THE ASTROPHYSICAL JOURNAL SUPPLEMENT SERIES, 101:309–334, 1995 December

GENETIC ALGORITHMS IN ASTRONOMY AND ASTROPHYSICS

P. CHARBONNEAU
High Altitude Observatory, National Center for Atmospheric Research,[1] P.O. Box 3000, Boulder, CO 80307-3000;
paulchar@hao.ucar.edu

Received 1994 December 30; accepted 1995 June 9

ABSTRACT

This paper aims at demonstrating, through examples, the applicability of *genetic algorithms* to wide classes of problems encountered in astronomy and astrophysics. Genetic algorithms are heuristic search techniques that incorporate, in a computational setting, the biological notion of evolution by means of natural selection. While increasingly in use in the fields of computer science, artificial intelligence, and computed-aided engineering design, genetic algorithms seem to have attracted comparatively little attention in the physical sciences thus far.

The following three problems are treated: (1) modeling the rotation curve of galaxies, (2) extracting pulsation periods from Doppler velocities measurements in spectral lines of δ Scuti stars, and (3) constructing spherically symmetric wind models for rotating, magnetized solar-type stars. A listing of the genetic algorithm-based general purpose optimization subroutine PIKAIA, used to solve these problems, is given in the Appendix.

Subject headings: galaxies: kinematics and dynamics — methods: numerical — stars: mass loss — stars: oscillations

1. INTRODUCTION: GENETIC ALGORITHMS

The class of computational techniques that form the subject matter of this paper addresses problems of *optimization.* Perhaps the most common mathematical optimization tasks are *minimization* and *maximization,* i.e., finding extrema of a function. Optimization is typically first encountered in a calculus class and is often remembered as a boring exercise in differentiation. What is perhaps not widely appreciated is that many classes of problems arising in the physical sciences can be recast in the form of optimization problems. Data fitting using least-squares is clearly an optimization (minimization) problem. Root finding problems for nonlinear, coupled systems of equations can be recast in the form of minimization problems. Any algebraic system of equations (including matrix systems arising from the discretization of differential equations) can actually be solved as a residual minimization problem.

It is therefore no surprise that most introductory textbooks on numerical analysis devote a substantial part of their discourse to optimization. Especially in the area of continuous function optimization, some very powerful numerical techniques have been developed, the archetype being conjugate gradient methods (see, e.g., Press et al. 1992, § 10.6). However, these powerful techniques can sometimes fail miserably on problems that are readily solved by hand using the "set the derivative to zero and solve for x" time-proven strategy. It is important to understand how and why this happens in order to later better appreciate the power and versatility of genetic algorithms-based optimization.

1.1. *A Model Optimization Problem*

It will prove useful to begin by constructing an optimization problem that, while conceptually straightforward, presents formidable difficulties for many classical optimization methods. Consider the following function of two variables:

$$f(x, y) = [16x(1-x)y(1-y)\sin(n\pi x)\sin(n\pi y)]^2\,, \qquad x, y \in [0, 1], \quad n = 1, 2, \cdots \qquad (1)$$

and the associated optimization task of finding the pair of values $[x_{\max}, y_{\max}]$ returning the maximum evaluation of f. Analytically, this is a straightforward, if somewhat tedious, exercise in partial differentiation. Numerically, the *steepest ascent method* represents a particularly simple mean of achieving the same result. Starting from an initial guess for $[x_0, y_0]$, one computes the local gradient $\nabla f(x, y)$ to establish the direction of maximum slope. One then takes a small "step" in that direction, reevaluates the new local gradient, takes another step, and so on until the maximum is reached. Clearly, steepest ascent/descent algorithms—or any other type of hill climbing technique such as the conjugate gradient method—are *local* methods that will work well if (1) the landscape is relatively well behaved, i.e., $f(x, y)$ is a smooth function that can be differentiated at least once, and (2) a single maximum exists in the domain under consideration.

Figure 1 is plot of the function defined by equation (1) for n = 1 (Fig. 1*a*) and n = 9 (Fig. 1*b*). It reveals immediately why the associated optimization problem, numerically, is an absolute nightmare for large n. Equation (1) with $n = 9$ defines a function having 81 local maxima in the domain under consideration. To make matters worse, many local maxima differ very little in function value and are separated by deep "valleys" in the two-dimensional landscape. In this context, the steepest ascent method is analogous to flipping the two-dimensional surface upside down, dropping a ball somewhere ($[x_0, y_0]$) in the (inverted) landscape, and letting it roll all the way down to the lowest point. In the case of Figure 1*a*, this will work without difficulties. But clearly, in the case of Figure 1*b* this strategy will succeed only if the ball is dropped in the (very restricted)

[1] The National Center for Atmospheric Research is sponsored by the National Science Foundation.

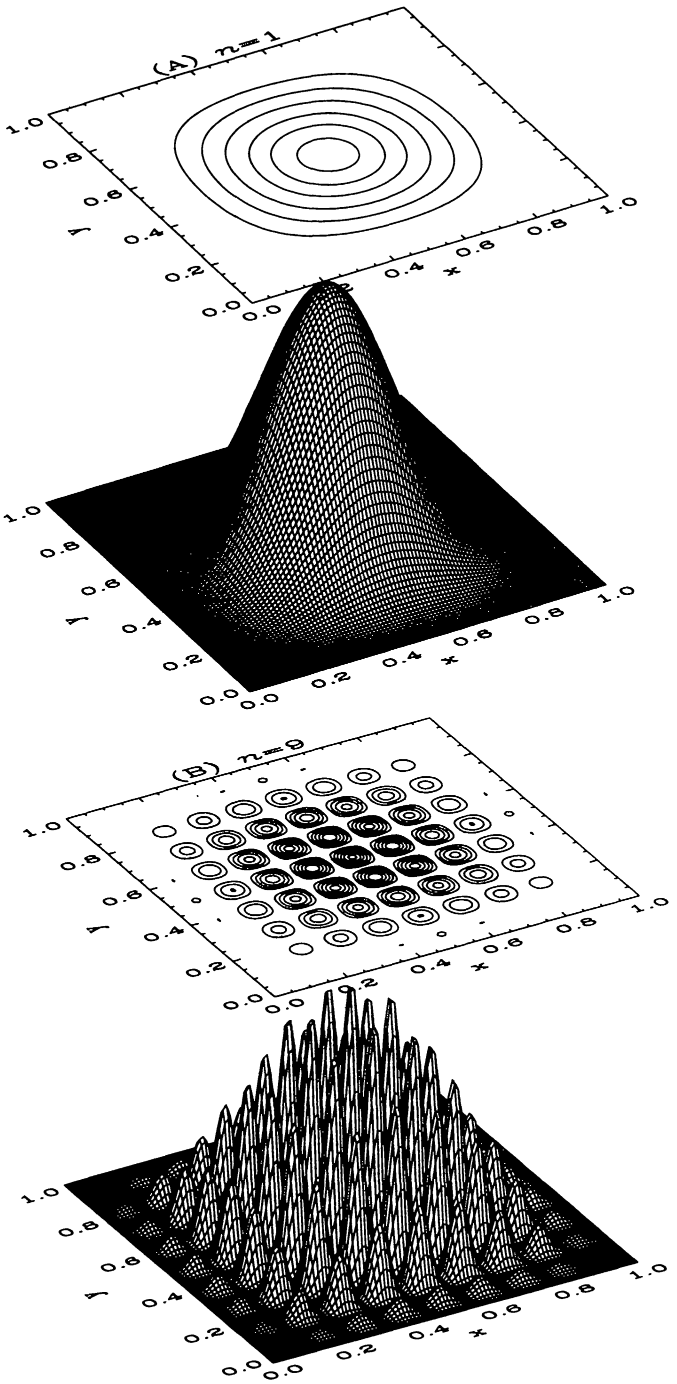


FIG. 1.—Surface and contour plots of the function $f(x, y)$ defined by eq. (1) with (a) $n = 1$ and (b) $n = 9$. In this latter case there are 81 local maxima in the domain $x, y \in [0, 1]$. In both cases the absolute maximum is $f(0.5, 0.5) = 1$.

interval ($\frac{4}{9} \le x \le \frac{5}{9}, \frac{4}{9} \le y \le \frac{5}{9}$) centered on $(x, y) = (0.5, 0.5)$. Any other starting point will cause the steepest ascent method to find a secondary maximum. A number of options are available to circumvent this difficulty. One can repeatedly restart the algorithm at other randomly chosen points, keeping track of all extrema located in this fashion. Such *iterated hill climbing* schemes become impractical as the dimensions and complexity of the search space are increased. In fact, for a M-dimensional generalization of equation (1) one can readily show that the probability of a randomly chosen starting point landing within reach of the central maximum scales as $1/n^M$. Iterated hill climbing schemes are in principle *global,* but they lack *efficiency.*

Another line of attack consists in incorporating in the search process problem-specific a priori knowledge (for example, concerning the overall structure of parameter space, such as asymptotic behavior, periodicities, etc.). Such fixes can be easy to design, but often end up requiring repeated manual intervention on the part of the modeler if applied to a wide spectrum of problems. In other words, these modified hill-climbing techniques lack *robustness.*

Ideally, what is needed here is a method that is both robust, efficient, and global. One class of such techniques are the *simulated annealing* methods (see § 10.9 of Press et al. 1992, and references therein). These methods are based on an analogy with thermodynamics, namely, the way in which a liquid that is slowly cooled crystallizes into what can end up being a highly organized large-scale molecular structure that corresponds to a global minimum energy state for the ensemble of molecular constituents. Simulated annealing has successfully solved a number of difficult global optimization problems, including the (in)famous traveling salesman combinatorial problem, but in many instances the performance of the method depends rather sensitively on the so-called annealing schedule, and on the specification of a suitable set of alternate pseudothermodynamic states for the algorithm to choose from (see Press et al. 1992, § 10.9). This is probably in part why the technique has not yet attracted much attention in astronomy and astrophysics, although it has been successfully applied to a few problems of interest (see, e.g., Jeffrey & Rosner 1986b).

Like simulated annealing, the techniques discussed in this paper are global, but their underlying motivation lies in the realm of biology, rather than physics.

1.2. *Biological Evolution as an Optimization Problem*

Darwin's 1859 *Origin of Species* brought *natural selection* to the forefront of evolutionary thought. Natural selection is the process whereby individuals better adapted to their environment (i.e., "fitter") tend to produce more offspring, on average, than their less well-adapted competitors. Darwin saw very well that two additional key ingredients were needed in order for natural selection to lead to large-scale *evolution:* (1) *inheritance,* or parents somehow "passing on" their fitness to their offspring, and (2) *variation,* or the existence of a fitness spectrum in a given population, so that natural selection can actually operate. The understanding of how variation is maintained and inheritance mediated, however, had to await the rediscovery of Mendel's law at the turn of this century, and the subsequent rise of experimental and mathematical genetics (see Stebbins 1966; Watson et al. 1987, chap. 1; Maynard Smith 1989).

The starting point of this discussion is to establish the distinction between *genotype* and *phenotype.* The genotype refers to the genetic makeup of an individual, stored on chromosomes in the form of linear gene sequences. The phenotype is the outward manifestation of the genotype, i.e., the individual that actually feeds, competes, and breeds in real space. At the level of the individual, the relationship between phenotype and genotype is very much unidirectional. The phenotype is simply a decoded version of its genotype, the decoding being what one usually refers to as development and growth (which implies, incidentally, that there exists in general environmental influences on the phenotype that are *not* encoded in the genotype). However, because high (low) phenotype fitness translates, *on average* into high (low) reproductive success, the phenotype has a direct influence on the "gene pool" (i.e., the ensemble of genotypes) of the *next* generation; high fitness phenotypes "copy" their genotype more frequently into the next generation, which leads to an increasingly higher proportion of high fitness phenotypes in the population. For fixed environmental conditions, the population as a whole will naturally (!) converge on the highest fitness genotype present in the original gene pool. How fast this convergence process operates is a function of *selection pressure,* the functional relation between fitness and reproductive success.

If this were all there is to evolution the initial spectrum of variation present in the population would gradually be annihilated. Once convergence is completed, severe changes in environmental conditions may suddenly alter the rules of the game, and make *all* individuals extremely unfit, with potentially disastrous consequences. This is has certainly happened in the geological past, as evidenced by the numerous episodes of mass extinction laid out in the fossil record. But the very fact that life has evolved—and continues to evolve—in response to changing environmental conditions implies that variation is maintained. It turns out to be maintained to a large extent by the very process of inheritance. Sexual reproduction is not an exact copying of the parent genotypes into the offspring. It frequently entails large-scale *recombination* of genetic material by the processes of chromosomal *crossover* and *inversion* (see, e.g., Stebbins 1966, chap. 3). Further variations are introduced by *mutation,* either in the form of copying mistakes or true random events affecting a gene in isolation.

Genetic algorithms are a class of heuristic search techniques that incorporate these ideas in a nonbiological, computational setting. Strictly speaking, genetic algorithms are not function optimizers (De Jong 1993), although they turn out to be extremely well suited for that task. Nevertheless, throughout this paper the term "genetic algorithm" (or "GA") will sometimes be used as a shortcut for what should formally be referred to as "genetic algorithm–based optimizer". The following (brief) presentation is in part adapted from Davis (1991), a highly recommended tutorial text. Goldberg (1989) is the standard textbook on the subject. The monograph by Holland (1975) is the classical reference on genetic algorithms.

1.3. *A Basic Genetic Algorithm*

In their most basic implementation, genetic algorithms make use of the following simplified version of the biological evolutionary process: the gene pool (and associated phenotypic population) evolves in response to (1) differential reproductive success in the parent phenotype population, (2) crossover (recombination at the chromosome level) at breeding, and (3) random mutation occurring directly at the gene level. Admittedly, evolutionary biologists are actively engaged in a number of debates concerning many aspects of evolutionary theory, including the molecular biology of genetic recombination (e.g., Watson et al. 1987, chap. 19), the tempo and mode of evolutionary change (Eldredge 1985; Gould 1989), gene versus individual as the focal point of the evolutionary process (Dawkins 1982), to name but a few. Nevertheless, the basic aspects of the process, as embodied in genetic algorithms, are beyond dispute.

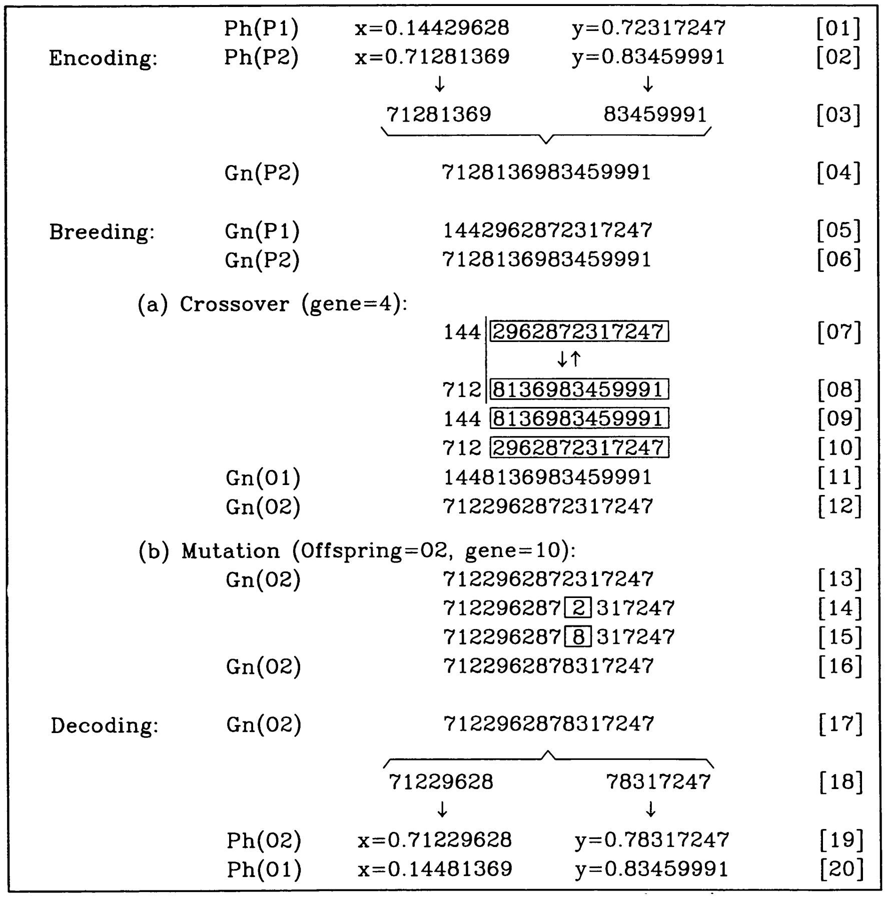
A top-level view of a genetic algorithm is as follows: given a target phenotype and a tolerance criterion,

1. Construct a random initial population and evaluate the fitness of its members.
2. Construct a new population by breeding selected individuals from the old population.
3. Evaluate the fitness of each member of the new population.
4. Replace the old population by the new population.
5. Test convergence; unless fittest phenotype matches target phenotype within tolerance, `goto` step 2.

