IOPscience iopscience.iop.org

Home Search Collections Journals About Contact us My IOPscience

Applications of genetic algorithms to the solution of ordinary differential equations

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

1993 J. Phys. A: Math. Gen. 26 3503

(http://iopscience.iop.org/0305-4470/26/14/017)

View the table of contents for this issue, or go to the journal homepage for more

Download details:
IP Address: 171.66.16.62
The article was downloaded on 01/06/2010 at 18:58

Please note that terms and conditions apply.

J. Phys. A: Math. Gen. 26 (1993) 3503–3513. Printed in the UK

Applications of genetic algorithms to the solution of ordinary differential equations

D A Diver
Department of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, UK

Received 11 November 1992, in final form 8 February 1993

Abstract. A novel genetic algorithm has been developed and applied to the solution of ordinary differential equations. The algorithm solves the equations by a process of breeding better candidate solutions from a family of estimates, and learns to retain the best features as it progresses. This self-learning system is intrinsically parallel, and capable of handling linear and nonlinear equations, both stiff and non-stiff.

Genetic algorithms are a key element in artificial intelligence and machine learning, playing a significant role in optimization and robotics. In this document, the application of genetic algorithms to the solution of ordinary differential equations is presented as a radically different way of approaching numerical simulations, and is of importance to many disciplines.

1. Introduction

The genetic algorithm (GA) is a self-adapting strategy for searching, based on the random exploration of the solution space coupled with a memory component which enables the algorithm to learn from experience the optimal search path. The analogy with nature is clear: the evolution of a species can be regarded in an elementary way as a random exchange of genetic information (that is, breeding) with natural selection ensuring that the surviving variants are only those evolved forms which are advantageous to the species' ability to exploit its environment. The GA is based on the same premise. Given a large database or solution space to search according to some external criterion, the GA will find an optimal route to the ultimate goal by a process of trial and error, learning from mistakes and preserving the most successful aspects of performance at each stage, until finally the best possible combination of information is attained. The governing process is natural selection, and the performance criterion encapsulates the physical meaning of the task in hand. Of fundamental importance is the fact that the GA is essentially a parallel algorithm, with the generation of each new pool of knowledge a naturally independent process. Thus such algorithms are the most efficient exploiters of parallel architecture in computer hardware, since their intrinsic design is based on the assumption of non-interrelated steps.

Since the design philosophy of the GA was first proposed (Holland 1975) many textbooks have been written explaining the technique in detail and illustrating its use in simple situations; the reader is encouraged to consult these for a fuller description of the process (see for example Goldberg (1989) and Davidor (1990)).

Typically, GAs are applied to sorting problems in which the size of the database to be searched and ordered is too big to enumerate each and every possible permutation, and so

0305-4470/93/143503+11$07.50 © 1993 IOP Publishing Ltd

a GA is harnessed to accomplish the goal. There are many diverse applications of such a useful technique, from discrete optimization (Herdy 1991), robot nagivation (Jo and Didier 1991), pattern recognition (Mahlab *et al* 1991, Bhattacharjya *et al* 1992) and seismic data processing (Wilson and Vasudevan 1991), to finding the most efficient orientation of magnets in an electron accelerator (Hajima *et al* 1992).

An example familiar to all is the travelling salesman problem (Grefenstette *et al* 1985) in which an optimal route around a territory must be computed, such a route being the shortest path that visits each and every designated site once only. Numbering each site, a specific GA approach in this case is to represent each possible route by the character string comprising the concatenation of the site indices in the order visited. Associated with each string is the total travelling distance. The GA attempts to construct better routes by a breeding process, which involves selecting two routes at random from an initial pool of possibilities, and randomly reassembling sections of each of the parent strings into a new string. This process is repeated many times to assemble a large collection of such objects, all derived from the initial stock. The distance associated with each route is then calculated, and only the best from the entire stock of parents and offspring are selected to be the seed corn for the next iteration. In this way, desirable features are propagated from one iteration to the next, with shorter routes multiplying at the expense of less successful ones. This crucial feature of self-learning makes the GA adaptable to many complex, multi-parameter problem solving situations arising in many diverse areas of system modelling, whether physical, biological or statistical.

