Robot trajectory planning and introductory stealth design

Abstract

Now that we have seen some basic applications of both binary and continuous GAs and discussed some of the fine points of their implementation, it will be fun to look at what can be accomplished with a GA and a bit of imagination. The examples in this chapter make use of some of the advanced topics discussed in Chapter 5 and add variety to the examples presented in Chapter 4. They cover a wide range of areas and include technical as well as nontechnical examples. The first example is the infamous traveling salesperson problem where the GA must order the cities visited by the salesperson. The second example revisits the locating-an-emergency-response unit from Chapter 4 but this time uses a Gray code. Next comes a search for an alphabet that decodes a secret message. The next examples come from engineering design and include robot trajectory planning and introductory stealth design. We end with several examples from science and mathematics that demonstrate some of the ways in which GAs are being used in research: two use data to build inverse models, one couples a simulation with a GA to identify allocations of sources to an air pollution monitor, one combines the GA with another artificial intelligence technique—the neural network—and the final one finds solutions to a nonlinear fifth-order differential equation.

1.TRAVELING SALESPERSON PROBLEM

Chapter 5 presented several methods to modify the crossover and mutation operators in order for a GA to tackle reordering or permutation problems. It's time to try this brand of GA on the famous traveling salesperson problem, which represents a classic optimization problem that cannot be solved using traditional techniques (although it has been successfully attacked with simulated annealing; Kirkpatrick et al., 1983). The goal is to find the shortest route for a salesperson to take in visiting N cities. This type of problem appears in many forms, with some engineering applications that include the optimal

layout of a gas pipeline, design of an antenna feed system, configuration of transistors on a very large-scale integration (VLSI) circuit, or sorting objects to match a particular configuration. Euler introduced a form of the traveling salesperson problem in 1759, and it was formally named and introduced by the Rand Corporation in 1948 (Michalewicz, 1992).

The cost function for the simplest form of the problem is just the distance traveled by the salesperson for the given ordering (x_n, y_n), $n = 1, \ldots, N$ given by

$$\cos t = \sum_{n=0}^{N} \sqrt{(x_n - x_{n+1})^2 + (y_n - y_{n+1})^2} \tag{6.1}$$

where (x_n, y_n) are the coordinates of the nth city visited. For our example, let's put the starting and ending point at the origin, so $(x_0, y_0) = (x_{N+1}, y_{N+1}) = (0, 0)$ = starting and ending point. This requirement ties the hands starting of the algorithm somewhat. Letting the starting city float provides more possibilities of optimal solutions.

The crossover operator is a variation of the cycle crossover (CX) described in Chapter 5. Here, however, we randomly select a location in the chromosome where the integers are exchanged between the two parents. Unless the exchanged integers are the same, each offspring has a duplicate integer. Next the repeated integer in $offspring_1$ is switched with the integer at that site in $offspring_2$. Now a different integer is duplicated, so the process iterates until we return to the first exchanged site. At this point each offspring contains exactly one copy of each integer from 1 to N. The mutation operator randomly chooses a string, selecting two random sites within that string, and exchanges the integers at those sites.

We'll initially look at this problem with $N = 13$ cities. Given the fixed starting and ending points, there are a total of $13!/2 = 3.1135 \times 10^9$ possible combinations to check. To test the algorithm, we will start with a configuration where all the cities lie in a rectangle as shown in Figure 6.1. We know that the minimum distance is 14. The GA parameters for this case are $N_{pop} = 400$, $N_{keep} = 200$, and $\mu = 0.04$. The algorithm found the solution in 35 generations as shown in Figure 6.2.

Now let's try a more difficult configuration. Randomly placing the 25 cities in a 1×1 square doesn't have an obvious minimum path. How do we know that the GA has arrived at the solution? The optimal solution will have no crossing paths. So we'll plot the solution and check. The algorithm had $N_{pop} = 100$, $N_{keep} = 50$, and $\mu = 0.04$. This algorithm found the minimum in 130 generations. Figure 6.3 shows the convergence of the algorithm, and Figure 6.4 is the optimal solution. We found that low population sizes and high mutation rates do not work as well for the permutation problems. For more details, see Whitley et al. (1991).

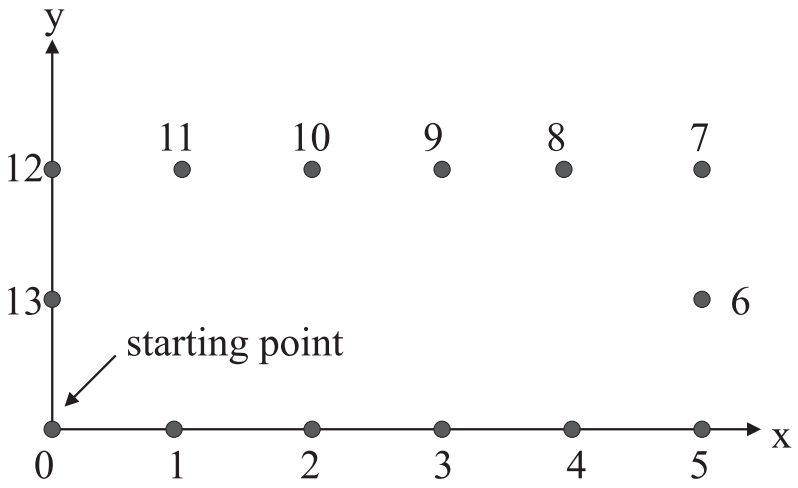

Figure 6.1 Graph of 13 cities arranged in a rectangle. The salesperson starts at the origin and visits all 13 cities once and returns to the starting point. The obvious solution is to trace the rectangle, which has a distance of 14.

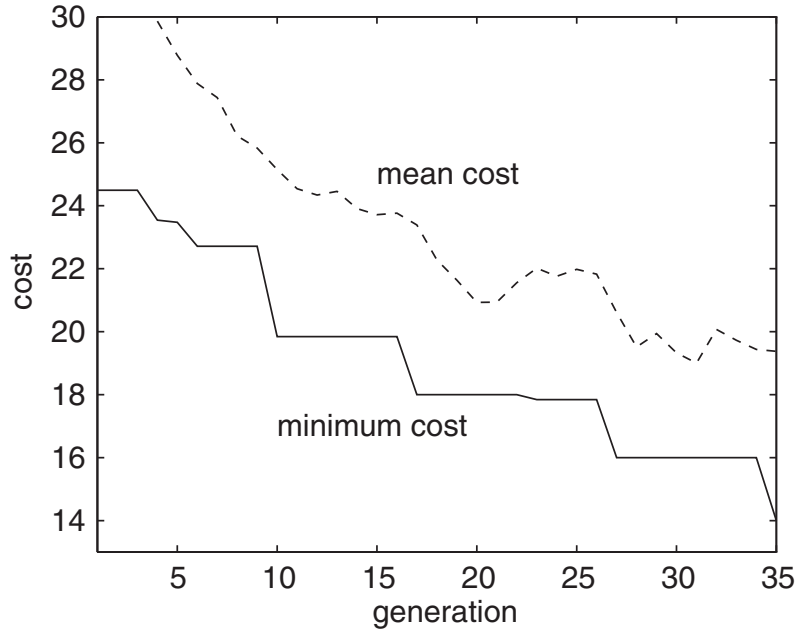

Figure 6.2 Convergence of the genetic algorithm when there are 13 cities on a rectangle as shown in Figure 6.1.

6.2 LOCATING AN EMERGENCY RESPONSE UNIT REVISITED

Finding the location of an emergency response unit described in Chapter 4 had a cost surface with two minima. Running the continuous and binary GAs revealed that the continuous GA was superior. One of the problems with the binary GA is the use of binary numbers to represent variable values. In this

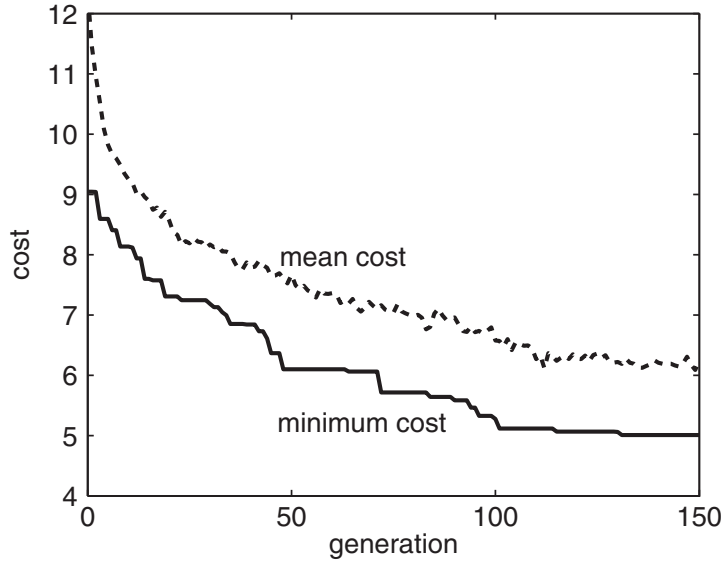

Figure 6.3 Convergence of the GA for the 25 city traveling salesperson problem.

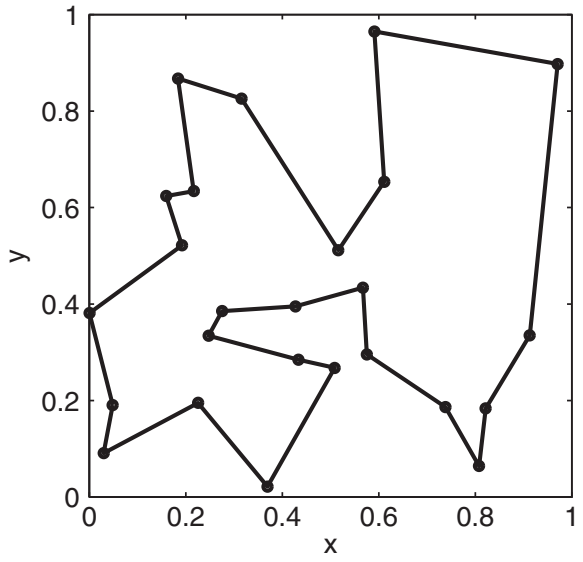

Figure 6.4 GA solution to 25 city traveling salesperson problem.

chapter we solve the same problem with a binary GA but use a Gray code to represent the variables.

Gray codes don't always improve the convergence of a GA. The convergence graph in Figure 6.5 shows that the Gray code did improve performance in this instance. However, implementing the Gray code in the GA slows down the algorithm because the translation of the binary code into binary numbers is time-consuming. We're somewhat skeptical of adding the Gray code translation to our GAs, so we usually don't. However, the result here shows that a small improvement is possible with the Gray code.

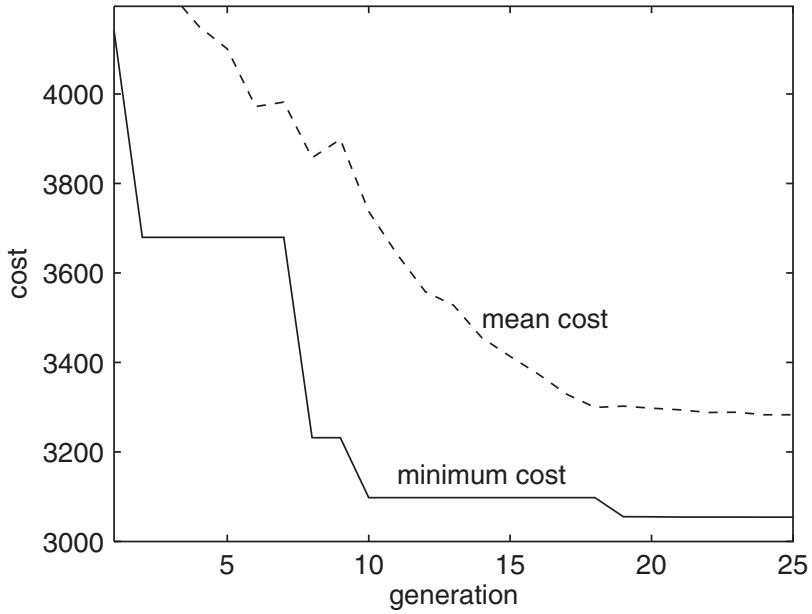

Figure 6.5 Convergence graph for the emergency response unit problem from Chapter 4 when a Gray code is used.

6.3 DECODING A SECRET MESSAGE

This example uses a continuous GA to break a secret code. A message consisting of letters and spaces is encoded by randomly changing one letter to another letter. For instance, all *d*'s may be changed to *c*'s and spaces changed to *q*'s. If the message uses every letter in the alphabet plus a space, then there are a total of 27! possible codes, with only one being correct. If the message uses *S* symbols, then there are 27! – *S*! possible encodings that work.