In the case of the model problem of Figure 1, an individual (phenotype) is an (x, y) pair, and fitness could simply be (but does not have to be) the evaluation of $f(x, y)$. In themselves, the steps listed above define only an *adaptive plan* or *evolution program,* with nothing obviously "genetic" about the whole thing. In fact, steps 1–5 look more like some variation on the Monte Carlo theme. Genetic algorithms actually differ from random heuristics in three essential ways, all related to breeding (which is also where the "genetic" comes in). First, what is being bred in step 2 are not the phenotypes, but the genotypes that encode the selected phenotypes. Second, new genotypes are produced by applying crossover and mutation operators to the parent genotypes, processes with distinct random attributes, as opposed to a deterministic scheme such as, e.g., averaging. Finally, the probability of an individual being selected for breeding is proportional to its "fitness."

Figure 2 illustrates the details of the breeding procedure, in the case of breeding two "solutions" to the optimization problem of § 1.1, with the target phenotype being the pair (0.5, 0.5) for which $f(x, y) = 1$. The first step consists in *encoding* the two parent phenotypes Ph(P1) and Ph(P2). Lines [02] through [04] in Figure 2 illustrate this procedure for Ph(P2). Here the genotype consists of a single chromosome of length 16, where each of the 16 "genes" can assume an integer value between zero and nine.[2] Breeding is a multistep procedure. Be-

[2] Classical implementations of genetic algorithms most often make use of *binary encoding,* whereby phenotype values are converted to binary numbers, and concatenated into what ends up being a long string of "0" and "1." The encoding scheme illustrated on Figure 2 functions in base 10 instead. One could argue that there is an inherent simplicity in binary encoding that makes it *the* fundamental encoding scheme. Yet one may also recall that DNA-based life, which includes all known biological life forms with the exception of some classes of viruses and possibly my next-door neighbor's roommate, is encoded in base four (the digits being A-T-C-G: yes indeed, those unforgettable Adenine, Thymine, Cytosine, and Guanine of high school biology . . .). The choice of an encoding scheme is far from a trivial issue, as it is known to affect the performance of genetic algorithm to varying degrees, depending on the problem at hand. Considerable effort has gone into developing encoding schemes based on floating point representation together with suitably altered genetic operators, and



```
                 Ph(P1)    x=0.14429628      y=0.72317247       [01]
Encoding:        Ph(P2)    x=0.71281369      y=0.83459991       [02]
                                ↓                 ↓
                            71281369          83459991          [03]

                 Gn(P2)          7128136983459991               [04]

Breeding:        Gn(P1)          1442962872317247               [05]
                 Gn(P2)          7128136983459991               [06]

           (a) Crossover (gene=4):
                                 144|2962872317247|             [07]
                                      ↓↑
                                 712|8136983459991|             [08]
                                 144 8136983459991              [09]
                                 712 2962872317247              [10]
                 Gn(O1)          1448136983459991               [11]
                 Gn(O2)          7122962872317247               [12]

           (b) Mutation (Offspring=O2, gene=10):
                 Gn(O2)          7122962872317247               [13]
                                 712296287[2]317247             [14]
                                 712296287[8]317247             [15]
                 Gn(O2)          7122962878317247               [16]

Decoding:        Gn(O2)          7122962878317247               [17]

                            71229628          78317247          [18]
                                ↓                 ↓
                 Ph(O2)    x=0.71229628      y=0.78317247       [19]
                 Ph(O1)    x=0.14481369      y=0.83459991       [20]
```

FIG. 2.—Encoding, breeding, and decoding in genetic algorithms. Phenotypes are defined in terms of two real numbers and are encoded as a string of 16 decimal digits (eight per real number). "Ph(P1)" means "phenotype of parent P1," "Gn(O2)" is "genotype of offspring 2," and so on. Encoding is shown only for Ph(P2), and decoding for Gn(O2). Note that a breeding event produces *two* offspring, and that both crossover and mutation operations occur only if a probability test yields true (*see text*).

gin by copying the parent genotypes Gn(P1) and Gn(P2) (lines [05]–[06] in Fig. 2). Whether or not crossover is to occur depends on the outcome of a probability test; this may consist in generating a random number $R \in [0, 1]$ and applying the *crossover operator* (line [07]–[12]) only if R is inferior to some preset crossover rate p_c (≤ 1, in fact a probability measure); if the probability test yields true, select randomly a gene (the fourth here) and cut both parent chromosomes at that point (lines [07]–[08]). Interchange the chromosome segments right of the cutting point to form the offspring genotypes Gn(O1) and Gn(O2) (lines [09]–[12]). This completes the crossover operation.[3] The genotype of each offspring now contains discrete, intact chunks of chromosome material from each parent. Now for each offspring, run a probability test for each gene making up the chromosome. If the test yields true, apply the *mutation operator* (lines [13]–[16], shown here operating on the 10th gene). This consists in replacing the corresponding gene value by a random integer in the range [0, 9] ("8" here, line [15]). This completes the mutation operation. Something has been inserted in the offspring genotype that was present in neither of the parents.

Comparing the parent phenotype pair [(0.14429628, 0.72317247), (0.71281369, 0.83459991)] to the offspring phenotypes [(0.71229628, 0.78317247), (0.14481369, 0.83459991)], it is clear that a substantial jump in parameter space has taken place. The magnitude of the effects of crossover and mutation clearly depends on the genes at which these op-

assessing their performance as compared to binary representation. Empirical evidence suggests that floating point-based schemes do at least as well as conventional binary-based schemes on many classes of optimization problems (see Wright 1991; Michalewicz 1994, chap. 5).

[3] What has been described is *one-point crossover.* Other types of crossover operators, involving multiple splicing points, can also be defined (see, e.g., Goldberg, chap. 4). One-point crossover seems to be the dominant mode of crossover recombination in the few natural genetic systems studied in detail to date (e.g., the *E-coli* bacterium and the fly *Drosophila*). Note also that in natural systems, crossover refers to exchange of segments between two homologous chromosomes during cell division (cf. Watson 1987, chap. 1); exchange between nonhomologous chromosomes is referred to as translocation and is usually deleterious. Strictly speaking, the form of crossover implemented in genetic algorithms—exchange between two parent chromosomes at breeding—has no direct counterpart in biological systems.

erations take place. In the case illustrated in Figure 2, mutation can induce variations by factors of order 1, down to one part in 10^8. Because each gene has equal probability of being affected, there results a logarithmic distribution of effects in x or y values, which can translate into almost anything in terms of function evaluation (eq. [1]) given the nonlinear nature of $f(x, y)$. Crossover is harder to quantify. Typically, the numerical value of a single component of the decoded genotype is again affected. But complete components are also interchanged, as compared to the parent genotypes. In the case considered on Figure 2 this amounts to exchanging the (intact) y values of the parent phenotypes. But in cases where the relationship between phenotype and decoded genotype is less direct, the effects of crossover on the offspring phenotypes can be subtle and far reaching. Note that there is in general a finite probability that neither crossover nor mutation occurs during a breeding event, in which case the two offspring are true replications of the two parents.

1.4. *A Solution to the Model Problem: Introducing* PIKAIA

The Appendix to this paper gives a listing and description of PIKAIA, a genetic algorithm-based general purpose optimization subroutine. Figure 3 shows a genetic solution to the model optimization problem defined equation (1) with $n = 9$ (cf. Fig. 1*b*), obtained with PIKAIA. A population of 100 individuals evolved over 100 generations with crossover and mutation rates of 0.65 and 0.003, respectively. Figure 3*a* shows the spatial distribution of the initial (random) population. Few individuals lie anywhere near the tallest, central peak in the landscape, or even near the summit of any major peak. More important, *none* are lying close enough to the central peak for local hill climbing methods to find the absolute maximum. After only 10 generations (Fig. 3*b*) the part of the population initially occupying the outer, low-altitude regions of the landscape has been effectively decimated, and the population converges on a few peaks that do *not,* however, include the central, highest peak. As the next few subsequent generation are bred, mutation and crossover catapult a few individuals on the slope of the highest peak, where a foothold is rapidly established. By the 20th generation (Fig. 3*c*), a few individuals have attained heights on the central peak exceeding the maximal altitude of neighboring peak. Natural selection now favors this subgroup, so that differential reproductive success gradually populates the central peak at the expense of its neighbors. By the 40th generation (Fig. 3*d*), the population as a whole resides on the central peak, with only a few (doomed) mutants appearing occasionally in other locations. From this point on most individuals have nearly identical genotypes, so that crossover in itself supplies fewer novelties to the gene pool. The occasional favorable mutation, however, is rapidly disseminated through the population if selection pressure is maintained. The population as a whole then gradually shifts closer and closer to the summit (Fig. 3*e*). Figure 3*f* shows the corresponding evolution of the fittest individual of a given generation (*solid line*), and median individual in the ranked population (*dashed line*). The key mutation event occurring between Figures 3*b* and 3*c* does not produce, in itself, a sharp increase in fitness (i.e., a drop in $1 - f$).

The evolutionary process can be subdivided into two more or less distinct phases. The first is characterized by a large-scale exploration of parameter space, made possible by the action of crossover on the extreme variety of genotypes making up the gene pool. This phase of rapid evolution effectively comes to a close when a large subset of the population has "found" the highest peak in the landscape. From that point on evolution occurs more slowly and leads to slow, incremental improvements in the population's fitness as favorable mutation becomes fixed in the population. Note that in Figure 3*f* how in this evolutionary phase a generational delay always exists between the appearance of a favorable mutation (the solid line for the fittest individual dropping down significantly), and its spread throughout the population (as evidenced by the dashed line for the median-fitness individual dropping down later, but to the same level).

The final accuracy (i.e., the evaluation of f for the fittest individual of the last generation of the run, here $f_{\max} = f(0.498456, 0.503658) = 0.987335$) of the solution shown in Figure 3 is not particularly impressive, although 100 generations is a *very* short evolutionary run. It is also important to realize that the rate of convergence of the algorithm varies from one run to another, and the final accuracies can easily vary by an order of magnitude. "Convergence" can only be defined in a statistical sense here. The solution shown on Figure 3 is actually somewhat worse than a "typical" solution; it was chosen because it illustrates an important operational advantage of genetic algorithms, namely, the capability of moving away from secondary maxima through crossover and mutation.

Figure 4 shows the effects of various assumptions concerning mutation and crossover rates, selection pressure, and population size. Each curve is a 100 run average and so is now more representative of the true performance under each set of conditions. A crossover rate $p_c = 0.65$ means that crossover happens (on average) for 65% of all breeding events; a mutation rate $p_m = 0.003$ means that in a given offspring, 0.3% of all chromosome locii (on average) are affected by mutation. High selection pressure (SP = H) means that the probability of being selected for breeding is heavily weighted toward the top end of the population's fitness distribution, while low selection pressure (SP = L) means that even very unfit individuals have a significant probability of breeding. A more quantitative description of how selection pressure is defined is given in the Appendix (§ A3).

It would appear from Figure 4 that an optimal genetic algorithm should work on a large population, maintaining high selection pressure and high rates of mutation and crossover throughout the evolution. This is not so. Genetic algorithms mimic life and, very much like life, are complex and unpredictable. Crossover, mutation, selection pressure, and population size interact in ways that are extremely nonlinear and, moreover, cannot be cleanly separated, as may superficially appear to be the case in Figure 4. For example, a large population converges rapidly at first, to a large extent because the probability of one individual in the random initial population landing very close to the target is directly proportional to the population size. However, during later evolutionary stages, a large population has a lot of inertia when the possibility arises to spread a favorable mutation. High mutation rates are a mixed

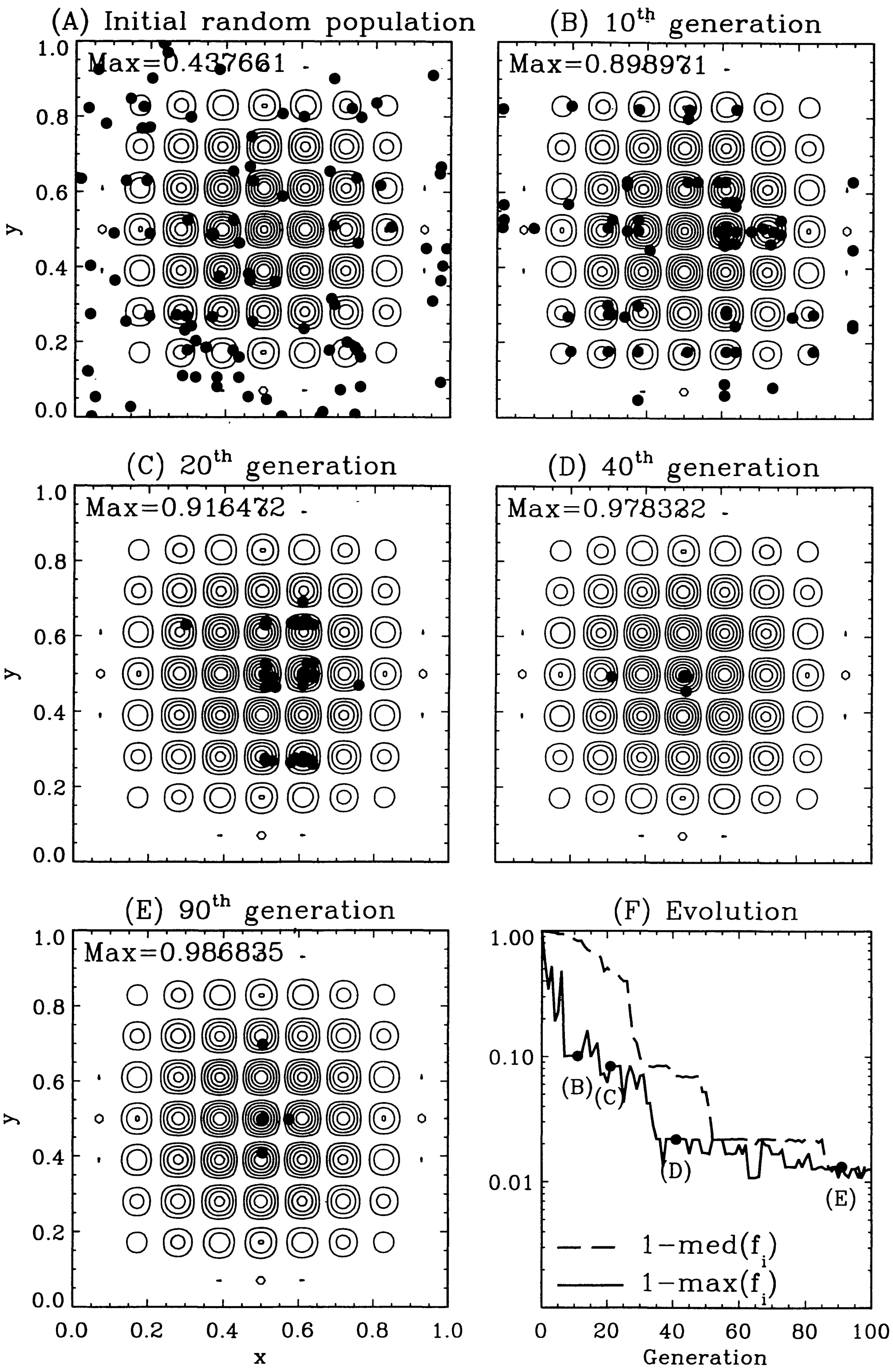


FIG. 3.—A genetic solution to the model optimization problem of § 1.1. The first five panels show the population distribution in space, overlaying elevation contours of constant *f*, starting with the initial (random genotype) population in (*a*) and proceeding on through the 90th on (*e*). The large filled circle is the fittest individual of its generation. The contours values are 0.05, 0.15, 0.3, 0.5, 0.7, and 0.9. (*f*) The evolution of the fittest phenotype (*solid line*, measured as one minus its associated function evaluation), and the median-fitness individual (*dashed line*) in each generation.