This paper details how the same general technique may be applied to the solution of ordinary differential equations (ODEs). The candidate solutions are represented as strings, as in the travelling salesman problem, but this time the strings represent indirectly the numerical value of the ODE solution. This is a crucial step, allowing a compact encoding of quantitative information. The performance criterion in this context is a measure of how well a particular candidate solution satisfies the ODE and its boundary conditions. The breeding process is then one in which successively better approximations to the actual discrete representation of the solution are manufactured by a self adapting process designed to converge with the minimum of user interference, and the maximum flexibility and numerical stability. Loosely speaking, the GA guesses what the solution is initially, and then chooses the best pieces and assembles the finest educated estimate of the solution possible after finitely many iterations.

The solution of ODEs by this technique is a fundamentally different type of problem from the sorting exercise exemplified by the travelling salesman puzzle, in that the latter has a finite domain in which to search for the definitive solution, but the former has, in principle, an infinite domain, since the evaluation of a solution to a differential equation must necessarily range over the entire continuum of real numbers, with no prior information limiting the excursions available in principle from the differential operator or its boundary conditions.

This paper sets out how one such algorithm, GENODE, attempts to implement the solution of ODEs using a GA. The breeding algorithm is more complex than the simple crossover of early GAs (Goldberg 1989), since it embodies problem specific knowledge and recovery strategies to avert premature convergence. The penalty weighting is multifaceted, designed to assess not only the quality of satisfaction of the differential equation, but also boundary conditions including gradient criteria, continuity and point resolution. The following sections analyse each aspect of the GA approach in detail, before presenting examples of its performance and a discussion on how further progress might be made.

2. The algorithm

2.1. Breeding a solution

The algorithm developed for use in the solution of ODEs is detailed in this section. Consider a discrete representation of the function $y(x)$, with values y_i at the points x_i, that is $y_i = y(x_i)$. We will choose to span the space of y_i values by forcing y_i to be integer multiples of two real numbers μ and ν, selected to yield unique representations of the values of y_i in a predetermined appropriate range. Thus we have

$$y_i = m\mu + n\nu \qquad \text{where } m, n \in Z. \tag{2.1}$$

GENODE takes the internal representation of this numerical profile as the character string constructed by concatenating the ms and ns for each discrete value of the function y. For example, the kth member of the pool of candidates, $y^{(k)}$ is represented as

$$y^{(k)} = m_1 n_1 \ldots m_s n_s \ldots m_f n_f \tag{2.2}$$

where the function y is evaluated at the points $x_1 \ldots x_f$. The m_i and n_i are functions of k, but this has been omitted from the notation in the interests of clarity. The breeding algorithms starts with N_p such strings representing the parental stock, and generates offspring as follows:

(i) two parents are selected randomly from a choice of N_p candidates;
(ii) random sections of the character strings are selected from each parent;
(iii) an offspring is generated by randomly assembling these sections into a string of the same length.

Clearly (i)–(iii) describe only superficially the breeding process. At each stage, the character strings are of the same length. Hence, though the length of section chosen randomly from the first parent is arbitrary, it cannot exceed the maximum length. Consequently the section chosen from the second parent cannot exceed the difference in length between the first section and the maximum string size. When these two sections fall short of the fixed length required to construct a valid offspring, the remaining spaces are filled by a mutant string generated randomly to fit the gap. This process ensures that there is no bias in choosing sections of genetic material, and that there is always a mechanism to introduce fresh variations in the stock. This latter point is particularly important, since the solution space spanned by a differential equation is not predetermined by the boundary conditions or the magnitudes of the coefficients in the operator, and so the generation of the exact solution cannot be merely a permutation of elements in a finite stock of candidates as in the travelling salesman problem, for example. This fundamental concept is also at the heart of the string representation defined by (2.1) and (2.2), which ensures that the function values are discretised in a controllable way. Normally the values of $y_i = y(x_i)$ would be drawn from the continuum of the real numbers, regardless of the discretized x-axis. This is impractical in this algorithm, since a full binary representation of each number would be required, instead of the compact notation of (2.2).