A chromosome consists of 27 genes with unique values from 1 to 27. A 1 corresponds to a space and 2 through 27 correspond to the letters of the alphabet. Letters and spaces in the *message* receive the appropriate numeric values. The cost is calculated by subtracting the guess of the message from the known message, taking the absolute value, and summing:

$$\text{cost} = \sum_{n=1}^{N} |message(n) - guess(n)| \tag{6.2}$$

We know the message when the cost is zero.

As an example, let's see how long it takes the GA to find the encoding for the message "bonny and amy are our children." This message has 30 total symbols, of which 15 are distinct. Thus 15 of the letters must be in the proper order, while the remaining 12 letters can be in any order. The GA used the following constants: $N_{pop} = 400$, $N_{keep} = 40$, and $\mu = 0.02$. It found the message in 68 generations as shown in Figure 6.6. Progress on the decoding is shown in Table 6.1.

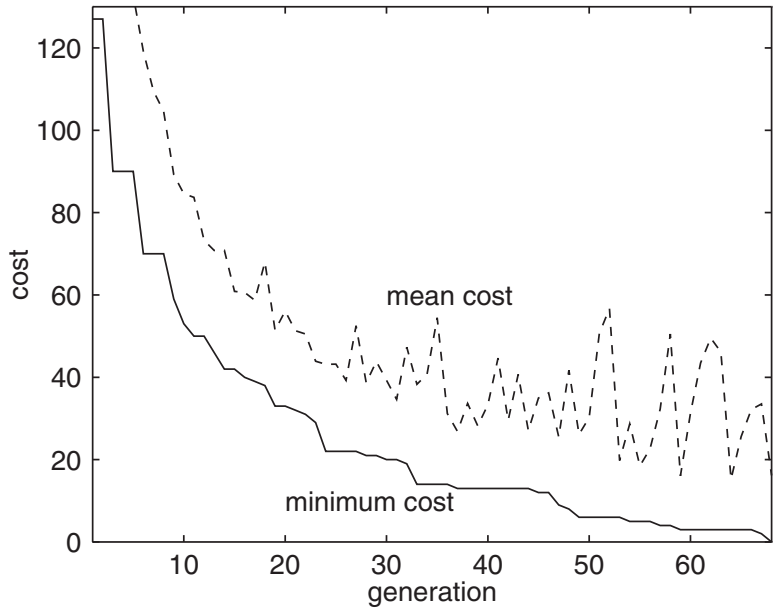

Figure 6.6 Genetic algorithm that decodes the message "bonny and amy are our children" in 68 generations.

TABLE 6.1 Progress of the GA as It Decodes the Secret Message

Generation	Message
1	amiizbditbdxzbdqfbmvqbeoystqfi
10	krooy aoe any aqf rwq gbpseqfo
20	crooy aoe any aqf rwq gdiheqfo
30	dpooy aoe any arf pwr ghikerfo
40	bqmmz amd anz are qur cfildrem
50	bonnz and amz are osr cghldren
60	bonny and ajy are our children
68	bonny and amy are our children

A more difficult message is "jake can go out with my beautiful pets and quickly drive to see you." This message lacks only x and z. It has 25 distinct symbols and a total of 65 total symbols. Figure 6.7 shows the convergence in this case with $N_{pop} = 500$, $N_{good} = 40$, and $\mu = 0.02$. The algorithm found the solution in 86 generations. Progress on the decoding is shown in Table 6.2.

6.4 ROBOT TRAJECTORY PLANNING

Robots imitate biological movement, and GAs imitate biological survival. The two topics seem to be a perfect match, and many researchers have made that connection. Several studies have investigated the use of GAs for robot tra-

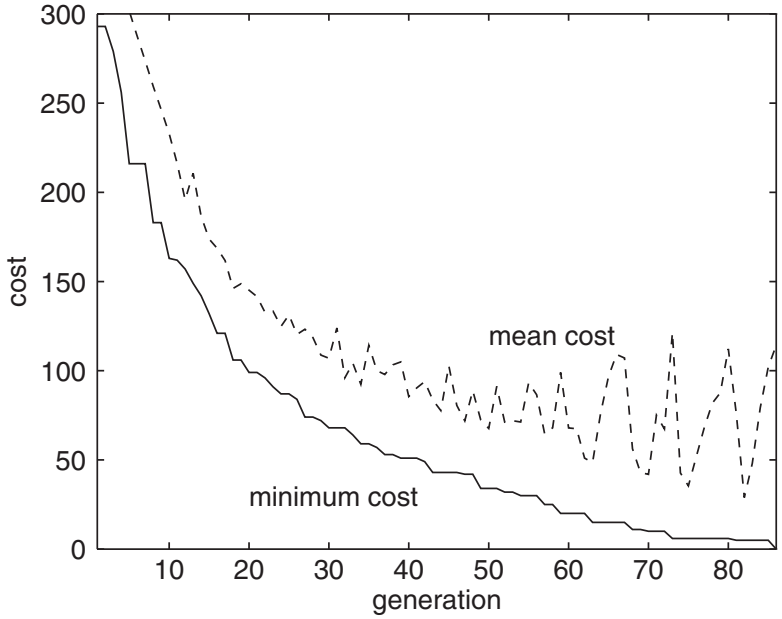

Figure 6.7 Genetic algorithm that decodes the message "jake can go out with my beautiful pets and quickly drive to see you" in 86 generations.

TABLE 6.2 Progress of the GA as It Decodes the Secret Message

Generation	Message
1	vhte fhb po olq zjqk ds mehlqjxlu neqr hbg wljftus gcjae qo ree sol
10	cahd bas np pxt iqtf kz edaxtqwxj vdtl asg oxqbhjz grqud tp ldd zpx
20	jakh dar go out wftb mx nhautfcui phty are sufdkix ezfqh to yhh xou
30	faje can gp pvs yish mx reavsikvl ueso and qvicjlx dwize sp oee xpv
40	kaje can dp pvt yitg mx reavtifvl oets anb qvicjlx bwize tp see xpv
50	jake can dp pvt xitg my heavtifvl oets anb qvickly bwize tp see ypv
60	jake can gp put xith my deautiful oets anb quickly bvize tp see ypu
70	jake can go out xith my beautiful pets and quickly dwize to see you
80	jake can go out xith my beautiful pets and quickly dwive to see you
86	jake can go out with my beautiful pets and quickly drive to see you

jectory planning (Davidor, 1991; Davis, 1991; Pack et al., 1996). For example, the goal is to move a robot arm in an efficient manner while avoiding obstacles and impossible motions. The even more complicated scenario of moving the robot arm when obstacles are in motion has been implemented with a parallel version of a GA (Chambers, 1995). Another application simulated two robots fighting. A GA was used to evolve a robot's strategy to defeat its opponent (Yao, 1995).

A robot trajectory describes the position, orientation, velocity, and acceleration of each robot component as a function of time. In this example, the robot is a two-link arm having two degrees of freedom in a plane called the robot workspace (Figure 6.8) (Pack et al., 1996). For calculation purposes this

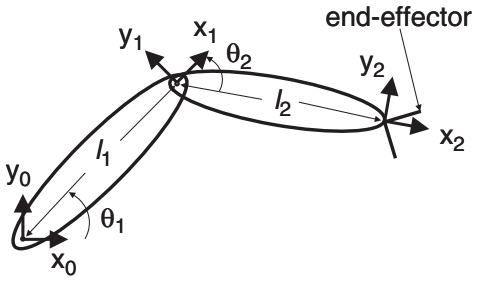

Figure 6.8 Diagram of a two-dimensional robot arm with two links. Link 1 pivots about coordinate system 0 and link 2 pivots about coordinate system 1. Coordinate system 3 has an origin at the end-effector.

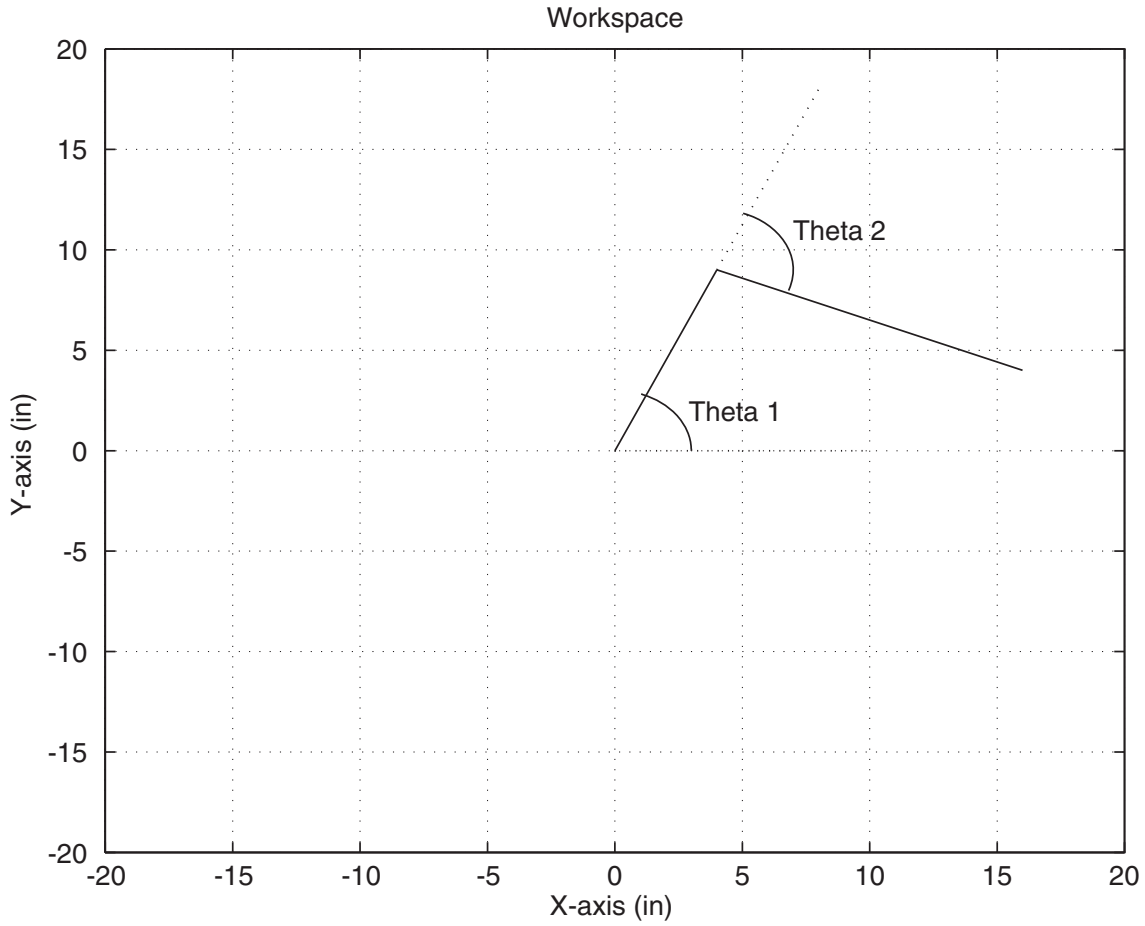

Figure 6.9 Robot arm in Figure 6.6 when more simply described by two line segments.

arm is approximated by two line segments in Cartesian coordinates as shown in Figure 6.9. Each joint has its own local coordinate system that can be related to the base x_0, y_0 coordinate system (located at the shoulder joint). The end-effector or tip of the robot arm is of most interest and has a local coordinate system defined by x_2, y_2. An intermediate coordinate system at the elbow joint is defined by x_1, y_1. Using the Donauit-Hartenberg parameters, one can transform an end-effector position in terms of the x_0, y_0 coordinates by

$$\begin{bmatrix} \cos\theta_{12} & -\sin\theta_{12} & 0 & \ell_1\cos\theta_1 + \ell_2\cos\theta_{12} \\ \sin\theta_{12} & \cos\theta_{12} & 0 & \ell_1\sin\theta_1 + \ell_2\sin\theta_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} \tag{6.3}$$

where

x_2, y_2, z_2 = position of end-effector with respect to coordinate system 2 (end-effector based coordinate system)

x_0, y_0, z_0 = position of end-effector with respect to the base coordinate system

$\cos \upsilon_{12} = \cos \upsilon_1 \cos \upsilon_2 - \sin \upsilon_1 \sin \upsilon_2$

$\sin \upsilon_{12} = \sin \upsilon_1 \cos \upsilon_2 + \cos \upsilon_1 \sin \upsilon_2$

ℓ_1 = length of link 1

ℓ_2 = length of link 1

υ_1 = angle between x_0-axis and link 1

υ_2 = angle between x_1-axis and link 1

Thus knowing the length of the links and the angles allows us to transform any points on the robot arm from the x_2, y_2 coordinate system to the x_0, y_0 coordinate system. Our goal is to find the optimal path for the robot to move through its environment without colliding with any obstacles in the robot workspace.