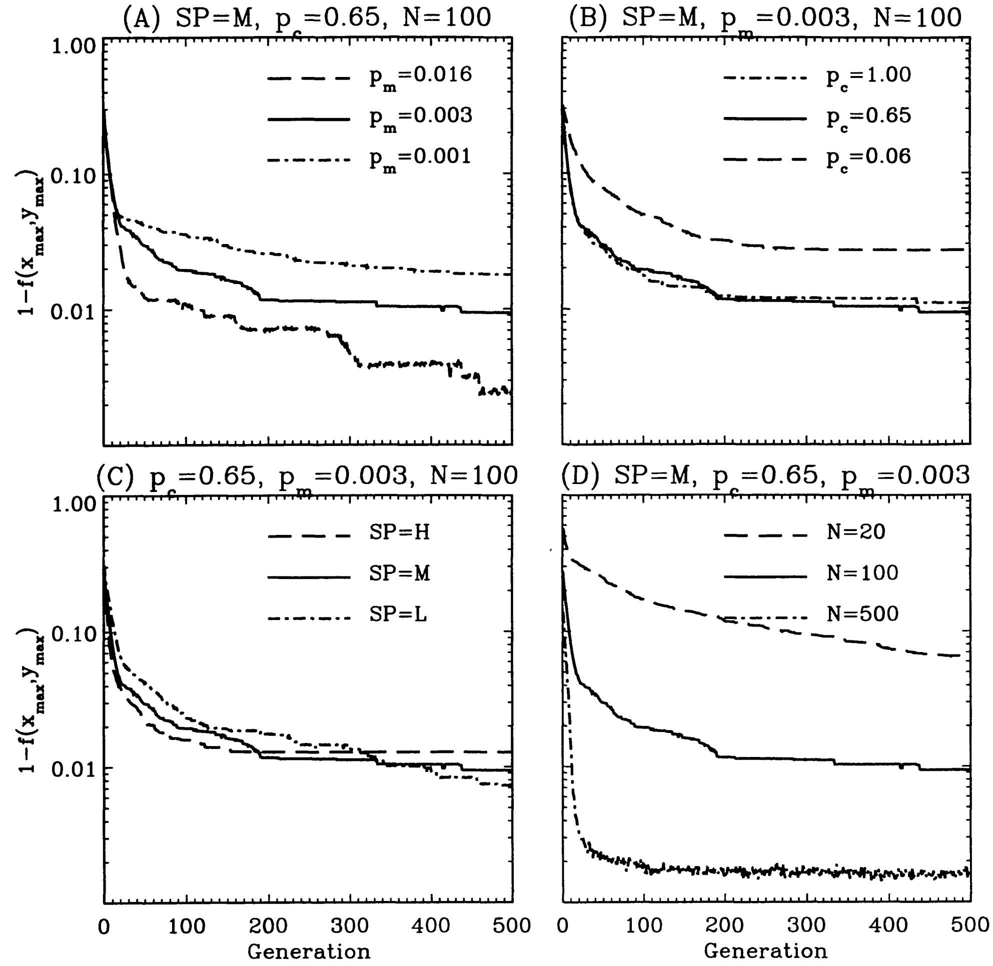


FIG. 4.—Fitness [as measured by $1 - f(x, y)$] vs generation count under various (*a*) mutation rates, (*b*) crossover rates, (*c*) selection pressures, and (*d*) population sizes. Each curve corresponds to the fittest individual averaged over 100 runs of the genetic algorithm.

blessing; they enhance the exploration of parameter space, but can impede accuracy. In fact, the optimal way to explore parameter space rapidly and efficiently involves a small population subjected to high mutation rates and moderate selection pressure.

The downhill simplex method of Nelder & Mead (1965) is formally a local optimization method and as such should not be expected to do particularly well on the model problem (Fig. 1*b*). Somewhat surprisingly, it turns out to do quite well, in fact, it exhibits pseudoglobal behavior, in the sense that the method (routine *amoeba* of Press et al. 1992, § 10.4) manages to locate the absolute maximum at (0.5, 0.5) over a wide range of sizes and locations of the initial simplex. This is probably due at least in part to the low dimensionality of the search space, since the set of "moves" executed by a two-dimensional simplex actually leads to a rather efficient exploration of two-dimensional space if the starting simplex is large enough. Simulated annealing, as embodied in the routine amebsa of Press et al. (1992, § 10.9), also succeeds in locating the global maximum of Figure 1*b*, as long as the annealing schedule is suitably defined. This is easy to do if prior information exists as to the relative depths of secondary extrema, but can be more problematic when no such information is available.

1.5. *Additional Strategies and Techniques*

The use of decimal—as opposed to binary—encoding excepted, what has been described up to this point could be labeled a "canonical genetic algorithm," in the sense that it incorporates the basic set of genetic operators originally suggested by Holland (1975). The solutions discussed in the preceding section were obtained by forcing PIKAIA to operate in this basic mode. In most cases, however, this is *not* optimal for numerical optimization. Numerous other operators and strategies have been developed to improve the performance of genetic algorithms in that context, as discussed for example in Davis (1991, chaps. 3 and 4) and Goldberg (1989, chap. 5). While such techniques can easily accelerate convergence—and improve final accuracy—many often require precise fine tuning of some model parameters and perform well only for a specific subclass of problems. In other words, they lack *robustness*. The following is a brief description of a few generally robust strategies and techniques that are incorporated as options in PIKAIA.

Fitness is Ranking.—Ranking, i.e., equating breeding probability to fitness-based rank in the population (as opposed to fitness *value*) is a useful way to maintain adequate selection pressure throughout the evolutionary run. Consider, for example, the solution shown on Figure 3; in the initial random population, the fittest and median individuals have function evaluations $f = 0.437661$ and 0.015896, respectively. If this is directly taken as a measure of breeding probability, then the fittest individual has a probability of being selected for breeding 28 times greater than the median individual. This seems fine, since the fittest individual is clearly superior, but consider what happens later in the evolutionary run. At the 40th generation, the fittest and median individuals have evaluations $f = 0.978322$ and 0.930098, giving them essentially identical breeding probabilities, even though the fittest is still much closer to target in a relative sense. A related problem occurs when, early in the evolution, one individual happens to have much higher fitness than its fellow population members. Making breeding probability equal (or linearly proportional) to fitness has the consequence that this one "superindividual" breeds too often for the health of the population, with the corresponding genotype rapidly dominating the gene pool. The loss of variation resulting from what rapidly becomes inbreeding often causes premature convergence to secondary extrema, with only mutation left to save the day. Ranking, on the other hand, ensures a constant fitness differential across the population, independently of what the distribution of absolute (true) fitnesses actually is. Ranking is the strategy used in PIKAIA.

Elitism.—This consists in ensuring that the fittest genotype of generation n is copied at least once without alteration into generation $n + 1$. This extremely simple procedure significantly improves the performance of GA-based optimizers, especially when used in conjunction with variable mutation rates.

Variable mutation rate.—Another useful strategy consists in gradually increasing the mutation rate once the difference in fitnesses between the median and fittest individuals falls below 20% (say). This represents a particularly simple way of maintaining variability in the gene pool and avoiding premature convergence of the population. This strategy should however be used in conjunction with elitism, so as to avoid destroying favorable genotypes through the effects of overly enhanced mutation rates.[4] Variable mutation rate with elitism are default settings in PIKAIA.

Reproduction plans: generational replacement versus Steady-State.—The algorithm outlined at the beginning of this section does not impose a priori restrictions on how step 4 is to proceed. A *reproduction* plan is needed to control how newly bred offspring are inserted in the population. One can imagine two extreme strategies: (1) offspring are accumulated in temporary storage, and once enough offspring have been bred they replace the entire parent population; this goes under the name of *full generational replacement.* (2) Alternately, offspring can be inserted in the population as soon as they are bred. This is known as *steady state reproduction.* Under this alternate plan, one must further specify how older individuals are to be deleted from the population in order to make room for newly bred individuals. Full generational replacement is the default in PIKAIA.

1.6. *Hybrid Methods*

In choosing values for genetic algorithm parameters or adopting one or the other specific technique described above, one typically strives to maintain adequate balance between *exploration* and *exploitation.* Efficient exploration of parameter space is a defining characteristic of global optimization schemes, but this requirement often conflicts with the exploitation of the characteristics of the current best solution to produce accelerated convergence to the true optimal solution. The general shape of the "convergence curves" in Figures 3*f* and 4 is quite typical for GA-based optimizers and, more interestingly, markedly different from the corresponding curves for conjugate gradient-type techniques. The latter tend to be characterized by slow convergence in the first few iterations, followed by rapid convergence (often at a well-defined rate) once in the vicinity of the nearest extremum. This is because conjugate gradient-type methods are often specifically designed to efficiently make use of local and prior information (values of local derivatives, "memory" of previous conjugate directions in parameter space, etc.. . .). Genetic algorithms, on the other hand, converge rather slowly near extrema, since they rely primarily on mutation to generate small incremental changes in the population. But the crossover operation, acting on a diverse gene pool, constantly produces large jumps through parameter space (cf. Fig. 2), And so favors exploration.[5] This suggests that if high accuracy is required, it would likely be advantageous to use a GA solution as a starting guess for more standard schemes such as conjugate gradient methods. Alternately, one can incorporate hill climbing–like capabilities *within* the genetic algorithm. The resulting *hybrid algorithms* combine the best of both classes of techniques, namely, the good exploratory capabilities of GAs with the rapid convergence of more conventional techniques in the vicinity of extrema.

1.7. *Defining Fitness*

In the model problem that has been the focus of attention thus far, there is a direct correspondence between "altitude," as returned by the evaluation of $f(x, y)$, and "fitness," as implemented in the genetic algorithm. But the correspondence need not be that direct. Consider a standard fitting problem, where one is given a discrete set of N data points $[x_i, y_i]$ with associated measurement errors $\sigma_i \equiv \sigma(y_i)$, and is asked to construct the best possible fit to these data using a specific functional form for the fitting function, e.g., $y(x) = mx + b$ for a

[4] A related technique known as dynamical or *nonuniform mutation* (Michalewicz 1994, § 6.2) consists in using mutation rates that vary in time *and* as a function of position along the chromosome, so that genes corresponding to least significant digits have higher mutation rates than those mapping onto the most significant digits. Efficient use of this technique, however, requires the introduction and adjustment of some additional parameters.

[5] That crossover achieves this exploratory effect is a central aspect of genetic algorithm theory, as embodied in the *schemata theorem.* See chap. 2 of Goldberg (1989) for further discussion.

linear fit. How well the linear function generated by a given $[m, b]$ pair fits the data can be quantified via the χ^2 merit function:

$$\chi^2(m, b) = \sum_{i=1}^{N} \left[\frac{y(x_i; m, b) - y_i}{\sigma_i} \right]^2 . \tag{2}$$

Calculating the χ^2 for all possible values of $[m, b]$ defines a "landscape" in the two-dimensional $[m, b]$ parameter space. The task becomes one of *minimizing* χ^2, but the procedure is conceptually identical to the maximization problem considered above.

The following sections illustrate the use of genetic algorithms in treating three real astronomical and astrophysical applications. The purpose is *not* to provide a technical discussion of the application of genetic algorithms to this or that specific problem, nor to carry out systematic comparisons of their performance against that of conventional methods. Instead, the aim is simply to demonstrate, through examples, the power and versatility (as well as the shortcomings) of genetic algorithms as a class of robust, general-purpose optimization techniques. The following three problems, as well as the model problem of this section, were all treated with the GA-based optimization subroutine PIKAIA listed in the Appendix.

2. FITTING ROTATION CURVES OF GALAXIES

2.1. *Problem Statement*

The *rotation curve* of a galaxy refers to the variations of the orbital velocity of stars and gas about the galactic center with galactocentric distance r in the galactic plane. The shape of a galaxy's rotation curve is usually interpreted in terms of an equilibrium between gravitational and centrifugal accelerations. If this is the case, then the rotational velocity v at galactocentric distance r is related to the gravitational potential $\Phi(r)$ by the relation

$$v^2(r) = r \frac{\partial \Phi}{\partial r} , \tag{3}$$

where $\Phi(r)$ is the gravitational potential of all mass located within a sphere of radius r. The rotation curves of galaxies thus provide, in principle, the means to determine their total mass, *including mass that is not directly detectable optically.* Early studies of galactic rotation curves immediately suggested the presence of "missing mass" or "dark matter" surrounding most galaxies, and extending to much larger galactocentric distances than the optically visible mass components (Faber & Gallaher 1979). Four distinct mass contributions are usually taken into account in modeling galactic rotation curves: (*a*) a *bulge* component, sometimes weak or even insignificant but in other cases dominating the mass distribution near galactic center, (*b*) a *disk* component, usually dominating the rotation curve throughout the optically visible portion of galaxies, (*c*) an interstellar gas component, primarily neutral hydrogen, and (*d*) a *halo* component composed of dark matter. In the nonrelativistic limit the total gravitational potential can then be written as

$$\Phi(r) = \Phi_B(r) + \Phi_D(r) + \Phi_G(r) + \Phi_H(r) . \tag{4}$$

In view of equation (3), this linear addition implies that the corresponding contributions to the total velocity curve add quadratically:

$$v^2(r) = v_D^2(r) + v_B^2(r) + v_H^2(r) + v_G^2(r) . \tag{5}$$

In practice, the *brightness* profiles for the bulge and disk are obtained observationally. The conversion to mass (and thus gravitational potential and then velocities) requires a knowledge of the mass-to-light ratios (M/L) for the stellar populations making up the disk and bulge. While in principle these ratios can be estimated if the age and mass functions of the populations are known, in practice they are both treated as adjustable parameters in the fitting procedure. Considerable uncertainties exist as to the functional form of the halo's mass distribution and corresponding contribution to the total velocity. The following two-parameter form, used by Kent (1986), is amply adequate here:

$$v_H^2(r) = 2\sigma^2 \left[1 - \left(\frac{r}{\alpha} \right) \tan^{-1} \left(\frac{\alpha}{r} \right) \right] , \tag{6}$$

but other prescriptions are possible (see, e.g., Carignan & Freeman 1985). Here σ is a velocity dispersion, and α a characteristic length scale. The contribution from neutral hydrogen (H I) is often measured directly at radio wavelengths (see, e.g., Carignan et al. 1990). The fitting of rotation curves is thus, in general, a four-parameter fitting problem for which genetic solutions can be obtained. The fitting is nonlinear, in the sense that the rotation curve is a nonlinear function of the fitting parameters; equation (6), defining the halo contribution in terms of α and σ, is a nonlinear function of α and is added (quadratically) to other contributions in order to produce the phenotype (eq. [5]) whose fitness is then evaluated in terms of its closeness to the data, as quantified by the χ^2.

A final point of importance concerns the allowed ranges for the fitting parameters. For the fitting problem under consideration, a minimal constraint is to require that the fitting parameters be positive quantities. But there are *physical* constraints that can (and often must) be taken into account. For example, a ratio $M/L_D \ll 1$ would imply that the stellar population making up the disk is of a rather peculiar sort. Unless there is strong evidence to the contrary, one may argue that a constraint of the form $1 \lesssim M/L_D \lesssim 5$ (say) should be introduced. The "maximum disk" procedure (e.g., Carignan & Freeman 1985) consists in pushing M/L_D to the largest possible value consistent with a reasonable fit of the rotation curve close to galactic center. As the following example demonstrates, such additional constraints may override choices of "best" fit based solely on minimal χ^2 values.

