the most popular ones amongst the researchers. This is because firstly in most cases the implementation process involves the completion of some straightforward steps which is the introduction of a cost criterion that needs to be minimised or maximised depended on the nature of the problem and the establishment of a search algorithm that will achieve that. Secondly, they can be easily coded into programming languages as MATLAB for simulation purposes providing very accurate performance results. Some of the most essential optimisation based method for ILC systems will be discussed in this section.

The first researchers to introduce optimality into ILC were [10]. They introduced a steepest-descent approach to minimise the norm of $||e_k||^2$. The input update law that was used

$$u_{k+1} = u_k + \varepsilon_k G^*[e_k] \tag{9}$$

where G^* is the adjoint operator of the plant G and ε is the scalar determining the step length. The optimal value of ε_k is calculated from

$$\varepsilon_k = \frac{||G^* e_k||^2}{||GG^* e_k||^2} \tag{10}$$

In general the above algorithm gives good convergence results for stable, controllable and observable linear systems. Furthermore, the computation of $G^* \varepsilon_k$ requires the precise knowledge of the plant model. However this is theoretically impossible to achieve since there is always some uncertainty between the actual plant and the nominal plant that is used for simulation purposes. Also the learning law 9 is quite complicated since it requires to solve a differential equation of the same order as the plant model between trials to find $G^*[e_k]$.

Another advanced optimisation based procedure for ILC systems with good performance results was proposed by [6]. A Newton-Raphson search technique is used at each trial in order to find an ideal input that minimises the cost criterion described by equation 11. The plant data changes from trial to trial while the cost criterion stays the same. Convergence speed for various systems is not formally examined but on the other hand the proposed scheme is applied into a real system with satisfactory performance results.

$$J_{k+1} = \frac{1}{T}\int_0^T (|e_{k+1}| + \varepsilon \dot{u}_{k+1}^2) dt \tag{11}$$

Also many optimisation based algorithms have been proposed for purpose of reducing the noise sensitivity. [17] proposed a discrete-time ILC algorithm based on the following least-squares objective function with an input penalty term

$$J_k(u_k) = ||e_k||^2 + ||u_k||^2 \tag{12}$$

The introduction of this quadratic penalty term resulted in noise reduction but also resulted in poor convergence speed of the cost function. Additionally the use of an input weight matrix which could improve convergence speed was not comprehensively investigated. A similar approach for continuous-time systems has also been considered by [15].

Finally, [2] designed a more advanced optimisation based method for ILC systems. This particular approach is the so-called Norm-Optimal ILC method. The basic idea behind this method is to solve the following optimisation problem on-line during each iteration:

$$J_{k+1} = ||e_{k+1}||^2 + ||u_{k+1} - u_k||^2 \tag{13}$$

with the constraint equation

$$y_{k+1}(t) = [Gu_{k+1}](t), \tag{14}$$

where G is the plant in question. The advantages of this approach are immediate from the simple interlacing result (16) which is a consequence of optimality (it is assumed that (14) has at least one optimal solution) and furthermore from the fact that the choice of

$$u_{k+1} = u_k \tag{15}$$

would lead to the relation $J_{k+1}(u_k) = ||e_k||^2$ and hence

$$||e_{k+1}||^2 \leq J_{k+1}(u_{k+1}) \leq ||e_k||^2 \tag{16}$$

in other words the algorithm results in monotonic decrease of the error norm. If the plant G in the constraint equation $y_{k+1} = [Gu_{k+1}](t)$ is a linear time-invariant (LTI) system, it is straightforward to show that the optimising solution is given by

$$u_{k+1}(t) = u_k(t) + [G^*e](t) \tag{17}$$

where $G*$ is the adjoint operator of G. This is a non-causal implementation of algorithm but it can be shown that with LIT systems there exists an equivalent causal feedback-law [2]. Furthermore, in the case of invertible discrete-time LTI systems one can show [1] that

$$||e_{k+1}|| \leq \frac{1}{1+\sigma}||e_k|| \tag{18}$$

where $\sigma > 0$ is the smallest singular value of the plant G. Therefore (18) shows that the convergence is in fact geometric for this particular class of plants. However, with nonlinear plants it is not clear how to use the adjoint of the plant to implement the algorithm (the adjoint does not exist or it is not clear how to find an equivalent causal implementation).

Hence in this paper it is suggested that for nonlinear plants the optimisation problem (13) is to be solved numerically between trials by using the $GA - ILC$ approach. It is important to understand that if the optimisation problem (13) has at least one optimising solution with the given nonlinear plant, and the chosen GA method is able to find one of the optimising solutions, then the interlacing result (31) still holds. In addition, the performance of the proposed $GA-ILC$ approach is also evaluated using linear dynamical systems.

4 A note on convergence for Nonlinear Optimal ILC

For the application of GAs to nonlinear optimal ILC some indication of the potential for convergence is valuable. Convergence may depend on the reference signal $r(t)$ and the initial control choice. To formalize this idea, consider the ILC problem with performance index, (eq 13), and the nonlinear model in (eq 1) written in matrix form as

$$y_k = G(u_k) \ , \ k \geq 0 \tag{19}$$

or

$$e_k = r - G(u_k) \tag{20}$$

where G is a continuous differentiable nonlinear mapping $\Re^N \rightarrow \Re^N$. According to [1], minimization of the performance index yields the equation

$$u_{k+1} = u_k + \left(\frac{\partial G}{\partial u}\right)^*_{u_{k+1}} e_{k+1} \tag{21}$$

where $\left(\frac{\partial G}{\partial u}\right)^*$ is the adjoint of the Jacobean $\left(\frac{\partial G}{\partial u}\right)$. This simple analysis can be turned into a geometric convergence theory as follows: substitute (eq21) into the performance index to obtain the relationship (at the optimal value)

$$||e_{k+1}||^2 + ||\left(\frac{\partial G}{\partial u}\right)^*_{u_{k+1}} e_{k+1}||^2 = J^{optimal}_{k+1} \leq ||e_k||^2 \tag{22}$$

which again implies (eq16) i.e the norm of the error is non-increasing. This (eq22), is difficult to analyse as there is no explicit formula for u_{k+1}. It is clear however that

$$u_{k+1} \in \mathrm{S}(e_k) := \{u \in \Re^N : ||e|| \leq ||e_k||, e = r - G(u)\} \tag{23}$$

and that

$$\mathbf{S}(e_0) \supset \mathbf{S}(e_1) \supset \cdots \supset \mathbf{S}(e_k) \supset \ldots \tag{24}$$

(which is hence true for any closed ball containing $\mathbf{S}\ (e_k)$).

Suppose now that

$$\sigma^2 = inf_{u \in \mathbf{S}(e_0)} p\left(\left(\frac{\partial G}{\partial u}\right)\left(\frac{\partial G}{\partial u}\right)^*\right) = inf_{u \in \mathbf{S}(e_0)} \underline{\sigma}^2\left(\frac{\partial G}{\partial u}\right) > 0 \tag{25}$$

where p denotes the spectral radious and $\underline{\sigma}$ denotes smallest singular value. This condition is equivalent to the invertibility of the linearisation of G close to the point $e = 0$.

It then follows that (eq22) implies

$$||e_k||^2 \geq (1 + \sigma^2)||e_{k+1}||^2 \ , \ k \geq 0 \tag{26}$$

and hence geometric convergence of the error to zero is guaranteed i.e.

$$||e_{k+1}||^2 \leq \frac{1}{1+\sigma^2}||e_k||^2 \ , \ k \geq 0 \tag{27}$$

$$\lim_{k \to \infty} ||e_k|| = 0 \tag{28}$$

The above discusion can be stated as a theorem as follows:

Theorem 1 *With the above notation, the nonlinear optimal ILC algorithm converges geometrically to a zero tracking error if the initial error e_0 is sufficiently small and the linearisation of the system dynamics about the point $e = 0$ is invertible.*

The practical outcome of the above analysis is to point out the potential for convergence but also that plant invertibility and the need for a good initial guess u_0 (e.g by a well-defined feedback controller) are also important factors. For the remainder of this paper, the possibility of convergence is assumed as the technical conditions derived above cannot be easily checked in practice.

5 The proposed GA-ILC framework

As mentioned earlier, the main motivation in this paper is to propose an optimisation based method which could be applied to any class of dynamical systems regardless if the system

under investigation is linear or nonlinear, continuous or discrete, minimum or nonminimum phase and also if there exist some hard constraints. Most of the created optimality based approaches cannot cope with this demand. On the other hand GAs capability to overcome limitations in the current algorithms has been indicated by many researchers [8], [9]. This is mainly due to the fact that GAs differ substantially from more traditional search and optimisation methods. First, GAs use probabilistic transition rules, not deterministic ones. Secondly, they do not require derivative information or other auxiliary knowledge, i.e. nonlinearities and external disturbances do not affect the search algorithm; only the objective function and the corresponding fitness levels influence the directions of search. Finally, GAs work on an encoding of the parameter set rather than the parameter set itself which allows adjusting the boundaries of the search space as appropriate.