Although following the end-effector path through Cartesian space (x_0- and y_0-axes) is easiest to visualize, it is not of the most practical value for optimization. First, the calculation of joint angles at each point along the path is difficult. Second, the calculations can encounter singularities that are difficult to avoid. An alternative approach is to formulate the trajectory problem in the configuration space (υ_1- and υ_2-axes) that governs the position of the end-effector. Although numerically easier, it can result in complicated end-effector paths. We will go with the numerically easier version and let the GA optimize in configuration space for this example.

Obstacles in the form of impossible robot joint angle combinations must be taken into account when designing the cost function. It can be shown that point obstacles are contained within an elliptical region in configuration space (Pack et al., 1996). As an example, a point obstacle in the world space transforms into a curved line in configuration space (Figure 6.10) (Pack et al., 1996). This line is nicely contained within an ellipse, and an ellipse is much easier to model as an obstacle.

The cost function is merely the length of the line needed to get from the starting point to the ending point in the configuration space. Rather than attempt to find a continuous path between the start and destination points, piecewise line segments are used. This example establishes a set number of line segments before the GA begins. Consequently the length of all the chro-

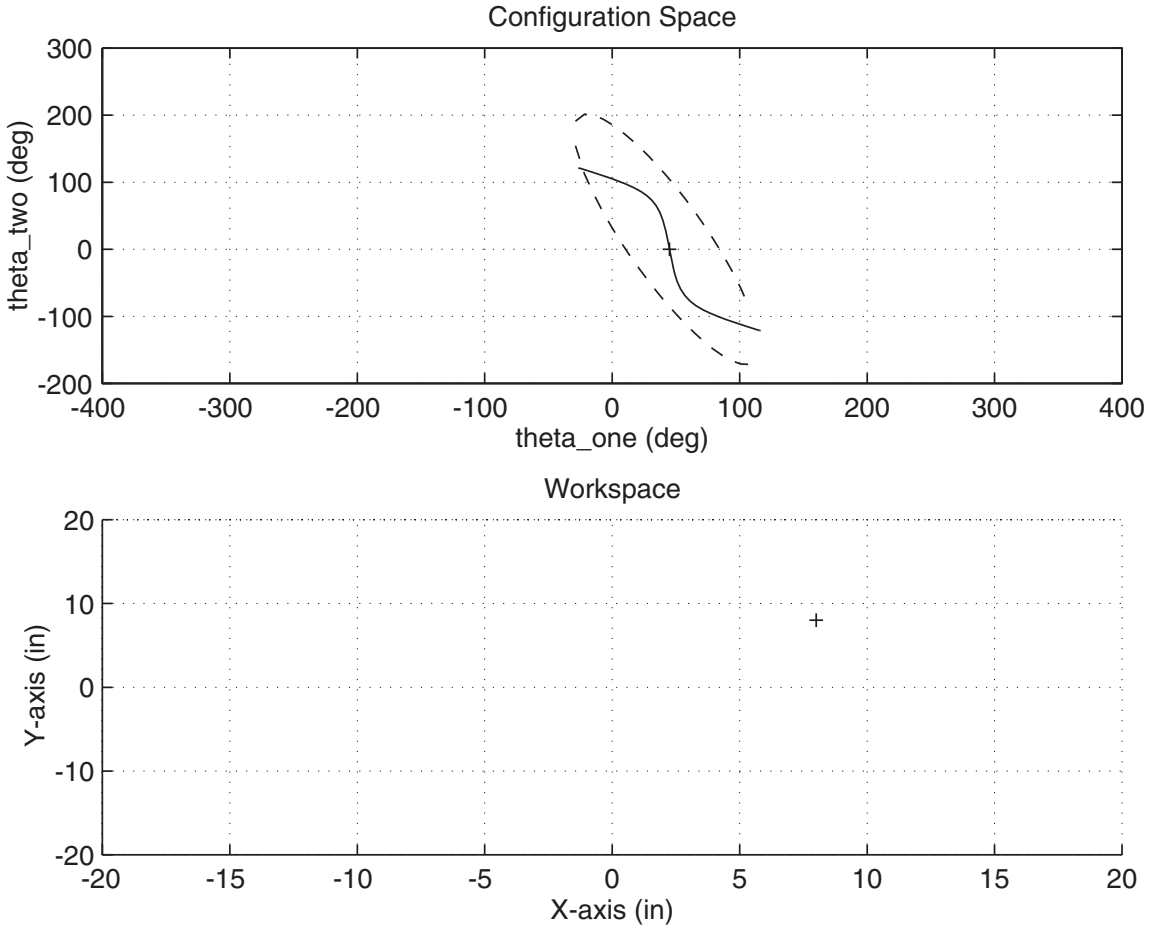

Figure 6.10 Point obstacle in the lower graph transformed into curved line in configuration space in upper graph. This curved line is contained within an elliptical region denoted by the dashed line. In configuration space this ellipse forms a boundary that the robot arm cannot pass through.

mosomes are the same. Others have used variable length chromosomes to find the optimum path. (The reader interested in this approach is referred to Davidor, 1991.)

The first example has four obstacles in the configuration space with start and stop points in obscure parts of the space. Only three intermediate points or four line segments are permitted to complete the shortest path from the start to the finish. The binary GA had $N_{pop} = 80$ members in the population and ran for 10 generations. The first generation had an optimal path length of 11.06 units, as shown in Figure 6.11. After 10 generations the minimum cost reduced to 9.656 units, and its path in configuration space is shown in Figure 6.12. Adding more intermediate points would give the algorithm more freedom to find a better solution.

A second example begins with a real world problem with five-point obstacles in world space that transformed into an ellipse in the configuration space. Again, the binary GA had $N_{pop} = 80$ members in the population and ran for 10 generations. The path after the first generation is shown in Figure 6.13 and

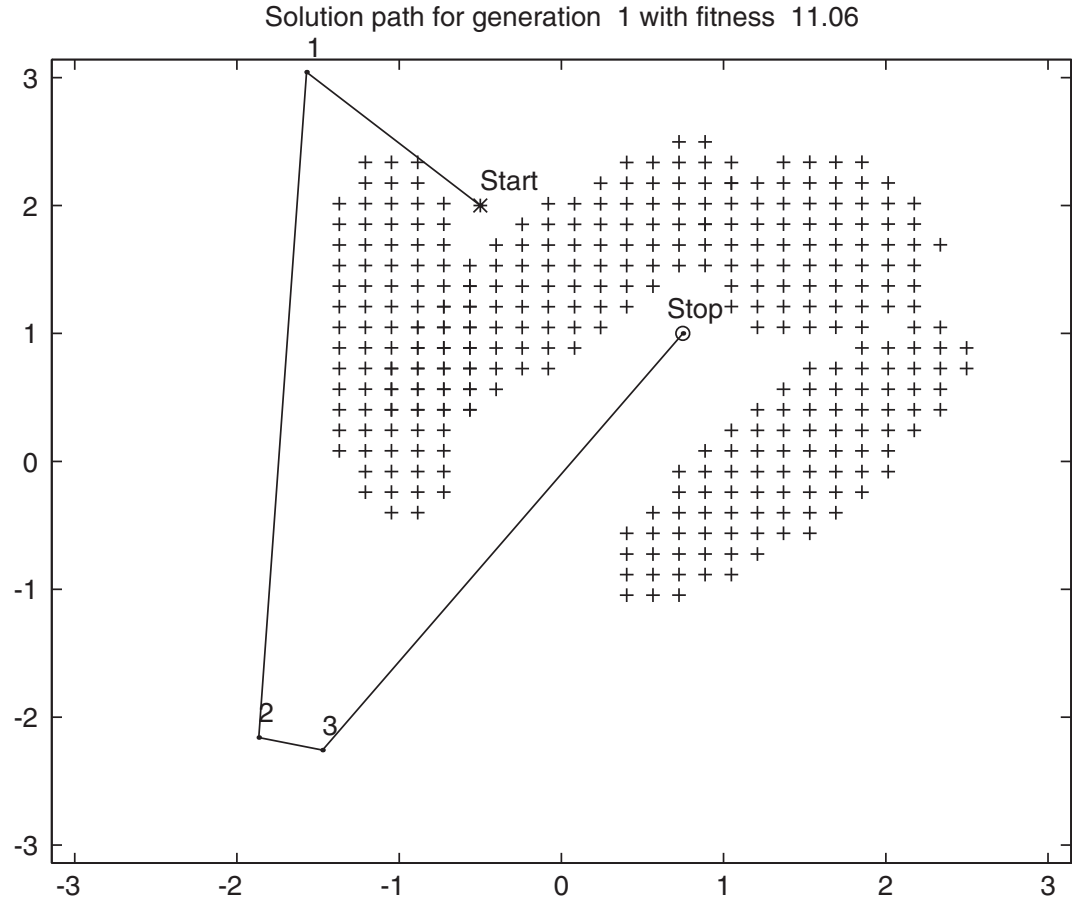

Figure 6.11 The best path between the obstacles after generation 1 is 11.06 units long.

has a cost of 7.321 units. After 10 generations the minimum cost reduced to 6.43 units, and its path in configuration space is shown in Figure 6.14. This optimal solution translates back to world space, as shown in Figure 6.15, where the symbols * and • denote the starting and ending robot end-effector positions, respectively. The elliptical obstacle shapes in Figure 6.13 and 6.14 translate into points (denoted by + signs) in Figure 6.15.

6.5 STEALTH DESIGN

A stealth airplane is difficult to detect with conventional radar. Engineers use a combination of materials, size, orientation, and shaping to reduce the radar cross section of an airplane. The radar cross section of a simple two-dimensional reflector can be modified by the placement of absorbing materials next to it. This type of reflector design is also of interest to satellite antenna manufacturers to lower sidelobe levels and reduce the possibility of interference with the desired signal.

This example demonstrates how to use GAs to find resistive loads that produce the lowest maximum backscatter relative sidelobe level from a per-

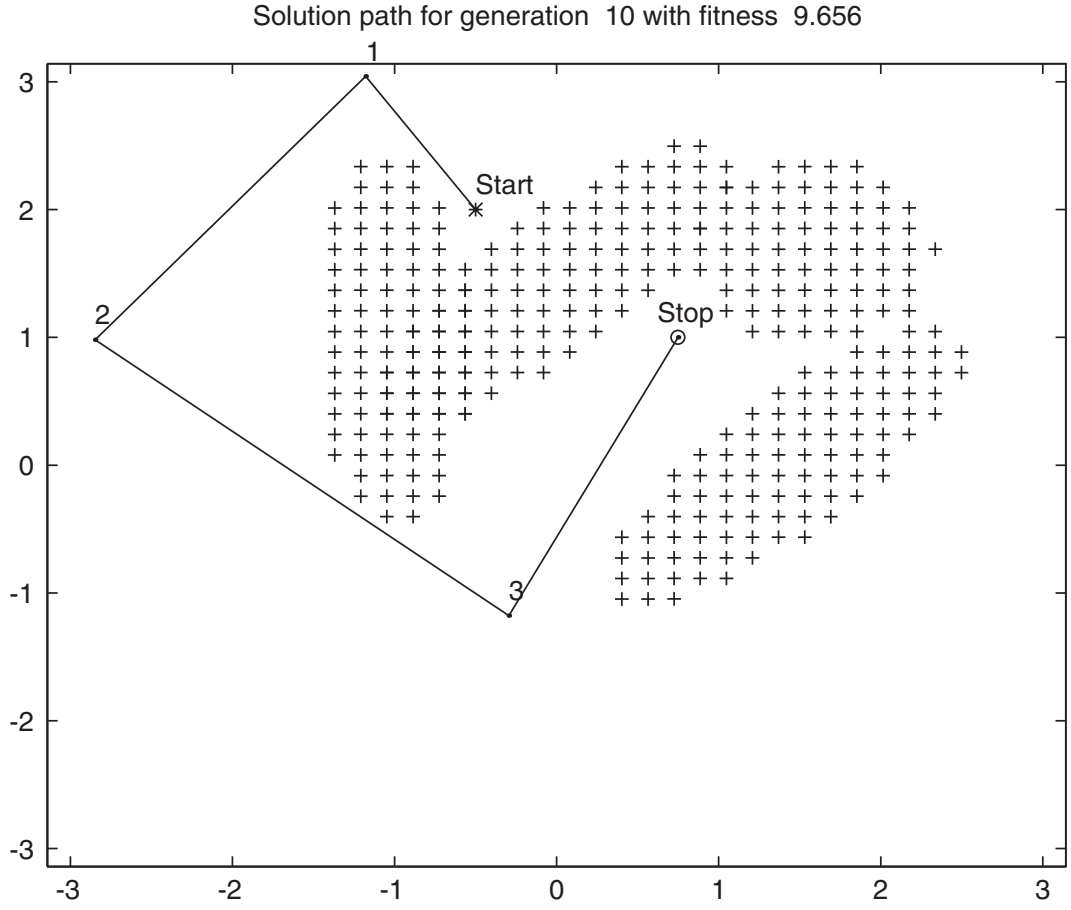

Figure 6.12 The best path between the obstacles after generation 10 is 9.656 units long.

