



Nonlinear model structure identification using genetic programming

Gary J. Gray, David J. Murray-Smith*, Yun Li, Ken C. Sharman, Thomas Weinbrenner

Centre for Systems & Control and Department of Electronics & Electrical Engineering, University of Glasgow, Scotland G12 8LT, U.K.

Received 29 August 1997

Abstract

Genetic Programming is an optimisation procedure which may be applied to the identification of the nonlinear structure of a dynamic model from experimental data. In such applications, the model structure may be described either by differential equations or by a block diagram and the algorithm is configured to minimise the sum of the squares of the error between the recorded experimental response from the real system and the corresponding simulation model output. The technique has been applied successfully to the modelling of a laboratory scale process involving a coupled water tank system and to the identification of a helicopter rotor speed controller and engine from flight test data. The resulting models provide useful physical insight. © 1998 Elsevier Science Ltd. All rights reserved.

Keywords: Genetic programming; genetic algorithms; nonlinear models; system identification; helicopter dynamics

1. Introduction

Identification of nonlinear models which are based in part at least on the underlying physics of the real system presents many problems since both the structure and parameters of the model may need to be determined. Many methods exist for the estimation of parameters from measured response data (Beck and Arnold, 1977) but structural identification is more difficult. Often a trial and error approach involving a combination of expert knowledge and experimental investigation is adopted to choose between a number of candidate models. Possible structures are deduced from engineering knowledge of the system and the parameters of these models are estimated from available experimental data. This procedure is time consuming and sub-optimal. Automation of this process would mean that a much larger range of potential model structures could be investigated more quickly.

Genetic Programming (GP) is an optimisation method which can be used to optimise the nonlinear structure of a dynamic system by automatically selecting model structure elements from a database and combining them optimally to form a complete mathematical model. Gen-

etic Programming works by emulating natural evolution to generate a model structure that maximises (or minimises) some objective function (Koza, 1992) involving an appropriate measure of the level of agreement between the model and system responses. A population of model structures evolves through many generations towards a solution using certain evolutionary operators and a 'survival-of-the-fittest' selection scheme. The parameters of these models may be estimated in a separate and more conventional phase of the complete identification process.

Fig. 1 illustrates the process used to identify a nonlinear model from experimental data using Genetic Programming. Expert knowledge is important in this process and is used both to select an appropriate function library and for the experimental design. The output of the Genetic Programming algorithm is a model or set of models which must then be validated. As a result of this validation, it may be desirable to change the experiment or modify the function library available to the GP algorithm as part of an iterative model building approach.

2. Genetic programming based modelling method

Genetic Programming works by emulating natural evolution to generate a model structure that maximises

*Corresponding author. E-mail: D. Murray-Smith@eng.gla.ac.uk

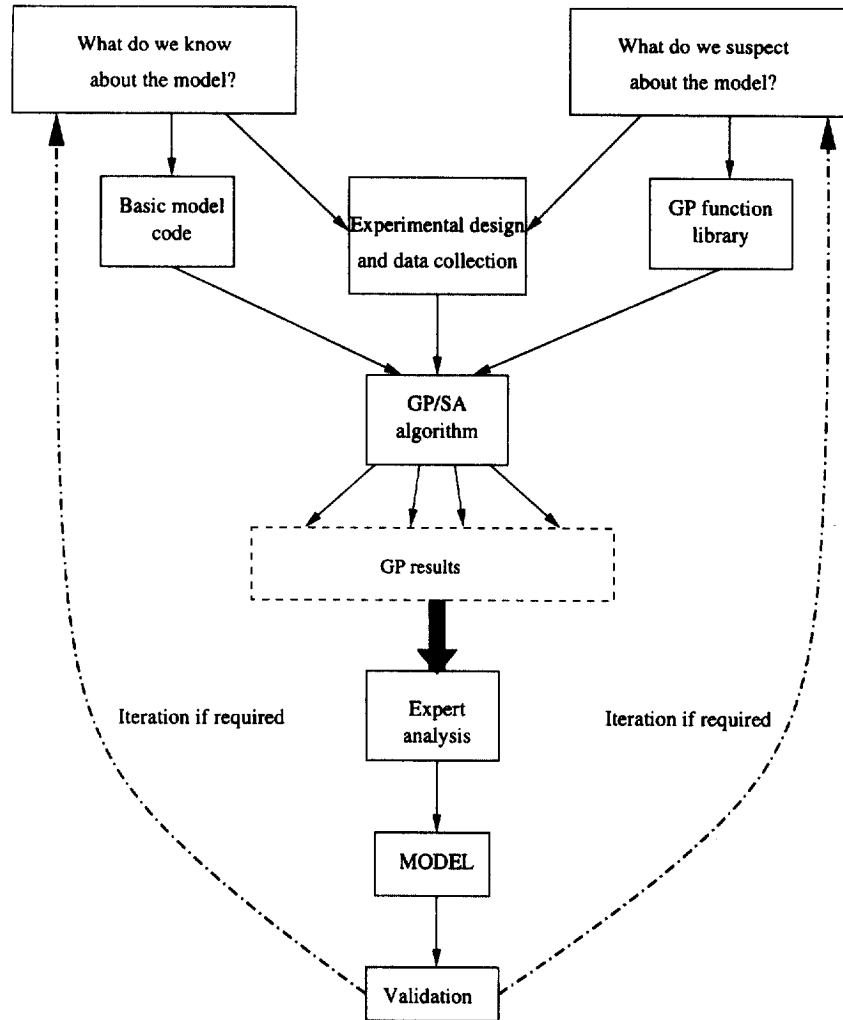


Fig. 1. Flow chart showing Genetic Programming modelling process.