2.2. Assessing fitness

Given a full generation of parents and offspring, the natural selection process requires that only the best be selected as the parents of the next breeding cycle. The assessment exercise

in this context is the evaluation of how well the candidate satisfies the differential equation and boundary conditions. The latter aspect is automatic, in respect of boundary values, since each candidate has the required boundary values assigned at the breeding stage, overwriting the generated information at the endpoints with the required representation. Assessing the quality of fit to the differential equation is rather more involved.

Let the differential equation be of the form

$$a(x)y'' + b(x)y' + c(x)y = 0 \tag{2.3}$$

where $'$ denotes $\mathrm{d}/\mathrm{d}x$. Central different operators are used to represent derivatives, so that for example

$$y'(x_i) = y'_i = (y_{i+1} - y_{i-1})/(2h) \qquad y''_i = (y_{i+1} - 2y_i + y_{i+1})/h^2 \tag{2.4}$$

where $h = x_i - x_{i-1}$ is the spacing in the x coordinate, assumed constant. The differential equation (2.3) is then represented by the central finite different algorithms (2.4) and coded as

$$\{a_i/h^2 + b_i/(2h)\}y_{i+1} + \{c_i - 2a_i/h^2\}y_i + \{a_i/h^2 - b_i/(2h)\}y_{i-1} \tag{2.5}$$

where $a_i, b_i = a(x_i), b(x_i)$. Clearly nearest-neighbour interaction is defined by (2.5), and therefore special arrangements must be made at the endpoints. Using quadratic extrapolants at each end, the full central difference operators can be applied, and expressed as

$$y'_1 = (-3y_1 + 4y_2 - y_3)/(2h) \qquad y''_1 = (y_1 - 2y_2 + y_3)/(h^2). \tag{2.6}$$

Note that in fact the form of the second derivative at the end point is identical to that at the penultimate; this is a consequence of second-order differences applied to a quadratic extrapolant. Using (2.5) together with the amended form via (2.6) for the end points, the full differential operator is discretized, and can be applied to a discrete representation of a candidate solution stored as an array of y-values. Hence, given a string representing the numerical profile of a possible solution to (2.3), the compact form (2.2) is expanded into a set of real numbers to which (2.5) is applied. If the candidate is indeed the exact solution, then (2.5) will yield 0 at every point x_i (to within the finite difference approximation). Suppose that R_i is the result of applying the differential operator to every point x_i of the candidate solution, $i = 1, \ldots, N_{\mathrm{pts}}$. Then crude measures of the quality of fit are

$$\epsilon_1 = \sum |R_i|, \epsilon_2 = \prod |R_i| \qquad i = 1, N_{\mathrm{pts}}$$

If gradient information is given in addition to boundary values for the equation, this may also be used to measure fitness, by assessing deviations in the candidate from the desired trend. GENODE uses the penalty

$$\epsilon_3 = \mathrm{e}^{\delta} - 1 \qquad \text{where } \delta = |y' - y'^*| \text{ evaluated at the end points}$$

y' being the actual value for the candidate in question, and y'^* denoting the desired gradient specified by the user. Note that although a second order ordinary differential equation requires only two pieces of information formally to specify completely the solution, in the case of the numerical solution of boundary value problems (the focus of attention in the next section) it is normal practice to overspecify the boundary conditions in that the

unknown and its gradient are given at both boundaries, with the proviso that only two of these pieces of information are exact, and that both cannot be assigned at the same point in a boundary value problem. GENODE uses the same tactic, making the actual boundary values unalterable, but assigning penalties if the calculated gradients differ from the desired trend.