2.2. *A Fit to the Rotation Curve of NCG 6946*

NGC 6946 is a well studied late-type (Scd) spiral galaxy, having an extended rotation curve with a well defined flat outer portion, effectively no bulge, and a significant H I component (Carignan et al. 1990). The rotational data and the contributions from the various visible components are shown on Figure 5. Because there is no bulge contribution, the fitting of the rotation curve reduces to a three (instead of four) parameter fit-

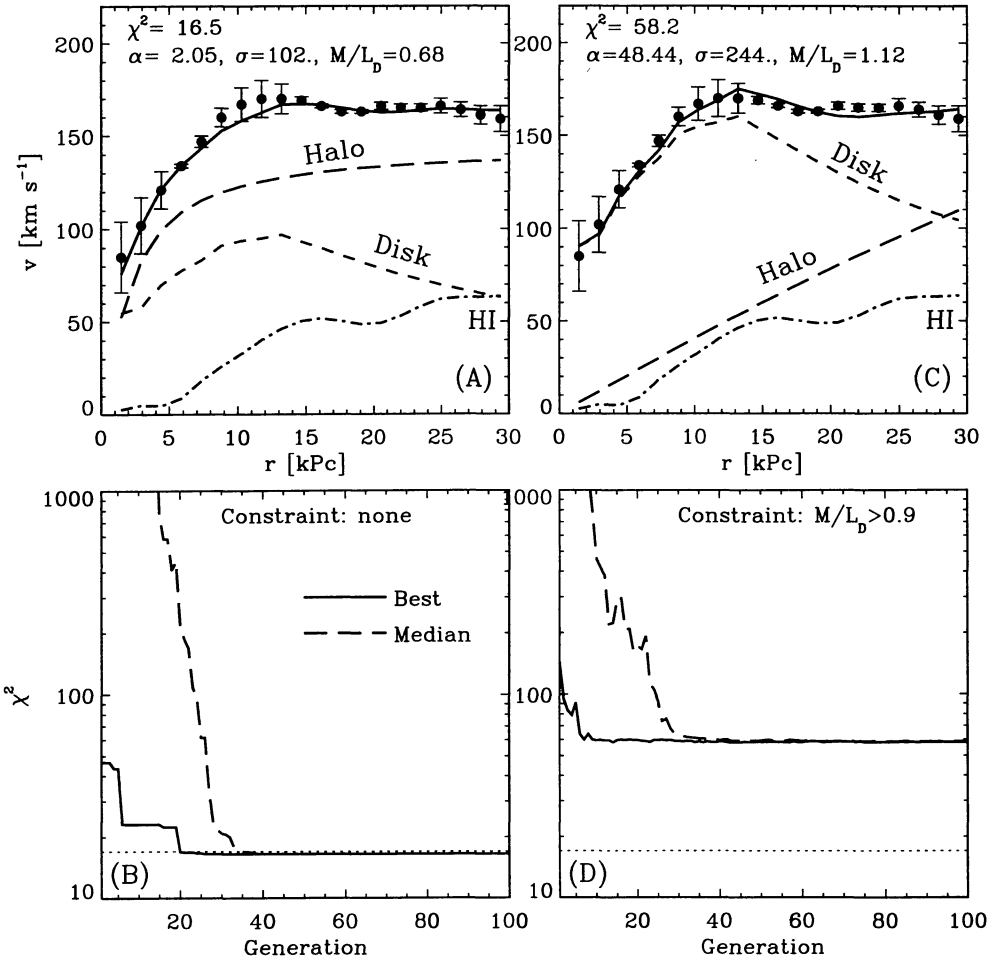


FIG. 5.—Two genetic solutions to the rotation curve fitting problem. (*a*, *c*) The optimal solutions on which the algorithm has converged after 100 generations. The filled circles are the data, and the solid line the fits evolved genetically. The corresponding halo, H I, disk, and halo contributions are also plotted. Fit (*a*) is unconstrained, while for fit (*c*) the restriction $M/L_D > 0.9$ has been enforced. (*b*, *d*) the corresponding χ^2 evolution for the fittest individual of each generation (*solid line*), and median individual (*dashed line*). The dotted line is the number of degrees of freedom present in the fit. While in terms of χ^2 fit (*a*) is clearly superior to (*c*), the latter is to be preferred on physical grounds (see text).

ting problem for M/L_D, the mass-to-light ratio of the disk, and the two halo parameters σ and α.

Figure 5 shows two genetic solutions obtained by evolving a population of 100 individuals over 100 generations, using crossover and mutation rates of 0.65 and 0.003, respectively. These two solutions differ only in the allowed M/L_D range; except for the basic positivity requirement, the fit shown in Figure 5*a* is otherwise unconstrained. The fit in Figure 5*c*, on the other hand, was obtained under the constraint that $M/L_D \geq 0.9$, in the spirit of the maximum disk procedure. The final fitting parameters are $(M/L_D, \alpha, \sigma) = (0.68, 2.05, 102.)$, yielding $\chi^2 = 16.5$ for fit (Fig. 5*a*), and $(M/L_D, \alpha, \sigma) = (1.12, 48.4, 244.)$, yielding $\chi^2 = 58.2$ for (Fig. 5*c*).

With an average rms deviation of ~ 5 km s^{-1} for velocities of ~ 200 km s^{-1}, both these fits "look good." Estimating goodness of fit via "χ-by-eye," however, is a guaranteed recipe for disaster. Among the various deterministic criteria available (see, e.g., Press et al. 1992, chap. 15) a particularly simple one—and one sufficient for the present purpose—consists in comparing the χ^2 of the fit to the number of available degrees of freedom in the fitting procedure, defined simply as

$$\nu = N - M, \qquad (7)$$

where N is the number of data points to be fitted and M the number of parameters involved in the fit. If $\chi^2 \sim \nu$, one has a moderately good fit. $\chi^2 > \nu$ indicates that either (1) the error estimates σ_i are larger than assumed and/or the errors are not normally distributed or (2) there is something wrong and/or incomplete about the model that is being used to fit the data. For the fitting problem of Figure 5 one has $N = 20$ and $M = 3$, so that $\nu = 17$, corresponding to the horizontal dotted lines on Figures 5*b* and 5*d*. In this light, fit (Fig. 5*a*) is clearly a better fit than (Fig. 5*c*). Or is it? Should one be fooled in accepting fit (Fig. 5*a*) and rejecting fit (Fig. 5*c*) solely on the basis of χ^2 considerations? Probably not. There are no compelling reasons to believe that the mass distribution of the halo should strictly obey equation (6) (or be any absolutely smooth function of r, for that matter!). It may well be that both fits on Figure 5 are good enough after all. So which one is to be preferred?

The existence of sets of widely different fitting parameters yielding fits of similar quality is actually a common occurrence in fitting rotation curves of galaxies. A casual exploration of parameter space reveals that many secondary minima are present, but this difficulty is compounded by the fact that the minima are clustered in a long, flat-bottomed valley with very little "contrast" between neighboring local minima (see, e.g., Fig. 8 of Carignan et al. 1990). This situation arises when the rotation curve does not extend very far beyond the peak in the disk's light curve, a common state of affairs in this business (see Kent 1986). There exists then a series of tradeoffs between M/L_D, α, and σ that yield very similar rotation curves, as exemplified on Figure 5. This is no moot point, as the physical parameters defining the various "acceptable" fits are markedly different and lead to very different pictures of the mass budget among the various components, and so to very different estimates for the mass of galactic dark matter. Only if the rotation curve extends well beyond the peak in the light curve can the contributions from disk and halo be unambiguously separated. Otherwise various physically based arguments, such as the maximum disk conjecture, must be put forth to pick "the" solution among the subset of low-χ^2 solutions (see discussions in Kent 1986 and Carignan et al. 1990). For NGC 6946, $M/L_D = 0.68$ is likely unreasonably low, so that fit (Fig. 5*b*) is to be preferred; fitting rotation curves of galaxies is just trickier than simply finding the lowest possible χ^2.

With or without additional constraints, genetic algorithms do find the *absolute* minimum in parameter space without difficulty, even though the fitting landscape is rather ill behaved; more interestingly, this problem illustrates another important operational advantage of GA-based optimization, namely, the easy handling of constraints. Many constraints can be incorporated directly in the encoding procedure. For the solution of Figure 5 (C/D), this simply involved writing $M/L_D = 0.9 + x$, and encoding x (≥ 0) in the usual way (cf. Fig. 2), a procedure that admittedly could be applied just as well to other methods. But the idea of "hardwiring" constraints directly at the encoding/decoding level is readily generalized to other classes of constraints that are much harder to accommodate within conventional optimization methods (for some specific examples, see chap. 7 of Michalewicz 1994).

3. FITTING A MULTIPLY PERIODIC SIGNAL WITH NOISE AND DATA GAPS

Fitting time-series data is perhaps one of the most common tasks in observational astronomy. Simple examples include the computation of orbital elements for spectroscopic binaries, period determination in pulsating stars, photometric determination of the rotation periods of late-type stars, and so on. The growing field of asteroseismology (see Brown & Gilliland 1994) is producing an ever-increasing set of fitting problems involving multiply periodic signals. The fitting of such signals using genetic algorithms is the topic of this section.

3.1. *Problem Statement and a Synthetic Example*

Consider a multiply-periodic signal defined by

$$f(t) = A_0 + \sum_{k=1}^{N} A_k \sin\left(\frac{2\pi t}{P_k} + \varphi_k\right). \qquad (8)$$

One such function is defined by N amplitude/period/phase triplets ($[A_k, P_k, \varphi_k]$), plus the base level constant A_0, for a total of $3N + 1$ parameters. The dotted line on Figure 6*a* is the time series of total length $\Delta t = 240$ resulting from setting $N = 5$ in equation (7), with $A_0 = 5$, $\{A_k\} = \{0.8, 0.75, 0.5, 0.4, 0.85\}$, $\{P_k\} = \{9.6, 12, 18, 21.6, 30\}$, and $\{\varphi_k\} = \{0.4, 0.15, 0.1, 0.2, 0.25\} \times 2\pi$. The solid dots are "data" generated by sampling the dotted curve over a period $P_S = 24$, with gaps of duration $\Delta t_g = P_S$ between successive sampling events, and with random noise at the level $\sigma = 1/\sqrt{12}$ also thrown into the signal. Such regularly spaced gaps are meant to mimic the day/night cycle plaguing all (low-latitude) single-site astronomical observations of variable stars. The fitting problem is again nonlinear, as the right-hand side of equation (8) involves nonlinear functions of some fitting parameters. The "cleaning algorithm" is a standard analysis technique for such multiply-periodic signals and proceeds as follows:

1. Compute the power spectra of the signal and identify the dominant period
2. Perform a least-squares fit to the data to obtain the amplitude and phase of the mode whose period was identified in step (1).
3. Construct the time series corresponding to that single mode, and subtract it from the original signal to obtain a new signal.
4. Repeat steps 1–3 until convinced that all that is left is noise.

Step 4, wisely deciding when to stop, is one of the aspects that make multiply-periodic signal fitting somewhat of an art. It may also happen that the *order* in which periods are succes-

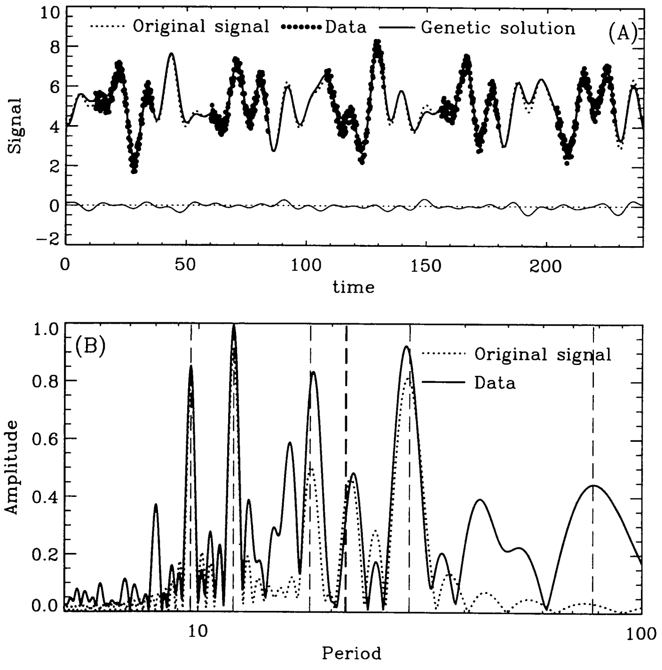


FIG. 6.—A noisy time series with data gaps. Panel (*a*) shows the time series, and panel (*b*) their power spectra. The dotted lines refer to the original signal (cf. eq. [8]), and the filled circles synthetic "data" generated from the original signal. The thin solid line at the bottom of the panel is $f - f^*$, i.e., the difference between the original signal and the fitted signal. The thick solid line in (*a*) is a genetic fit to the data, and the vertical dashed lines in (*b*) indicate the periods "found" by the genetic algorithm.

sively extracted from the signal ends up affecting the final set of periods and amplitudes fitted to the signal.

The thick solid line on Figure 6*a* is a genetic solution f^* to the fitting problem defined by the "data," and the thin solid line the difference between the original signal and the fitted signal. This solution was obtained by evolving 200 individuals over 200 generations under full generational replacement, using a variable mutation rate with elitism. With $\chi^2 = 812$, this is a decent fit, as $\nu = 750-22 = 728$ here, and 200 generations being a rather short evolutionary run. In fact, the original signal used to construct the noisy "data" of Figure 6 returns $\chi^2 = 778$, hardly better than the fit! The genetic solution was set up so that *seven* periods be sought (although $N = 5$ in the synthetic signal, but one is not supposed to know that). The periods found by the algorithm are $\{P_k^*\} = \{9.5998, 12.013, 17.879, 21.480, 21.588, 30.012, 77.674\}$, with corresponding amplitudes $\{A_k^*\} = \{0.6997, 0.8999, 0.4589, 0.2869, 0.1611, 0.9453, 0.0519\}$. Note how one of the original periods ($P_4 = 21.6$) has been "divided" into two contributions ($P_4^* = 21.480$, $P_5^* = 21.588$), with $A_4^* + A_5^* \simeq A_4$. The genetic solution has also "found" a period $P_7^* = 77.674$ absent in the original signal, but with an amplitude $A_7^* = 0.0519$ this hardly contributes to the total signal and is most likely a leftover of an earlier evolutionary phase.

Although one may rejoice in the fact that all periods present in the original signal have been properly detected and fitted by the genetic solution, readers seasoned in the art of time series analysis will not be overly impressed, and rightfully so; despite noise and data gaps, the power spectrum of the data (solid line in Fig. 6*b*) still carries rather clearly the signature of the five periods present in the original signal, whose power spectrum is shown as a dotted line on Figure 6*b*. Although some of the peaks in the data power spectrum exceed in amplitude two of the components of the original signal, all five true periods remain clearly delineated. In other words, fitting the data of Figure 6*a* is a fairly straightforward task for the standard cleaning algorithm.

It is certainly reassuring that the genetic algorithm does as well as more conventional techniques on easy problems, and noteworthy that the genetic fit proceeds completely autonomously (i.e., without outside intervention on the part of the modeler); but can genetic techniques do significantly *better* on hard problems? More difficult signals, for example signals where the sidelobes of one strong period completely wash out neighboring periods of small amplitudes, are not so well fitted by the standard genetic algorithm (although they can also be quite difficult to fit with the cleaning algorithm). More worrisome, in many trial runs on harder synthetic problems the genetic algorithm often misses completely small amplitude, short-period components that, disturbingly, stand out quite clearly in the corresponding power spectrum. There is indeed very important information in power spectra, but unlike the cleaning algorithm, direct genetic fitting does not know about Fourier space. One potential solution lies with *hybrid algorithms*, i.e., algorithms that incorporate the standard genetic algorithm *within* the cleaning algorithm (for example). Another possibility is to use a standard genetic algorithm, but incorporate a goodness-of-fit measure for the power spectrum (i.e., do χ^2 minimization in Fourier space and in real space simultaneously).

3.2. *Fitting Pulsation Periods in δ Scuti Stars*

The δ Scuti stars are Population I post–main-sequence objects, showing complex multiply periodic brightness variations, typically in the 10–100 millimagnitude range (Matthews 1993). Their position in the H-R diagram coincides with the low-luminosity extension of the Cepheid instability strip, which suggests that the κ-mechanism is responsible for the driving of the pulsations. Observationally, pulsations show up photometrically and/or as Doppler shifts in strong spectral lines. With multiple periods in the range 0.5–6 hr, δ Scuti are particularly sensitive to the day/night cycle problems discussed above (compare the "data" of Fig. 6*a* herein to Fig. 1 in Matthews 1993).