(or minimises) some fitness function (Koza, 1992). A population of model structures (represented as trees as in Fig. 2) evolves through many generations towards a solution using certain evolutionary operators and a 'survival-of-the-fittest' selection scheme. The example shown in Fig. 2 represents a block diagram description of a nonlinear system. If the tree is turned clockwise through 90°, it can be viewed as a conventional block diagram. The tree can also represent an algebraic expression using Polish notation. Each individual tree is of variable length, is constructed of nodes and represents one candidate model structure for the system. The nodes can be terminal nodes at the end of a branch signifying an input or constant, or non-terminal nodes representing functions performing some action on one or more signals within the structure to produce an output signal. In the case of the system of Fig. 2, there is one output and the terminal nodes are the system inputs u and v . The non-terminal nodes are functions from the library selected for

Model output:
 $x = \exp(uv) + \sin(u)$

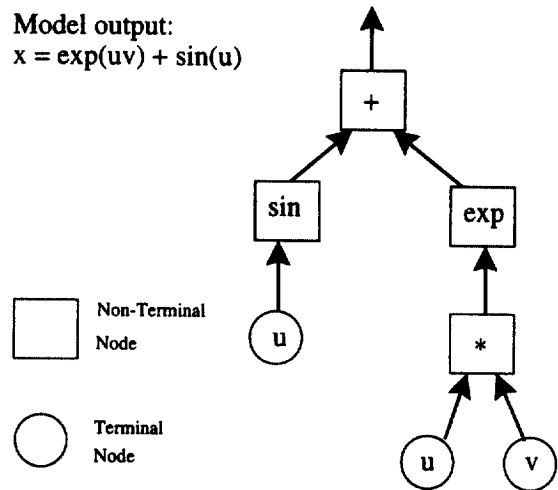


Fig. 2. GP Tree for a typical block diagram function.

this dynamic system. The library consists of functions which could conceivably form part of the dynamic system. The selection of the library functions is an important part of the GP modelling process. The nodes are combined to build the trees according to grammar rules specifying the number of inputs of each node type. Each tree is evaluated by identifying any numerical parameters, simulating the corresponding dynamic model and calculating some fitness function defining the quality of that model with respect to recorded experimental data.

Typically the Genetic Programming population contains a few hundred trees and evolves through the action of operators known as crossover, mutation and selection. The initial population is created entirely at random. The tree structure is limited by the GP grammar and by a pre-defined maximum tree size. Crossover and mutation processes are applied to branches, i.e. that part of the structure lying below a randomly selected point in the tree. Typically crossover is applied to 80% of each generation and involves the branches from two parent trees being interchanged as in Fig. 3. This means that characteristics of both parents will survive to the next generation but are combined differently, sometimes leading to offspring fitter than either parent. Mutation, at a low rate, means the creation of a completely new branch determined at random. This procedure is less likely to improve a specific structure but it can help the optimisation algorithm to escape from a local minimum. Selection involves evaluating the fitness of each population member and choosing the fittest to continue to the next generation. There are various selection strategies which are used to determine which of the models will survive to the next generation (Koza, 1992). The selection method used for this work is tournament selection (Koza, 1992). A number of trees (perhaps between 5 and 10) is selected from the population and the best of this group survives whilst the worst die off. This is repeated until a new generation is created. Other selection schemes include the roulette-wheel selection method where fitter models have a better chance of selection, and ranking selection where the population is ordered with its offspring and the fittest members of that entire group survive. The fitness function defines the quality of the model. It could be the sum

of the squares of the error between experimental data and simulated model output or a correlation function (McKay et al., 1996), or indeed any quantitative measure of model fidelity. The fitness function need not be differentiable or even continuous. Crossover, mutation and selection improve the general fitness of the population from one generation to the next. The algorithm repeats through many generations until some convergence criterion is satisfied. The resulting near-optimal model can then be used for nonlinear simulation of the system or for further investigation of the physical system, or to validate the structure of an existing model developed in some other way.

2.1 Application of genetic programming to nonlinear modelling

Genetic Programming is an established technique which has been applied to several nonlinear modelling tasks including the development of signal processing algorithms (Sharman and Esparcia-Alcázar, 1996) and the identification of chemical processes (Bettenhausen et al., 1995, Marenbach and Bettenhausen, 1996). In the identification of continuous time system models, the application of a block diagram oriented simulation approach to GP optimisation is discussed in Marenbach and Bettenhausen (1996) and Gray et al. (1996) and the issues involved in the application of GP to nonlinear system identification are discussed in (Gray et al. 1997b). In this paper, Genetic Programming is applied to the identification of model structures from experimental data. The systems under investigation are to be represented as nonlinear time domain continuous dynamic models.

The model structure evolves as the GP algorithm minimises some objective function involving an appropriate measure of the level of agreement between the model and system responses. One example is,

$$J = \sum_{i=1}^N e_i^2, \quad (1)$$

where e_i is the error between model output and experimental data for each of N data points. The GP algorithm

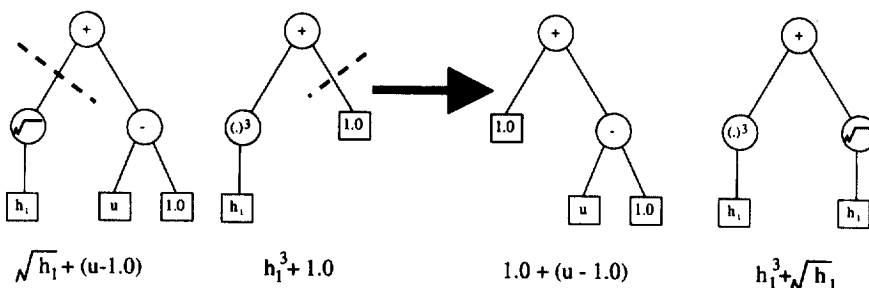


Fig. 3. Genetic Programming crossover mechanism illustrated using tree notation.

constructs and re-constructs model structures from the function library. The parameters of each candidate model structure are identified using a combined Nelder-Simplex and simulated annealing method (see Section 2.5) and the fitness of that model is evaluated using a fitness function such as that in Eq. (1). The general fitness of the population improves until the GP eventually converges to a model description of the system.

2.2. The function library

Any prior knowledge of the physical system should be included in an initial model and in the function library. The function library is the pool of potential nonlinear model structural components. The Genetic Programming algorithm selects elements from this library and builds a model structure which best represents the training data. For example, the GP optimised expression could represent the behaviour of a system state or some variable in the system such as a friction or stiffness term. The unknown dynamics evolve as the Genetic Programming algorithm builds each candidate model from this library of available functions.