A block diagram representing the proposed GA-ILC structure can be seen in Figure 1. The idea is to solve the optimisation problem introduced in equation (30)

$$\min_{u_{k+1}} J(u_{k+1}) \tag{29}$$

$$J_{k+1} = ||e_{k+1}||^2 + \alpha||u_{k+1} - u_k||^2 \tag{30}$$

with the constraint equation $y_{k+1} = Gu_{k+1}$ between trial k and $k + 1$, α is a weight factor and G is the equivalent input-output mapping. Also, $||e_{k+1}|| = e^T Qe$ where Q is a symmetric positive-definite weighting matrix and in a similar fashion $||u_{k+1} - u_k|| = (u_{k+1} - u_k)^T R(u_{k+1} - u_k)$. In general, Q and R can be used to balance the relative importance of accurate tracking. The GAs are used in order to search for the optimal u^*_{k+1} that minimises (30) between trials using a simulation model of the plant G. Also the GAs needs to be able to satisfy the following inequality based on (29) and (30)

$$||e_{k+1}||^2 \leq J_{k+1}(u^*_{k+1}) \leq ||e_k||^2 \tag{31}$$

If the optimisation problem in (29) has at least one solution for $k = 1, 2, \ldots$, the approach results in monotonic convergence of the cost criterion value (see $Proposition 1$).

The optimisation procedure is implemented in Matlab. A simulation model for the given dynamical system can be constructed in Matlab's Simulink environment while the GA optimization procedure can be coded in Matlab's workspace. These two working areas are connected by the cost function. In each iteration a new input u_{k+1} is introduced into the Simulink model by the GA.

Some of the advantages provided by Matlab is that it has a variety of functions useful for the GA practitioner. Given the versatility of Matlab's high-level language, problems can

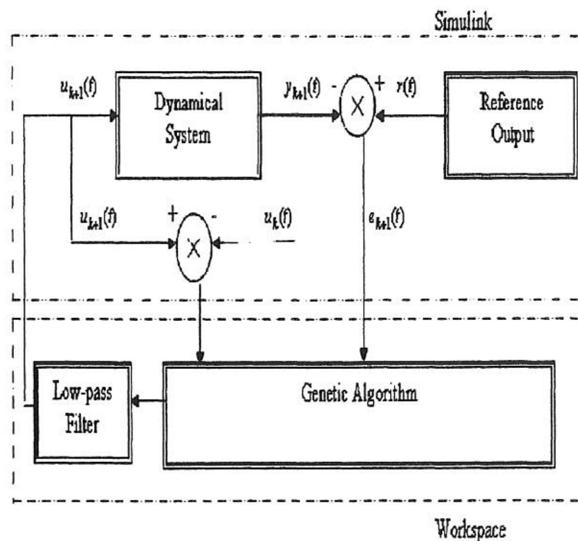

Figure 1: the proposed GA-ILC structure.

be coded in m-files in a fraction of time that it would take to create C or $Pascal$ programs for the same purpose. Also the GA toolbox [7] in Matlab is a collection of routines, written mostly in m-files, which implement the most important functions in GAs.

As mentioned above, a simulation model for a given system can be designed in Matlabs Simulink. More specific, for each iteration the data of the cost function can be collected. It′s then passed into the workspace and the optimisation procedure takes place. Then a new input is initialized into the Simulink model by the GA. For real life plants this kind of process is a so-called on-line optimisation [9]. The operation of the system is directly influenced by the GA. Overall, GA based on-line optimisation tend to be quite rare in engineering problems mainly due to difficulties associated with computation burden.

6 The Genetic Algorithm process

Overall, GAs is search algorithms based on the mechanics of natural selection and natural genetics. Typically a GA work on binary string of a fixed length L, i.e $I = [0,1]^L$, which is referred to as chromosomes. It starts with a population of potential solutions (chromosomes) applying the principle of survival of the fittest to produce better and better approximations to a solution. In every generation, a new set of chromosomes is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure.

For more information about the GA method the reader is referred in [11], [14] , [7].

The block diagram of the proposed GA procedure is shown in Fig. 2. The procedure starts by generating a population P of functions S (possible solutions to the optimisation problem)

$$P(0) = [S_1(0), ..., S_n(0)] \in I^n \tag{32}$$

The space of chromosomes is then converted to actual search space M by a decoding function $\Gamma : I \rightarrow M$ and the objective function $f : M \rightarrow R$ is calculated. A more detailed description of the GA operators can be found in [18]. Based on the objective function information the algorithm evaluates the fitness function $F(S_i) : I \rightarrow R$ of each chromosome in the population which is a measure of the chromosomes performance in the problem domain. After that, the selection of the fittest chromosome takes place. This chromosome is the initial input u_0, which is imported into the Simulink model. The chromosomes are then reproduced using the genetic operators (crossover and mutation) according to their fitness value. This means that fittest ones have better change of being chosen for reproduction. More specific, for crossover whose local operator is $r[p_c] : I^2 \rightarrow I^2$ where probability of crossover is $p_c \in [0.8, 0.9]$ select bits from the selected chromosomes and creates new offsprings i.e multiple point crossover

$$\begin{aligned} r(s_1) &= (s_{11} \dots s_{1p}, s_{2(p+1)} \dots s_{2(p+n)}, s_{1(p+n+1)} \dots s_{1(k)}, s_{2(k+1)} \dots s_{2L}) \\ r(s_2) &= (s_{21} \dots s_{2p}, s_{1(p+1)} \dots s_{1(p+n)}, s_{2(p+n+1)} \dots s_{2(k)}, s_{1(k+1)} \dots s_{1L}) \end{aligned} \tag{33}$$

Then with a probability $p_m = 0.001$ the mutation operator is applied to the selected chromosomes $m(p_m) : I \rightarrow I$ i.e it alters a bit of the chromosome. Finally, the developed offsprings are reinserted into the population replacing the old chromosomes using an *elitist* strategy (generation gap). This means that the fittest chromosomes from the old will always propagate into the next population resulting in monotonic convergence of the objective function.

Proposition 1 *Suppose that for* $k = 1, 2, \dots$ *the optimisation problem* $\min_{u_{k+1}} J_{k+1}(u_{k+1})$ *has at least one optimal solution* u^*_{k+1}, *for each k, and implementing an elitist strategy the GA is able to find* u^*_{k+1}, *then* $||J_{k+1}|| \leq ||J_k||$.

Proof. Using the proposed GA-ILC structure (see figure 2) we can measure the performance of each chromosome in the current population according to their fitness value. So let $u^*{}_k$ be the fittest chromosome in k population which means that

$$u^*{}_k \rightarrow J^*{}_k \rightarrow F^*{}_k \tag{34}$$

where J^*_k is the corresponding objective value when the selected u^*_k is imported into the plant under investigation and F^*_k is the fitness value of the input. Then the GA proceeds to the production of the next $k+1$ population by reproducing the old chromosomes. However using the generation gap mechanism the u^*_k will propagate into this generation.

$$\begin{bmatrix} u_{k+1}(0) \\ u_{k+1}(1) \\ u_{k+1}(2) \\ \vdots \\ u^*_k(n) \end{bmatrix} \quad (35)$$

If non of the new offsprings $u_{k+1}(n)$ performs better than the $u^*_k(n)$ then the fittest chromosome of the $k+1$ population will be $u^*_k(n) = u^*_{k+1}(n)$. Which means that

$$J_{k+1} = J_k \quad (36)$$

So monotonic convergence is achieved since $J_{k+1} \leq J_k$ is always true.

$\square$

For this particular paper, this loop is repeated 100 times (generations) before selecting the next input to be introduced into the real model. The algorithm ends when the norm of the error of the system is minimized to an optimal solution. Note that in this implementation it is straightforward to include hard constraints in the input variable u_{k+1} by adjusting the boundaries of the GAs search space within the appropriate values.

Generally the GA-ILC structure appears to be very robust and capable of given good tracking performance for various classes of dynamical systems. However the proposed GA mechanism seems to have the tendency of producing what appears to be $noisy$ inputs. This is due to the fact that the GA creates randomly an initial population of sampled input function values at each sample time from a large search space without having any information about the plant or correlation. The proposed solution to this $computational noise$ problem is to filter the optimal input, produced from the GA before applying it into the real plant between sample instants.

7 Design of a low-pass FIR filter

The filtering process involves the design of a low-pass filter for removing the unwanted $noisy$ parts of the selected input signal. As mentioned above, this is due to the fact that the search space in the GA process is not restricted and sample instants are not correlated, and

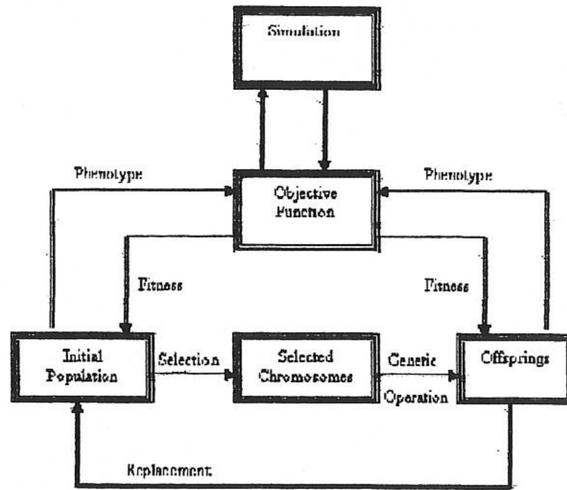

Figure 2: the proposed GA-ILC structure.

hence the algorithm does not necessarily produce *smooth* input functions. Also, numerical experience and statistical intuition indicate that the noise will typically be high frequency. So, filtering of the input signal using a low-pass filter with a bandwidth greater than that of the plant will hence smooth the input and have little effect on the output (and hence tracking error) as in practice, the plant itself is a low pass filter. To be more precise, in each iteration the idea is to filter the input chosen by the GA using the formula 37 to reshape the input signal within each time instance.