fectly conducting strip. The radar cross section of a 6λ strip appears in Figure 6.16, and its highest relative sidelobe level is about 13.33 dB below the peak of the main beam. The radar cross section is given in terms of dBlambda or decibels above one wavelength. The wavelength is associated with the center frequency of the electromagnetic wave incident on the strip. A model of the loaded strip is shown in Figure 6.17. Assuming the incident electric field is parallel to the edge of the strip, the physical optics backscattering radar cross section is given by Haupt (1995):

$$\sigma(\phi) = \frac{k}{4}\left|4asSa(2kau) + \sum_{n+1}^{N}\left(\frac{2b_n s}{0.5+\eta_n s}\right)Sa(kb_n u)\cos\left[2k\left(a + \sum_{m+1}^{n-1} b_m + \frac{b_n}{2}\right)u\right]\right|^2 \tag{6.4}$$

where

$s = \sin\phi$

$u = \cos\phi$

$2a =$ width of perfectly conducting strip

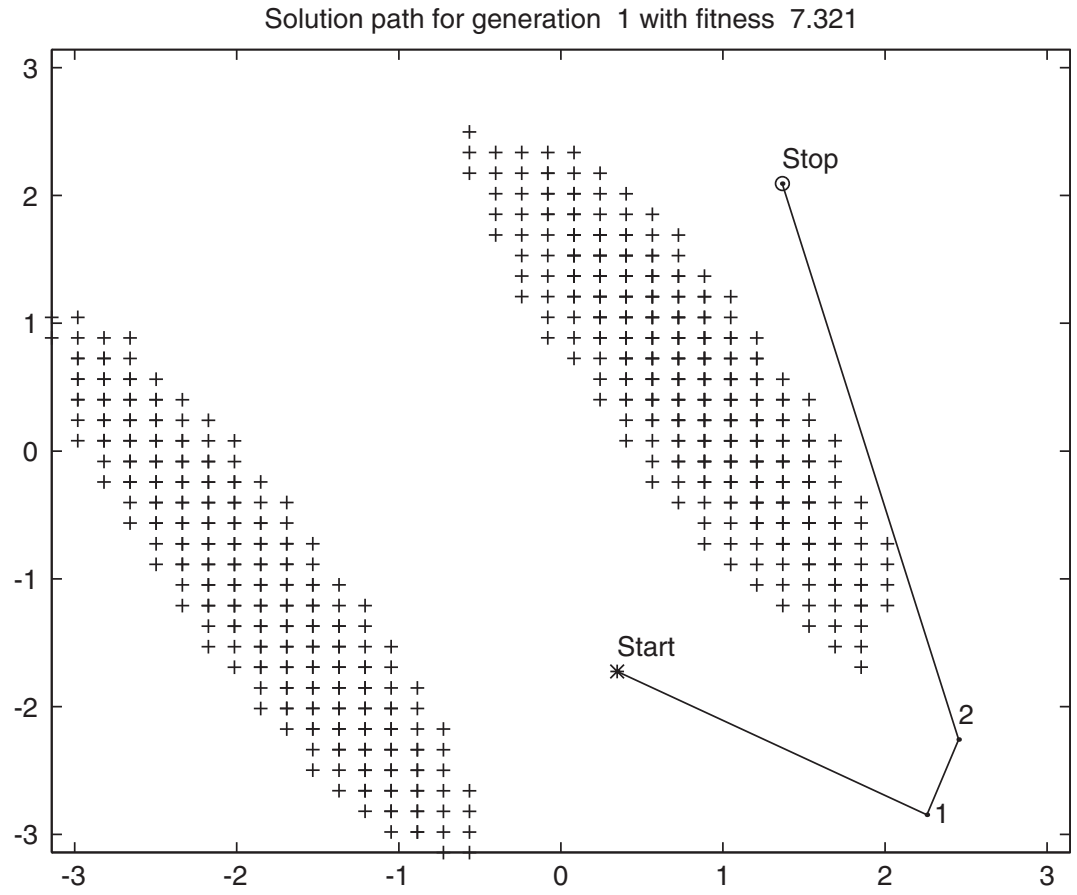

Figure 6.13 The best path between the obstacles after generation 1 has a length of 7.32 units.

$b_n =$ width of load $n = \sum_{m=1}^{B_w} b_w[m]2^{1-m}W$

$\eta_n =$ resistivity of load $n = \sum_{m=1}^{B_r} b_r[m]2^{1-m}R$

$Sa = (\sin x)/x$

$B_w, B_r =$ number of bits representing the strip width and resistivity

$b_w, b_r =$ array of binary digits that encode the values for the strip widths and resistivities.

$W, R =$ width and resistivity of the largest quantization bit

Eight resistive loads are placed on each side of a perfectly conducting strip that is 6λ wide. The widths and resistivities of these loads are optimized to reduce the maximum relative sidelobe level of the radar cross section. Both the width and resistivity of each load are represented by 5 quantization bits, and $W = 1$ and $R = 5$. The optimized values arrived at by the GA are

$$\eta_n = 0.16, 0.31, 0.78, 1.41, 1.88, 3.13, 4.53, 4.22$$

$$w_n = 1.31, 1.56, 1.94, 0.88, 0.81, 0.69, 1.00, 0.63\lambda$$

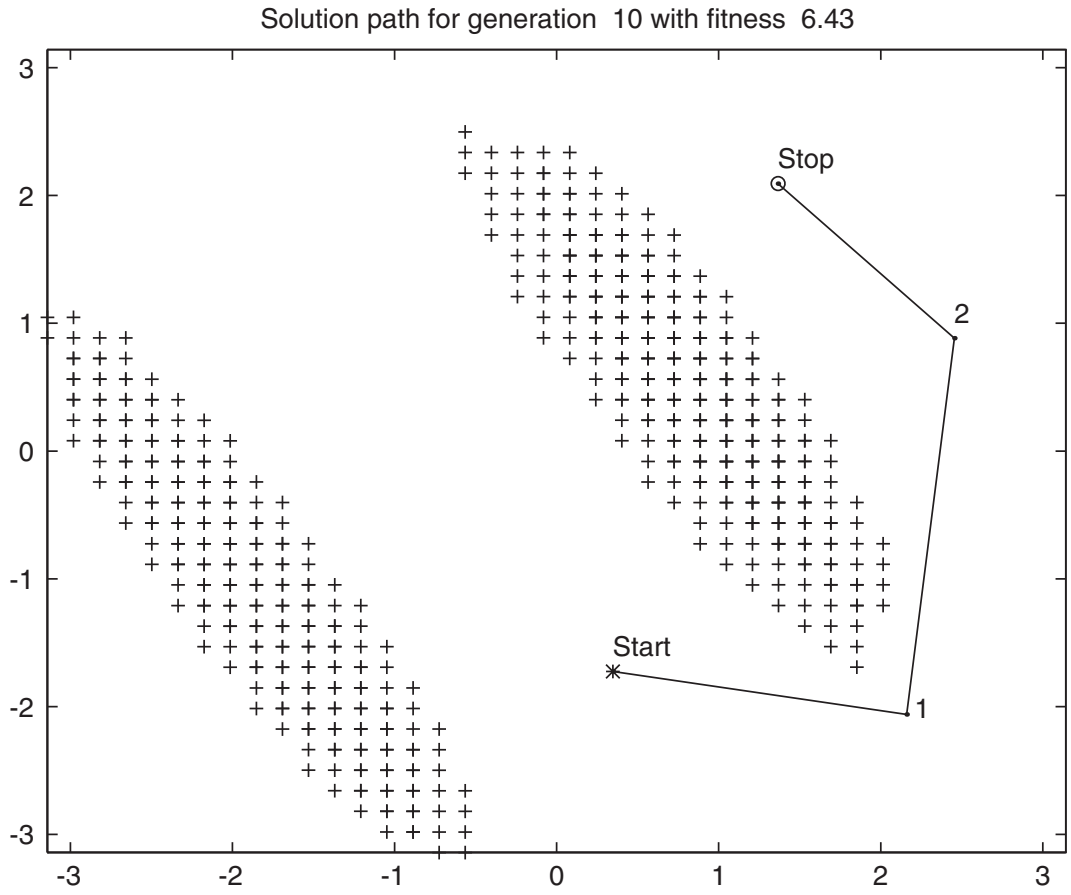

Figure 6.14 The best path between the obstacles after generation 10 has a length of 6.43 units.

These values result in a maximum relative radar cross section sidelobe level of –33.98 dB. Figure 6.18 shows the optimized radar cross section. The peak of the mainbeam is about 6 dB higher than the peak of the mainbeam of the 6λ perfectly conducting strip radar cross section in Figure 6.16. In exchange for the increase in the mainbeam, the peak sidelobe level is 15 dB less than the peak sidelobe level in Figure 6.16. In other words, compared to the 6λ perfectly conducting strip, this object is easier to detect by a radar looking at it from the broadside, but it is more difficult to detect looking off broadside.

The resistive loads attached to the perfectly conducting strip were also optimized using a quasi-Newtonian method that updates the Hessian matrix using the Broyden–Fletcher–Goldgarb–Shanno (BFGS) formula. A true gradient search was not used because the derivative of (6.4) is difficult to calculate. The quasi-Newtonian algorithm performed better than the GA for 10 or less loads. Using the quasi-Newtonian method in the previous example resulted in a maximum relative sidelobe level of –36.86 dB. When 15 loads were optimized, GAs were clearly superior.

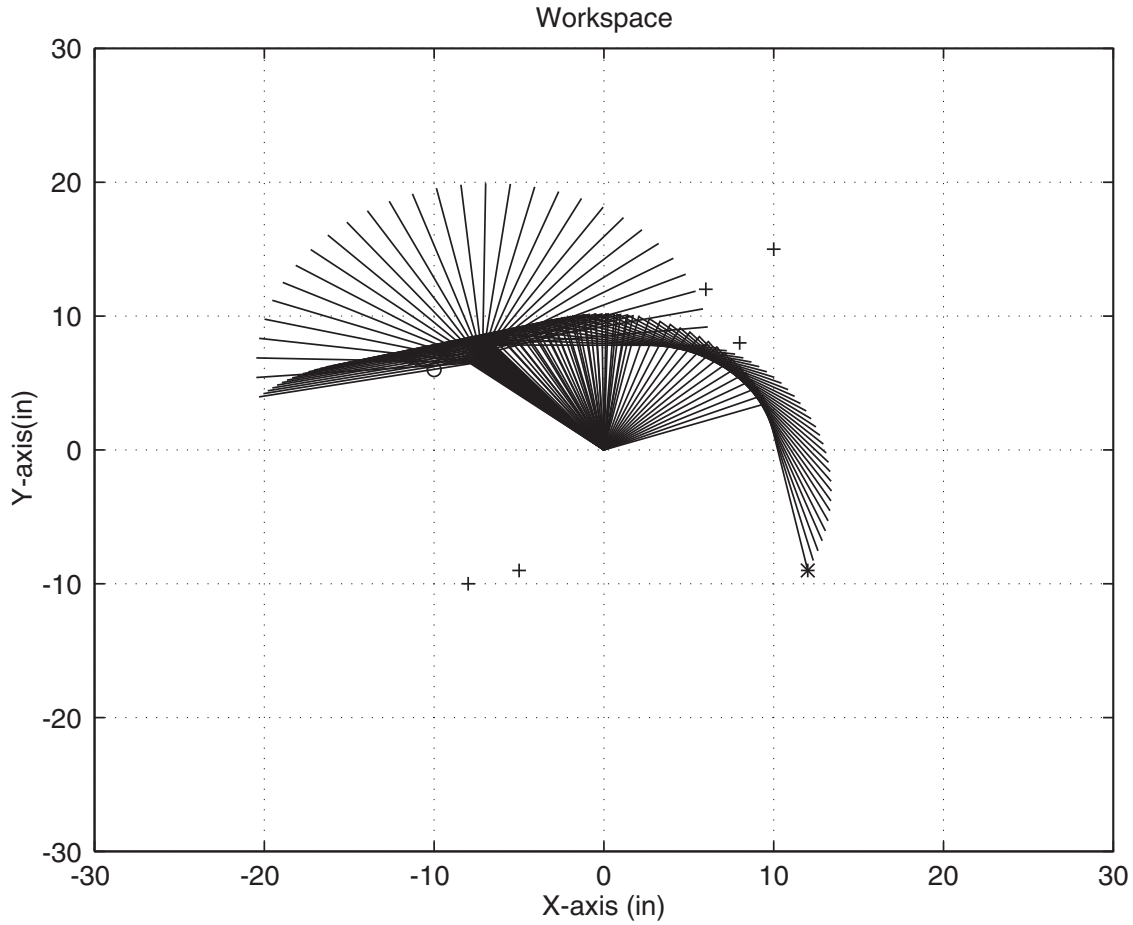

Figure 6.15 Actual movement of the robot arm through the obstacles in world space (denoted by + signs). The plus signs transform into the elliptical regions shown in configuration space (Figures 6.13 and 6.14).