This function library is very important and should be flexible enough to be able to represent a wide range of systems. However, if it is too general, experience has shown that the GP algorithm tends to produce an empirical 'best-fit' rather than a meaningful model structure. The function library can include the basic algebraic functions (+, −) as well as complex sub-functions which can include features suspected from prior knowledge of the system.

2.3. The Genetic Programming algorithm

For this research, a steady-state Genetic Programming algorithm was used. At each generation, two parents are selected from the population and the offspring resulting from their crossover operation replace an existing member of the same population. The number of crossover operations is equal to the size of the population i.e. the crossover rate is 100%. The crossover algorithm used was a subtree crossover with a limit on the depth of the resulting tree.

Genetic Programming parameters such as mutation rate and population size varied according to the application. More difficult problems where the expected model structure is complex or where the data are noisy generally require larger population sizes. Mutation rate did not appear to have a significant effect for the systems investigated during this research. Typically, a value of about 2% was chosen.

The function library varied according to application and what type of nonlinearity might be expected in the system being identified. A core of linear blocks was always available. It was found that specific nonlinearities

such as look-up tables which represented a physical phenomenon would only be selected by the Genetic Programming algorithm if that nonlinearity actually existed in the dynamic system. This allows the system to be tested for specific nonlinearities.

2.4. Evaluation of candidate models for Genetic Programming model structure identification

Each member of the Genetic Programming population represents a candidate model for the system. It is necessary to evaluate each model and assign to it some fitness value. Each candidate model is integrated using a numerical integration routine to produce a time response. This simulation time response is compared with experimental data to give a fitness value for that model. A sum of squared error function (Eq. 1) is used in all the work described in this paper, although many other fitness functions could be used.

The simulation routine must be robust. Inevitably, some of the candidate models will be unstable and therefore, the simulation program must protect against overflow errors. Also, all systems must return a fitness value if the GP algorithm is to work properly even if those systems are unstable.

2.5. Parameter estimation

Many of the nodes of the GP trees contain numerical parameters. These could be the coefficients of the transfer functions, a gain value or in the case of a time delay, the delay itself. It is necessary to identify the numerical parameters of each nonlinear model before evaluating its fitness. The models are randomly generated and can therefore contain linearly dependent parameters and parameters which have no effect on the output. Because of this, gradient based methods cannot be used. Genetic Programming can be used to identify numerical parameters (Koza, 1992) but it is less efficient than other methods. The approach chosen involves a combination of the Nelder-Simplex and simulated annealing methods (Press et al., 1992). Simulated annealing optimises by a method which is analogous to the cooling process of a metal. As a metal cools, the atoms organise themselves into an ordered minimum energy structure. The amount of vibration or movement in the atoms is dependent on temperature. As the temperature decreases, the movements, though still random, become smaller in amplitude and as long as the temperature decreases slowly enough, the atoms order themselves into the minimum energy structure. In simulated annealing, the parameters start off at some random value and they are allowed to change their values within the search space by an amount related to a quantity defined as system 'temperature'. If a parameter change improves overall fitness, it is accepted, if it reduces fitness it is accepted with a certain probability.

The temperature decreases according to some pre-determined 'cooling' schedule and the parameter values should converge to some solution as the temperature drops. Simulated annealing has proved particularly effective when combined with other numerical optimisation techniques.

One such combination is simulated annealing with Nelder-simplex optimisation (Press et al., 1992). The simplex is an $(n + 1)$ dimensional shape where n is the number of parameters. This simplex explores the search space slowly by changing its shape around the optimum solution. It closes in on the optimum of the function converging on the solution. The simulated annealing adds a random component and the temperature scheduling to the simplex algorithm thus improving the robustness of the method.

This has been found to be a robust and reasonably efficient numerical optimisation algorithm (Gray et al., 1997a). The parameter estimation phase can also be used to identify other numerical parameters in parts of the model where the structure is known but where there are uncertainties about the parameter values.

2.6. Representation of a GP candidate model

Nonlinear time domain continuous dynamic models can take a number of different forms. Two common representations involve sets of differential equations or block diagrams. Both these forms of model are well known and relatively easy to simulate. Each has advantages and disadvantages for simulation, visualisation and implementation in a Genetic Programming algorithm. Block diagram and equation based representations are considered in this paper along with a third hybrid representation incorporating integral and differential operators into an equation based representation.

2.6.1. Block diagram representation

Block-diagram descriptions of dynamic systems are easily interpreted and can represent a wide range of systems of any order. Many block diagram based simulation tools are available (e.g. SIMULINK (Checkoway and Kirk, 1992) which is a simulation toolbox for MATLAB (Moler et al., 1992)). Such tools use a library of system elements, both linear and nonlinear to construct a block diagram of the dynamic system to be simulated.

A GP tree can be configured to represent a SIMULINK block diagram of the dynamic system under investigation and the GP algorithm can then choose from a library of SIMULINK blocks. This library contains candidate building blocks for the system model and should reflect any prior knowledge of the system as well as including basic building blocks such as summing junctions, simple transfer functions involving first and second order sub-models, time delays etc. The terminal nodes in this case are system inputs and the non-terminal nodes

are SIMULINK function blocks. These function blocks can contain both linear and nonlinear elements enabling the GP to identify the order of a system as well as any nonlinearities. The grammar inherent in the GP tree ensures that the blocks are connected logically and with the correct number of inputs.

To evaluate an expression tree, that tree must be converted to a SIMULINK block diagram. To do this, the C++ program writes a SIMULINK script file describing the dynamic system and this SIMULINK file is executed from a MATLAB engine process running alongside the C++ executable on a UNIX workstation.

2.6.2. Differential equations

Equation based model representations include difference equations and differential equations. Discrete time difference equations are easier to simulate and show advantages in terms of speed at run-time but the continuous time differential equation representation produces more meaningful physical models. Both these methods can be implemented in Genetic Programming by using the GP tree to form all or part of the right hand side of one or more of the system equations. In this paper, the differential equation representation is described. One disadvantage of this method is that the order of the system must be selected before-hand and with a nonlinear system this order will not necessarily be known. As with the block diagram method, the terminal nodes are the system inputs and the non-terminal nodes are the library functions.