$$b_1 u^{*'}(n) = F[\alpha_1 u^*(n+2) + \alpha_2 u^*(n+1) + \alpha_3 u^*(n) + \alpha_4 u^*(n-1) + \alpha_5 u^*(n-2)] \quad (37)$$

where $u^{*'}$ is the obtained 'smooth' input signal, u^* is the selected 'noisy' signal and $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_5]$ is the coefficients matrix which is used for the filtering procedure. It can be seen that (eq 37) is a so called non-causal equation which cannot be directly applied into dynamical systems. The solution to this problem is quite straightforward. The filtering process can take place offline between trials while the proposed GA-ILC algorithm can be applied online into the system under investigation. The only disadvantage of the proposed

approach is a possible increase in the simulation time.

$$\begin{bmatrix} u^{*'}(0) \\ u^{*'}(1) \\ u^{*'}(2) \\ \vdots \\ u^{*'}(n) \end{bmatrix} = \begin{bmatrix} u^*(2) & u^*(1) & u^*{}_1(0) & u^*{}_2(0) & u^*{}_3(0) \\ u^*(3) & u^*(2) & u^*(1) & u^*{}_1(0) & u^*{}_2(0) \\ u^*(4) & u^*(3) & u^*(2) & u^*(1) & u^*{}_1(0) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ u^*(n+2) & u^*(n+1) & u^*(n) & u^*(n-1) & u^*(n-2) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix}, \tag{38}$$

where $u^*(0)$ are the filter's preselected initial conditions, they can be described by equation 39 and they are calculated from the initial values of the 'noisy' input signal for the first three time instances.

$$U^*(0) = [u_1^*(0) \quad u_2^*(0) \quad u_3^*(0)] \tag{39}$$

In conclusion, it can be said that filtering on the selected input signal has little effect on the error value since all ILC dynamical systems are by nature low-pass devices. That is way the technique of trial and error was used for the selection of the $\alpha^{'}$s values.

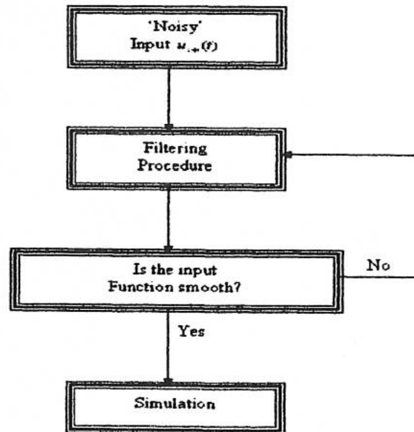

Figure 3: A block diagram representing filtering process

8 Simulation results

To demonstrate the effectiveness of the proposed GA-ILC method simulation results are provided in this section. The GA-ILC algorithm is applied to three different types of dynamical systems and the obtained results are analysed in terms of convergence speed and tracking of the reference signal.

8.1 Simulation example 1

The selected dynamical system is a typical continuous time minimum phase linear system described by the following transfer function

$$G(s) = \frac{1}{s^2 + 5s + 6} \tag{40}$$

where the system is defined over the time interval $t \in [0, 6]$ with a sampling rate $T_s = 0.1$. The output of the system needs to track the reference signal $r(t) = \sin(t)$. The free parameters of the GA used for this particular example are shown in Table 2. The settings of the genetic operators was done according to guidelines in [7].

SGA parameter	Setting
Population size	100
Total Generations	100
Number of iterations	6
Coding	Binary representation, 30 bits per decision variable
Selection	Low-level stochastic universal sampling routine
Recombination	Shuffle crossover with reduced surrogate, probability=0.7
Mutation	Bit-flipping, random probability
Generation gap	0.95
Elitism	Best five chromosomes of previous population forward to next one

Table 1: Algorithm parameters for the linear case

The cost function was chosen to be

$$J(u_{k+1}) = ||e_{k+1}||^2 + 0.1||u_{k+1} - u_k||^2, \tag{41}$$

Using the above settings, the obtained results were satisfactory. Fig. (4) shows the value of the objective function as a function of the iteration round. It indicates that the GA-ILC framework is able to find the optimal solution after only a few iterations and the convergence is monotonic which is a very strong property for ILC. Also the tracking of the reference signal is extremely accurate see Fig. (5) .

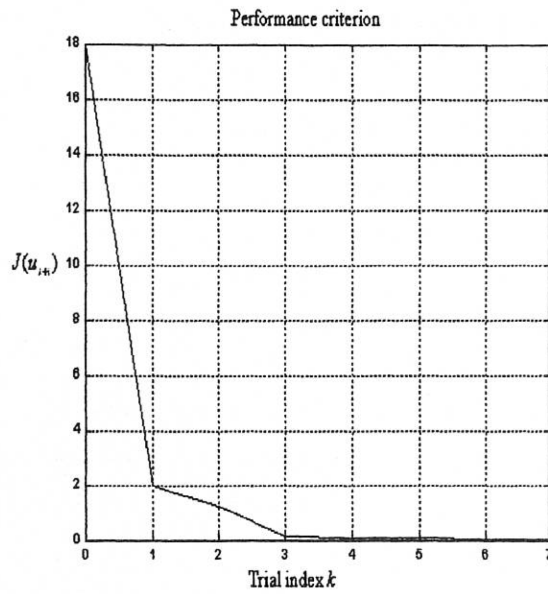

Figure 4: $||\vec{e}(k)||$ as a function of iteration round k .

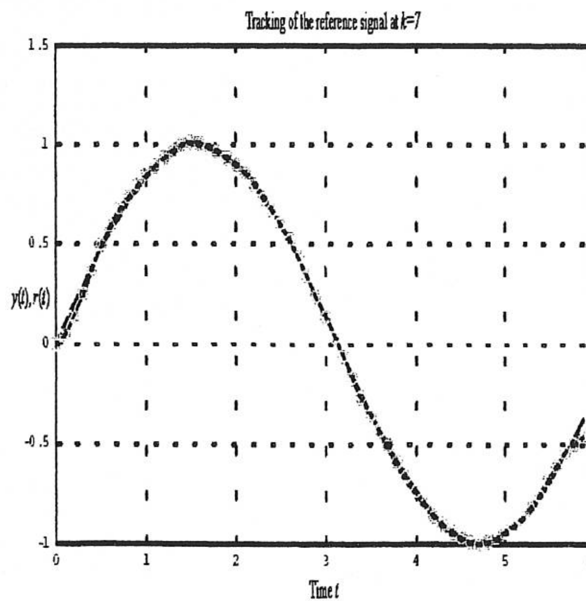

Figure 5: Tracking of the reference signal after iteration $k = 6$.

The performance of the FIR filter is indicated in Fig. 6, where it can be obtained that

the filter successfully removes the unwanted noise from the selected input signal. For this particular simulation example the following low-pass filter was used:

$$u^{*'}{}_{k+1}(n) = \begin{bmatrix} 0.25u^*_{k+1}(n) + 0.2u^*_{k+1}(n-1) + 0.2u^*_{k+1}(n-2) + \dots \\ 0.2u^*_{k+1}(n+1) + 0.15u^*_{k+1}(n+2) \end{bmatrix} \tag{42}$$

with initial conditions

$$u^*(0) = \begin{bmatrix} u^*{}_{k+1}(0) & 0.6 * u^*{}_{k+1}(1) + 0.4 * u^*{}_{k+1}(0) \dots \\ & 0.4 * u^*{}_{k+1}(2) + 0.3u^*_{k+1}(1) + 0.3u^*_{k+1}(0) \end{bmatrix} \tag{43}$$

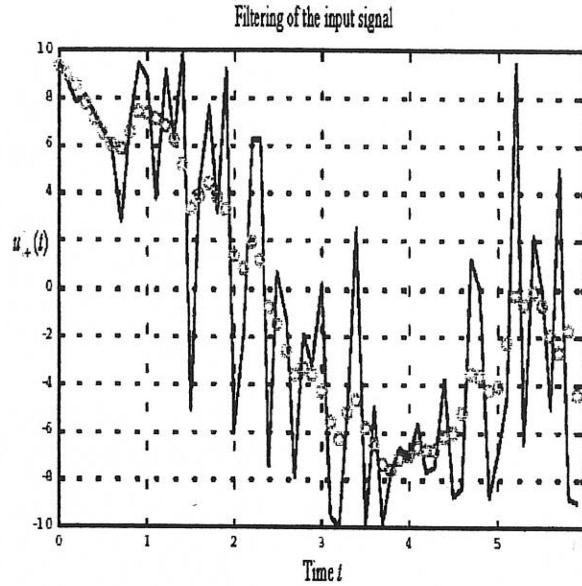

Figure 6: Filtering the selected *noisy* input signal.

8.2 Simulation example 2

In this simulation example the proposed GA-ILC algorithmŠs performance is evaluated using a discrete time nonminimum phase dynamical system. The equations describing the dynamical model are:

$$\begin{aligned} x_1(i+1) &= -0.1x_3(i) + u(i) \\ x_2(i+1) &= x_i \\ x_3(i+1) &= x_2(i) \\ y(i) &= x_1(i) + 2.5 * x_2(i) + x_3(i) \end{aligned} \tag{44}$$

The GA-ILC algorithm needs to be able to find the input that tracks the following desired output.

$$y_d(i) = 0, i = 0, 1$$
$$y_d(i) = sin(0.05\pi(i-2)), 2 \leq i \leq 22 \tag{45}$$

The GA specifications used for simulation purpose are shown in Table 2.

SGA parameter	Setting
Population size	100
Total Generations	100
Number of iterations	4
Coding	Binary representation, 40 bits per decision variable
Selection	Low-level stochastic universal sampling routine
Recombination	Shuffle crossover with reduced surrogate, probability=0.8
Mutation	Bit-flipping, random probability
Generation gap	0.95
Elitism	Best five chromosomes of previous population forward to next one

Table 2: Algorithm parameters for the linear case

Once again the algorithms performance is satisfactory, figure (7). Objective function is minimized after a few iterations and the convergence is monotonic while tracking of the reference signal is ideally (see Figure 8).