A further penalty weighting is used in GENODE, namely

$$\epsilon_4 = \max|R_i| \qquad i = 1, N_{\text{pts}}$$

which measures the continuity of the proffered candidate solution. The final penalty W_k associated with the kth candidate $y^{(k)}$ is the weighted sum of $\epsilon_i, i = 1, \ldots, 4$, the weights being supplied by the user at run time. In this way, the quality of fitness of each candidate is assessed with respect to the major components, namely boundary conditions, continuity and membership of the kernel of the differential operator. When this penalty is attached to each candidate, and the entire stock ranked, only those with the lowest score will be selected as parents for the next breeding cycle.

Note that both ϵ_1 and ϵ_2 are used in assessing the overall suitability of a candidate as a member of the operator kernel. This is because ϵ_2 helps to differentiate between otherwise similar solutions, since the candidate with the lower ϵ_2 has a more accurate string section somewhere in its make-up, and consequently will be ranked higher since local precision is the ultimate goal.

2.3. Avoiding premature convergence

The goal of the breeding programme is to promote the emergence of a dominant genetic code, which is the compact representation of the solution to the ODE. At each breeding stage, improvements are promoted to the parent gene pool by virtue of their lower penalty weighting, so that better information is used in subsequent generations with the ultimate aim of converging on the best possible form. In order to keep the process of evolution in operation, the 'environment' must change or pose sufficient challenges that the breeding stock is induced to strive for still better results. If conditions are only slowly changing, there is a danger that a sub-optimal form will come to dominate the entire stock, wiping out all diversity and effectively creating a genetic cul-de-sac. The environment in this analogy corresponds to the sampled part of the solution space represented in the entire breeding stock, and weighted by the differential operator. The genetic cul-de-sac then takes the form of an inadequate candidate for the solution of the ODE which is better than its contemporaries, but has features which must be improved by surmounting a local maximum in the weighting function to achieve a much better improvement. The GA becomes very inefficient under such circumstances, since the accumulated experience of the algorithm forms too narrow a base from which to evolve better forms (see discussion by Davidor (1991)).

In order to prevent this occurrence, GENODE employs the following strategies. After a specific number of iterations, GENODE suspends the normal breeding process for one cycle, replacing it by a 'tweaking' process in which all the parents have small changes made to their genetic code in order to test whether such a minor alteration can improve the overall weighting. This relaxation is done in several ways.

If the best parent has a weighting greater than some preset threshold, then each parent is taken in turn and its entire string is altered by adding random noise to the m values. The usual number of derived offspring are generated by this coarse tuning process, and the sorting and evaluation proceeds as before. An element of learning is retained by restricting

the amplitude of the noise, so that offspring generated in this fashion have some memory of their origin.

If the best weight is below the threshold, a fine tuning process is instigated, not on the whole string, but on the worst part based on the location of the maximum excursion measure ϵ_4. A section of the parental string is altered by adding noise to the n values, so fine-tuning the worst point and its neighbours. The fact that only the n values are being altered is sufficient to retain the learning component, and so no restriction is placed on the noise amplitude.

These two basic strategies apply after every N_{crit} iterations, N_{crit} having been set at execution time. However, the situation can arise where the parental stock is just N_{par} copies of the same string, with an associated non-zero weighting. In this circumstance, the relaxation procedure takes place automatically, using a combination of focused and general fine and coarse tuning where appropriate, and in alternate cycles, depending on whether the weight initially is above or below the threshold level.

These recovery strategies help to avoid premature convergence in almost all cases, although there are very occasionally situations in which the best option is to restart the calculation with a fresh data set.