Simulation of the equation based models is more efficient than the block diagram representation because the models are coded in C++ and form part of the GP program.

2.6.3. Integro-differential equations

This model representation is equation based but the function library includes an integrator and a differentiator. This allows the GP algorithm to build nonlinear models of variable order. Also, sub-trees known as Automatically Defined Functions (ADF's) (Koza, 1992) were included in the GP optimisation program. These ADF's are defined as terminal nodes and are GP trees in their own right evolving in the same way as the main tree. They can be considered as sub models within the main model.

2.6.4. Choice of experimental data set – experimental design

The identification of nonlinear systems presents particular problems regarding experimental design. The system must be excited across the frequency range of interest as with a linear system, but it must also cover the range of any nonlinearities in the system. This could mean ensuring that the input shape is sufficiently varied to excite

different modes of the system and that the data covers the operational range of the system state space.

A large training data set will be required to identify an accurate model. However the simulation time will be proportional to the number of data points so optimisation time must be balanced against quantity of data. A recommendation on how to select efficient step and PRBS signals to cover the entire frequency range of interest may be found in texts such as Godfrey (1993) and Ljung (1987).

2.7. Results analysis

As the Genetic Programming evolution continues, the population gradually converges towards a solution. The optimisation used in this paper is stopped after 25 generations since it is evident that any improvement in the overall fitness of the population beyond this point is likely to be minimal. The fittest tree is taken as the optimum model. This expression tree can be quite lengthy and complex but it often contains much redundant information. For example, a complete branch could be nullified by a gain of zero or by having an output which is subtracted from a copy of itself. After post-processing to decode and simplify this GP tree, the result is an algebraic expression or block diagram describing the unmodelled part of the system with estimated values of any numerical parameters.

The GP optimisation process is normally carried out several times. Even after simplification, it is not expected that each optimisation will converge to the same solution. The solutions will look different and some will have better fitness values than others. Generally, however, each solution will be a sum of terms and many of these terms will occur in most of the solutions. Moreover, common terms will often be the major contributors to the main expression. In this way, examination of several separate optimisations gives an indication of likely components of the nonlinear model of the system. More conventional modelling procedures can be used to further develop the model of Murray-Smith (1995b).

2.8. Model validation

An important part of any modelling procedure is model validation. The new model structure must be validated with a different data set from that used for the

optimisation. There are many techniques for validation of nonlinear models the simplest of which is analogue matching where the time response of the model is compared with available response data from the real system. The model validation results can be used to refine the Genetic Programming algorithm as part of an iterative model development process.

3. Examples

The Genetic Programming structure identification technique was applied to a simple simulated system, to the modelling of a coupled water tank system and to the identification of a helicopter rotor speed controller and engine. The GP optimisation program is written in C++. The C++ code is based on a general GP program called gpc++ (Fraser, 1994). It controls the GP part of the program. The remainder of the program is dependent on the model representation being used (Section 2.6).

3.1. Example 1: Simulated system

This example identifies a simple model structure from simulated data. It is intended to illustrate the application of Genetic Programming to model structure identification. The system to be identified is shown in Fig. 4. It is a second order system with a time delay acting on the input.

The block diagram representation described in Section 2.6.1 is used for this example. The GP algorithm can choose elements from a variety of linear and nonlinear SIMULINK blocks defined in the GP function library (Table 1). A small population size of ten trees was used to demonstrate the GP modelling process. A square wave was used as the input and the sum of the squared error samples was used as the cost function indicating candidate model quality.

The time response of the model used to generate the simulated data is shown in Fig. 5. The progress of the optimisation is shown in Table 2. As the optimisation proceeds, different system blocks and model characteristics are evaluated. System elements which improve the model fitness are more likely to be retained for recombination with other successful system elements. In this example, the time delay is identified as important in the first generation and the second order characteristic is

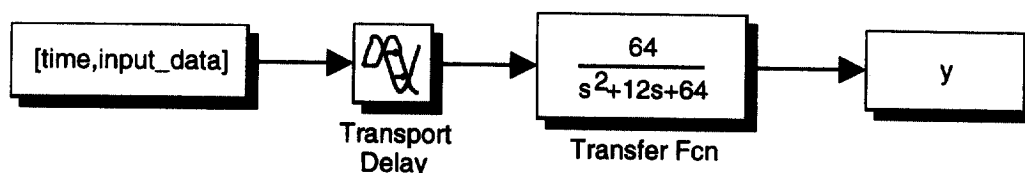


Fig. 4. Block diagram of system used to generate simulated data for example 1.

Table 1
Function library for Genetic Programming structure estimation routine

Function	Inputs	Parameters
Non-terminal nodes		
+	2	0
−	2	0
Gain	1	1
$1/(s + a)$ (1st order)	1	1
$\omega_n^2/(s^2 + 2\omega_n\zeta + \omega_n^2)$ (2nd order)	1	2
Time delay	1	1
Saturation	1	1
Squared	1	0
Terminal Node		
Input (u)	−	0
Zero	−	0

combined with this in generation four. Model fitness improves from one generation to the next towards the solution after seven generations. The parameters of the models are estimated at each stage.

3.2. Example 2: Equation based identification of flow through outlet pipe

This example shows the application of Genetic Programming to the identification of the outlet flow of a coupled water tank system as shown in Fig. 6 (Murray-Smith, 1995a). The system is nonlinear since even at the simplest approximation, the flow between the two tanks and from the second tank to the outlet varies as the square root of the water depth.