The *noisy* input signal is successfully transformed into a 'smooth' one by the use of a low-pass FIR filter as shown in in Fig. 9. For this particular simulation example the following α values where used:

$$u^{*'}{}_{k+1}(n) = \begin{bmatrix} 0.3 * u^*{}_{k+1}(n) + 0.25 * u^*{}_{k+1}(n-1) + 0.1 * u^*{}_{k+1}(n-2) + \dots \\ 0.25 * u^*{}_{k+1}(n+1) + 0.1 * u^*{}_{k+1}(n+2) \end{bmatrix} \tag{46}$$

with initial conditions

$$U^*(0) = \begin{bmatrix} u^*_{k+1}(0) & 0.4u^*_{k+1}(1) + 0.4u^*_{k+1}(0) + 0.2u^*_{k+1}(2) \dots \\ & 0.3u^*_{k+1}(2) + 0.25u^*_{k+1}(1) + 0.25u^*_{k+1}(0) + 0.2u^*_{k+1}(3) \end{bmatrix} \tag{47}$$

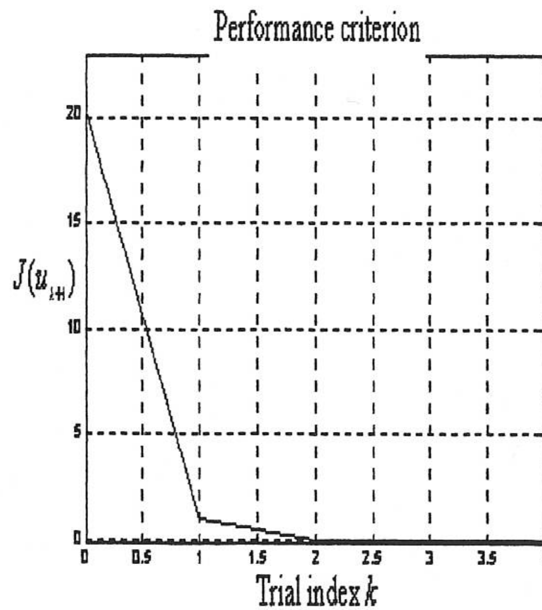

Figure 7: $||\vec{e}(k)||$ as a function of iteration round k .

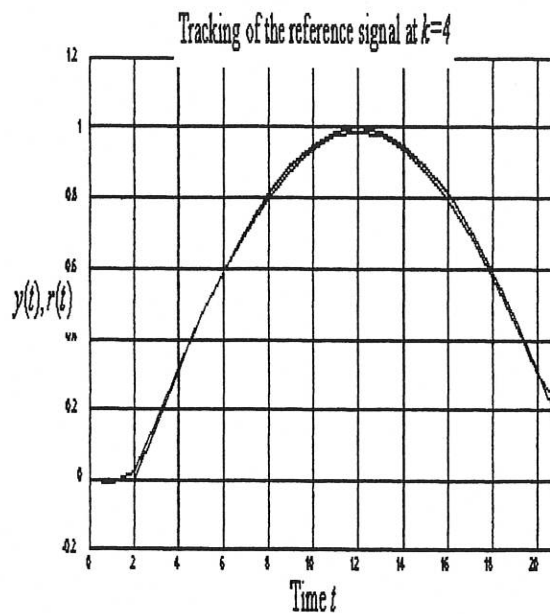

Figure 8: Tracking of the reference signal after iteration $k = 4$.

8.3 Simulation example 3

As a next simulation consider the following nonlinear model of a one-joint manipulator

$$\frac{d^2}{dt^2}\theta(t) = \frac{1}{J}(u(t) - F(t)) + \frac{1}{J}(\frac{1}{2}m + M)gl\sin\theta(t) \tag{48}$$

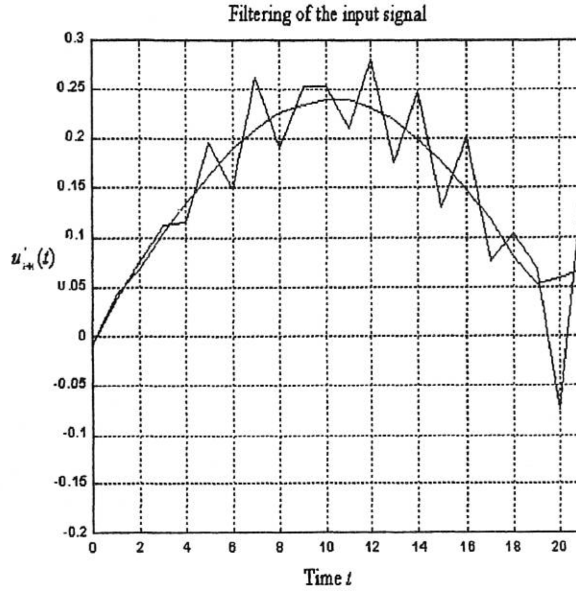

Figure 9: Filtering the selected *noisy* input signal.

where $\theta(t)$ is the angular position of the manipulator, $u(t)$ is the applied joint torque, $F(t)$ is the friction torque; m, l are the mass and length of the manipulator respectively, M is the mass of the tip load, g is the gravitational acceleration and J is the moment of inertia with respect to joint and is given by $J = Ml^2 + ml^2/3$. The reference signal is chosen to be

$$\theta_r(t) = \theta_b + (\theta_b - \theta_f)(15\tau^4 - 6\tau^5 - 10\tau^3) \tag{49}$$

where $\tau = t/(t_f - t_0)$. The numerical parameters used in the simulation are $t_0 = 0, t_f = 1, \theta_b = 0, \theta_f = 90^o$. Furthermore, the sampling interval used in the simulations is $T_s = 0.01$ and the friction $F(t) = 0$ in this simulation. The cost function in this case was chosen to be

$$J(u_{k+1}) = ||e_{k+1}||^2 + 0.01||u_{k+1} - u_k||^2 \tag{50}$$

The parameter values used in the simulation for the model (48) are shown in Table 3.

Parameter	m	l	g	M
Value	2	0.5	9.8	4
Unit	Kg	M	m/sec^2	Kg

Table 3: Numerical values for the nonlinear model

BFGA parameter	Setting
Population size	300
Total Generations	100
Number of iterations	9
Coding	Binary-value representation
Selection	Low-level stochastic universal sampling routine
Recombination	Shuffle crossover with reduced surrogate, probability=0.9
Mutation	Bit-flipping, random probability
Generation gap	0.98
Elitism	Best six chromosomes of previous population forward to nex one

Table 4: Algorithm parameters for the nonlinear case

Table 4 shows the parameter values used in the GA algorithm.

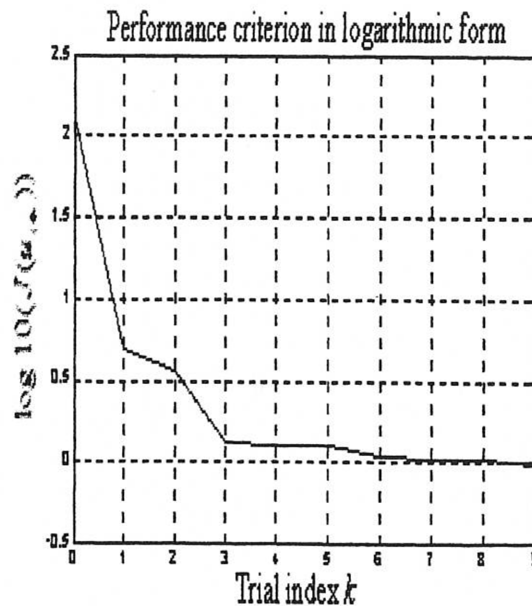

Figure 10: $||\vec{e}(k)||$ as a function of iteration round k with the nonlinear example.

Fig. 10 shows the corresponding error behaviour. It is clear that the Norm-Optimal al-

gorithm is able to produce the input function that gives ideal tracking even with this highly nonlinear simulation example. This input function is also smooth, as Fig. 11 illustrates.

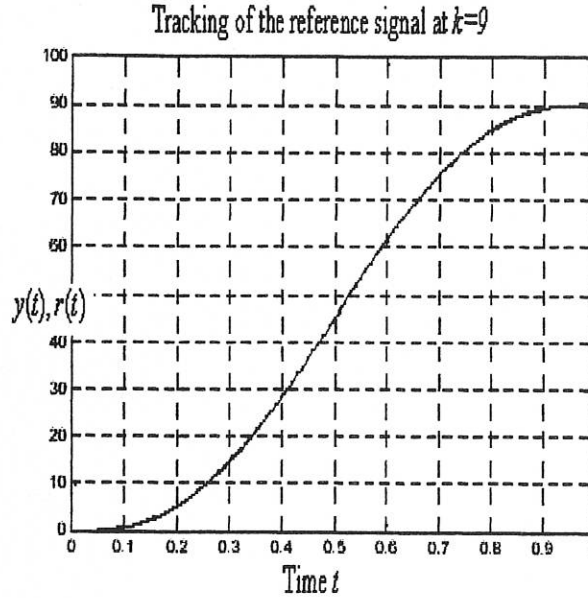

Figure 11: Input function after nine iteration rounds.