2.4. Point resolution

The use of second-order finite difference algorithms to represent the derivatives in GENODE is a potential source of error and inefficiency, because the finite difference method is subject to round off error, and because only the nearest neighbour interactions are involved in calculating gradients. The latter aspect is the more serious, since this means that information about the boundary conditions only filters into the interior points via the differential operator, and at a pace and strength of influence dicatated by the combination of derivatives in the differential operator. To be specific, any boundary information supplied has a direct influence only on the two points next to the actual endpoints; points interior to these are not exposed directly to these constraints, but learn to accommodate them through the application of penalty weighting. Clearly if the interior points predominate, then the influence of the boundary conditions will be difficult to assert, and the convergence will be very slow.

GENODE has a strategy to circumvent this problem by solving the equation at six points initially: two endpoints and four interior points. This choice ensures that all function points (that is values of the unknown) are influenced directly by the boundary conditions and operator application at the endpoints, since a second derivative at an endpoint embraces the next two interior points. In this way, GENODE constructs a coarse solution over the poorly resolved interval. If better resolution is desired, GENODE bisects each interval and repeats the process, retaining the information generated at every other point and assigning values to the new points simply by averaging the calculated data on either side. This is an efficient use of the GA, since no attempt is made to evolve large and unwieldy data strings with only an indirect influence over the evolution strategy of the majority of the components. The resolution is thus improved exponentially, taking a fraction of the time it would have required had the solution evolved at all points from the start. This is analogous to the shooting method of conventional numerical solution techniques.

2.5. Numerical stability

Since the GA does not attempt to iterate towards a numerical solution by prescribing explicitly the convergence strategy of an initial guess, the GA avoids the issue of the stability

of the numerical algorithm. For example, a stiff ODE such as

$$y'' - y = 0 \tag{2.8}$$

has solutions $y = e^{\pm x}$. Attempting to solve (2.8) over the interval $x \in [0, \infty]$ for the subdominant solution $y = e^{-x}$ with a simple Runge–Kutta algorithm may lead to poor convergence because numerical round-off can never permit the total exclusion of the dominant form $y = e^{+x}$, and ultimately this component will exert an ineradicable influence over the final shape of the solution. The GA approach will simply discard any intermediate construct exhibiting the significant presence of the dominant form, because the GA uses the differential operator only to assess a given candidate, and not to suggest an iterative strategy towards a better guess. Hence the GA will also generate solutions with the dominant component present, but these will not survive the breeding process, and will eventually be replaced by more acceptable forms.

3. Examples

GENODE has been applied to the solution of a few problematic ODEs in order to demonstrate the efficacy of using this method to solve boundary value problems.

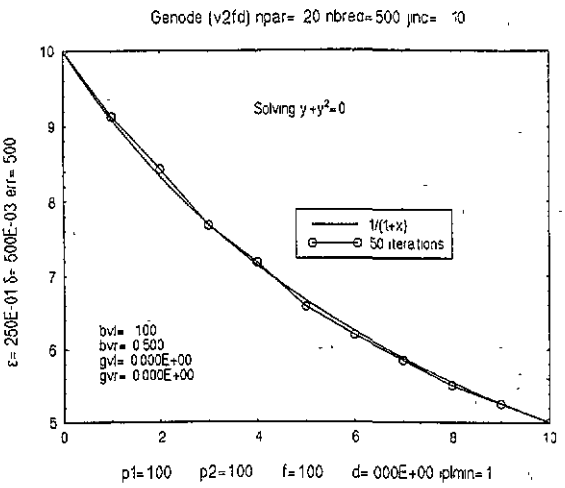

Figure 1. Solution of equation $y' + y^2 = 0$. With the given boundary conditions, the analytic solution is $y = (1 + x)^{-1}$; GENODE's output is shown as the curve with circles marking the data points.