The input signal which determines the input flow rate to tank one was a binary signal the amplitude and time pattern of which were chosen so that different modes of the system would be excited and the complete operating range of H_1 and H_2 would be covered. Two sets of data were recorded, one for identification and one

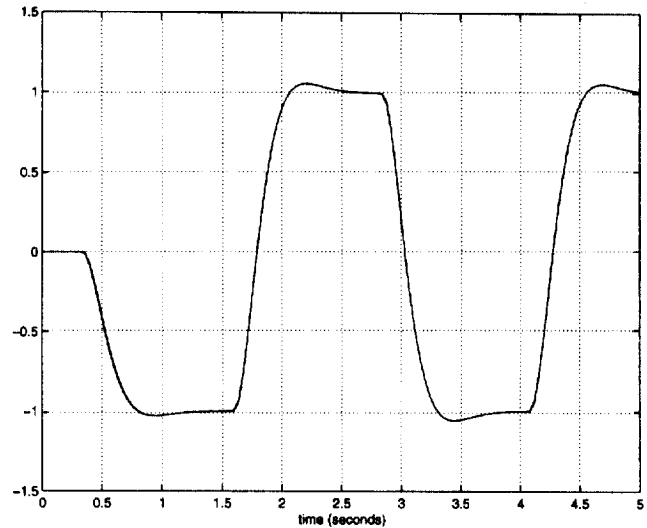


Fig. 5. Time response of example model used to demonstrate block diagram based model structure identification from simulated data.

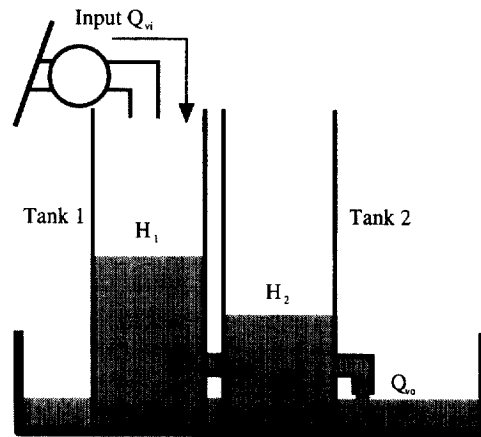


Fig. 6. Twin tank system.

Table 2
Progress of Genetic Programming optimisation of model structure

Gen.	$\sum e^2$	Best model of generation	Comments
1	32.3668	Td (Gain u)	Best of initial random population. Time delay is identified.
2	3.0154	Td($-1/(s + a) u$ (Gain u))	Time delay retained with additional dynamic element.
3	3.0154	Td($-1/(s + a) u$ (Gain u))	
4	1.79844	Td($-1/(s + a) u$ (Td($-2nd\ order\ u$ (Gain u))))	Second order characteristic identified.
5	1.79844	Td($-1/(s + a) u$ (Td($-2nd\ order\ u$ (Gain u))))	
6	1.79844	Td($-1/(s + a) u$ (Td($-2nd\ order\ u$ (Gain u))))	
7	0.00513	((Td(2nd order u)))	GP finds solution.
8	0.00513	((Td(2nd order u)))	
9	0.00513	((Td(2nd order u)))	
10	0.00513	((Td(2nd order u)))	

for validation. These involved the use of input signals of similar but not identical amplitude and frequency distributions. The Genetic Programming approach allowed an expression for flow out of tank two to be developed, including estimation of numerical parameters using the combined simplex-simulated annealing method.

The fluid flow in and out of the tanks is described by,

$$A_1 \frac{dH_1}{dt} = Q_{vi} - Q_{v1}, \quad (2)$$

$$A_2 \frac{dH_2}{dt} = Q_{v1} - Q_{vo} \quad (3)$$

where Q_{vi} is the volume flow into tank one, Q_{v1} is the flow from tank one to tank two, and Q_{vo} is the output flow from tank two. A_1 and A_2 are the cross-sectional areas of tanks one and two respectively and are each equal to 0.01m^2 . H_1 is the depth of the water in tank one and H_2 is the depth in tank two. If the connecting pipe and the output pipe are assumed to be orifices, Bernoulli's theorem applies and the flow can be calculated thus,

$$Q_{v1} = c_{d1}a_1\sqrt{2g(H_1 - H_2)}, \quad (4)$$

$$Q_{v2} = c_{d2}a_2\sqrt{2g(H_2 - H_3)} \quad (5)$$

where c_{d1} and c_{d2} are empirical discharge coefficients for the connecting pipe and the output pipe respectively, a_1 (39.56mm^2), is the area of the orifice between tanks one and two and a_2 (38.48mm^2) is the area of the orifice at the outlet of tank two. H_3 is the height of the outlet pipe above the floor of the tank. While this approximation may be valid for the pipe connecting the two tanks (it is 5.5mm long), it is known to be inaccurate for the outlet pipe from tank two (106mm long). The Genetic Programming optimisation was therefore configured to find an algebraic expression for the flow out of tank two. The discharge coefficient for the flow from tank one to two, c_{d1} , was estimated using the combined Nelder-simplex and simulated annealing routine as part of the optimisation.

Incompressible fluid flow can exist in two forms – laminar and turbulent. Both these forms are nonlinear and a system of equations has to be solved to calculate the flow rate for any particular pipe, environmental conditions, and pressure difference (see Appendix A). Whether flow is laminar or turbulent depends on the Reynolds number. There is a crucial regime between laminar and turbulent flow where the flow can be a combination of these or one or the other (Thomas, 1995). Empirical calculations based on the typical flow rate in this experimental apparatus suggested that the flow was in this region.

The only variable relevant to the flow of water through the outlet pipe in the water tank system is the pressure difference which is directly proportional to the depth

difference of the water at either end of the pipe (in this case, $H_2 - H_3$). The flow rate for every possible depth difference in this system was calculated by solving the nonlinear flow equations (Thomas, 1995) at each point and storing them in two look-up tables – one for laminar flow and one for turbulent. These look-up tables became non-terminal nodes for the Genetic Programming function library. They had one input – the depth variable used to determine the flow, and one output – the flow rate. The other non-terminal nodes included +, –, gain, time delay and square root (see Table 3). One additional non-terminal node was used. This was the basis function which is a Gaussian curve around a centre point with a certain width which depends on H_2 (Fig. 7). The basis value multiplies the input signal to generate an output. It was included in the Genetic Programming function set to permit the Genetic Programming to evolve certain dynamics for some values of H_2 and different dynamics for others. Such a combination of weighted models is known as a local model network (Murray-Smith and Johansen, 1997). This was not unreasonable since it is known that turbulent flow is more likely to occur at higher values of pressure differential. The terminal nodes were the system input, the states, and 0 and 1. The Genetic Programming function library is shown in Table 3. The gain is an estimated parameter. The square root was included since it appears in the Bernoulli equation (Eq. (5)).