6.6 BUILDING DYNAMIC INVERSE MODELS—THE LINEAR CASE

Inverse models are becoming increasingly common in science and engineering. Sometimes we have collected large amounts of data but have not developed adequate theories to explain the data. Other times the theoretical models are so complex that it is extremely computer intensive to use them. Whichever the circumstance, it is often useful to begin with available data and fit a stochastic model that minimizes some mathematical normed quantity, that is, a cost. Our motivation here lies in trying to predict environmental variables. In recent years many scientists have been using the theory of Markov processes combined with a least squares minimization technique to build stochastic models of environmental variables in atmospheric and oceanic science (Hasselmann, 1976; Penland, 1989; Penland and Ghil, 1993). One example is predicting the time evolution of sea surface temperatures in the western Pacific Ocean as a model of the rises and falls of the El Niño/Southern Oscillation (ENSO) cycle. This problem proved challenging. However, stochastic

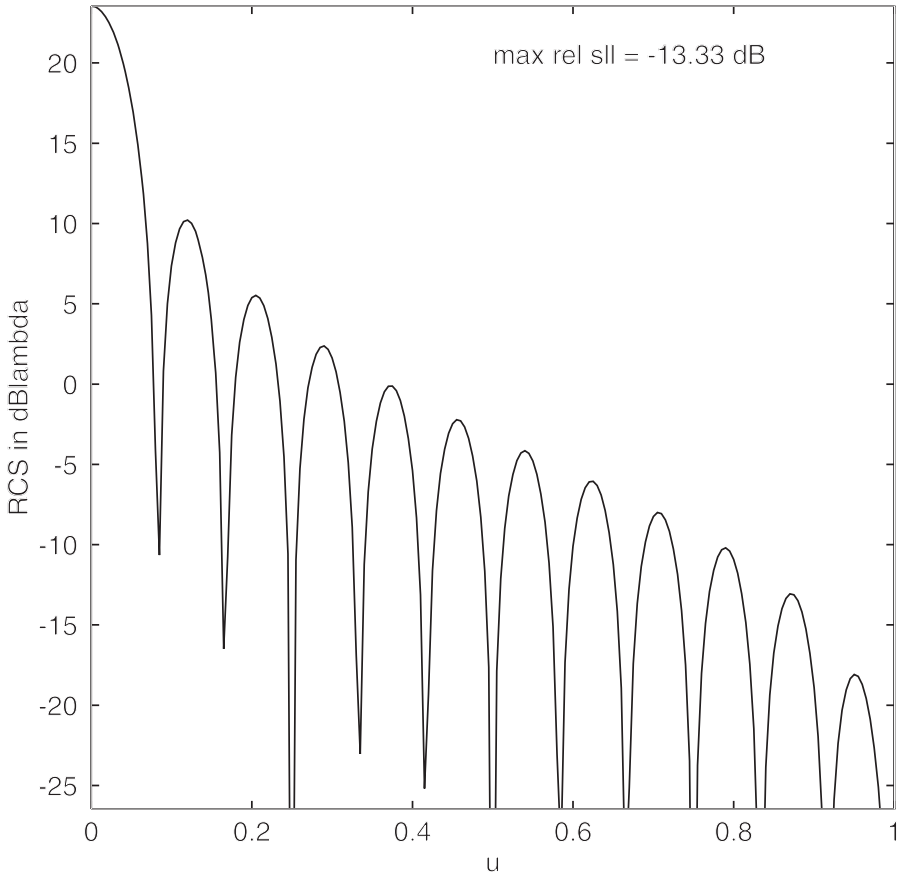

Figure 6.16 Radar cross section of a 6λ wide perfectly conducting strip.

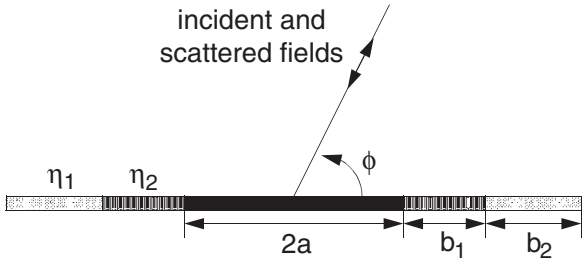

Figure 6.17 Diagram of a perfectly conducting strip with symmetric resistive loads placed at its edges.

models have performed as well as the dynamical ones in predicting future ENSO cycles (Penland and Magorian 1993; Penland, 1996). Another application involves predicting climate. We now build very complex climate models that require huge amounts of computer time to run. There are occasions when it would be useful to predict the stochastic behavior of just a few of the key variables in a large atmospheric model without concern for the details of day-to-day weather. One such application is when an atmospheric climate model is coupled to an ocean model. Since the time scale of change of the atmosphere is so much faster than that of the ocean, its scale dictates the Courant-Friedichs-Levy criteria, which limits the size of the allowable time step. For some problems it would be convenient to have a simple stochastic model of

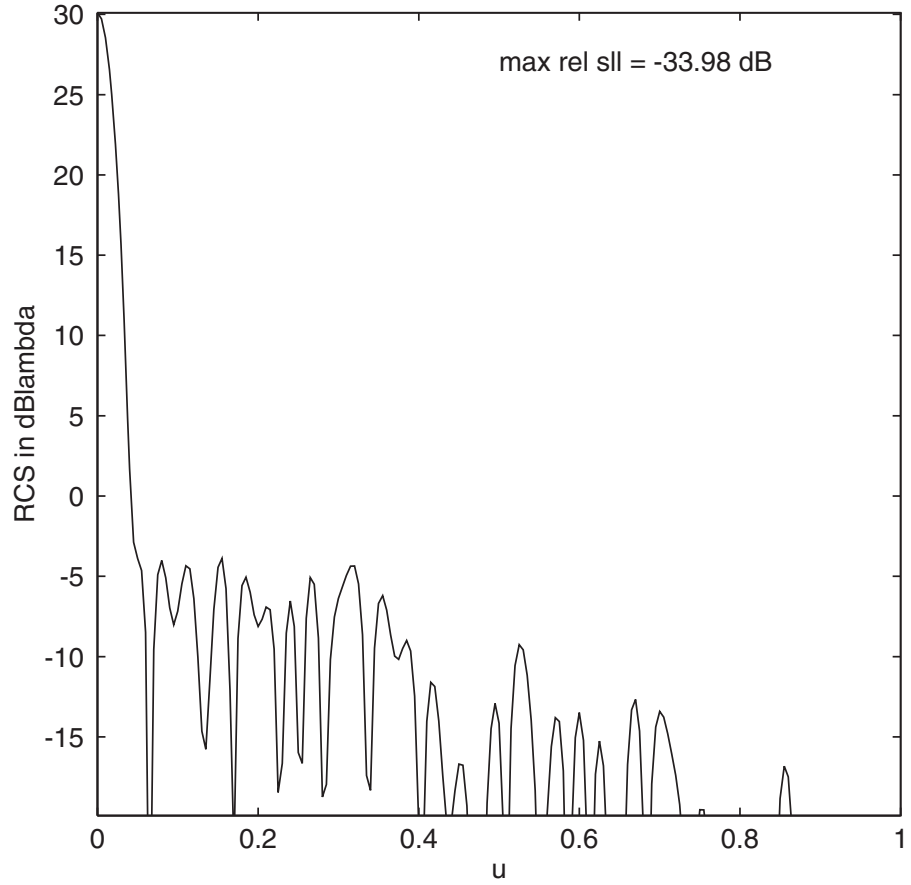

Figure 6.18 Radar cross section of the 6λ wide strip with 8 resistive loads placed at its edges. The level between the maximum sidelobe and the peak of the mainbeam is –33.98 dB, which is a 20.78 dB reduction.

the atmosphere to use in forcing an ocean model. Recent attempts have shown that such models are possible and perhaps useful for computing responses to forcing (Branstator and Haupt, 1998). However, the least squares techniques typically used to build these models assume a Markov process. This assumption is not valid for most environmental time series. Would a different method of minimizing the function produce a better match to the environmental time series? This is an interesting question without a clear answer. Before answering it using large climate models, it is convenient to begin with simple low-dimensional models of analytical curves.

We use a GA to compute parameters of a model of a simple curve that is parametric in time. In particular, we wish to fit a model

$$\frac{dx}{dt} = \mathbf{A}\mathbf{x} \tag{6.5}$$

to a time series of data. Here $\mathbf{x}$ is an N-dimensional vector, $dx/dt = \mathbf{x_t}$ is its time tendency, and $\mathbf{A}$ is an $N \times N$ matrix relating the two. Note that most first-order

time-dependent differential equations can be discretized to this form. Our goal is to find the matrix **A** that minimizes the cost

$$\cos t = \left\langle (\mathbf{x_t} - \mathbf{Ax})^p \right\rangle \tag{6.6}$$

where P is any appropriate power norm that we choose. The least squares methods use $P = 2$, or an L^2 norm. The angular brackets denote a sum over all of the data in the time series.

An example time series is a spiral curve generated by $(X, Y, Z) = (\sin(t), \cos(t), t)$, with $t = [0, 10\pi]$ in increments of $\pi/50$. The time evolution of this curve appears in Figure 6.19. Note that for this problem, computation of the cost function requires a summation over 500 time increments. However, even with the reasonably large population size and number of generations (70) that we computed, the computer time required was not excessive. (Note that for a bigger problem with a longer averaging period, this would no longer be true.) A continuous GA is applied to this curve with a population size of $N_{pop} = 100$, and a mutation rate of $\mu = 0.2$. Since the GA is oblivious to which value of P we choose, we experimented a bit and found the best results for moderate P. The solution displayed here uses $P = 4$. Evolution of the minimum cost appears in Figure 6.20. We notice that the cost decreases several orders of magnitude over the 70 generations. The result appears in Figure 6.21. We see that the general shape of the spiral curve is captured rather well. The bounds in X and Y are approximately correct, but the evolution in $Z = t$ is too slow. We found

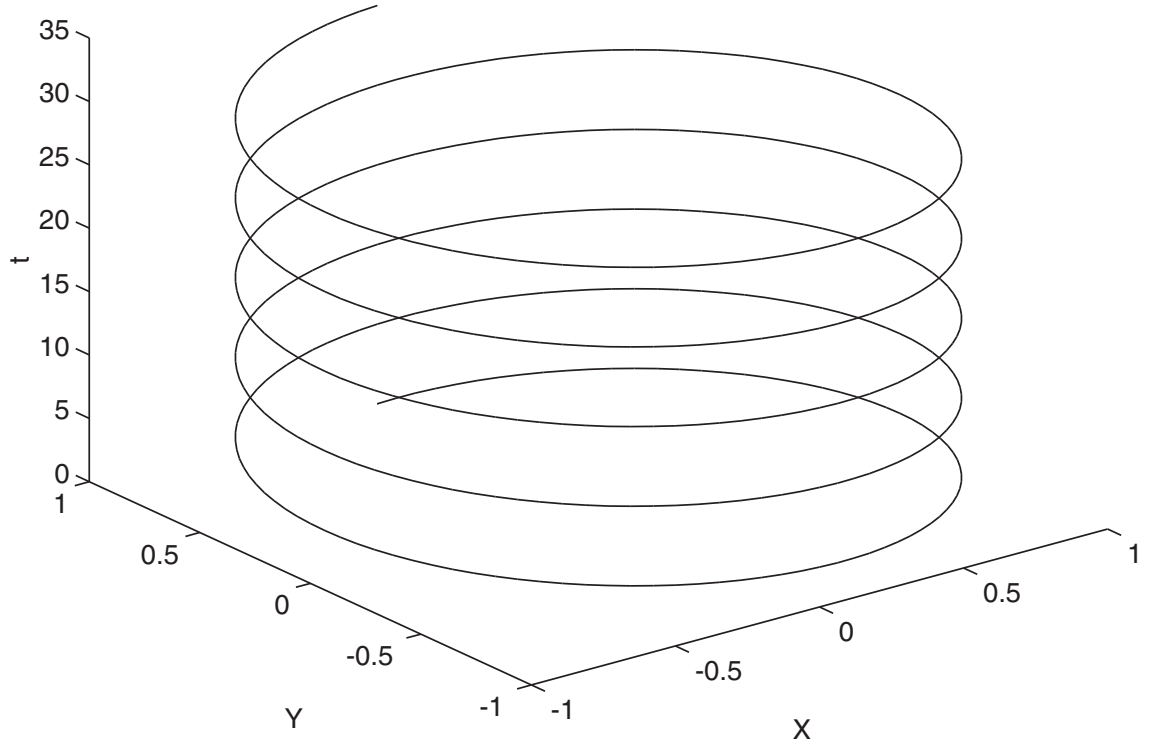

Figure 6.19 Spiral curve specified by $(X, Y, Z) = [\cos(t), \sin(t), t]$ for $t = [1, 10\pi]$.