Finally the following low-pass filter was used:

$$u^{*\prime}{}_{k+1}(n) = \begin{bmatrix} 0.3u^*_{k+1}(n) + 0.2u^*_{k+1}(n-1) + 0.15u^*_{k+1}(n-2) + \dots \\ 0.2u^*_{k+1}(n+1) + 0.15u^*_{k+1}(n+2) \end{bmatrix} \quad (51)$$

with initial conditions

$$u^*(0) = \begin{bmatrix} u^*_{k+1}(0) & 0.4u^*_{k+1}(1) + 0.3u^*_{k+1}(2) + 0.3u^*_{k+1}(0) \dots \\ & 0.4u^*_{k+1}(2) + 0.3u^*_{k+1}(1) + 0.3u^*_{k+1}(0) \end{bmatrix} \quad (52)$$

9 Conclusions

In this paper the possibility of using GAs in the context of Norm-Optimal ILC algorithms were investigated. The basic idea behind the GA approach is that it can be used to implement Norm-Optimal ILC for any kind of plant models unlike other traditional optimisation based ILC methods which work only for specific classes of linear plant models. This new approach was tried on two linear and a nonlinear example. These three examples showed

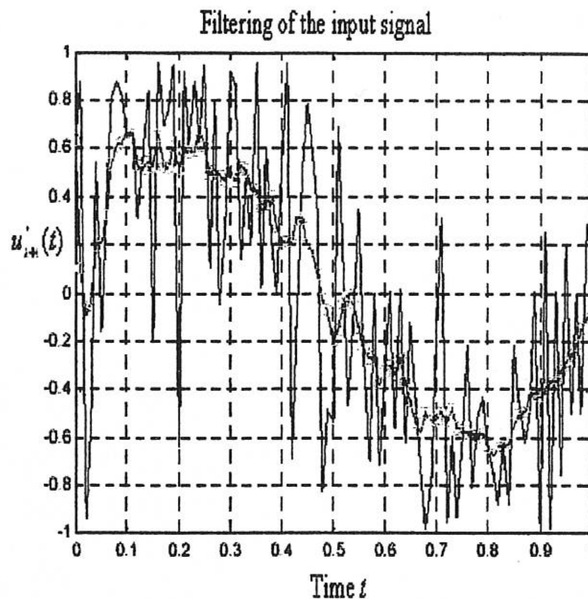

Figure 12: Filtering the selected *noisy* input signal.

that the proposed GA-ILC algorithm can find the optimal solution, and the convergence is monotonic, just as theory suggests. Furthermore, a low-pass FIR filter was also designed for reducing the unwanted *noise* from the selected input signal.

As a future work it should be investigated how modelling uncertainty can be taken into account in the proposed algorithm. Another interesting possibility is to extend the proposed GA approach to cover the Predictive Norm-Optimal ILC-algorithm presented in [3].

References

[1] N. Amann and D. Owens. *Optimal algorithms for iterative learning control.* PhD thesis, June 1996.

[2] N. Amann, D.H. Owens, and E. Rogers. Iterative learning control using optimal feedback and feedforward actions. *International Journal of Control*, 65(2):277–293, 1996.

[3] N. Amann, D.H. Owens, and E. Rogers. Predictive optimal iterative learning control. *International Journal of Control*, 69(2):203–226, 1998.

[4] S. Arimoto, S. Kawamura, and F. Miyazaki. Bettering operations of robots by learning. *Journal of Robotic Systems*, 1:123–140, 1984.

[5] P. Bondi, G. Casalino, and G. Bartolini. On the ilc theory for robotic manipulators. *IEEE Journal of Robotics*, pages 14–22, 1988.

[6] K. Buchheit, M. Pandit, and M. Befort. Optimal iterative learning control of an extrusion plant. In *Proceedings of IEE International Conference of Control*, pages 652–657, Coventry, UK, 1994.

[7] A. Chipperfield. Genetic algorithms toobox user's guide. Technical report, The University of Sheffield, 1996.

[8] L. Davis. *Handbook on Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[9] P. Fleming and R. Purchouse. Genetic algorithms in control systems engineering. Technical report, The University of Sheffield, 2001.

[10] K. Furuta and M. Yamakita. The design of a learning control system for multivariable systems. In *Proceedings of the 30th Conference on Decision and Control*, pages 371–376, Philadelphia, Pennsylvania, USA, 1987.

[11] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[12] V. Hatzikos, D. H. Owens, and J. Hatonen. An evolutionary based optimisation method for nonlinear iterative learning control systems. In *Proceedings of American Control Conference*, Denver, USA, June 2003.

[13] P. Lucibello. Learning control of linear systems. In *Proceedings of American Control Conference*, pages 1888–1992, Chicago, USA, June 1992.

[14] Z. Michalewics. *Genetic algorithms + Data structures = Evolution programs*. Springer-Verlag, 2nd Edition, 1994.

[15] T. Sogo and N. Adachi. A gradient-type learning control algorithm for linear systems. In *Proceedings of ASCC*, pages 227–230, Tokyo, Japan, July 1994.

[16] T. Sugie and T. Ono. An iterative learning control law for dynamical systems. *Automatica*, 27(4):729–732, 1991.

[17] T. Tao, R.L. Kosut, and G. Aral. Learning feedforward control. In *Proceedings of American Control Conference*, pages 2575–2579, Baltimore, USA, June 1994.

[18] Y. Yupu, X. Xu, and W. Zhang. Real-time stable self-learning fnn controller using genetic algorithm. *Fuzzy sets and Systems*, 100:173–178, 1998.

APPENDIX

An Introduction to the Genetic Algorithm Toolbox in Matlab

V. Hatzikos*, D. H. Owens*

1 Introduction

Over the years, Matlabs capabilities of implementing various control engineering methods and theories into dynamical systems have been explored with high degrees of success. Some of these traditional methods, as Root-locus design, Nyquist plots analysis and, Bode diagrams, can be coded in Matlab using m-files or even implemented by graphical user interfaces procedures. However lately, more advanced techniques became available for execution in Matlab mainly due to the fact that toolboxes can be added to extend the designed system, providing, for example, signal processing facilities, fuzzy logic option, or even genetic algorithm optimisation solutions.

In this report, the Genetic Algorithm *GA* toolbox in Matlab will be described. At first, an introduction to the basic GA operators will be presented. Then, the procedure of how the GA toolbox uses MATLAB's matrix functions to build a set of versatile tools for implementing a wide range of genetic algorithms methods will be discussed. Finally, GAs performance will be evaluated using simulation approaches.

2 Tutorial

Matlab has a wide variety of functions useful to the genetic algorithm practitioner and those wishing to experiment with the genetic algorithm for the first time. Furthermore, it is a modern programming language and problem solving environment: it has sophisticated data structures, contains built-in debugging and profiling tools, and supports object oriented programming. These factors make Matlab an excellent language with which the user is able to explore the potential of GAs. The different versions of Matlab are presented in chronological order in Table 1.

Year	Version	Characteristics
1983	Matlab 1	Capable on solving mathematical equations.
1985	Matlab 2	Extended number of commands and functions.
1987	Matlab 3	Improved graphics analysis(colour), faster interpreter.
1992	Matlab 4	Graphical user interface control, animation and visualisation tools, Microsoft Windows support and debugger options.
1997	Matlab 5	Object-oriented programming, extended number of toolboxes, profiler. cell arrays, new ordinary differential equation solver, structures, improved Simulink environment.
2000	Matlab 6	Improved GUI design, Real-time toolbox, Matlab desktop includes help browser options, graphics object transparency, Java support.

Table 1: Different versions of Matlab

3 Installation in Matlab

The GA toolbox is a collection of m-files (about 40). (In this report we will work with the GA toolbox provided by the University of Sheffield [5]. For more details the reader is referred to the references section where more information are available). It is recommended that all these files should be stored in a directory named Genetic of the main Matlab/toolbox directory. In each m-file a genetic operator function is implemented.

Also a number of demonstrations are accessible. Both binary or real value coded GA structures can be implemented using functions from the GA toolbox. Additionally, a set of test functions is also available, in a separate directory from the GA toolbox functions, called $test_{fns}$.

4 Introduction to Genetic Algorithms

The Genetic Algorithms (GAs) is a stochastic global search method that mimics, as the rest EA do, the metaphor of natural biological evolution. They were invented by John Holland in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s [13]. In contrast with evolution strategies and evolutionary programming, Hollands original idea was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be

imported into computer systems.

Thereafter, a series of literature and reports became available. Some of the most interesting books about GAs are [11], [17], [4], [22], [16], [6]. GAs is inspired by the mechanism of natural selection where stronger individuals are likely the winners in a competing environment. They combine survival of the fittest among string formations with a structured yet randomised information exchange to form a search algorithm with some of the innovative flair of human search. Even though rigorous mathematical analysis of the GAs is difficult and is still incomplete, GAs has been applied in a variety of areas (e.g. robot path-planning, stability analysis, fault diagnosis, combinatorial problems, and optimisation problems) with high degrees of success within each.

5 Major Elements of a simple GA

Genetic Algorithms are search algorithms based on the mechanics of natural selection and natural genetics. It starts with a population of strings (potential solutions) applying the principle of survival of the fittest to produce better and better approximations to a solution. In every generation, a new set of artificial creatures (chromosomes, strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure.

5.1 Population representation

Individuals are encoded as strings, chromosomes, composed over some alphabets, so that the genotypes (chromosomes values) are uniquely mapped onto the decision variable (phenotypic) domain, table 2. Each chromosome can be thought of as a point in the search space of candidate solutions.

genotypic domain	**phenotypic domain**
001 ⋮101 ⋮110	(1,5,6)

Table 2: Example mapping between chromosomes and decision variables

The coding of the chromosome representation may vary according to the nature of the problem itself. In general the binary alphabet 0, 1 bit encoding is the most classic method used by GA researchers because of its simplicity and trace ability. Here, each decision variable in the parameter set is encoded as a binary string and these are concatenated to form a chromosome. An initial population of chromosomes might look like this:

No.	Population of Chromosomes
1	010011101010110
2	101000011010001
3	010001011111000
4	111101100010101

Table 3: Population of Chromosomes

Other chromosomes representations have also been introduced. [23] initialised real-valued genes of chromosomes. According to him this kind of approach offers significant advantages over the *traditional* binary encoding in terms of computational time, i.e.