The GP derived expression for Q_{vo} was incorporated in the set of nonlinear ordinary differential equations which were numerically integrated over the experimental time range. The sum of the squares of the samples of the error between predicted H_1 and measured H_2 was calculated for each simulation and formed the basis for the optimisation process.

Table 3
Function library for Genetic Programming structure estimation of flow through outlet pipe

Function	Inputs	Parameters
Non-terminal nodes		
+	2	0
–	2	0
Gain	1	1
Square root	1	0
Basis function	1	2
Laminar flow look-up	1	0
Turbulent flow look-up	1	0
Unit time delay	1	0
Terminal Node		
H_1	–	0
H_2	–	0
Input	–	0
One	–	0
Zero	–	0

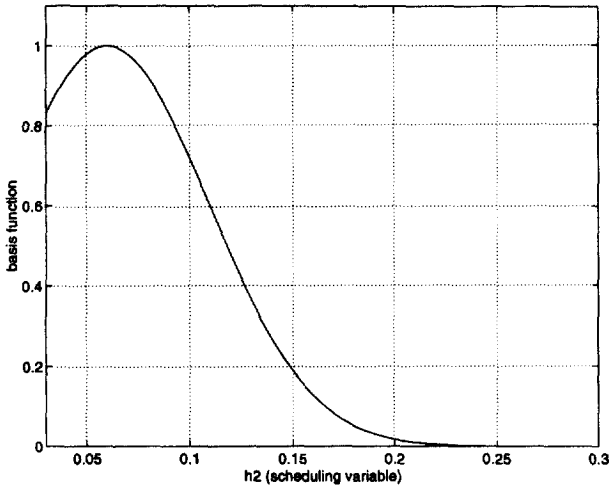


Fig. 7. Basis function of H_2 for a centre point of 0.06 and a width of 0.07 (the depth is always > 0.03).

The optimisation for the equation for flow from tank two and parameter c_{d1} was carried out four times. The resulting structure equations with their respective error norms were,

$$\text{flow} = \text{lam}(H_2) - \text{turb}(H_1) - p_0 H_1, \sum e^2 = 0.681, \quad (6)$$

$$\begin{aligned} \text{flow} &= \text{basis}(p_0, p_1) \times \text{basis}(p_2, p_3) \times \text{basis}(p_4, p_5), \\ &\quad \times \text{basis}(p_6, p_7) \times \text{basis}(p_8, p_9) \times \text{lam}(H_2), \\ \sum e^2 &= 0.0451 \end{aligned} \quad (7)$$

$$\text{flow} = \text{basis}(p_0, p_1) \times \text{lam}(H_2), \sum e^2 = 0.0424 \quad (8)$$

$$\text{flow} = \text{basis}(p_0, p_1) \times \text{lam}(H_2 - \text{lam}(H_2)), \sum e^2 = 0.0416 \quad (9)$$

One common feature of these four expressions is the function $\text{basis}(p_0, p_1) \times \text{lam}(H_2)$. Many of the other terms can be neglected. The turbulent flow term in Eq. (6) is not large although the error for this solution is significantly larger than the others. Substituting the identified parameter values, Eq. (7) can be approximated by one basis function multiplied by the laminar flow. In Eq. (9), the $\text{lam}(H_2)$ term added to H_2 is several orders of magnitude smaller than H_2 and is therefore insignificant. Eq. (8) was thus accepted as the best representation of flow out of tank two.

The parameters p_0 and p_1 were identified using the combined Nelder-simplex and simulated annealing routine within the GP optimisation. For the range of H_2 in the example, and the identified values of p_0 and p_1 , $\text{basis}(H_2)$ is equivalent to a constant of 0.75. Hence, the

following model for flow was derived;

$$\text{flow} = 0.75 \text{lam}(H_2) \quad (10)$$

During each function evaluation, the discharge coefficient from tank one to tank two, c_{d1} , (Eq. (4)) was estimated for each case. For this model, the estimated value for c_{d1} was 0.75.

3.2.1. Model validation

The parameters c_{d1} and c_{d2} of the model described by Eqs. (2)–(5) were estimated using the same binary input signal and data set as for the GP model identification. This model differs from the GP identified model in that the outlet flow from tank two is calculated using Eq. (10) rather than Eq. (5). The parameters were estimated with the same combined Nelder-Simplex and Simulated annealing routine as used in the GP algorithm. The cost function minimised was the sum of the squares of the error (Eq. (1)). The resulting parameter estimates were $c_{d1} = 0.608$ and $c_{d2} = 0.697$. The training data and the time responses of the identified GP model and the original model are illustrated in Fig. 8. Both models seem a reasonable match for the training data. However, both models were also simulated with a different set of experimental data and the time responses are compared in Fig. 9. The fit for H_2 for the GP identified model is of a reasonable quality indicating that the model described in Eq. 10 is a good representation of this system. It is also obvious that the GP identified model is a better match than the model based on the Bernoulli equations. The error in the prediction of outlet flow is integrated producing the discrepancy in terms of the level, H_2 shown in Fig. 9.

3.3. Example 3: Genetic Programming identification of a helicopter rotor speed controller and engine

Genetic Programming has also been applied to the structure identification of helicopter engine dynamics using flight test data from a MBB Bo105 helicopter. The helicopter main rotor is designed to rotate at a constant speed. This speed is maintained by an automatic control system which has measured rotor speed as the controller input and fuel flow to the engine as its output. Changes in rotor blade pitch angles cause changes in the drag on each rotor blade and this alters the load torque on the main rotor. In the MBB Bo105 helicopter, a mechanical governor is used to control the fuel flow to the engine to compensate for this changing load and maintain a constant rotor speed. The dynamics of the engine are slow and changes in the main rotor collective torque can cause a transient variation in rotor speed of up to about 2%. These variations in rotor speed and rotor shaft torque have significant impact in terms of aircraft heave (vertical movement) and yaw dynamics. An accurate model of this subsystem is therefore important for accurate simulation of the helicopter flight mechanics.