The examples are detailed in the figures 1 to 4. The notation on each graph is as follows: ϵ and δ are as stated earlier, *err* is the threshold value of weighting discussed in section 2.3, where the weighting for the kth candidate is constructed as

$$W_k = p_1\epsilon_4 + p_2\epsilon_1 + f\epsilon_2 + d\epsilon_3 \tag{3.1}$$

iplmin is a control parameter used to restrict the search range (*iplmin* $= 1$ means do not select integers m and n which give negative y values, *iplmin* $= -1$ means no positive

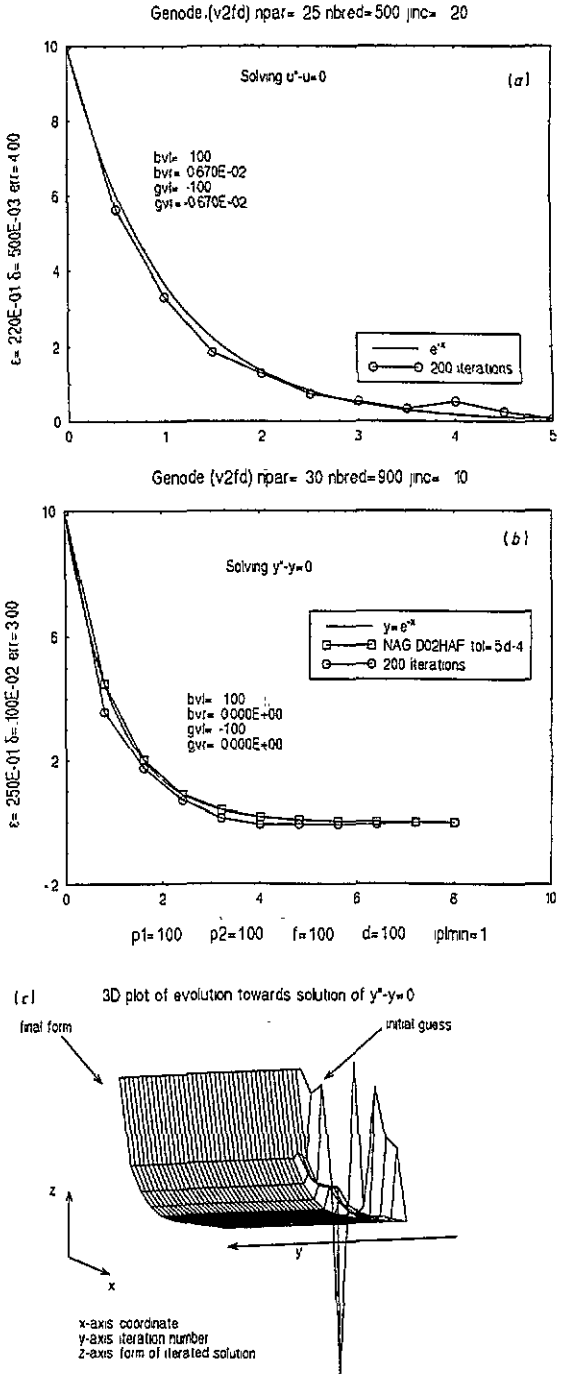

Figure 2. (*a*) Solution of the equation $y'' - y = 0$, with general solution $y = ae^{-x} + be^{+x}$, where a and b are constants. Given the boundary conditions, only the decaying solution is correct. (*b*) Solution of the same equation as in 2(*a*), but over a larger range of x values and for poorer point resolution. The circle symbols denote GENODE's output, and the square ones are those produced by a standard NAG routine. (*c*) Showing the evolution towards the final form, with the shape of the best parent shown at every 10th interation, thus creating a 3D picture of the development. Note that GENODE converges relatively quickly to the rough shape, leaving only fine detail to be adjusted in the later iterates. The equation and parameter values are as for (b).