1. No need to convert chromosomes to phenotypes before each chromosome evaluation
2. Less computer memory is required
3. Freedom to use different genetic operators
4. Easier incorporation of domain-specific knowledge

The use of real-valued encoding is described in detail by [17]. Also, integer representation of the chomosome's genotype have been used by [15] providing improved results compared to binary encoding.

5.2 The Objective and Fitness functions

Each chromosome in the GA population is assigned an objective function value which describes the chromosomes performance in the problem domain. However, during the reproduction phase an additional fitness value is required by the GA, i.e each chromosome is assigned a fitness value f derived from its raw performance measure given by the objective function. Intuitively, we can think of the function f as some measure of profit, utility, or goodness that we want to maximize [11]. Selecting strings according to their fitness values means that strings with a higher value have a higher probability of contributing one or more offspring in the next generation.

5.3 Selection phase

Selection mechanism is the process of determining the number of times a particular chromosome is chosen for reproduction and thus the number of offsprings that a chromosome

will produce for the next population. In theory, the fittest chromosomes of the old population will be the ideal choice for the reproduction phase. However, this may lead to premature convergence into specific unwanted areas of the search space (local minimum). Thus, it is essentially important that most of the existence chromosomes of the current population will contribute in the production of new offsprings.

A scheme called Roulette Wheel Selection is one of the most common techniques being used for such a proportionate selection mechanism (see equation 1). Each chromosome occupies an interval space within the roulette wheel according to the fitness value. In general, the probability of a string to be selected for reproduction is given by

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{1}$$

where f_i is the fitness value of the $i-th$ chromosome and N is the population size [8]. Thus the fittest chromosome occupies the largest interval. For example, lets consider the population in table 3. From Figure 1 it can been seen that chromosome No.3 is the fittest one.

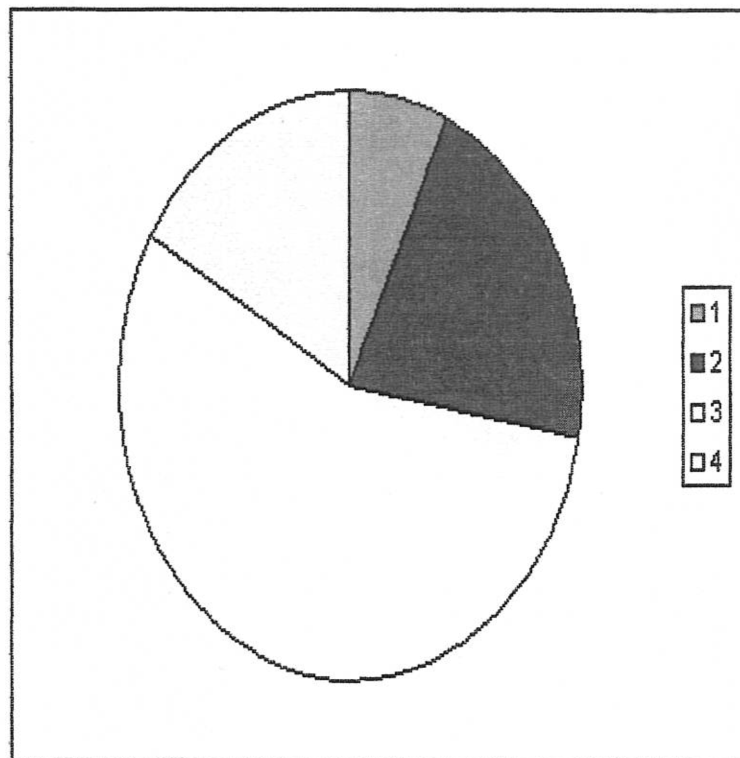

Figure 1: Roulette wheel selection

Also a tournament selection mechanism has been introduced by [12]. Tournaments are played between two chromosomes and the better one is selected for reproduction. All the chromosomes in the population are eventually picked to participate in the tournament. In fact, it was shown that the tournament selection mechanism has better or equivalent convergence and computational time complexity properties when compared to any other reproduction operator that exists in the literature.

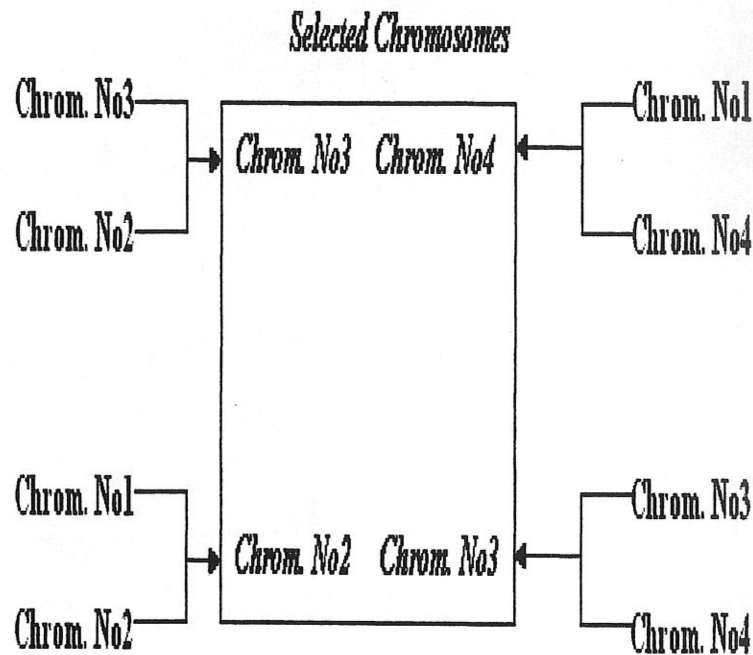

Figure 2: Tournament selection

[1] proposed a stochastic universal sampling (SUS) technique. In this version, chromosomes are placed into a roulette wheel according to their fitness value. However, only one random number n is selected for the whole selection procedure and a set of N equi-spaced numbers is created (see equation 2). Thereafter, a solution corresponding to each member of S is chosen from the roulette wheel.

$$S = [n, n + 1/N, s + 2/N, \ldots, s + (N-1)/N] \tag{2}$$

From figure 3 it can be seen that Chrom. No3 is the fittest one and will be selected three times while the rest chromosomes will be selected only one time.

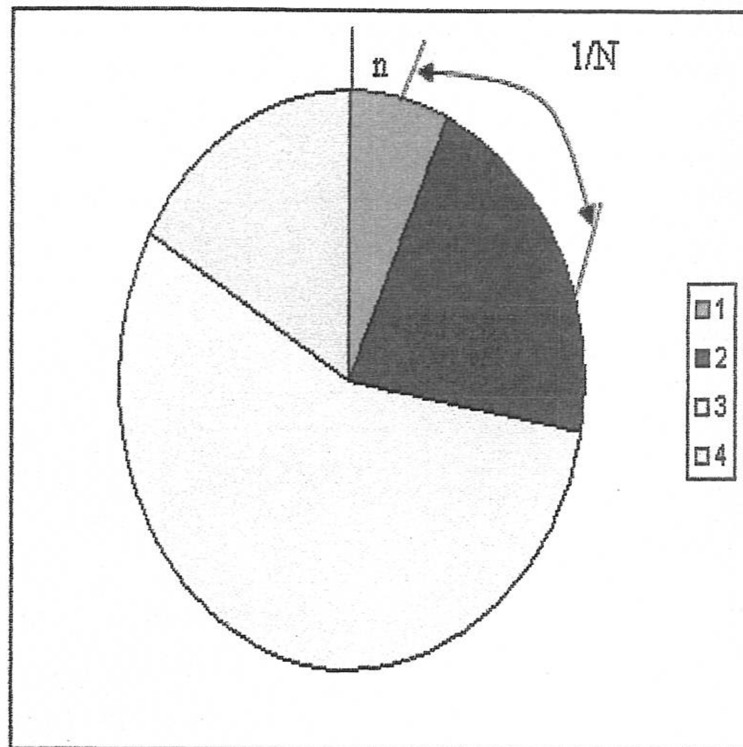

Figure 3: Stochastic universal sampling mechanism

The *ranking* of the chromosomes can provide some robustness to the selection mechanism. The chromosomes are sorted according to their fitness value, from worst (rank 1) to the best (rank 4). Each chromosome in the sorted list is then assigned a new fitness value according to the rank of the chromosome in the list. Thereafter, one of the selection operators described above i.e roulette wheel, tournament, stochastic universal sampling, is applied to the ranked chromosomes and N chromosomes are selected for the reproduction phase.

5.4 Genetic Operators

After reproduction, two fundamental operators: Crossover and Mutation are required. Crossover selects genes (bits) from parent chromosomes and creates a new offspring. The simplest way to do this, is to choose randomly some crossover point, at random between 1 and the chromosome length less one $[1, l-1]$ and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent. For example, consider strings No.1 and No.4 from the initial population in table 3 to

be selected for single point crossover. This process is illustrated in Table 7. It needs to be mentioned that according to famous biologists and evolutionists [7] and [10], simple crossover of chromosomes it is the mechanism used for the creation of complex life forms from simple ones.