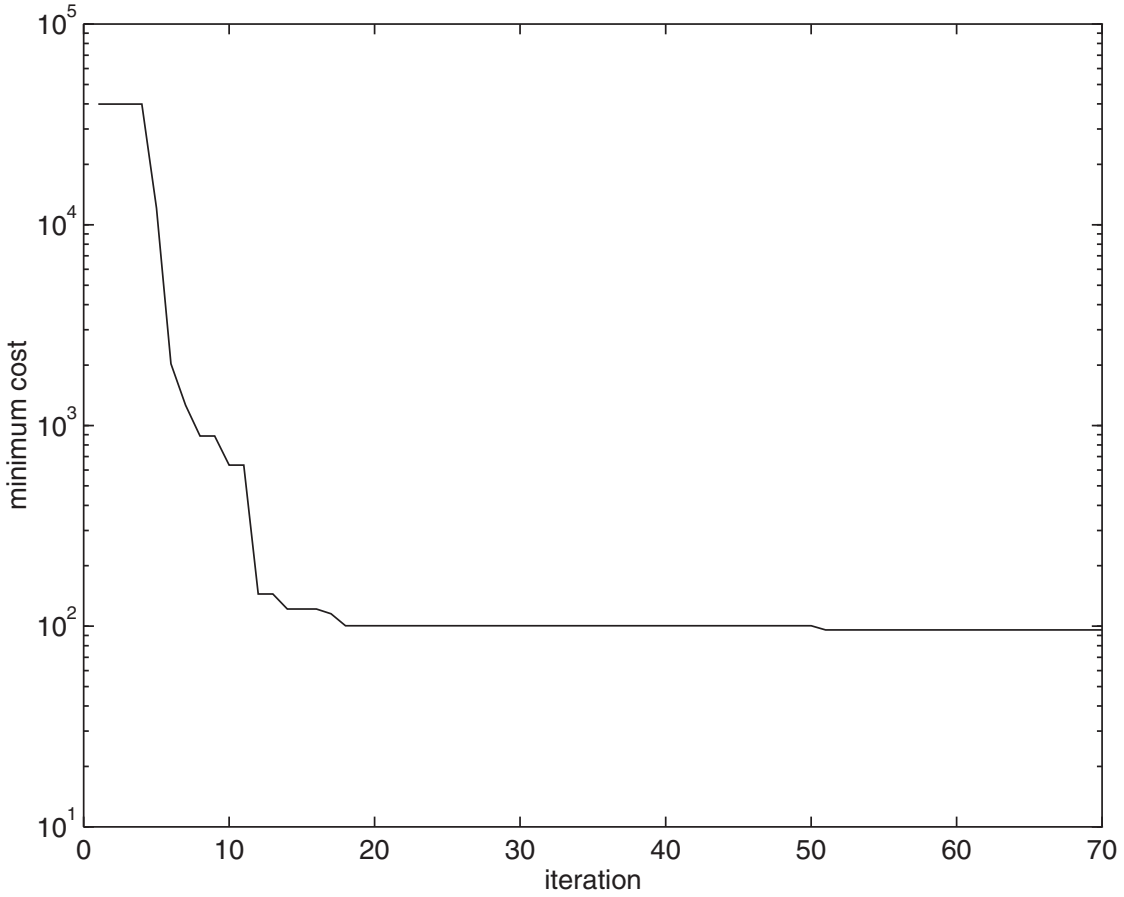

Figure 6.20 Evolution of the mimimum cost of the genetic algorithm, which produces a dynamical inverse model of the spiral curve in Figure 6.19.

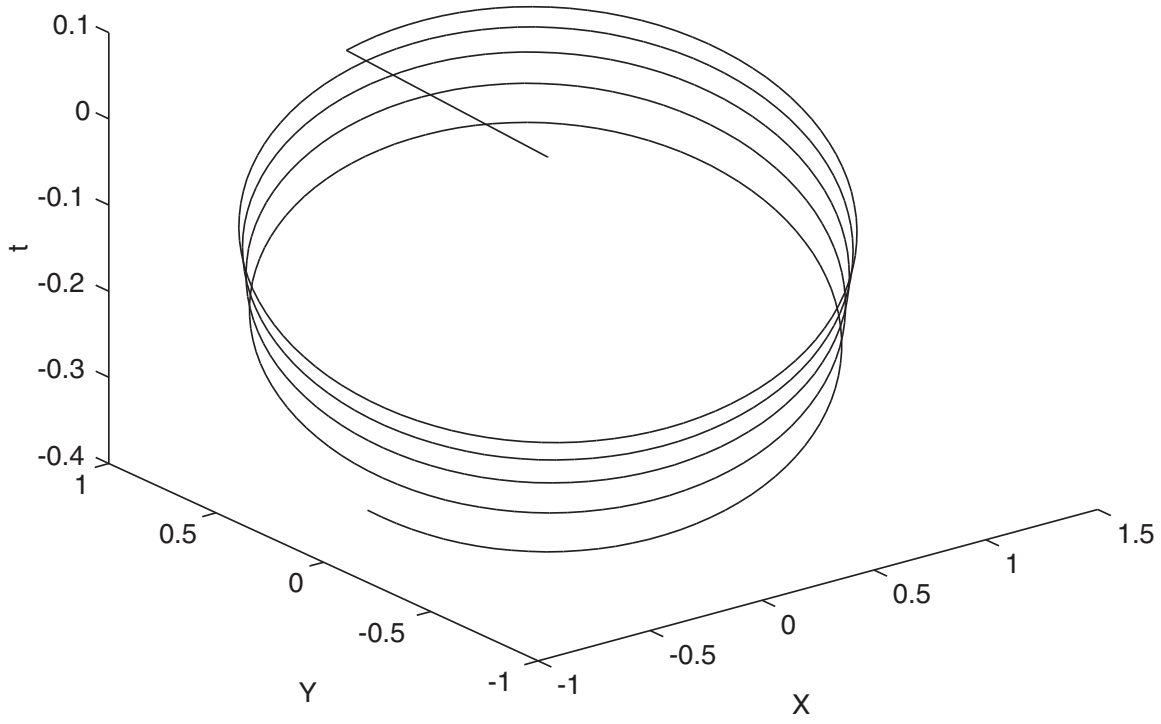

Figure 6.21 Genetic algorithm's dynamical fit of a model based on the time series of the spiral curve in Figure 6.19.

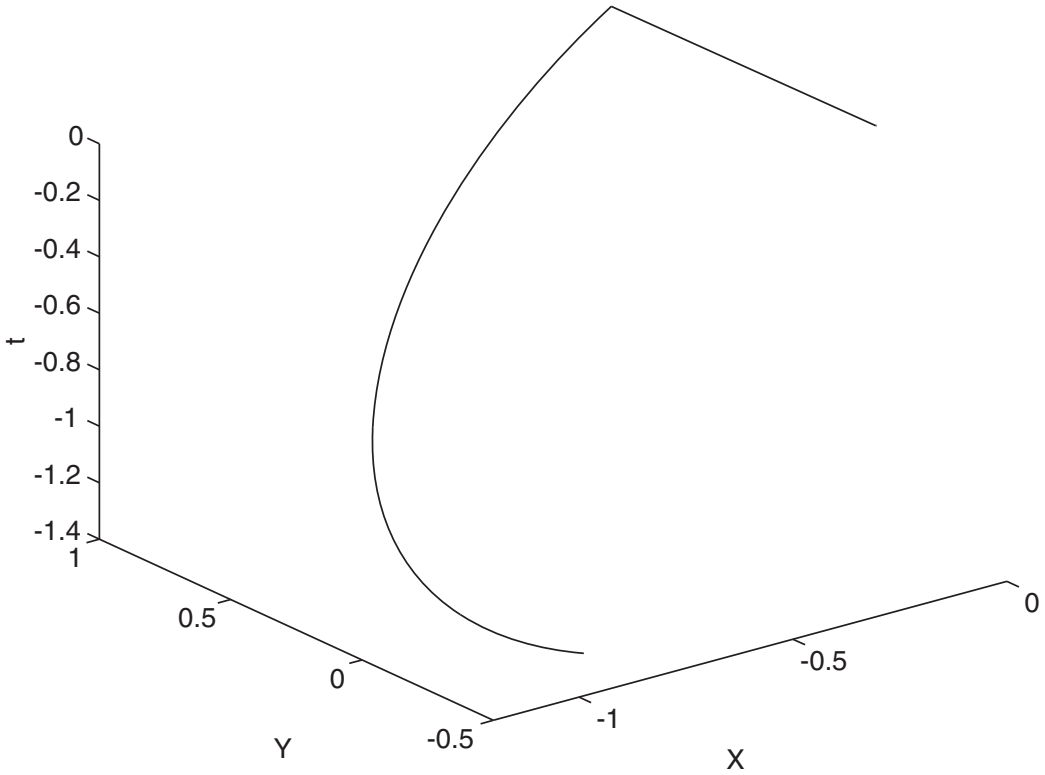

Figure 6.22 A linear least square dynamical fit of a model based on the time series of the spiral curve in Figure 6.19.

that this aspect of our model was rather difficult to capture. In terms of dynamical systems, we were able to find the attractor but not able to exactly model the evolution along it. For comparison a standard least squares technique is used to solve the same problem. The result appears as Figure 6.22. We can see that the least squares method could not even come close to capturing the shape of the attractor. Of course, we can fine-tune the least squares method by adding a noise term in the cost function. We can do that for the GA as well. The advantage of the GA is that it is easy to add complexity to the cost function. Feeding this simple model more variables adds nothing to the solution of the problem, since it can be completely specified with the nine degrees of freedom in the matrix.

6.7 BUILDING DYNAMIC INVERSE MODELS—THE NONLINEAR CASE

An enhancement to the application of the previous section on empirical modeling is including higher order terms in the calculation. Many dynamical problems are not linear in nature, so we cannot expect them to reproduce

the shape of the data using linear stochastic models. We saw this in the traditional least square fit to the spiral model in the preceding section (see Figure 6.22). The spiral model was sinusoidal and that behavior could not be captured with the linear fit. In this section we expand the inverse model to include quadratically nonlinear terms, often the form that appears in fluid dynamics problems.

The example problem that we consider is predator-prey model (also known as the Lotka-Volterra equations), namely

$$\frac{dx}{dt} = ax - bxy$$

$$\frac{dy}{dt} = -cy + dxy \tag{6.7}$$

where x is the number of prey and y the number of predators. The prey growth rate is a while the predator death rate is c. Variables b and d characterize the interactions. Equations (6.7) were integrated using a fourth order Runge Kutta with a time step of 0.01 and variables $a = 1.2$, $b = 0.6$, $c = 0.8$, and $d = 0.3$. The time series showing the interaction between the two appears in Figure 6.23. This time series serves as the data for computing the inverse models. The phase space plot is shown in Figure 6.24 where we see the limit cycle between the predators and the prey.

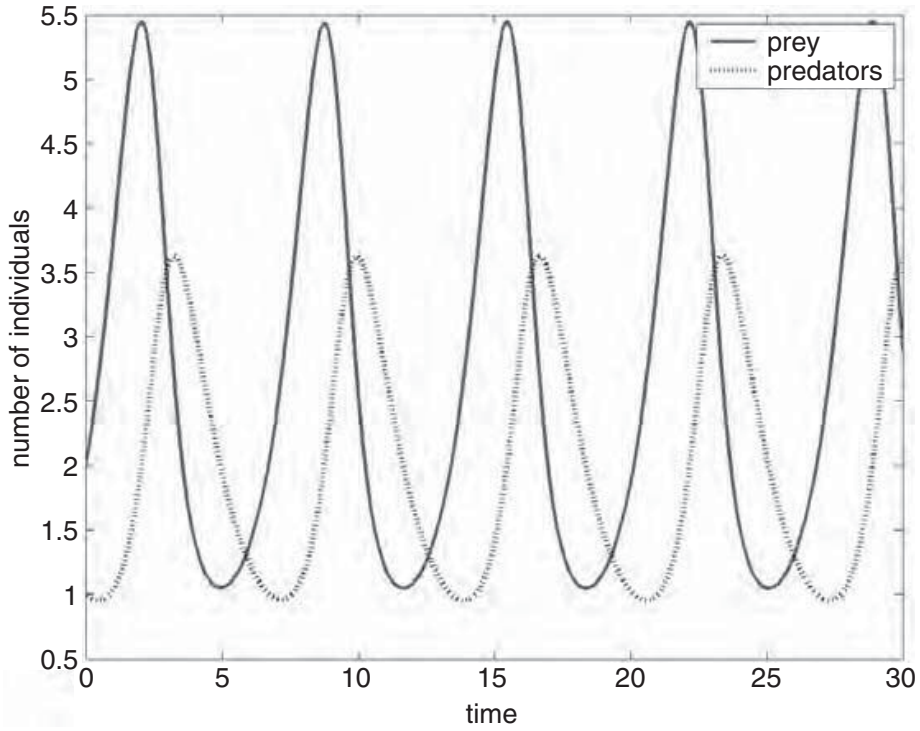

Figure 6.23 Time series showing predator and prey variations over time according to (6.7).