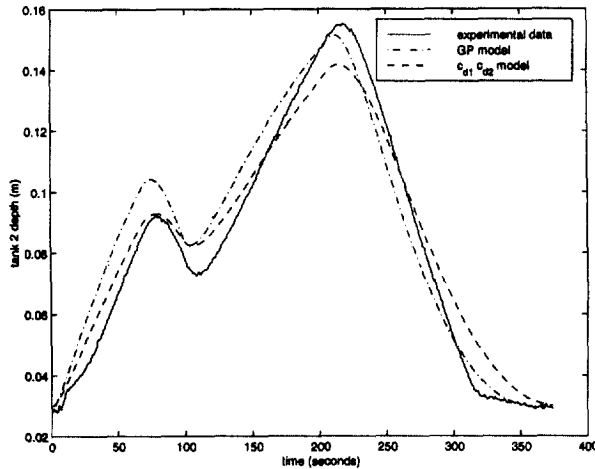


Fig. 8. Time response of coupled tank system output, GP evolved model and Bernoulli derived model (training data).

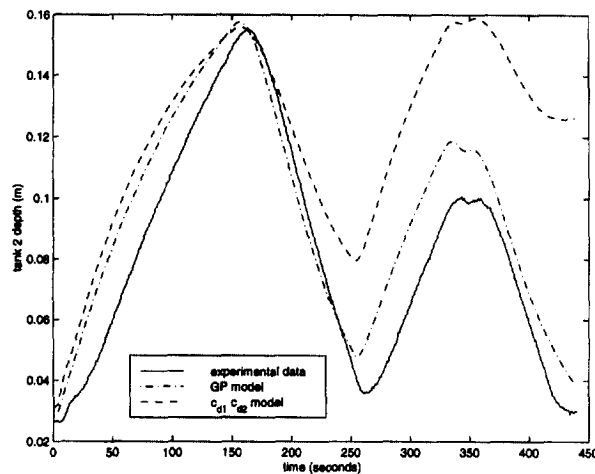


Fig. 9. Time response of coupled tank system output, GP evolved model and Bernoulli derived model (validation data).

Genetic Programming was applied to finding a non-linear dynamic model for the rotor speed controller and the engine. The input for this identification was rotor speed, and the output was engine torque. The experi-

mental data set came from flight tests in near hover condition and involved a step input in main rotor collective pitch. This means that the pitch of all four helicopter blades was increased simultaneously giving an abrupt change in main rotor load torque. The GP algorithm used automatically defined functions and a single tree structure with algebraic, nonlinear, differential and integral operators. The GP converged to the model illustrated in Fig. 10. Fig. 11 shows that the corresponding model time response matches the training data very well. The model was validated by applying to it a data set recorded for a similar experiment but with the opposite sign of input. The match is shown in Fig. 12. The bias parameter in Fig. 10 was identified for the new data set because the initial conditions were different but all other parameters and the model structure were those identified from the training data (Fig. 11).

4. Application of GP model structure identification to other problems

This technique could be applied to many structural modelling problems. However, as with all modelling methods, some knowledge of the likely dynamics of the system is required. It is also important to have representative experimental data for the identification. The model is being derived from these data.

The GP can be configured to identify a complete model structure but it is more likely that some parts of the model will be known and that the GP is used to generate a representation of some specific nonlinear component of the system. Careful consideration should be given to the selection of the function library. It may be necessary to run several optimisations with different library components until a suitable model is identified. Analysis of all the runs should reveal consistent model structure elements as likely candidates for the final model.

Finally, model validation is very important to show confidence in the identified model. Different data sets

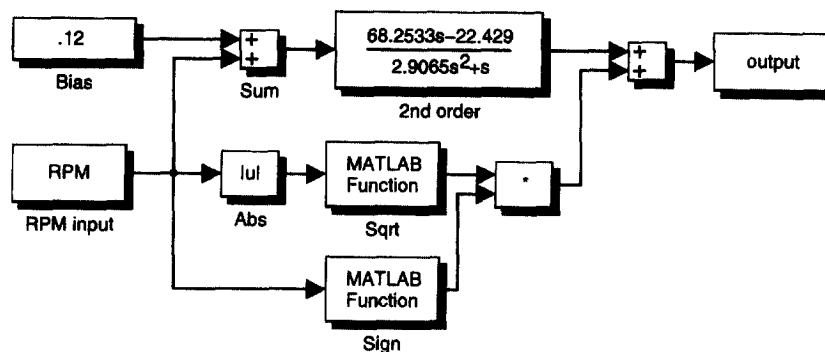


Fig. 10. GP identified system for helicopter rotor speed controller and engine.

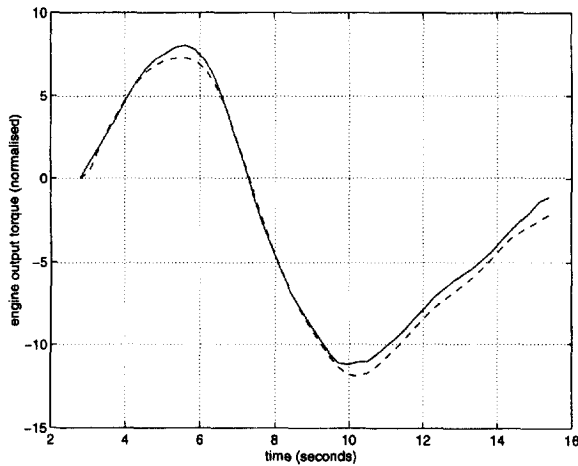


Fig. 11. Time response of helicopter engine torque real data (solid) and GP evolved model (dashed) – training data.