Selected Chromosomes	Simple crossover mechanism
Chrom. No1	0100111 ⋮01010110
Chrom. No4	**1111011 ⋮00010101**
Offspring 1	0100111 ⋮ **00010101**
Offspring 2	**1111011**⋮01010110

Table 4: Crossover mechanism during evolution process

Other crossover mechanisms have also been introduced. The multi-point crossover select m crossover positions, $m \in (1, 2, \ldots, l-1)$ sorted into ascending order. Then the bits between successive crossover points are exchanged between the two pairs to produce two new offsprings (see Table 5). According to [20] the disruptive nature of multi-point crossover appears to encourage the exploration of the search space, rather than favoring the convergence to highly fit individuals early in the search, thus making the search more robust.

Selected Chromosomes	Multiple point crossover mechanism
Chrom. No2	101 ⋮000011 ⋮01 ⋮0001
Chrom. No3	**010 ⋮001011 ⋮11 ⋮1000**
Offspring 1	101 ⋮**001011** ⋮01 ⋮**1000**
Offspring 2	**010** ⋮000011 ⋮**11** ⋮0001

Table 5: Multiple point Crossover mechanism during evolution process

More complicated crossover operators are:

- **Uniform Crossover** which was introduced by [21]. This crossover mechanism makes every gene in the chromosome a potential crossover point. In order to achieve that a crossover mask is created, having the same length as the chromosome, see Table 6. Each gene in the mask indicates which parent will supply the offspring with which bits. For example in table 6 the first offspring is produced by taking the bit

from Chrom. No1 if the corresponding mask gene is 1 or the bit from Chrom. No4 if the corresponding mask gene is 0. Offspring 2 is created using the inverse of the mask.

Selected Chromosomes	**Uniform crossover mechanism**
Chrom. No1	010011101010110
Chrom. No4	**111101100010101**
Facade	010111011000010
Offspring 1	111011101010111
Offspring 2	0 10101100010100

Table 6: Uniform crossover mechanism during evolution process

- **shuffle crossover** where the selected parents are randomly shuffled before the crossover operator takes place [3].
- **reduced surrogate crossover** where crossover points occur only where gene values of the selected parents differ [2].
- furthermore, a special crossover operators have been introduced for recombination of real-valued chromosomes. **Intermediate Crossover** which is a method of producing new offsprings whose genes are around and between of parent genes [19]. To be more specific, the offspring's genes are calculated from the equation:

$$O_1 = P_1 * \alpha(P_2 - P_1) \tag{3}$$

where $\alpha \in \Re$ and is usually in the range $[-0.25\ 1.25]$. A new value of α is introduced for each pair of parent genes. According to [19] using this crossover operator we can avoid premature convergence thus satisfactory explore the search space.

In general, the crossover operation is not necessarily performed on all strings in the population. Instead it is applied with a probability rate P_c, typical value between 0.6 and 1.0. After a crossover is performed, mutation takes place. This is to prevent falling all solutions in population into a local optimum of solved problem. It alters each bit of a chromosome randomly with a small probability P_m with a typical value less than 0.1. For binary encoding, a few randomly chosen bits can be switched from 1 to 0 or from 0 to 1.

For real-valued chromosomes the mutation operator is achieved by either flipping the selected gene values of the parent chromosomes or by selecting new values which replace

Selected Chromosomes	Mutation mechanism
Chrom. No1	01001110 :1 :010110
Chrom. No4	**11110110 :0 :010101**
Offspring 1	01001110 :**0** :010110
Offspring 2	**11110110**:1 :**010101**

Table 7: Crossover mechanism during evolution process

some genes in the selected parents. However, it has been suggested by many researchers [14], [23] that a higher mutation rate P_m should be used for real-valued mutation than for binary-valued mutation. Overall, the choice of Pc and Pm can be a complex nonlinear optimisation problem to solve. Furthermore, their settings are critically dependent upon the nature of the objective function. According to [16] some guidelines can be introduced:
For large population size (100)

- Crossover rate Pc: 0.6
- Mutation rate Pm: 0. 001

For small population size (30)

- Crossover rate Pc: 0.9
- Mutation rate Pm: 0.01

5.5 Reinsertion

After crossover and mutation, the new chromosomes are then decoded, the objective function evaluated, a fitness value assigned to each chromosome and the fittest ones selected for mating and so on the process through subsequent generations (this approach is also known as simple genetic algorithm (SGA) figure 4). This is motivated by a hope, that the new population will be better than the old one.

It is usually required that the algorithm follows an *elitist strategy*. This means that fittest chromosomes of the old population always propagate through successive generations. However, care must be taken when this approach is used because the new population should not be dominated by old chromosomes i.e premature convergence. The whole process is terminated when the fitness of a population remain static for a number of generations. In this case the fittest chromosomes of the last generation are possibly solutions to

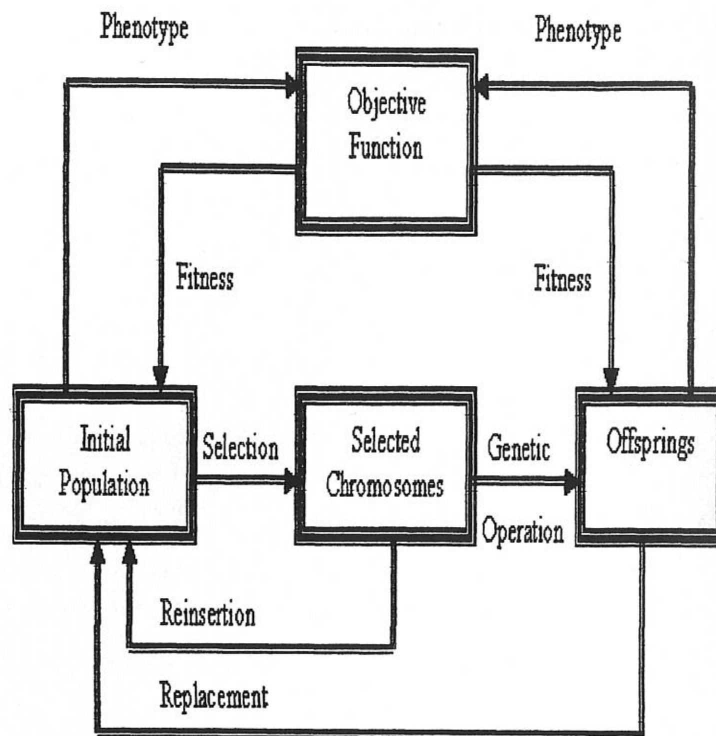

Figure 4: A genaral description of a simple genetic algorithm

the problem. If however these chromosomes are not acceptable solutions for the simulated problem, the GA may be restarted or a fresh search initiated.

6 Simple Genetic Algorithm

As mentioned earlier, the simple genetic algorithm approach was initially introduced by [11]. Since then, many variations of the original SGA have been proposed and applied to a variety of search and optimisation problems with high degrees of success. Overall, SGA has proved to be a robust evolutionary based algorithm in terms of performance results and adaptation to particular problems domain. Most SGA methodologies have some elements in common, see table 8.

This pseudo-code structure can easily be coded in Matlab using functions from the GA toolbox. Some of the available functions can be seen in table 9. As previously mentioned these functions are written in m-files and implement the most important routines in genetic algorithms. This is the main reason why Matlab is preferred for the implementation of

1. **Start**. Generate random population of chromosomes
2. **Fitness**. Evaluate the fitness $f(x)$ of each chromosome in the population
3. **New population**. Create a new population by repeating following steps until the new population is complete
3.1 **Selection**. Select two parent chromosomes from a population according to their fitness
3.2 **Crossover**. With a crossover probability cross over the parents to form a new offspring
3.3 **Mutation**. With a mutation probability mutate new offspring at each locus
3.4 **Reinsert**. Place new offspring in a new population
4. **Replace**. Use new generated population for a further run of algorithm
5. **Test**. If the end condition is satisfied, **stop**, and return the best solution in current population
6. **Loop**. Go to step 2

Table 8: A pseudo-code description of a *SGA*

GAs rather than other computer languages. However, other *GA* coding structures have been proposed over the years. [11] implemented *GA* using *Pascal* while [18] preferred *C* computer language.

1.[Start]	crtbp (create binary population), crtrp (real-value population)
2.[fitness]	ranking (most popular one), scaling
3.1[selection]	rws (roulette wheel selection), sus (stohastic universal sampling) select (high level selection)
3.2[crossover]	xovsp (single-point crossover), xovdp, xovsh (shuffle crossover) xovshrs (shuffle reduced surrogate crossover) etc.
3.3[mutation]	mut (discrete mutation), mutate, mutbga(real-value mutation)
3.4[reinsert]	reins (reinsertion of new offsprings)

Table 9: m-files of the GA toolbox in Matlab

7 GAs versus traditional methods

Generally speaking, it can be seen that GAs differs substantially from more traditional search and optimisation methods. According to [4] the four most significant differences are:

- GAs searches a population of points in parallel, not a single point.
- GAs does not require derivative information or other auxiliary knowledge; only the objective function and the corresponding fitness levels influence the directions of search.
- GAs use probabilistic transition rules, not deterministic ones.
- GAs work on an encoding of the parameter set rather than the parameter set itself

8 Simulation

To demonstrate the effectiveness of the *GAs* in *Matlab*, a simple optimisation problem is used for the simulation study. This example reveals how a *SGA* can be constructed using routines from the *GA* Toolbox to solve an optimisation problem. The objective function to be minimised is an extended version of De Jongs [20] first test function:

$$f(x) = \sum_{i=1}^{k} x_i^2 \ , \ -500 \leq x_i \leq 500 \tag{4}$$

where the search space is between $[-500\ 500]$ and $k \in [1, 2, \ldots, 20]$. The minimum of this function is, of course, located at $x_i = 0$. In reality, we can think of x_i being the error of a system that is needed to be minimised to zero as soon as possible.