The Hyades star θ^2 Tau is a particularly well-observed δ Scuti star. Figure 7*a* shows Doppler velocity measurements determined from multisite observations of θ^2 Tau, spanning 4 days and obtained as part of the MUSICOS campaign (Catala et al. 1993). Figure 7*b* shows the corresponding power spectrum. It is a mess. A superposition of closely spaced modes likely gives rise to the complex structure at $P \sim 2$ hr, and hints of low-amplitude signals at $P \simeq 0.9$, 4, and possibly 1.2 hr are also present. Using an implementation of the cleaning algorithm due to Breger (1980), Kennelly et al. (1995) find in these data the following set of periods and amplitudes:

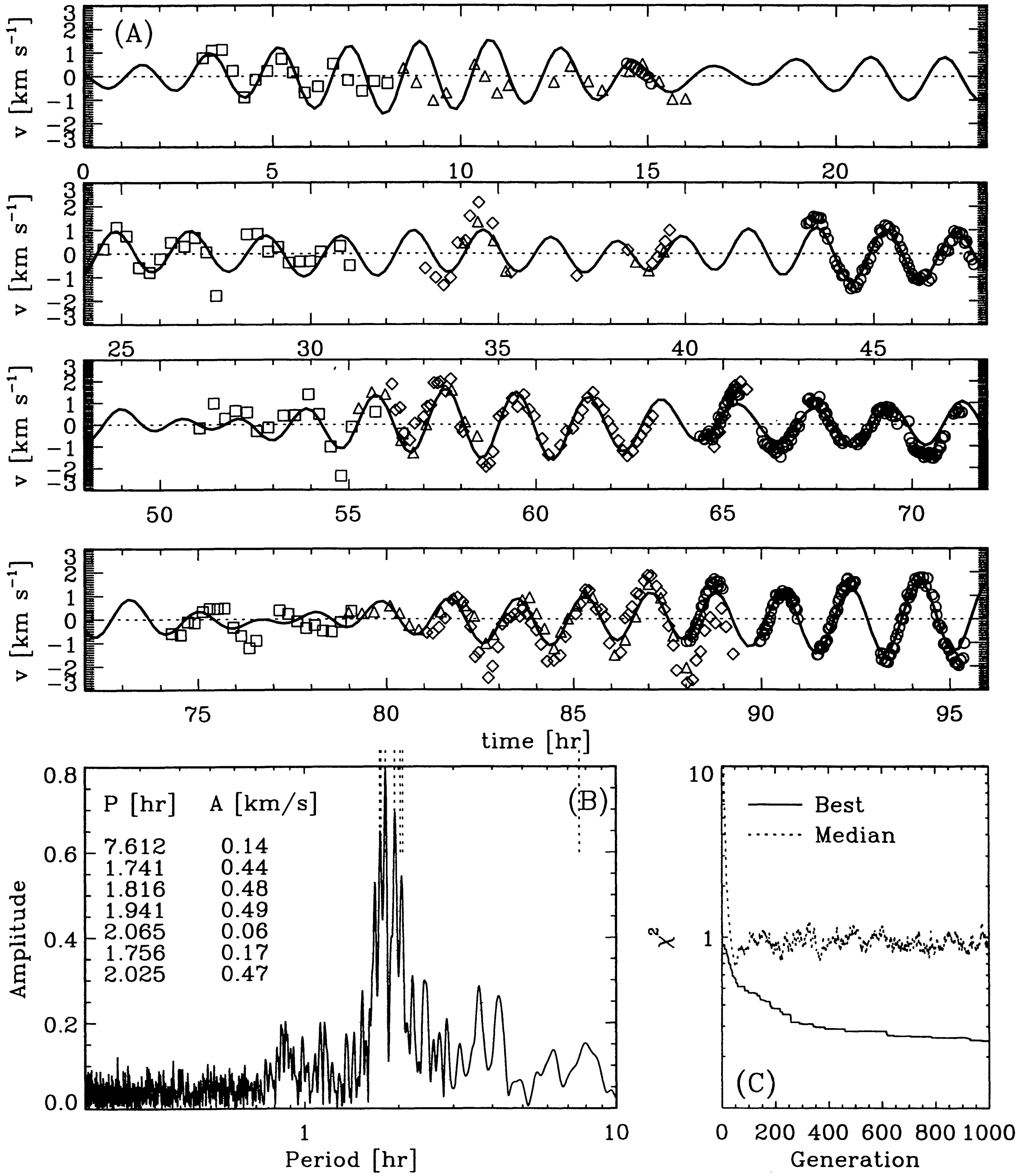
$$P_k = \{2.042, 1.948, 1.811, 1.751, 1.675\} \quad \text{hr}\,, \qquad (9a)$$

$$A_k = \{0.50, 0.62, 0.48, 0.50, 0.21\} \quad \text{km s}^{-1}\,, \qquad (9b)$$

with, however, low confidence level for the actual existence of P_5 (E. J. Kennelly, personal communication).

The solid line on Figure 7*a* is a genetic fit, obtained by trying to fit *seven* sinusoidal components to the data. In obtaining this fit, four out of the seven available periods were initialized to {0.870, 1.120, 1.820, 1.950} hr in half the individuals of the initial (no longer completely) random population. This was based on a casual inspection of the data power spectrum, and represents one particularly simple way in which information obtained from the power spectrum can be introduced in the genetic algorithm. Except for this initialization, the genetic fit is a standard one, evolving 200 individuals over 1000 generations under moderate selection pressure and with a variable mutation rate. Figure 7*c* shows the evolution of a normalized χ^2 (the data came without error estimates, so that σ was assigned an arbitrary value of 1 for all data points). Note how the χ^2 of the median individual (*dotted line*) remains approximately constant beyond the 100th generation, while the χ^2 for the fittest individual (*solid line*) continues to decrease. This is due to a gradual increase in mutation rate beginning at about the 50th generation. High variability is then maintained in the gene pool, which, in combination with elitism, allows favorable mutations to appear at a relatively high rate and become fixed in the "top end" of the population. After 10^3 generations, the genetic solution is characterized by the following set of periods and amplitudes:

$$P_k^* = \{7.612, 2.065, 2.025, 1.941, \\ 1.816, 1.756, 1.741\} \quad \text{hr}\,, \qquad (10a)$$



P [hr]	A [km/s]
7.612	0.14
1.741	0.44
1.816	0.48
1.941	0.49
2.065	0.06
1.756	0.17
2.025	0.47

FIG. 7.—Genetic fit to the velocity variations observed in the δ Scuti star θ^2 Tau. (*a*) Data obtained as part of the second MUSICOS campaign (*circles:* Kitt Peak/McMath telescope, USA; *triangles:* Observatoire de Haute-Provence, France; *diamonds:* William Herschel telescope, Canary Islands; *squares:* Xinlong Observatory, China). The zero point of the timescale corresponds to JD8964.0. The solid line is a 10^3 generations, 200 individuals genetic fit. (*b*) The data power spectrum, together with a listing of periods and amplitudes found by the genetic algorithm. The dotted line segments indicate the positions of the periods. (*c*) The evolution of the (normalized) χ^2 for the fittest and median individuals in the population.

$$A_k^* = \{0.14, 0.06, 0.47, 0.49, 0.48, 0.17, 0.44\} \quad \mathrm{km\ s^{-1}} . \qquad (10b)$$

Note how two of the periods artificially introduced in the initial population have disappeared, while the other two have been slightly altered. Identifying $P_k^{**} = \{P_3^*, P_4^*, P_5^*, P_7^*\}$ as the relevant subset of periods, the genetic solution is compatible with the first four modes found by Kennelly et al. (1995) within an accuracy

$$\Delta P_k = \{-1.1, 0.36, -0.28, 0.57\} \quad (\%) , \qquad (11a)$$

$$\Delta A_k = \{6.0, 21, 0.0, 12\} \quad (\%) . \qquad (11b)$$

This represents a rather impressive agreement. What is perhaps equally noteworthy is that the fit "evolved" in a completely autonomous fashion, without human intervention.

4. MAGNETOHYDRODYNAMIC WIND MODELS WITH MULTIPLE CRITICAL POINTS

It was claimed in the introductory paragraphs of this paper that a large number of problems can be cast in terms of minimization problems even though they may not look like classical "fitting problems," such as the ones dealt with in §§ 2 and 3; this third and final example is chosen to illustrate precisely this point.

4.1. *Problem Statement*

The existence of a continuous, more or less radial and steady corpuscular outflow (or *wind*) emanating from the Sun results from the impossibility of maintaining the hot ($T \sim 10^6$ K) solar coronal gas in hydrostatic equilibrium in the Sun's gravitational field (see, e.g., Parker 1963, Chap. 10). The in situ detection of the solar wind by early Earth-orbiting satellites was a truly spectacular confirmation of the wind model earlier developed by Parker (1958). The early solar wind models were strictly hydrodynamical, in the sense that they neglected the dynamical effects of magnetic fields. These were first incorporated into hydrodynamic wind models by Weber & Davis (1967; see also Belcher & MacGregor 1976). Although it involves some rather restrictive assumptions concerning the assumed geometry of the magnetic field, the Weber-Davis model applied to the solar case yields flow speeds and densities in good agreement with observations at 1 AU (Pizzo et al. 1983), and remains to this day the primary (deterministic) model used to estimate quantities such as mass and angular momentum loss from rotating, magnetized stars (see, e.g., Charbonneau, Schrijver, & MacGregor 1995, § 4).

A detailed description of the mathematical formulation of the MHD wind problem would entail an overly lengthy digression, and so is only outlined here. The solutions are constructed in the framework of single fluid magnetohydrodynamics. Adopting spherical polar coordinates (r, θ, ϕ), the solutions sought are steady ($\partial / \partial t = 0$), axisymmetric ($\partial / \partial \phi = 0$) and restricted to the equatorial plane ($\partial / \partial \theta = 0$ and $v_\theta = 0$). Quantities such as the rotation rate Ω, the base temperature, density, and radial magnetic field strength (T_0, ρ_0, and B_{r0}) are assumed known at some fiducial reference radius r_0 ($\gtrsim R_\odot$, typically), and the solution is to be constructed in $r > r_0$. The interested reader is referred to Weber & Davis (1967) and Belcher & MacGregor (1976) for further discussion and for a listing of the basic governing equations of single-fluid magnetohydrodynamics. Belcher & MacGregor show that the r-component of the equation of motion can be manipulated into the form:

$$\frac{\partial v_r}{\partial r} = \left(\frac{v_r}{r}\right)\left[\frac{(v_r^2 - A_r^2)(2c_s^2 + v_\phi^2 - GM_\odot/r) + 2v_r v_\phi A_r A_\phi}{(v_r^2 - A_r^2)(v_r^2 - c_s^2) - v_r^2 A_\phi^2}\right] , \qquad (12)$$

where $c_s^2 = \alpha p / \rho$ is the polytropic sound speed with α the polytropic index, and $A_{r(\phi)} = B_{r(\phi)} / (4\pi\rho)^{1/2}$ are Alfvén speeds. Equation (12) is basically the magnetohydrodynamic equivalent of $F = ma$, i.e., a statement of momentum conservation governing the dynamics of the flow. This expression can be integrated once to yield

$$E(r, v_r) = \frac{1}{2}(v_r^2 + v_\phi^2) - \frac{GM_\odot}{r} + \frac{c_{s0}^2}{\alpha - 1}\left(\frac{\rho}{\rho_0}\right)^{\alpha - 1} - \frac{(\Omega r) A_r A_\phi}{v_r} , \qquad (13)$$

where $E(r, v_r)$ is a constant, corresponding to the total energy per unit mass in the flow, so that equation (13) simply expresses energy conservation. The right-hand side of equation (12) looks like it may diverge if there exists values of v_r for which the denominator vanishes. This (unfortunately) does occur, whenever v_r becomes equal to the phase speed of either the fast or slow magnetosonic wave mode, corresponding to two distinct ways in which information can be transmitted in a magnetized, electrically conducting, compressible fluid. Denote the corresponding radial position by $r_f$and r_s. Now, divergence of the right-hand side means $\partial v_r / \partial r \rightarrow \infty$, implying infinite accelerations and other related nastiness which are best avoided. There is one avenue open to save the day, namely ensuring that the *numerator* on the right-hand side of equation (12) also vanishes at r_s and r_f(the corresponding pairs of positions and fluid velocities, $[r_s, v_{rs}]$ and $[r_f, v_{rf}]$, are *critical points* of the governing differential equation). Denoting by N and D the numerator and denominator within the square brackets on the right-hand side of equation (12), one then requires that

$$N(r_f, v_f) = 0 , \qquad (14a)$$

$$D(r_f, v_f) = 0 , \qquad (14b)$$

$$N(r_s, v_s) = 0 , \qquad (14c)$$

$$D(r_s, v_s) = 0 , \qquad (14d)$$

complemented by the constraints

$$E(r_f, v_f) = E(v_{r0}, r_0) , \qquad (14e)$$

$$E(r_s, v_s) = E(v_{r0}, r_0) , \qquad (14f)$$

so that the solution conserves energy. Equations (14a)–(14f) must be solved simultaneously for the elements of the solution 6-vector

$$\boldsymbol{w} = (v_{r0}, v_{\phi 0}, r_s, v_{rs}, r_f, v_{rf}) . \qquad (15)$$

These six quantities turn out to fully determine the wind solution, as shown by Belcher & MacGregor (1976).

4.2. *Recasting the Solution Procedure as a Minimization Problem*

Equations (14) define a six-dimensional nonlinear root finding problem for a set of six coupled equations. Unfortunately, there are just no robust, efficient algorithms to solve this kind of root finding problem. Section 9.6 of Press et al. (1992) contains a brief, yet accessible and insightful discussion of why this is the case. Belcher & MacGregor (1976) actually tackle the problem directly, using a conjugate gradient method. Not surprisingly, the technique converges well as long as a *very* good starting guess can be provided. In general, sufficiently good guesses can only be made in some very specific asymptotic limits. To construct a wind solution for a strongly magnetized rapidly rotating Sun (say) starting from a (known) solar-type solution, Belcher & MacGregor use a continuation scheme whereby one of the solution's input parameters (e.g., the rotation rate) is increased by a small amount, a new wind solution computed and used as a starting guess after the input parameter is further incremented, and so on, until the desired value for the input parameter is attained. The procedure is then repeated for the other input parameters (e.g., magnetic field strength), as needed. For small enough increments the procedure works quite well and can be automated to a large degree. But it becomes rather time consuming if large distances need be traveled through parameter space, and requires some degree of premeditation on the part of the modeler. Clearly, a more robust and efficient technique would be preferable.

An alternate approach consists in converting the root finding problem defined by equations (14a)–(14f) to a minimization problem. This begins by defining a function

$$f(\boldsymbol{w}) = \{N^2(r_f, v_f) + D^2(r_f, v_f) + N^2(r_s, v_s) + D^2(r_s, v_s) + [E(r_f, v_f) - E(v_{r0}, r_0)]^2 + [E(r_s, v_s) - E(v_{r0}, r_0)]^2\} , \qquad (16)$$

so that the root finding problem is now simply

$$f(\boldsymbol{w}) = 0 , \qquad (17)$$

which already looks friendlier, although it must be kept in mind that one has *six* quantities to adjust, as opposed to one, in order to find the root. Clearly any root (a 6-vector $\boldsymbol{w}_R$ where $f(\boldsymbol{w}_R) = 0$) is also a minimum since $f(\boldsymbol{w}) > 0$ by construction. The root finding problem can be then cast as a *minimization problem* for a single, strongly nonlinear multidimensional function, which is usually easier than a root finding problem for a set of coupled nonlinear equations. However, while all roots of equations (14a)–(14f) will correspond to a minimum (in fact, a zero) of $f(\boldsymbol{w})$, not all minima of $f(\boldsymbol{w})$ should be expected to be roots of equations (14a)–(14f). Furthermore, it is generally the case that $f(\boldsymbol{w})$ has *a lot* more secondary minima than equations (14a)–(14f) have solutions (simultaneous roots). One is facing a familiar difficulty (cf. the model problem of § 1); to avoid getting "stuck" in a minimum that is not a zero one will want to either (1) have a pretty good starting guess for $\boldsymbol{w}$, in which case one may as well use Belcher & MacGregor's original method, or (2) have available a scheme that can efficiently explore parameter space and climb out of secondary minima . . . enter genetic algorithms!