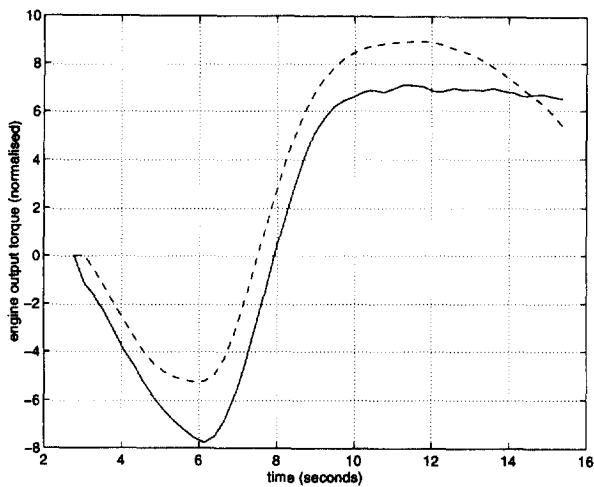


Fig. 12. Time response of helicopter engine torque real data (solid) and GP evolved model (dashed) – validation data.

should be used to ensure that the derived mathematical model is an accurate representation of the dynamic system in the specified operating regime.

5. Conclusion

Genetic Programming is a valuable tool for the modelling of nonlinear dynamic systems. It can allow nonlinear model structures to be developed that best fit experimental data. The GP automates the trial and error process of structure estimation and can therefore test a lot more potential model components and structures. It is capable of retaining the best parts of these structures and of recombining them to give new and often better models. However, certain issues should be taken into account. Calculation of a fitness value for each candidate model is crucial. A cost function involving an appropriate

measure of the level of agreement between the model and system responses must be selected. Since the number of evaluations of the fitness function may be very large, the necessary simulation routines should be fast and should return a numerical value of model fitness suitably scaled for the GP selection operator. The GP program must include a robust parameter estimation routine and as with all nonlinear modelling methods, experimental design and model validation are important parts of the process. The model should be represented in a format relevant to the modelling problem.

The model resulting from the GP can reveal important aspects of the physical structure of the system. Specific nonlinearities such as look-up tables representing suspected system characteristics can be included in the GP library in the knowledge that if it does not improve model fidelity, it will not be included in the final model. The dynamics of the system are described by the GP solution and the GP optimisation method can be configured to select the order of the model as well as the form of any nonlinearities present. Such information can be used for further investigation of the physical system or to validate the structure of an existing model developed in some other way.

Unlike many other nonlinear modelling techniques GP can be used to build a model structure that best fits experimental test data from a library of different types of linear and nonlinear components.

Acknowledgement

This research is supported by the UK Engineering and Physical Sciences Research Council under grant GR/K24987 (“Evolutionary Programming for Nonlinear Control”). The authors would like to thank the other members of the evolutionary computing research group for their useful discussions and Mr. P.J. Thomas for his contribution to the theoretical derivation of the fluid flow through a pipe (Thomas, 1995). Support from the British Council under the British-German ARC Programme is also gratefully acknowledged. Thanks are also due to the members of the DLR Institute of Flight Mechanics, Braunschweig, Germany who provided the helicopter flight test data.

References

- Bettenhausen, K., Marenbach, P., Freyer, S., Rettenmaier, H., & Nieken, U. (1995). Self-organizing structured modelling of a biotechnological fed-batch fermentation by means of genetic programming. In *Proceedings of First IEE/IEEE Int. conf on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, No. 414, pp. 481–486. IEE, London.
- Checkoway, C., & Kirk, K. (1992). *SIMULINK Users Guide*. The MathWorks, Inc.

- Fraser, A. (1994). Genetic programming in C++. Technical report, Dept. Electronics and Electrical Engineering, University of Salford, Salford, England. available at <ftp://ftp.salford.ac.uk/pub/gp/gpc++-0.4.ps.gz>.
- Godfrey, K. (1993). *Perturbation Signals for System Identification*. Prentice Hall International, UK.
- Gray, G., Li, Y., Murray-Smith, D., & Sharman, K. (1996). Structural system identification using genetic programming and a block diagram oriented simulation tool. *Electronics Letters*, 32(15), 1422–1424.
- Gray, G.J., Murray-Smith, D.J., Li, Y., & Sharman, K.C. (1997a). Nonlinear structural system identification using Genetic Programming. In Troch, I. and Breitenecker, F., (Eds), *Proceedings of the Second International Symposium on Mathematical modelling (MATHMOD)*, volume 1 of ARGESIM Report, pp. 301–306, Vienna. ARGESIM.
- Gray, G.J., Weinbrenner, T., Murray-Smith, D.J., Li, Y., & Sharman, K.C. (1997b). Issues in nonlinear model structure identification using Genetic Programming. In *Proceedings of the Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA 97)*, Conference Series, 446, 308–313, Institution of Electrical Engineers.
- Koza, J. (1992). *Genetic Programming: On the Programming of Computers by means of Natural Selection*. The MIT Press, Cambridge, Mass.
- Ljung, L. (1987). *System Identification – Theory for the User*. Prentice-Hall, Englewood cliffs, New Jersey, USA.
- Marenbach, P. and Bettenhausen, K. (1996). Signal path oriented approach for generation of dynamic process models. In *Proceedings of Genetic Programming '96 Conference*, Stanford, CA.
- McKay, B., Willis, M., Hiden, H., Montague, G., and Barton, G. (1996). Identification of industrial processes using Genetic Programming. In *Proceedings of Identification in Engineering Systems*, Swansea, UK. Available at <http://lorien.ncl.ac.uk/sorg/paper4.ps>.
- Moler, C., Little, J., & Bangert, S. (1992). *MATLAB user's Guide*. Mathworks Inc.
- Murray-Smith, D. (1995a). Advances in simulation model validation: theory, software and applications. In Breitenecker, F. and Husinsky, I., (eds.), *EUROSIM'95 Simulation Congress*, pp. 75–84, Amsterdam, The Netherlands. Elsevier.
- Murray-Smith, D. (1995b). *Continuous System Simulation*. Chapman and Hall, London.
- Murray-Smith, R., & Johansen, T. (1997). *Multiple Model Approaches to Modelling and Control*. Taylor and Francis, London.
- Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. (1992). *Numerical Recipes in C*, chapter 10. Cambridge University Press.
- Sharman, K., & Esparcia-Alcázar, A. (1996). Some applications of genetic programming in digital signal processing. In *Late breaking papers at the Genetic Programming '96 Conference*, pp. 473–480, Stanford, CA.
- Thomas, P. (1995). Unpublished note to D.J. Murray-Smith.