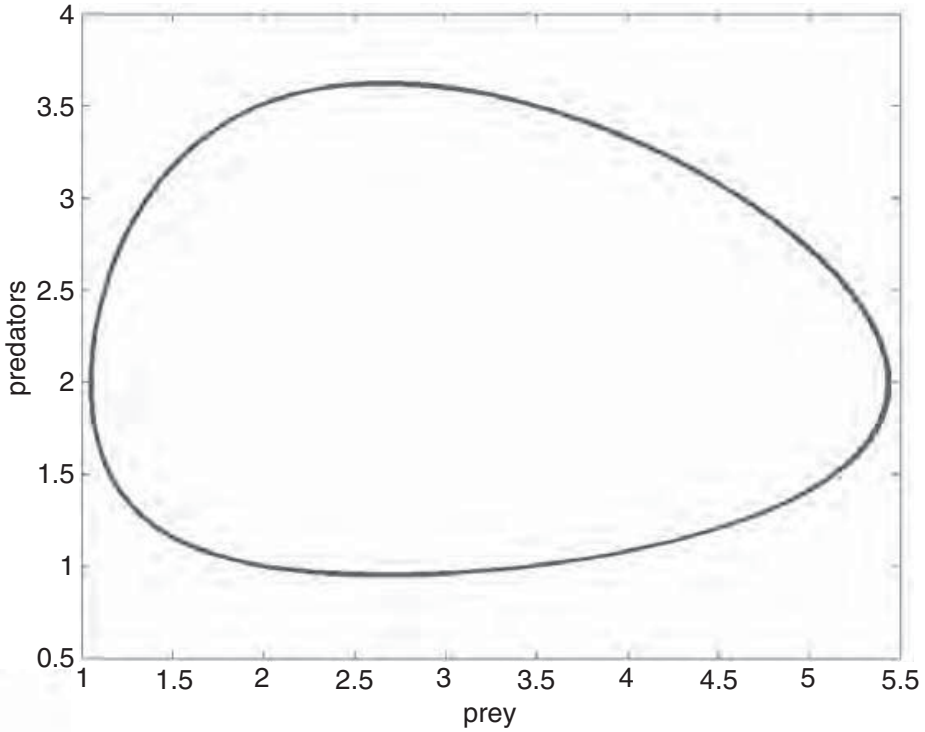

Figure 6.24 State space showing predator-prey interactions.

To fit a linear model, we would use (6.5). The least squares fit to the linear model produces the time series of Figure 6.25. We note that the agreement is quite poor, as one would expect given that the system (6.7) is highly nonlinear. With no nonlinear interaction available, the number of prey grows while the number of predators remains stationary.

To obtain a more appropriate nonlinear fit, we now choose to model the data with a nonlinear model:

$$x_t = Nx^T x + Ax + C \tag{6.8}$$

We allow nonlinear interaction through the nonlinear third-order tensor operator, N, and include a constant, C. Although one can still find a closed form solution for this nonlinear problem, it involves inverting a fourth-order tensor. For problems larger than this simple two-dimensional one, such an inversion is not trivial. Therefore we choose to use a GA to find variables that minimize the least square error between the model and the data. The cost function is

$$\text{cost} = \left\langle \left(x_t - Nx^T x + Ax + C\right)^p \right\rangle \tag{6.9}$$

The GA used a population size of 100, and a mutation rate of 0.2. A time series of the solution as computed by the GA appears in Figure 6.26. Note that although the time series does not exactly reproduce the data, the oscillations

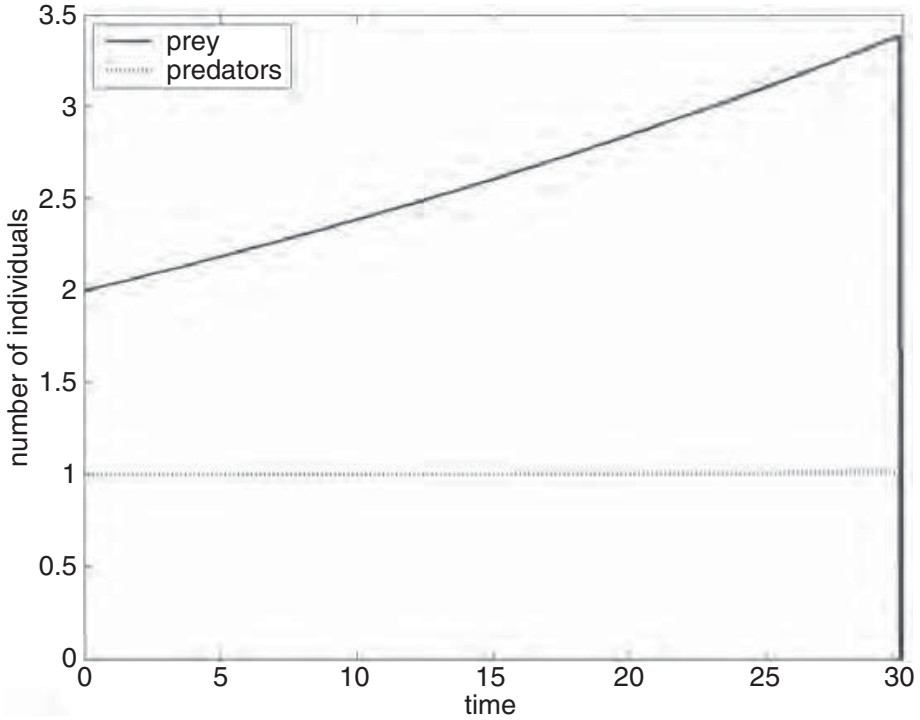

Figure 6.25 Least squares time series fit to predator-prey model.

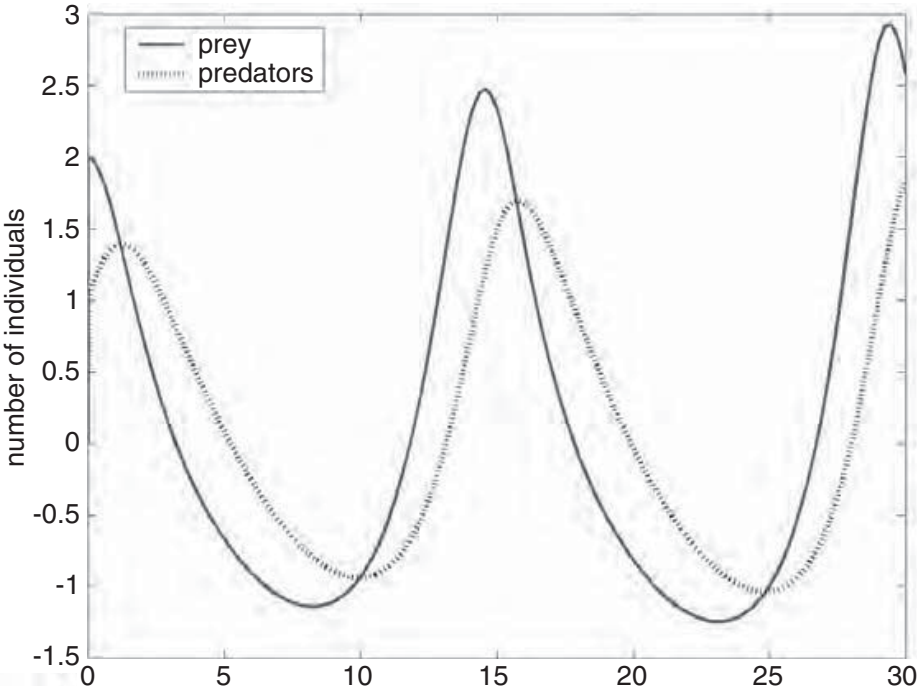

Figure 6.26 Time series of predator-prey interactions as computed by the genetic algorithm.

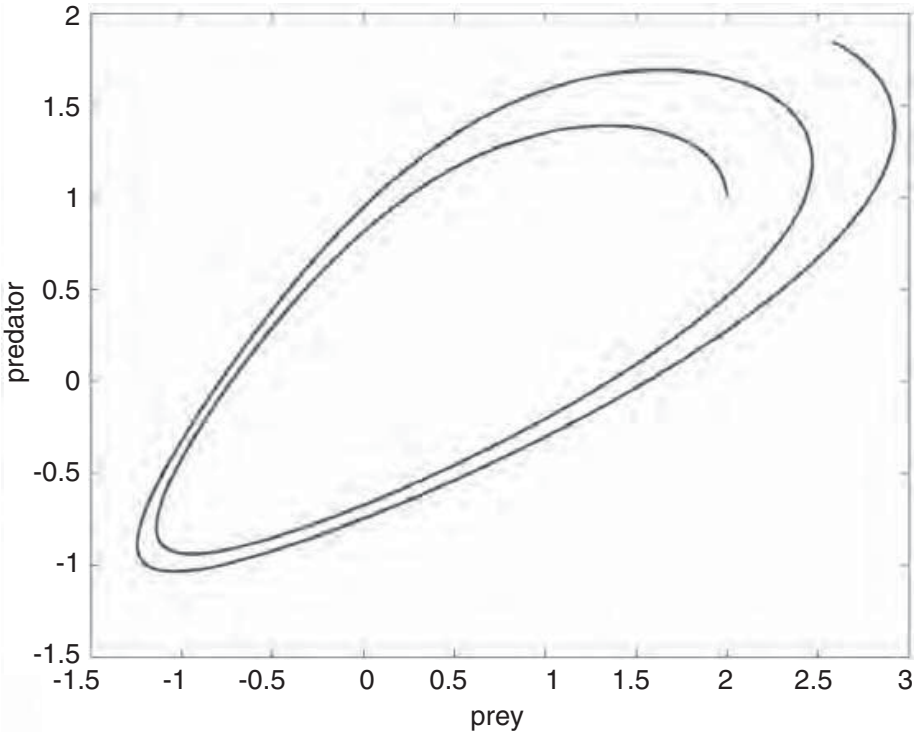

Figure 6.27 The predator-prey relation in state space as computed by the nonlinear model with parameters fit by the GA.

are reproduced including the phase shift of roughly a quarter period. The wavelength is not exact and the amplitudes grow in time, indicating an instability. This instability is likely inherent in the way that the model is matched. However, the reproduction of such a difficult nonlinear system is amazing given the comparison to traditional linear models.

The state space plot appears in Figure 6.27. The limit cycle is not exactly reproduced. The nonlinear model instead appears unstable and slowly grows. For comparison, however, the linear least squares model resulted in a single slowly growing curve (Figure 6.25) that was a much worse match. The GA nonlinear model was able to capture the cyclical nature of the oscillations, a huge improvement.

Finally Figure 6.28 shows the convergence of the GA for a typical run of fitting the nonlinear model (6.8) to the data. Due to their random nature, the results of a GA are never exactly the same. In particular, the convergence plots will differ each time. However, the results are quite reliable. For this simple two-dimensional nonlinear system describing predator-prey relations, the GA fit the variables of a nonlinear model so that the attractor was much better produced than by a traditional linear least squares fit. Although the match is not perfect, the nonlinear GA model captures the essence of the dynamics.

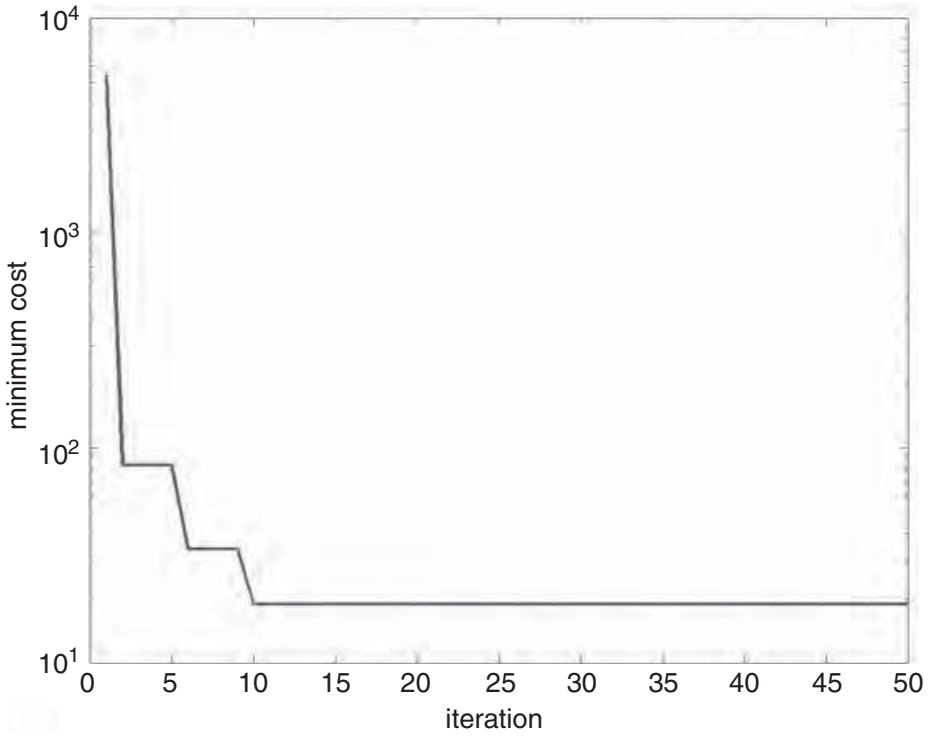

Figure 6.28 Evolution of the minimum cost for the GA fit to the nonlinear model parameters.