4.3. *A Genetic Algorithm Solution*

The construction of the function to be minimized is clearly not unique, and other options than the quadratic sum of equations (16) may yield better results. A series of experiments indicate that in the present context the following is preferable to equation (16):

$$f(\boldsymbol{w}) = [|N(r_f, v_f)|^{1/2} + |D(r_f, v_f)|^{1/2} + |N(r_s, v_s)|^{1/2} + |D(r_s, v_s)|^{1/2} + |E(r_f, v_f) - E(v_{r0}, r_0)|^{1/2} + |E(r_s, v_s) - E(v_{r0}, r_0)|^{1/2}] . \qquad (18)$$

This function is also positive definite by construction, and the exponent 1/2 (instead of 2 in eq. [16]) helps to maintain a better balance between the six distinct contributions on the right-hand side as the solution evolves. Figure 8*a* shows the evolution of the elements of the solution 6-vector. This genetic solution was obtained under full generational replacement, with variable mutation rate and elitism. After 500 generations, the fittest of 100 individuals has a phenotype

$$\boldsymbol{w}^*_\odot = \{0.0122779, 0.0140085, 6.59223, 0.668323, 31.5335, 1.34764\} , \qquad (19)$$

where lengths (r_s and r_f) are normalized to the reference radius r_0, and velocities (v_{r0}, $v_{\phi 0}$, v_{rs}, and v_{rf}) to the base sound speed c_{s0}. For comparison, the corresponding values obtained using the method of Belcher & MacGregor are

$$\boldsymbol{w}_\odot = \{0.0122947, 0.0140226, 6.60039, 0.676146, 29.5224, 1.37785\} \qquad (20)$$

and are shown as horizontal line segments on Figure 8*a*. The numbers in square brackets are the percent difference between the two solutions. Figure 8*b* shows the evolution of $f(\boldsymbol{w}^*_\odot)$ for the best and median individual. As on Figure 7*c*, the lack of convergence of the population as a whole, revealed by the fact that the "median" curve never joins the "best" curve, can be traced to the gradually increasing mutation as the genetic solution evolves.

It may appear surprising that relatively good accuracy ($\sim$0.1%) could be obtained for three elements of the solution 6-vector, while the remaining three ($w^*_4 \equiv v_{rs}$, $w^*_5 \equiv r_f$, $w^*_6 \equiv v_{rf}$) are markedly less accurate (1%–10%). This can be traced to the lack of contrast in the $[v_{rs}, r_f, v_{rf}]$ hyperplane of parameter space for this solution. This situation can be corrected by

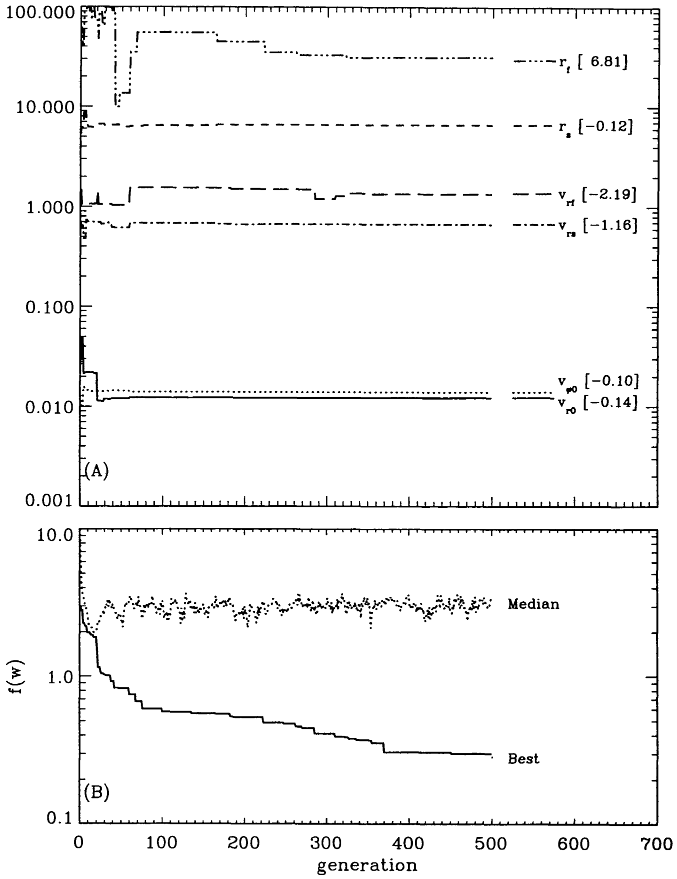


FIG. 8.—A genetic solution to the Weber-Davis MHD wind problem. (*a*) The evolution of the six elements of the solution six-vector (cf. eq. [15]) over 500 generations for a population of 100 individuals. The horizontal line segments on the right indicate the values obtained using the conjugate gradient method, iterating to a tolerance 10^{-8}. The numbers in square brackets are the percent difference between the genetic and conjugate gradient solutions. (*b*) The evolution of the function value $f(\boldsymbol{w})$ for the best and median individual.

introducing weights to the six contributions to $f(\boldsymbol{w})$, for example by replacing equation (18) by

$$f(\boldsymbol{w}) = [a_1|N(r_f, v_f)|^{1/2} + a_2|D(r_f, v_f)|^{1/2} + a_3|N(r_s, v_s)|^{1/2} + a_4|D(r_s, v_s)|^{1/2} + a_5|E(r_f, v_f) - E(v_{r0}, r_0)|^{1/2} + a_6|E(r_s, v_s) - E(v_{r0}, r_0)|^{1/2}], \quad (21)$$

where the weights $a_1, \ldots, a_6$ are adjusted so that proper contrast is maintained along all dimensions of parameter space. This trick works, but in practice lacks robustness, in the sense that solutions for other input parameters (e.g., a "young Sun" rotating at 25 times its present rate) require a different set of a_k values for comparable accuracy. This is yet another situation where a hybrid algorithm would appear optimal. This may involve, for example, running a genetic algorithm over a few hundred generations, and then using the resulting best phenotype to initialize a conventional conjugate gradient run.

Simulated Annealing (specifically, routine `amebsa` of Press et al. 1992) failed repeatedly on the six-dimensional minimization problem defined by equation (18). However, this may reflect nothing more than my own lack of experience with this class of techniques, in particular in the design of suitable annealing schedules.

5. CONCLUSION: GENETIC ALGORITHMS IN ASTRONOMY AND ASTROPHYSICS

In hailing the power and virtues of genetic algorithms, the discussion in the preceeding sections has focused primarily on aspects related to robustness and versatility. There exists, in addition, a number of operational advantages to GA-based optimizers, as compared to conventional optimization methods. In their standard incarnation, genetic algorithms are easy to code and modify. From the model problem of § 1.4 to the Weber-Davis MHD wind problem of § 4, all genetic solutions presented in this paper were obtained using the same basic code, with a single subroutine being replaced in going from one problem to the next. While GA-based optimization can be CPU intensive, the memory requirements are usually quite moderate, as no large matrices need to be constructed and stored. Genetic codes can thus easily run (perhaps overnight) on fairly small computers/workstations. Because they do not require the computation of derivatives with respect to modeling parameters, GA-based optimizers are essentially impervious to ill-behaved search spaces having vanishing derivatives and other similar nastiness leading to singular behavior in Jacobian and Hessian matrices; all that is required is that a relatively unambiguous measure of fitness be computable for any point in parameter space.

In practical terms, genetic algorithms can be expected to complement, rather than displace, conventional optimization techniques. The resulting hybrid algorithms can combine the best of both classes of techniques, namely, efficient and adaptive exploration of parameter space (genetic algorithms), combined with rapid convergence to high accuracy in the neighborhood of extrema (e.g., conjugate gradient methods). The problems treated in §§ 3 and 4 are cases in point. Where GA-based optimization stands with respect to other optimization schemes is not easy to ascertain as yet. Ackley (1987) applied seven global optimization methods (including GA-based optimizers, simulated annealing, and various incarnations of iterated/stochastic hill climbing) to six test problems of varying difficulty. While carefully restraining from granting superior status to one method over another, his results do confirm that GA-based optimizers are a strong and promising contender in the field of global optimization.

Like Monte Carlo methods, GAs are characterized by a very high level of intrinsic parallelism. In most applications, the two most CPU-intensive subtasks are phenotype construction and fitness evaluation. These two steps can be carried out completely independently for each individual comprising the population with, ideally, one processor assigned per individual. Communication between processors is minimal, being only required (1) for ranking individuals once fitness has been computed for each individual, and (2) at breeding, when the two parent phenotypes are used to construct the two offspring genotypes.

Again like Monte Carlo methods, GAs can be an attractive alternative for some difficult *inverse problems.* Examples of such problems abound in astronomy (see Craig & Brown 1986;

Jeffrey & Rosner 1986a), including the determination of plasma physical properties from spectroscopic emission measures, Doppler imaging, helioseismic inversion, etc. In all these cases, the inverse problem is hard and computationally intensive, while the corresponding forward problem is often a much simpler and faster task. Genetic techniques require only the solution of the forward problem for fitness evaluation, making them particularly attractive if the forward problem is *very much* easier than the original inverse problem. Constraints such as positivity, often requiring an iterative approach in conventional inversion methods, can be trivially hardwired at the encoding level, and so pose no difficulty. Hakala (1995) has successfully used GAs, in conjunction with maximum entropy constraints, to perform inverse modeling of the brightness distribution along the accretion stream of eclipsing AM Her systems. Tomczyk et al. (1995) have recently applied genetic techniques to the reconstruction of the internal solar differential rotation profile from the observed amplitudes and splittings of p-mode frequencies (Christensen-Dalsgaard, Gough & Toomre 1985; Gough & Toomre 1991). Their results, while having still much to be improved upon, nevertheless suggest that GAs can be a powerful complement to conventional helioseismic inversion methods.

I cannot help ending this discourse on a more personal note. In playing with various incarnations of genetic algorithms, I have often monitored the convergence of genetic solutions in animation form. This may involve, for example, plotting the best phenotype of successive generations along with the "data" of Figure 6*a*, and subsequently animating the resulting sequence of images. There is something deeply fascinating, if not troubling, about watching a genetic run search for, lock on, and pin down an optimal solution. This is perhaps because conceptually, genetic algorithms embody the very mechanisms that led to our own existence. Such mystical considerations notwithstanding, genetic algorithms offer an extremely robust and completely different way to do optimization. In the delightful introductory essay at the beginning of his monograph on genetic programming, Koza (1992) identifies seven basic principles of good conventional optimization techniques: correctness, consistency, justifiability, certainty, orderliness, parsimony, and decisiveness. He then goes on to argue that genetic algorithms incorporate *none* of these presumably sound principles. This may explain why, while being increasingly popular in the fields of artificial intelligence and computer-aided engineering design, genetic algorithms have only been making slow headway in other scientific fields. Yet the bottom line, if there is to be one, is that genetic algorithms *work,* and often frightfully well. If the examples and discussion herein have managed to awaken the curiosity of even only a few of my readers, then this paper has served its purpose.

I wish to thank Claude Carignan and Ted Kennelly for kindly making available to me the data shown on Figures 5 and 7. Constructive comments and suggestions by the referee, Melanie Mitchell, are also gratefully acknowledged. Special thanks are due to Barry Knapp, who put his software design expertise to the gruesome task of converting (and improving) my original opaque incarnation of the PIKAIA code to the user-friendly subroutine listed in the Appendix.

APPENDIX
A Short User's Guide to PIKAIA

A1. INTRODUCTION

The subroutine PIKAIA described in this Appendix maximizes a function $f(\boldsymbol{x})$ in the n-dimensional parameter space $\boldsymbol{x} = (x_1, x_2, . . , x_n)$. PIKAIA is meant primarily to be a learning instrument, not a production code. In most instances, where conflict arose between efficiency and clarity, the former was sacrificed for the sake of the latter.

PIKAIA adheres strictly to the ANSI FORTRAN-77 standard, with the exception of (1) the use of lowercase alphabet and (2) the use of implicit none statements. Readers having strong allergic reactions to FORTRAN and/or interested in more elaborated genetic algorithm packages may wish to take a look at the Genetic Algorithm Archive Web Page (http://www.aic.nrl.navy.mil/galist) for listings and directions to various public domain GA packages available electronically.

A2. CALLING SEQUENCE AND I/O

The calling sequence for PIKAIA is

```
call PIKAIA (funk,n,ctrl,xb,fb,status) ,
```

where funk is the name of a user-supplied external function to be maximized, and n is the parameter space dimension, i.e., the number of adjustable parameters in funk. For example, the model problem of § 1.1 requires n=2, the fitting problem of § 2.2 n=3, that of § 3.2 n=21, and so on. The maximum allowed size for n is set in a PARAMETER statement within PIKAIA, and in the listing below is set at 32. Ideally, the user should set this value equal to n, so that PIKAIA does not defines its internal arrays as having sizes larger than necessary. The floating point array ctrl has length 12 and contains flags and parameter values that control PIKAIA's evolutionary behavior; a detailed description of each element of ctrl is given below.

PIKAIA first calls subroutine SETCTL to perform a minimal set of run-time tests on its input parameters. Note that if any element of the control vector ctrl is *negative,* then PIKAIA will supply its own default values. If any invalid but positive values are supplied, PIKAIA aborts and returns with a positive (nonzero) value for the status output variable. One or many warning messages will be issued if some combinations of parameter values, while formally valid, risk producing an inefficient algorithm. It is important to realize that only a very basic set of tests is carried out, as optimal parameter settings are usually to some extent a function of the problem under consideration. Consequently, it is impossible to guarantee that all potentially inefficient combinations of parameters will elicit a warning from PIKAIA. Upon successful termination PIKAIA returns with status=0. Additional run-time information can be routed to standard output by choosing appropriate settings of the control flag ivrb, as described below.

PIKAIA is currently set up to produce minimal output, but this is an area where the user may wish to tailor the code to his/her specific needs. In addition to the status variable, the only output returned as arguments by PIKAIA is (1) a real array xb of length n, containing the n parameter values associated with the best phenotype in the final population, and (2) a floating point scalar fb corresponding to the evaluation of funk for that phenotype. In many cases it can be useful to output more information to file, for example the best and median phenotypes at each generation together with their associated fitness values, or maybe even the complete phenotype population. A logical place to output such information is at the end of the main generation loop, as indicated in the PIKAIA listing provided below. Internally, the population is stored in the two-dimensional array oldph (the two-dimensional array newph is only used as temporary storage when operating under full generational replacement). The ranking array ifit allows access to the population in terms of rank; the fittest phenotype is stored in oldph(1:n,ifit(np)), the second fittest in oldph(1:n,ifit(np-1)), and so on all the way to oldph(1:n,ifit(1)) for the worst. The complementary array jfit contains the actual ranks: jfit(i) is the rank of the individual stored in oldph(1:n,i). The array fitns contains the true fitness (i.e., funk evaluation) for each population member, with again the evaluation for the fittest being stored in fitns(ifit(np)), etc., as for oldph.

A3. DESCRIPTION OF INPUT QUANTITIES

Internally, PIKAIA associates the control vector ctrl to the following flags and parameters:

```
ctrl(1:12)=(np,ngen,nd,pcross,imut,pmut,pmutmn,pmutmx,fdif,irepp,ielite,ivrb) .
```

These correspond to the following:

Population number [np (≡ctrl(1)); default is np=100].—The number of individuals in the population. Note that this remains constant throughout the run. The population size is internally restricted to np ≤ 512.

Number of Generation [ngen (≡ctrl(2)); default is ngen=500].—PIKAIA evolves the population over a predetermined number of generations set by the value of the parameter ngen, instead of trying to meet a preset tolerance criterion. The latter approach is potentially dangerous when approaching a new problem, in view of the usual convergence trends exhibited by GA-based optimizers (cf. Figs. 3*f* and 4).

Encoding accuracy [nd (≡ctrl(3)); default is nd=5].—This sets the number of digits retained in encoding the phenotype into a genotype. This is internally restricted to nd ≤ 8; in most real applications, if more than 4 digits accuracy are required it would generally be preferable to use a 4 digit–accurate genetic solution as a starting guess for a more conventional optimization method. Note that the genotype ends up being an integer array of length n*nd, where each element ("gene") takes values in the range [0, 9]. The encoding scheme used in PIKAIA is clearly far from optimal in terms of efficient use of storage, but it is easy to code, understand, and modify . . . and to keep track of when something goes wrong.

Crossover rate [pcross (≡ctrl(4)); default is pcross=0.85].—Once two parents have been selected for breeding, a random number $R \in [0.0, 1.0]$ is generated, and the crossover operation (cf. Fig. 2) is applied only if $R \le$ pcross.

Mutation mode [imut(≡ctrl(5)) default is imut=2].—Integer flag controlling the behavior of the mutation operator. Setting imut=1 enforces a constant mutation rate, at a value set by pmut (see below). For imut=2, the mutation rate varies in the range [pmutmn,pmutmx] throughout the evolution, with starting value pmut. The mutation rate increases (decreases) only when the relative difference in the absolute fitnesses of the best and median member of the population falls below (exceeds) the value rdiflo (rdifhi). The mutation is varied by logarithmically constant increments delta, i.e. pmut←pmut*delta (pmut/delta). The values rdiflo=0.05, rdifhi=0.25 and delta=1.5 are set in a PARAMETER statement in subroutine adjmut, inspection of which should further clarify how variable mutation rate is implemented. Depending on the fitness contrast in parameter space, the user may wish to adjust the values of rdifhi and rdiflo.