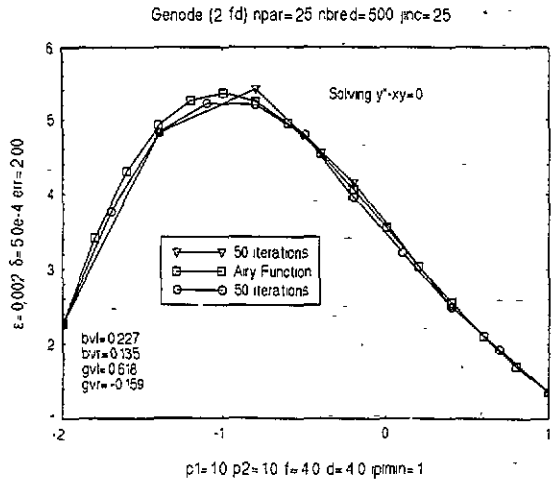

Figure 3. Solution of the Airy equation $y'' - xy = 0$, with general solution $y = c_1\mathrm{Ai}(x) + c_2\mathrm{Bi}(x)$. The boundary conditions pick out the decaying solution $\mathrm{Ai}(x)$. The graph shows GENODE's output for a six-point coarse solution after 50 iterations (triangular symbols), and then the refinement of that solution achieved by doubling the number of points and continuing for a further 50 iterations (circular symbols). The tabulated values of the Airy function taken from Abramowitz and Stegun are shown (square symbols).

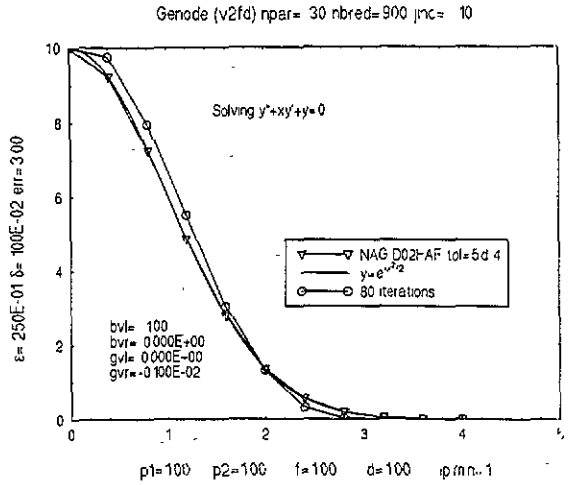

Figure 4. Solution of the equation $y'' + xy' + y = 0$, with subdominant solution $y = \exp(-x^2)$ using 11 data points. GENODE's output after 80 iterations is shown (circular symbols) and a standard NAG routine is displayed for comparison (triangular symbols).

values, and *iplmin* = 0 means no restrictions), *npar* is the number of parents, from which *nbred* children are manufactured, and the 'tweaking' process referred to in section 2.3 is performed every *jinc* iterations. The boundary values are given as parameters *bvl* and *bvr*, referring to the leftmost and rightmost y values respectively, and the gradients are specified in an analogous way using the variables *gvl* and *gvr*. The parenthetical *v2fd* refers to

the second-order central finite-difference formula version of the algorithm. The number of iterations quoted indicates the maximum number of iterations requested at the start of the computation, and therefore not necessarily the minimum number required to reproduce the solution.

Examining figure 1, it can be seen that this simple nonlinear ODE presents little difficulty: GENODE converges on the appropriate form of the solution with no trouble.

Figures 2(a) and (b) show the results of a much tougher challenge, solving a stiff ODE over a significant range of x values with comparatively poor point resolution. This justifies the claim made in section 2.5 that stiff systems present no particular hurdle to GA solution methods. The surface plot shown in figure 2(c) illustrates the evolution of the final solution form, as the shape of the solution is plotted as a function of iterate number. Note that in the latter stages, only fine-scale tuning of the solution takes place.

This stability aspect is further emphasized by solving the Airy equation, as shown in figure 3. This shows GENODE's point resolution strategy, where the curve marked with the triangular symbol has only six data points, giving a coarse fit after 50 iterations. This coarse fit was refined by doubling the number of points, and iterated a further 50 times to produce the curve with the circular symbols. The third curve is a plot of values of the Airy function taken from Abramovitz and Stegun (1972).

The last curve, figure 4 shows GENODE again performing a tough calculation, seeking the subdominant solution in a stiff ODE.