The *SGA* used to the optimisation of the cost function was developed to the specifications described in table 10. Overall, the *GA* structure is very similar to the one described in Figure 4. To make life easier, binary representation of the chromosomes was introduced. The precision of each decision variable was set to 20 binary values:

Chrom1: 01010100100111100011 ... 01111000101000110001 ....

$\Downarrow$ $\qquad$ $\Downarrow$

x_1 $\qquad$ x_2

The population size was decided to be 40 (number of chromosomes in each population). Furthermore, a generally popular, *select* function is used for the selection of the

chromosomes which are gone be recombined in order to produce the new offsprings. In addition, the recombination (crossover) procedure includes multiple point crossover of the selected chromosomes with a probability rate of 0.7. Finally the algorithm is set to implement an elitist strategy whereby the four most fit individuals (generation gap arranged to 0.9) always propagate through to successive generations. These settings may vary according to the nature of the problem.

SGA parameter	Setting
Population size	40
Total Generations	400
Coding	Binary-value representation, 30 bits per decision variable
Selection	Low-level stochastic universal sampling routine
Recombination	Shuffle crossover with reduced surrogate, probability=0.7
Mutation	Bit-flipping, random probability
Generation gap	0.9
Elitism	Best 4 chromosomes of previous population forward to nex one

Table 10: Algorithm parameters

Using the above specifications the SGA is able to minimize the cost function with satisfactory convergence rate. Different settings of the SGA may be proven capable to improve the performance of the algorithm. However, the algorithm performs reasonably well according to the following Figure. The initial value of the cost function is around 10000. The SGA needs 400 generations for the minimization of this enormous cost function. After the 400th generation the value is reduced to 0.45, an almost ideally solution.

As mentioned above the GA is able to minimize $f(x)$ to a very small value, almost optimum, of about 0.45. Overall, analysis of the convergence speed indicates that the algorithm needs around 2 minutes for the execution of the procedure, which is reasonably good. Most of others traditional optimisation methods require the computation of complicate mathematical formulas, which take lot of time and are very difficult to be implemented in software languages as $MATLAB$.

Finally, the reader is referred to a number of simulations approaches that will help him to further understanding of the GA procedure. More specific, [9] constructed a test

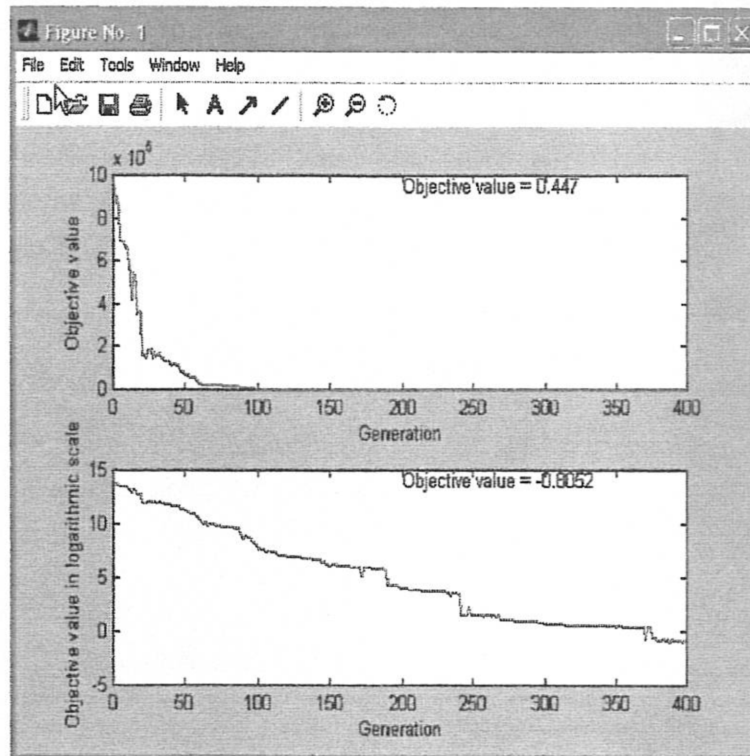

Figure 5: Convergence speed of $f(x)$

environment of five problems in function minimisation. He took care to include functions with the following characteristics:

1. Continuous/discontinuous
2. Convex/nonconvex
3. Unimodal/multimodal
4. Quadratic/nonquadratic
5. Deterministic/stochastic

The functions and their coding characteristics are presented in Table 11. Each can be implemented using approaches similar to Figure 4 structure.

Number	Function	search space
1	$f(x_i) = \sum_1^3 x_i^2$	$-5.12 \leq x_i \leq 5.12$
2	$f(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$2.048 \leq x_i \leq 2.048$
3	$f(x_i) = \sum_1^5 int(x_i)$	$-5.12 \leq x_i \leq 5.12$
4	$f(x_i = \sum_1^{30} ix_i^4 + Gauss(0,1)$	$-1.28 \leq x_i \leq 1.28$
5	$f(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i - a_{ij})^2}$	$-65.536 \leq x_i \leq 65.536$

Table 11: Algorithm parameters

9 Conclusions

Over the years, GA toolbox in $MATLAB$ has been developed to a very useful tool for optimisation procedures. Several GA toolboxes are available for the technical computing package $MATLAB$. In this chapter, the one developed by the University of Sheffield is described. This toolbox provides a wide-range of GA tools and is easily extensible.

Also, two Math works (company that created MATLAB) GA toolboxes are available. *Flextool* (GA), developed by *RKSites.com* is a modular user-interface driven tool. The alternative is the Genetic Search Toolbox, developed by *Optimal Synthesis Incorporated*. This toolbox features a graphical code writer and has apparently been tested at United States government research laboratories.

10 GA software

The GA toolbox, developed at the University Of Sheffield, provides a wide-range of GA tools and is easily extensible. For more information visit:
http://www.shef.ac.uk/uni/projects/gaipp/gatbx.html.
Also visit: http://www.geatbx.com.

The GA toolbox, developed by RKSites.com for Mathworks company. For more information about this FlexTool (GA) toolbox visit:
http://www.flextool.com.

The Genetic Search Toolbox, developed by Optimal Synthesis Incorporated can be found in the following address: http://www.optsyn.com.

A freeware toolbox, GAOT, developed by North Carolina State University provides several selection, crossover and mutation choices. GAOT is available for download at http://www.ie.ncsu.edu/mirage/GAToolbox/gaot/.

Parallel genetic algorithm Chipperfield Q629.8 no518

References

[1] J.E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of ICGA1*, pages 101–110, 1985.

[2] L. Booker. In genetic algorithms and simulated annealing. In *L. Davis (Ed), Morgan Kaufmann*, pages 61–73, 1987.

[3] R.A Caruana, L.A. Eshelman, and J.D. Schaffer. Representation and hidden bias2:eliminating defining length bias i genetic search via shuffle crossover. In *In 11th Joint conference on Artificial intelligence*, pages vol1, 750–755, 1989.

[4] A. Chipperfield. Genetic algorithms toobox user's guide. Technical report, The University of Sheffield, 1996.

[5] A. Chipperfield, H. Pohlheim, C. Fonseca, and P.J. Fleming. Genetic algorithms toobox in matlab. Technical report, The University of Sheffield, 1996.

[6] L. Davis. *Handbook on Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[7] R. Dawkins. *The selfish Gene.* New York: Oxford University Press, 1976.

[8] K. Deb. *Multi-objective optimization using evolutionary algorithms.* John Wiley sons LTD, 2001.

[9] K.A. DeJong. *An analusis on the behaviour of a class of genetic adaptive systems.* PhD thesis, The University of Michigan, 1975.

[10] N. Eldredge. *Macro-Evolutionary Dynamics: species, niches and adaptive peaks.* New York: McGraw-Hill, 1989.

[11] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, 1989.

[12] D.E. Goldberg and K. Deb. A comparison of selection schemes used in genetic algorithms. *In Foundations of Genetic Algorithms*, pages 69–93, 1991.

[13] J.H. Holland. *Adaptation in natural and artificial systems.* Ann Arbor:The University of Michigan Press, 1975.

[14] C.Z. Janikow and Z. Michalewicz. An experimental comparison of binary and floating-point representations in genetic algorithms. In *Proceedings of ICGA4*, pages 31–36, 1991.

[15] C.B. Lucasius and G. Kateman. Towards solving subset selection problems with the aid of the genetic algorithm. *In parallel problem solving from nature 2, R. Manner and B. Manderick(Ed.), Amsterdam:North-Holland*, pages 239–247, 1992.

[16] K.F. Man, K.S. Tang, and S. Kwong. *Genetic Algorithms*. Springer, 1999.

[17] Z. Michalewics. *Genetic algorithms + Data structures = Evolution programs*. Springer-Verlag, 2nd Edition, 1994.

[18] Z. Michalewicz. *Genetic algorithms + Data structures = Evolution Programs*. Springer-Verlag, 2nd Edition, 1994.

[19] H. Muhlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm. *Evolutionary Computation*, pages 25–49, 1993.

[20] W.M. Spears and K.A. De Jong. On the virtues of parameterised uniform crossover. In *Proceedings of ICGA4*, pages 230–236, 1991.

[21] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of ICGA3*, pages 2–9, 1989.

[22] M.D. Vose. *The simple genetic algorithm-foundations and theory*. The MIT Press, 1999.

[23] A.H. Wright. Genetic algorithms for real parameter optimization. *In foundations of genetic algorithms, J.E. Rawlins(Ed.), Morgan Kaufmann*, pages 205–211, 1991.