Initial mutation rate [pmut (≡ctrl (6)); default is pmut=0.005].—By convention, the value of pmut corresponds to the probability (≤ 1) that a given *gene* be affected by a mutation at breeding. For every gene a random number $R \in [0.0, 1.0]$ is generated, and mutation is carried out only if $R \le$ pmut. The mutation itself consists in generating a random integer $K \in [0, 9]$, and setting the gene value to K; note that there is 1:10 probability that K be equal to the original gene value, in which case mutation has effectively no phenotypic effect.

Minimum mutation rate [pmutmn (≡ctrl(7)); used only if imut=2; default is pmutmn=0.0005]: Minimum mutation rate attainable under variable mutation mode.

Maximum mutation rate [pmutmx(≡ctrl(8)); used only if imut=2; default is pmutmx=0.25]: Maximum mutation rate attainable under variable mutation mode. Typically, pmut and/or pmutmx must be much smaller than unity, otherwise near complete randomization is likely to occur in every offspring. Under full generational replacement (see *reproduction plan* below) and unless elitism has been turned on (by setting ielite=1), PIKAIA will issues warnings if pmut > 0.05, or if imut=2 and pmutmx > 0.05.

Fitness differential [fdif (≡ctrl(9)); default is fdif=1.0]: PIKAIA makes use of ranking to assign fitness (see § 1.5). Individuals are first ranked as [1, 2, . . . , np], according to "true" fitness, where by definition the fittest individual has rank 1 and the least fit rank np, respectively. The breeding probability of the *i*th individual in the population is then defined as

```
1/np + fdif*(np + 1 − 2*jfit(i))/(np*(np + 1)),
```

where jfit(i) is the rank of the individual stored in the *i*th column of the population array oldph, as described before. This defines a *linear* relationship with slope fdif between rank and breeding probability. Note in particular that setting fdif=0.0 corresponds to no selection pressure (i.e., equal breeding probability for everybody), while fdif=1.0 directly equates breeding probability to rank. On Figure 4 for example, high selection pressure (SP = H) was produced by setting fdif=1.0, moderate pressure by setting fdif=3./5. and low-pressure fdif=1./3. . PIKAIA will issue a warning if fdif is set to a value inferior to 1./3., which would correspond to a fitness differential too low for most practical applications. All breeding probability calculations are carried out internally in subroutine select.

Reproduction plans [irepp (≡ctrl(10)); default is irepp=1].—Integer flag controlling the choice of either one of the reproductive plan discussed in § 1.5. Setting irepp=1 selects full generational replacement, and irepp=2 or irepp=3 steady state reproduction. In these latter cases, an offspring is inserted only if (1) its fitness is superior to that of the *least-fit* population member and (2) its genotype differs in at least one gene from any genotype already present in the population. This latter restriction is an important safeguard against inbreeding. The two steady state reproduction plans differ only in how individuals from the old population are deleted to make room for new offspring fit enough for insertion; under irepp=3, the least fit is deleted. If, on the other hand, irep=2, an individual from the old population is chosen at random and deleted, independently of its actual fitness. When operating under a steady-state reproduction plan, a "generation" is not a well-defined concept. Internally, PIKAIA defines a generation as a group of np individuals. One should note, in particular, that if a user-supplied output subroutine is inserted within the generation loop in PIKAIA (see above), this routine will be called once every time np individuals have been bred, independently of how many of them have actually been inserted in the population. As a rule of thumb, going from irepp=1 through irepp=2 to irepp=3 represents to some extent a transition from enhanced exploration to enhanced exploitation, but neither plan seems to be clearly superior to the others in a general sense (see, e.g., Syswerda 1991). Both reproduction plans operate under the assumption of a fixed-sized population, and make use of the roulette wheel algorithm for parent selection (see, e.g., chap. 1 of Davis 1991; numerous other sampling mechanisms are possible, for a discussion see chap. 4 of Goldberg 1989, or § 4.1 of Michalewicz 1994).

Elitism [ielite (≡ctrl(11)); default is ielite=1].—Integer flag controlling the use of elitism. Elitism is enforced if ielite=1, otherwise no action is taken. Under irepp=2, setting ielite=1 ensures that the fittest individual cannot be selected for random deletion. This flag has no effect under irepp=3 ("delete-worst" steady state reproduction plan).

Verbose mode [irvb (≡ctrl(12)); default is ivrb=0].—Integer flag controlling the generation of additional run-time output to the screen. Setting ivrb=1 or ivrb=2 generate a listing of input parameters, as well as information concerning the current status of the population. This latter output work is carried out by subroutine report. The first line of output generated by report contains (1) the generation count, (2) the number of individuals inserted in the last round of breeding activity, and (3) the fitnesses of the best, second and median individuals in the current population. This is followed by n lines listing the phenotypes for the best, second, and median individuals. Under ivrb=3 this information is printed at every generation. Under ivrb=2 it is printed only if either (1) the mutation rate has been adjusted (under imut=2) or (2) the fitness of the best individual has improved since the last generation (under irepp=1) or pseudogeneration (under irepp=2 or irepp=3).

A4. USER-SUPPLIED FUNCTIONS AND SUBROUTINES

Random number generator.—PIKAIA requires a random number generator function that returns random or pseudorandom deviates from a uniformly distributed sequence in the interval [0.0, 1.0]. This function *must* be called urand and *must* be declared and called without arguments, i.e., function urand(), r=urand(), and so on. urand must also be declared as REAL and EXTERNAL in the program calling PIKAIA. Any required initialization of urand should also be carried out by the calling program. The use of generic system-supplied random number generators is *not* recommended. More robust generators, such as routine ran2 of Press et al. (1992, § 7.1), should be used instead.

Ranking subroutine.—The user must supply a subroutine that accepts as input a floating point array arrin of length n, and return an integer array indx of identical length, where element indx(i) is a key index giving the location in arrin of the ith smallest value in arrin (this is equivalent to the integer array ifit described above). This subroutine is called only in subroutine rnkpop. The calling sequence included in the listing below corresponds to a call to subroutine indexx in Press et al. (1992, § 8.4), a recommended choice.

Fitness evaluation function.—The function funk to be maximized must be declared as REAL and EXTERNAL in the calling program. funk *must* accept as argument a single integer n (the dimension of parameter space) and a single floating point array of length n (a point in parameter space). The evaluation of funk *must* return a positive-definite quantity that measures fitness (high/low values ≡ high/low fitnesses, goodness of fit, etc.). PIKAIA searches a bounded nondimensional search space spanning the range [0.0, 1.0] in all n direction of parameter space. funk *must consequently carry out internally all appropriate scalings to dimensional variables.* Likewise, all values in the "best phenotype" array xb returned by PIKAIA upon successful termination (status=0) are in the range [0.0, 1.0] and so must be rescaled to dimensional values in the same way.

Initialization subroutine (optional).—The function funk will be called np×ngen in the course of the evolutionary run, and so should be coded in as efficient a manner as possible. Any secondary computation in funk that is independent of its arguments should be carried out once and for all before calling PIKAIA, and the computed quantities passed to funk via one or more appropriately defined labeled COMMON blocks. The initialization subroutine would also be a logical place to read in data, and so on.

A5. CODE LISTING

Following the reference section is a reduced listing of the subroutine PIKAIA. The complete subroutine PIKAIA contains many more explanatory comments lines than the listing given here.

Following this listing is an example of a driver program xpikaia that seeks to maximize a two-dimensional function two__d, a listing of the latter being also provided. The function two__d is qualitatively similar to the test function of Figure 1 but actually a bit harder from the numerical optimization standpoint.

REFERENCES

Ackley, D. H. 1987, in Genetic Algorithms and Simulated Annealing, ed. L. Davis (London: Pitman), 170

Belcher, J. W., & MacGregor, K. B. 1976, ApJ, 210, 498

Breger, M. 1980, ApJ, 237, 850

Brown, T. M., & Gilliland, R. L. 1994, ARA&A, 32, 37

Carignan, C., Charbonneau, P., Boulanger, F., & Viallefond, F. 1990, A&A, 234, 43

Carignan, C., & Freeman, K. C. 1985, ApJ, 294, 494

Catala, C., et al. 1994, A&A, 275, 245

Charbonneau, P., Schrijver, C. J., & MacGregor, K. B. 1995, in Cosmic Winds and the Heliosphere, ed. J. R. Jopikii, C. P. Sonett, & M. S. Giampapa (Tucson: Univ. of Arizona Press), in press

Christensen-Dalsgaard, J., Gough, D., & Toomre, J. 1985, Science, 229, 923

Craig, I. J. D., & Brown, J. C. 1986, Inverse Problems in Astronomy, (Bristol, UK: Adam Hilger)

Davis, L. 1991, Handbook of Genetic Algorithms (New York: Van Nostrand Reinhold)

Dawkins, R. 1982, The Extended Phenotype (Oxford: Oxford Univ. Press)

De Jong, K. A. 1993, in Foundations of Genetic Algorithms 2, ed. L. D. Whitley (San Mateo: Morgan Kaufmann), 5

Eldredge, N. 1985, Time Frames (New York: Simon & Schuster)

Faber, S. M., & Gallaher, J. S. 1979, ARA&A, 17, 135

Goldberg, D. E. 1989, Genetic Algorithms in Search, Optimization, & Machine Learning (Reading, MA: Addison-Wesley)

Gough, D., & Toomre, J. 1991, ARA&A, 29, 627

Gould, S. J. 1989, Wonderful Life. The Burgess Shale and the Nature of History (New York: W. W. Norton)

Hakala, P. J. 1995, A&A, 296, 164

Holland, J. H. 1975, Adaptation in Natural and Artificial Systems (1st ed.; Ann Arbor: Univ. of Michigan Press; 2d ed.; 1992, Cambridge: MIT Press)

Jeffrey, W., & Rosner, R. 1986a, ApJ, 310, 463

Jeffrey, W., & Rosner, R. 1986b, ApJ, 310, 473

Kennelly, E. J., et al. 1995, in preparation

Kent, S. M. 1986, AJ 91, 1301

Koza, J. R. 1992, Genetic Programming: on the Programming of Computers by Means of Natural Selection (Cambridge: MIT Press)

Matthews, J. M. 1993, in ASP Conf. Ser. Vol. 42, GONG 1992: Seismic Investigation of the Sun and Stars, ed. T. M. Brown (San Francisco: ASP), 303

Maynard Smith, J. 1989, Evolutionary Genetics (Oxford: Oxford Univ. Press)

Michalewicz, Z. 1994, Genetic Algorithms + Data Structures = Evolution Programs (New York: Springer)

Nelder, J. A., & Mead, R. 1965, Comput. J., 7, 308

Parker, E. N. 1958, ApJ, 128, 664

———. 1963, Interplanetary Dynamical Processes (New York: Wiley)

Pizzo, V., Schwenn, R., Marsch, E., Rosenbauer, H., Mülhäuser, K.-H., & Neubauer, F. M. 1983, ApJ, 271, 335

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. 1992, Numerical Recipes (2d. ed.; Cambridge: Cambridge Univ. Press)

Stebbins, G. L. 1966, Processes of Organic Evolution (Englewood Cliffs, NJ: Prentice Hall)

Syswerda, G. 1991, in Foundations of Genetic Algorithms, ed. G. J. E. Rawlins (San Mateo: Morgan Kaufmann), 94

Tomczyk, S., Charbonneau, P., Schou, J., & Thompson, M. J. 1995, in Proc. 4th SOHO Workshop: Helioseismology, ed. T. Hoeksema (Noordwijk: ESA), in press

Watson, J. D., Hopkins, N. H., Roberts, J. W., Steitz, J. A., & Weiner, A. M. 1987, Molecular Biology of the Gene (Menlo Park, CA: Benjamin/Cummings)