These examples compare GENODE's performance with the NAG library routine D02HAF, a general purpose boundary value solver using a combination of Runge–Kutta–Merson and Newton iteration in a shooting and matching technique. Whilst the GA is slower, it is reasonably accurate considering it does not employ an advanced deterministic convergence strategy such as NAG uses.

4. Summary

GENODE has been applied to various ODEs, both stiff and non-stiff, and is capable of solving boundary value problems accurately and efficiently. Since the heart of the code is the breeding algorithm, GENODE does not require to be tailored to suit each ODE: one algorithm tackles all. In particular, the only real input required of the user is the value of the coefficients in the candidate ODE. Currently the user is invited to choose from a menu of possible ODE forms as follows. The general ODE takes the form

$$C_2y'' + (C_1 + C_{1x}x + C_{1xs}x^2)y' + (C_0 + C_{0x}x + C_{0xs}x^2)y + C_{ys}y^2 = 0$$

and a compact, second-order finite difference form of this equation is encoded within GENODE. The user then has to specify the values of the coefficients C_q, the solution interval, the number of points at which the solution is required and the boundary conditions. GENODE can then proceed with default parameter values for the number of parents and children, error levels, etc. if the user so desires. At no point is the user required to encode any Fortran, or alter the source code of GENODE in any way to tailor the solution method to a particular ODE: GENODE learns its own strategy. This is an important difference between this approach and that of the NAG library, for example. Moreover, stiff systems pose no threat, since those calculations displaying an undesirable trend are merely discarded.

Herein then lies the power of GENODE: a multipurpose, self-learning numerical solver that is stable in all situations and easy to use. Independent of library routines, it is readily

parallelized and general in application. Whilst it does not always offer the precision of advanced conventional solvers, it may be valuable in calculating an excellent estimate of the form of the solution as input to a high precision solver such as NAG D02RAF. It is in this area that major applications of GENODE are planned in future research plans: solving complicated boundary value problems in stellar structure of fluid flows in constrained geometry are of particular interest.

In order fully to exploit the power of GENODE, and develop its full potential in other applications, it must be run on a parallel machine. The algorithm strategy is intrinsically parallel, since offspring are independent: in fact the only serial process is the ordering after each step. Serial computers impose huge execution penalties on GENODE, making further development time consuming and impractical. New technology must be harnessed in this novel approach, and GENODE is an ideal example of new generation computing strategy. The flexibility and awesome power of parallel processing make it imperative that new approaches to numerical simulation be developed which are designed to exploit such advances. GENODE is a prime example.

GENODE is written in Fortran 77, using the Salford FTN386 compiler. The random number generator routine at the heart of the algorithm is taken from the NAG library, although FTN386 is supplied with an equally acceptable one.

Acknowledgments

The authors wishes to thank SERC and the Nuffield Foundation for financial support, and colleagues in the Department of Physics and Astronomy for their constructive comments and enthusiasm.

References

Abramowitz M and Stegun I A 1972 *Handbook of Mathematical Functions* (New York: Dover) p 475
Bhattarcharjya A K, Becker D E and Roysam B 1992 *Signal Processing* **28** 335–48
Davidor Y 1990 *Genetic Algorithms and Robotics* (London: World Scientific)
Greffenstette J J, Gopal R, Rosmaita B J and Van Gucht D 1985 *Proc. 1st Int. Conf. on Genetic Algorithms and Their Applications* (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 160–8
Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (New York: Addison-Wesley)
Hajima R, Takeda N, Ohashi H and Akimaya M 1992 *Nucl. Instrum. Methods* A **318** 822–4
Herdy M 1991 *Lecture Notes in Computer Science* **496** 188–92
Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor: The University of Michigan Press)
Jo D and Didier K 1991 *Lecture Notes in Computer Science* **496** 352–62
Mahlab U, Sharmir J and Caulfield H J 1991 *Optics Lett.* **16** 648–50
Wilson W G 1991 *Geophys. Res. Lett.* **18** 2181–4