Weber, E. J., & Davis, L., Jr. 1967, ApJ, 148, 217

Wright, A. H. 1991, in Foundations of Genetic Algorithms, ed. G. J. E. Rawlins (San Mateo, CA: Morgan Kaufmann), 205

```
      subroutine pikaia(ff,n,ctrl,x,f,status)
      implicit none
      real        ctrl(12),x(n),f,ff
      integer     n,status
      external    ff
c=======================================================================
c     Optimization (maximization) of user-supplied function ff over
c     n-dimensional parameter space  x  using a GA-based optimizer.
c
c     Paul Charbonneau & Barry Knapp
c     High Altitude Observatory
c     National Center for Atmospheric Research
c     Boulder CO 80307-3000
c     <paulchar@hao.ucar.edu>
c     <knapp@hao.ucar.edu>
c
c     Version of 1995 April 13
c
c     Genetic algorithms (GA) are heuristic search techniques that
c     incorporate in a computational setting, the biological notion
c     of evolution by means of natural selection.  This subroutine
c     is a general purpose GA-based optimizer, incorporating the basic
c     GA operators of one-point crossover and mutation, as well as a
c     few additional robust reproductive and ecological strategies
c     known to improve the performance of GA in the context of
c     function optimization. Parameters defining the function to be
c     optimized (maximized) are encoded as string of simple decimal
c     integers [0,9].
c
c     References:
c
c        Goldberg, David E.  Genetic Algorithms in Search, Optimization,
c           & Machine Learning.  Addison-Wesley, 1989.
c
c        Davis, Lawrence, ed.  Handbook of Genetic Algorithms.
c           Van Nostrand Reinhold, 1991.
c=====================================================================
c     USES: setctl, ff, urand, rnkpop, select, encode, decode, cross,
c           mutate, genrep, stdrep, newpop, adjmut, report
      integer     NMAX, PMAX, DMAX
      parameter   (NMAX = 32, PMAX = 512, DMAX = 8)
      integer     np, nd, ngen, imut, irep, ielite, ivrb, k, ip, ig,
     +            ip1, ip2, new, newtot, gn1(NMAX*DMAX), gn2(NMAX*DMAX),
     +            ifit(PMAX), jfit(PMAX)
      real        pcross, pmut, pmutmn, pmutmx, fdif, fitns(PMAX),
     +            ph(NMAX,2), oldph(NMAX,PMAX), newph(NMAX,PMAX), urand
      external    urand
c     Set control variables from input and defaults
```

```
      call setctl
     +   (ctrl,n,np,ngen,nd,pcross,pmutmn,pmutmx,pmut,imut,
     +    fdif,irep,ielite,ivrb,status)
      if (status .ne. 0) then
         write(*,*) ' Control vector (ctrl) argument(s) invalid'
         return
      endif
c     Make sure locally-dimensioned arrays are big enough
      if (n.gt.NMAX .or. np.gt.PMAX .or. nd.gt.DMAX) then
         write(*,*)
     +      ' Number of parameters, population, or genes too large'
         status = -1
         return
      endif
c     Compute initial (random but bounded) phenotype population
      do 1 ip=1,np
         do 2 k=1,n
            oldph(k,ip)=urand()
    2    continue
         fitns(ip) = ff(n,oldph(1,ip))
    1 continue
c     Rank initial population by fitness order
      call rnkpop(np,fitns,ifit,jfit)
c     Main Generation Loop
      do 10 ig=1,ngen
c        Main Population Loop
         newtot=0
         do 20 ip=1,np/2
c           1. pick two parents
            call select(np,jfit,fdif,ip1)
   21       call select(np,jfit,fdif,ip2)
            if (ip1.eq.ip2) goto 21
c           2. encode parent phenotypes
            call encode(n,nd,oldph(1,ip1),gn1)
            call encode(n,nd,oldph(1,ip2),gn2)
c           3. breed
            call cross(n,nd,pcross,gn1,gn2)
            call mutate(n,nd,pmut,gn1)
            call mutate(n,nd,pmut,gn2)
c           4. decode offspring genotypes
            call decode(n,nd,gn1,ph(1,1))
            call decode(n,nd,gn2,ph(1,2))
c           5. insert into population
            if (irep.eq.1) then
               call genrep(NMAX,n,np,ip,ph,newph)
            else
               call stdrep(ff,NMAX,n,np,irep,ielite,
     +                     ph,oldph,fitns,ifit,jfit,new)
```

```
               newtot = newtot+new
            endif
c        End of Main Population Loop
   20    continue
c        if running full generational replacement: swap populations
         if (irep.eq.1)
     +      call newpop(ff,ielite,NMAX,n,np,oldph,newph,
     +         ifit,jfit,fitns,newtot)
c        adjust mutation rate?
         if (imut.eq.2) call adjmut(np,fitns,ifit,pmutmn,pmutmx,pmut)
c        print generation report to standard output?
         if (ivrb.gt.0) call report
     +      (ivrb,NMAX,n,np,nd,oldph,fitns,ifit,pmut,ig,newtot)
c**** User-supplied (optional) output routine could go here
c     End of Main Generation Loop
   10 continue
c     Return best phenotype and its fitness
      do 30 k=1,n
         x(k) = oldph(k,ifit(np))
   30 continue
      f = fitns(ifit(np))
      end
c***********************************************************************
      subroutine setctl
     +   (ctrl,n,np,ngen,nd,pcross,pmutmn,pmutmx,pmut,imut,
     +    fdif,irep,ielite,ivrb,status)
      implicit none
      integer    n, np, ngen, nd, imut, irep, ielite, ivrb, status
      real       pcross, pmutmn, pmutmx, pmut, fdif, ctrl(12)
c=======================================================================
c     Set control variables and flags from input and defaults
c=======================================================================
      integer    i
      real       DFAULT(12)
      save       DFAULT
      data       DFAULT /100,500,6,.85,2,.005,.0005,.25,1,3,0,0/
      do 1 i=1,12
         if (ctrl(i).lt.0.) ctrl(i)=DFAULT(i)
    1 continue
      np = ctrl(1)
      ngen = ctrl(2)
      nd = ctrl(3)
      pcross = ctrl(4)
      imut = ctrl(5)
      pmut = ctrl(6)
      pmutmn = ctrl(7)
      pmutmx = ctrl(8)
      fdif = ctrl(9)
```

```
      irep = ctrl(10)
      ielite = ctrl(11)
      ivrb = ctrl(12)
      status = 0
c     Print a header
      if (ivrb.gt.0) then
         write(*,2) ngen,np,n,nd,pcross,pmut,pmutmn,pmutmx,fdif
    2    format(/1x,60('*'),/,
     +      ' *',13x,'PIKAIA Genetic Algorithm Report ',13x,'*',/,
     +           1x,60('*'),//,
     +      '   Number of Generations evolving: ',i4,/,
     +      '       Individuals per generation: ',i4,/,
     +      '    Number of Chromosome segments: ',i4,/,
     +      '    Length of Chromosome segments: ',i4,/,
     +      '            Crossover probability: ',f9.4,/,
     +      '            Initial mutation rate: ',f9.4,/,
     +      '            Minimum mutation rate: ',f9.4,/,
     +      '            Maximum mutation rate: ',f9.4,/,
     +      '    Relative fitness differential: ',f9.4)
         if (imut.eq.1) write(*,3) 'Constant'
         if (imut.eq.2) write(*,3) 'Variable'
    3    format(
     +      '                     Mutation Mode: ',A)
         if (irep.eq.1) write(*,4) 'Full generational replacement'
         if (irep.eq.2) write(*,4) 'Steady-state-replace-random'
         if (irep.eq.3) write(*,4) 'Steady-state-replace-worst'
    4    format(
     +      '                 Reproduction Plan: ',A)
      endif
c     Check some control values
      if (imut.ne.1 .and. imut.ne.2) then
         write(*,10)
         status = 5
      endif
   10 format(' ERROR: illegal value for imut (ctrl(5))')
      if (fdif.gt.1.) then
         write(*,11)
         status = 9
      endif
   11 format(' ERROR: illegal value for fdif (ctrl(9))')
      if (irep.ne.1 .and. irep.ne.2 .and. irep.ne.3) then
         write(*,12)
         status = 10
      endif
   12 format(' ERROR: illegal value for irep (ctrl(10))')
      if (pcross.gt.1.0 .or. pcross.lt.0.) then
         write(*,13)
         status = 4
      endif
   13 format(' ERROR: illegal value for pcross (ctrl(4))')
      if (irep.eq.1 .and. imut.eq.1 .and. pmut.gt.0.5 .and.
     +    ielite.eq.0) then
         write(*,14)
      endif
   14 format(' WARNING: dangerously high value for pmut (ctrl(6));',
     +      /' (Should enforce elitism with ctrl(11)=1.)')
      if (irep.eq.1 .and. imut.eq.2 .and. pmutmx.gt.0.5 .and.
     +    ielite.eq.0) then
         write(*,15)
      endif
   15 format(' WARNING: dangerously high value for pmutmx (ctrl(8));',
     +      /' (Should enforce elitism with ctrl(11)=1.)')
      if (fdif.lt.0.33) then
         write(*,16)
      endif
   16 format(' WARNING: dangerously low value of fdif (ctrl(9))')
      return
      end
c**********************************************************************
      subroutine report
     +   (ivrb,ndim,n,np,nd,oldph,fitns,ifit,pmut,ig,nnew)
      implicit none
      integer    ifit(np),ivrb,ndim,n,np,nd,ig,nnew
      real       oldph(ndim,np),fitns(np),pmut
c======================================================================
c     Write generation report to standard output
c======================================================================
      real       bestft,pmutpv
      save       bestft,pmutpv
      integer    ndpwr,k
      logical    rpt
      data       bestft,pmutpv /0,0/
      rpt=.false.
      if (pmut.ne.pmutpv) then
         pmutpv=pmut
         rpt=.true.
      endif
      if (fitns(ifit(np)).ne.bestft) then
         bestft=fitns(ifit(np))
         rpt=.true.
      endif
      if (rpt .or. ivrb.ge.2) then
c        Power of 10 to make integer genotypes for display
         ndpwr = nint(10.**nd)
         write(*,'(/i6,i6,f10.6,4f10.6)') ig,nnew,pmut,
     +      fitns(ifit(np)), fitns(ifit(np-1)), fitns(ifit(np/2))
         do 15 k=1,n
            write(*,'(22x,3i10)')
     +         nint(ndpwr*oldph(k,ifit(np  ))),
     +         nint(ndpwr*oldph(k,ifit(np-1))),
     +         nint(ndpwr*oldph(k,ifit(np/2)))
   15    continue
      endif
      end
c**********************************************************************
      subroutine encode(n,nd,ph,gn)
      implicit none
      integer    n, nd, gn(n*nd)
      real       ph(n)
c======================================================================
c     encode phenotype parameters into integer genotype
c     ph(k) are x,y coordinates [ 0 < x,y < 1 ]
c======================================================================
      integer    ip, i, j, ii
      real       z
      z=10.**nd
      ii=0
      do 1 i=1,n
         ip=int(ph(i)*z)
         do 2 j=nd,1,-1
            gn(ii+j)=mod(ip,10)
            ip=ip/10
    2    continue
         ii=ii+nd
    1 continue
      return
      end
c**********************************************************************
      subroutine decode(n,nd,gn,ph)
      implicit none
      integer    n, nd, gn(n*nd)
      real       ph(n)
c======================================================================
c     decode genotype into phenotype parameters
c     ph(k) are x,y coordinates [ 0 < x,y < 1 ]
c======================================================================
      integer    ip, i, j, ii
      real       z
      z=10.**(-nd)
      ii=0
      do 1 i=1,n
         ip=0
         do 2 j=1,nd
            ip=10*ip+gn(ii+j)
```

```
 2    continue
      ph(i)=ip*z
      ii=ii+nd
   1 continue
     return
     end
c***********************************************************************
     subroutine cross(n,nd,pcross,gn1,gn2)
     implicit none
     integer    n, nd, gn1(n*nd), gn2(n*nd)
     real       pcross
c=======================================================================
c    breeds two parent chromosomes into two offspring chromosomes
c    breeding occurs through crossover starting at position ispl
c=======================================================================
c    USES: urand
     integer    i, ispl, t
     real       urand
     external   urand
c    Use crossover probability to decide whether a crossover occurs
     if (urand().lt.pcross) then
c       Compute crossover point
        ispl=int(urand()*n*nd)+1
c       Swap genes at ispl and above
        do 10 i=ispl,n*nd
           t=gn2(i)
           gn2(i)=gn1(i)
           gn1(i)=t
 10     continue
     endif
     return
     end
c***********************************************************************
     subroutine mutate(n,nd,pmut,gn)
     implicit none
     integer    n, nd, gn(n*nd)
     real       pmut
c=======================================================================
c    Mutations occur at rate pmut at all gene loci
c=======================================================================
c    USES: urand
     integer    i
     real       urand
     external   urand
     do 10 i=1,n*nd
        if (urand().lt.pmut) then
           gn(i)=int(urand()*10.)
        endif
```

```
  10 continue
     return
     end
c***********************************************************************
     subroutine adjmut(np,fitns,ifit,pmutmn,pmutmx,pmut)
     implicit none
     integer    np, ifit(np)
     real       fitns(np), pmutmn, pmutmx, pmut
c=======================================================================
c    dynamical adjustment of mutation rate; criterion is relative
c    difference in absolute fitnesses of best and median individuals
c=======================================================================
     real       rdif, rdiflo, rdifhi, delta
     parameter  (rdiflo=0.05, rdifhi=0.25, delta=1.5)
     rdif=abs(fitns(ifit(np))-fitns(ifit(np/2)))/
    +        (fitns(ifit(np))+fitns(ifit(np/2)))
     if(rdif.le.rdiflo)then
        pmut=min(pmutmx,pmut*delta)
     else if(rdif.ge.rdifhi)then
        pmut=max(pmutmn,pmut/delta)
     endif
     return
     end
c***********************************************************************
     subroutine select(np,jfit,fdif,idad)
     implicit none
     integer    np, jfit(np), idad
     real       fdif
c=======================================================================
c    Selects a parent from the population, using roulette wheel
c    algorithm with the relative fitnesses of the phenotypes as
c    the "hit" probabilities [see Davis 1991, chap. 1].
c=======================================================================
c    USES: urand
     integer    np1, i
     real       dice, rtfit, urand
     np1 = np+1
     dice = urand()*np*np1
     rtfit = 0.
     do 1 i=1,np
        rtfit = rtfit+np1+fdif*(np1-2*jfit(i))
        if (rtfit.ge.dice) then
           idad=i
           goto 2
        endif
   1 continue
   2 return
     end
```

```
c***********************************************************************
      subroutine rnkpop(n,arrin,indx,rank)
      implicit none
      integer    n, indx(n),rank(n)
      real       arrin(n)
c=======================================================================
c     Calls external sort routine to produce key index and rank order
c     of input array arrin (which is not altered).
c=======================================================================
c     USES: indexx
      integer    n, indx(n),rank(n)
      real       arrin(n)
      integer    i
      external   indexx
c**** User-supplied routine: Compute the key index
      call indexx(n,arrin,indx)
c     ...and the rank order
      do 1 i=1,n
         rank(indx(i)) = n-i+1
    1 continue
      return
      end
c***********************************************************************
      subroutine genrep(ndim,n,np,ip,ph,newph)
      implicit none
      integer    ndim, n, np, ip
      real       ph(ndim,2), newph(ndim,np)
c=======================================================================
c     full generational replacement: accumulate offspring into new
c     population array
c=======================================================================
      integer    i1, i2, k
      i1=2*ip-1
      i2=i1+1
      do 1 k=1,n
         newph(k,i1)=ph(k,1)
         newph(k,i2)=ph(k,2)
    1 continue
      return
      end
c***********************************************************************
      subroutine stdrep
     +   (ff,ndim,n,np,irep,ielite,ph,oldph,fitns,ifit,jfit,nnew)
      implicit none
      integer    ndim, n, np, irep, ielite, ifit(np), jfit(np), nnew
      real       ff, ph(ndim,2), oldph(ndim,np), fitns(np)
      external   ff
c=======================================================================
c     steady-state reproduction: insert offspring pair into population
c     only if they are fit enough (replace-random if irep=1 or
c     replace-worst if irep=2).
c=======================================================================
c     USES: ff, urand
      integer    i, j, k, i1, if1
      real       fit, urand
      external   urand
      nnew = 0
      do 1 j=1,2
c        1. compute offspring fitness (with caller's fitness function)
         fit=ff(n,ph(1,j))
c        2. if fit enough, insert in population
         do 20 i=np,1,-1
            if (fit.gt.fitns(ifit(i))) then
c              make sure the phenotype is not already in the population
               if (i.lt.np) then
                  do 5 k=1,n
                     if (oldph(k,ifit(i+1)).ne.ph(k,j)) goto 6
    5             continue
                  goto 1
    6             continue
               endif
c              offspring is fit enough for insertion, and is unique
c              (i) insert phenotype at appropriate place in population
               if (irep.eq.3) then
                  i1=1
               else if (ielite.eq.0 .or. i.eq.np) then
                  i1=int(urand()*np)+1
               else
                  i1=int(urand()*(np-1))+1
               endif
               if1 = ifit(i1)
               fitns(if1)=fit
               do 21 k=1,n
                  oldph(k,if1)=ph(k,j)
   21          continue
c              (ii) shift and update ranking arrays
               if (i.lt.i1) then
c                 shift up
                  jfit(if1)=np-i
                  do 22 k=i1-1,i+1,-1
                     jfit(ifit(k))=jfit(ifit(k))-1
                     ifit(k+1)=ifit(k)
   22             continue
                  ifit(i+1)=if1
               else
c                 shift down
```

```
                  jfit(if1)=np-i+1
                  do 23 k=i1+1,i
                     jfit(ifit(k))=jfit(ifit(k))+1
                     ifit(k-1)=ifit(k)
   23             continue
                  ifit(i)=if1
               endif
               nnew = nnew+1
               goto 1
            endif
   20    continue
    1 continue
      return
      end
c***********************************************************************
      subroutine newpop
     +   (ff,ielite,ndim,n,np,oldph,newph,ifit,jfit,fitns,nnew)
      implicit none
      integer    ndim, np, n, ielite, ifit(np), jfit(np), nnew
      real       ff, fitns(np), oldph(ndim,np), newph(ndim,np)
      external   ff
c=======================================================================
c     replaces old population by new; recomputes fitnesses & ranks
c=======================================================================
c     USES: ff, rnkpop
      integer    i, k
      nnew = np
c     if using elitism, introduce in new population fittest of old
c     population (if greater than fitness of the individual it is
c     to replace)
      if (ielite.eq.1 .and. ff(n,newph(1,1)).lt.fitns(ifit(np))) then
         do 1 k=1,n
            newph(k,1)=oldph(k,ifit(np))
    1    continue
         nnew = nnew-1
      endif
c     replace population
      do 2 i=1,np
         do 3 k=1,n
            oldph(k,i)=newph(k,i)
    3    continue
c        get fitness using caller's fitness function
         fitns(i)=ff(n,oldph(1,i))
    2 continue
c     compute new population fitness rank order
      call rnkpop(np,fitns,ifit,jfit)
      return
      end
```

```
      program xpikaia
c=======================================================================
c     Exercise driver program for pikaia.f
c
c     This program performs repeated maximization of a 2-D function
c     named two_d, prompting the user for a random seed
c=======================================================================
      implicit none
      integer n, seed, i, status
      parameter (n=2)
      real ctrl(12), x(n), f, two_d
      external two_d
c**** First, initialize the random-number generator
c     (no other initialization required for this application)
    1 write(*,'(/A$)') ' Random number seed (I*4)? '
      read(*,*) seed
      seed = -abs(seed)
      call urand_init(seed)
c     Set control variables (use defaults)
      do 10 i=1,12
         ctrl(i) = -1
   10 continue
c     Now call pikaia
      call pikaia(two_d,n,ctrl,x,f,status)
c     Print the results
      write(*,*) ' status: ',status
      write(*,*) '      x: ',x
      write(*,*) '      f: ',f
      write(*,20) ctrl
 20   format(  '    ctrl: ',6f11.6/10x,6f11.6)
      goto 1
      end

      function  two_d(n,x)
      real      x(n)
c=======================================================================
c     Compute sample fitness function (altitude in 2-d landscape)
c=======================================================================
      implicit  none
      integer   n,nn
      real      pi, sigma2, x(n), rr, two_d
      parameter (pi=3.1415926536,sigma2=0.15,nn=9)
      if (x(1).gt.1..or.x(2).gt.1.) stop
      rr=sqrt( (0.5-x(1))**2+ (0.5-x(2))**2 )
      ff=cos(rr*nn*pi)
      two_d=cos(rr*nn*pi)*exp(-rr**2/sigma2)
      return
      end
```