6.8 COMBINING GAs WITH SIMULATIONS—AIR POLLUTION RECEPTOR MODELING

Now we move into problems that require running some sort of simulation as part of the cost function. In both design and in fitting some inverse models, we often know something about the physics of the problem that can be formulated into a numerical simulation. That simulation is often necessary to evaluate the quality of the chosen design or fitting model. For instance, several engineers have designed airplanes wings and airfoils by optimizing the shape through testing with a full fluid dynamics model (e.g., Karr, 2003; Obayashi et al., 2000).

The problem demonstrated here begins with air pollution data monitored at a receptor. Given general information about the regional source characteristics and meteorology during the period of interest, we wish to apportion the weighted average percentage of collected pollutant to the appropriate sources. This problem is known as air pollution receptor modeling. More specifically, our example problem is to apportion the contribution of local sources of air pollution in Cache Valley, Utah, to the measured pollutants received at a monitoring station owned by the Utah Department of Air Quality. This demonstration problem uses sixteen sources surrounding the receptor as seen in Figure 6.29. Of course, the spread and direction of pollutant plumes are highly dependent on wind speed and direction in addition to other meteorological

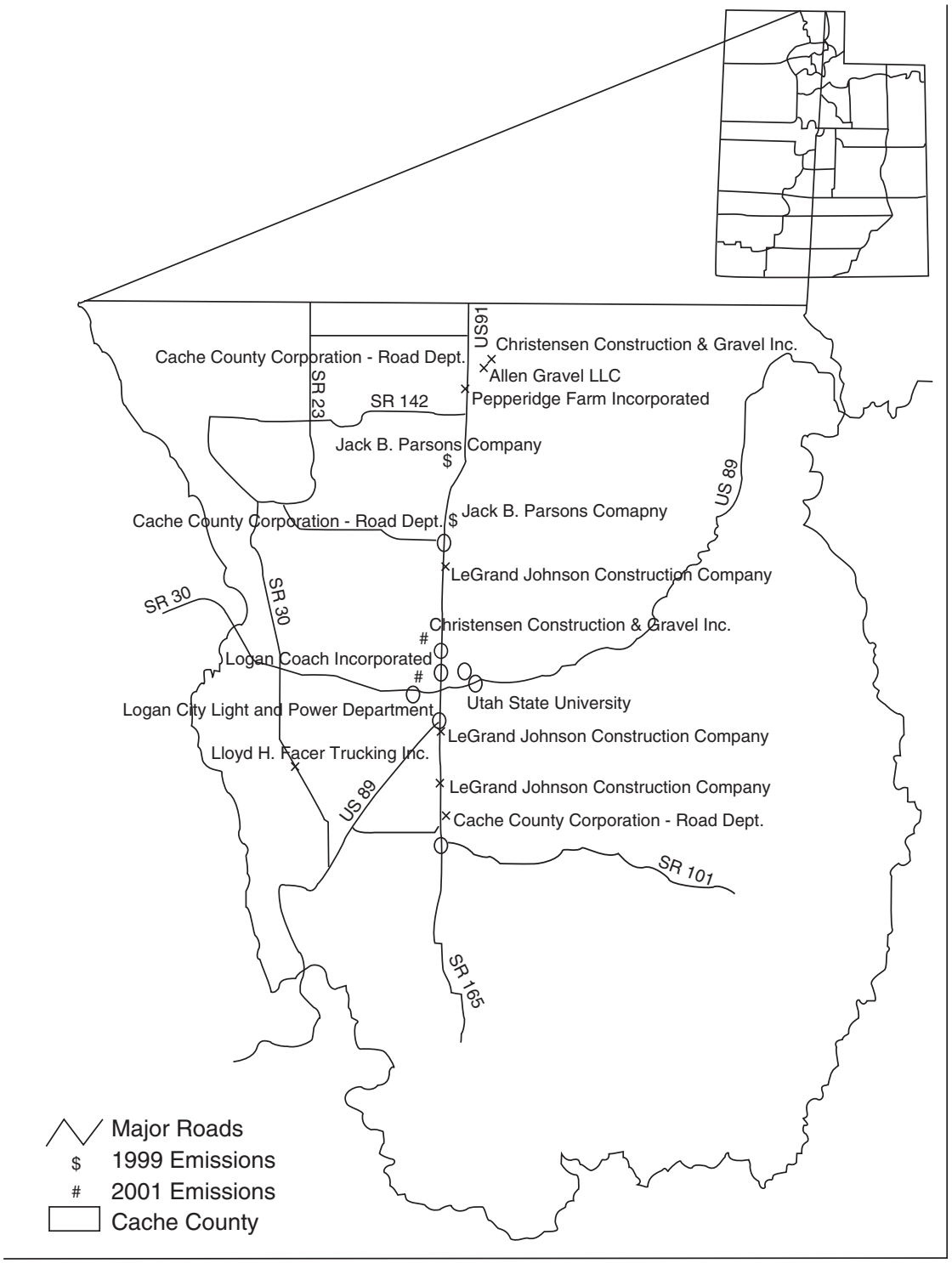

Figure 6.29 Air pollution sources in Cache Valley, Utah. The receptor is marked with an #.

variables. Cartwright and Harris (1993) used a GA to apportion sources to pollutant data at receptors. They began with a chemical mass balance model of the form

$$M \bullet S = R \tag{6.10}$$

where M is the source profile matrix, which denotes the effective strength of pollutant from a given source at the receptor; S is the fraction of the source that contributes to the concentration at the receptor, the unknown apportionments; and R is the concentration of each pollutant measured at a given receptor. In theory, these matrices are whatever size can incorporate as many sources, receptors, and pollutants as necessary. Here we demonstrate the technique with a single pollutant at a single receptor. Cartwright and Harris (1993) chose to use uniform dispersion in all directions, with a decrease of concentration with distance according to a $r^{-2.5}$ power law, where r is the distance from the source to the receptor. Here we, instead, choose to use the more refined dispersion law as found in Beychok (1994), together with actual wind data for the time period modeled:

$$C = \frac{Q}{u\sigma_z\sigma_y 2\pi}\exp\left(\frac{-y^2}{2\sigma_y^2}\right)\left[\exp\left(\frac{-(z_r - H_e)^2}{2\sigma_x^2}\right) + \exp\left(\frac{-(z_r + H_e)^2}{2\sigma_z^2}\right)\right] \quad (6.11)$$

where

C = concentration of emissions at a receptor

(x, y, z_r) = Cartesian coordinates of the receptor in the downwind direction from the source

Q = source emission rate

u = wind speed

H_e = effective height of the plume centerline above ground

σ_y, σ_z = standard deviations of the emission distribution in the y and z directions

Note that there are a myriad of assumptions hidden behind the problem. First, we assume that the wind speed and direction are constant over the entire time period. Although we know a priori that this assumption is poor, it is balanced by the assumption of Gaussian dispersion in a single direction. Although a plume of pollutants may meander throughout the time period, we only care about the weighted average statistical distribution of the concentrations. Next we are forced to assume a constant emission rate, in this case an average annual rate. The hourly rate is much different. Another major difficulty is in estimating reasonable average values for the dispersion coefficients, σ_y and σ_z. Again, we must assume a weighted average over time and use dispersion coefficients computed by

$$\sigma = \exp\left[I + J\left(\ln(x) + K(\ln(x))^2\right]\right. \quad (6.12)$$

where x is the downwind distance (in km) and I, J, and K are empirical coefficients dependent on the Pasquill stability class (documented in a lookup table; Beychok, 1994). The Pasquill stability class depends on wind speed,

direction, and insolation. For this demonstration problem, we assumed neutral stability (class D).

Equations (6.10) and (6.11) together with two lookup tables and (6.12) convert the source emission rates into the elements of the source matrix, M, which indicates an expected average concentration due to each source for a constant emission rate and actual hourly average wind data. This process is repeated for each source at each time. The measured concentrations are also time averaged (in this case over a three-day period) and go into the matrix, R. The goal is to solve for the fractions, S, that apportion the pollution to each source. That is where the GA comes in. If R and S were constant one-dimensional vectors, one could easily solve for S. However, the need to sum the matrix times the factors hourly for differing meteorological conditions precludes a simple matrix inversion. The chromosomes of the GA in this case represent the unknown elements of matrix S. The cost function is the difference between the pollutant values at the receptor and the summation of the hourly concentrations predicted for each source as computed from the dispersion model (6.11) times the apportionment factors supplied by the GA:

$$\text{cost} = R - M \bullet S \tag{6.13}$$

where $M = \sum_{h=1}^{H} C_h$ with the C_h computed from (6.11).

The receptor model was run using actual meteorological data for three-day periods in 2002 and comparing predicted weighted average concentrations of PM10 (particulate matter less than 10 micrometers in diameter) measured at the receptor. The dispersion coefficients were computed assuming a Pasquill stability class D. Three to four runs of the GA were done for each time period using a population size of 12 and mutation rate of 0.2. The fractions in the unknown vector, S, were normalized to sum to 1. Runs were made for 1000 generations. Four different runs were made for each of five different days and results appear in Table 6.3 for the run with the best convergence for each day. Those days were chosen to represent different concentrations and meteorology conditions, although we were careful to choose days where the assumption of stability D appeared to be good. For many of the runs the factors converged on the heaviest weighting of source 13, the Utah State University heating plant. Note that this does not necessarily imply that it contributed the most pollutant but rather that its average emission rate, when dispersed according to (6.11) using actual wind data, must have a heavier weighting to account for the monitored PM10. The second highest weighted source was number 9, a local construction company.

The point of this exercise is to demonstrate that the GA is a useful tool for problems that require including another model, in this case a dis-

TABLE 6.3 Factors Computed to Apportion Sources to Received PM10 Measurements

Received	22–Apr	21–Jun	27–Jul	11–Aug	21–Nov
($\mu g/m^3$)	8	27	39	36	33
Source					
1	0.000	0.002	0.001	0.003	0.002
2	0.000	0.017	0.043	0.000	0.000
3	0.184	0.124	0.063	0.001	0.001
4	0.128	0.023	0.002	0.024	0.001
5	0.101	0.004	0.143	0.041	0.001
6	0.022	0.022	0.013	0.000	0.231
7	0.012	0.011	0.037	0.000	0.001
8	0.005	0.014	0.005	0.045	0.000
9	0.001	0.295	0.033	0.257	0.159
10	0.001	0.114	0.005	0.039	0.005
11	0.281	0.027	0.026	0.037	0.119
12	0.055	0.022	0.000	0.004	0.013
13	0.180	0.206	0.571	0.193	0.248
14	0.014	0.026	0.002	0.063	0.005
15	0.000	0.001	0.004	0.273	0.120
16	0.016	0.093	0.053	0.063	0.094

persion model, to evaluate the cost of a function. Despite the large number of times that (6.13) was evaluated, it still not prohibitive in terms of CPU time required. Coupling GAs with simulations is becoming a more popular way to do searches. The work of Loughlin et al. (2000) coupled a full air quality model with a GA to design better control strategies to meet attainment of the ozone standard while minimizing total cost of controls at over 1000 sources. Such problems are requiring significant amounts of time on computers.

6.9 OPTIMIZING ARTIFICIAL NEURAL NETS WITH GAs

An increasingly popular use of GAs combines their ability to optimize with the strengths of other artificial intelligence methods. One of these methods is the neural network. Artificial neural networks (ANN) have found wide use in fields areas as signal processing, pattern recognition, medical diagnosis, speech production, speech recognition, identification of geophysical features, and mortgage evaluation.

ANNs model biological neurons in order to do numerical interpolation. Figure 6.30 provides a sketch of a biological neuron and a human-made neuron. The biological neuron acts as a processing element that receives many signals. These signals may be modified by a weight at the receiving synapse.

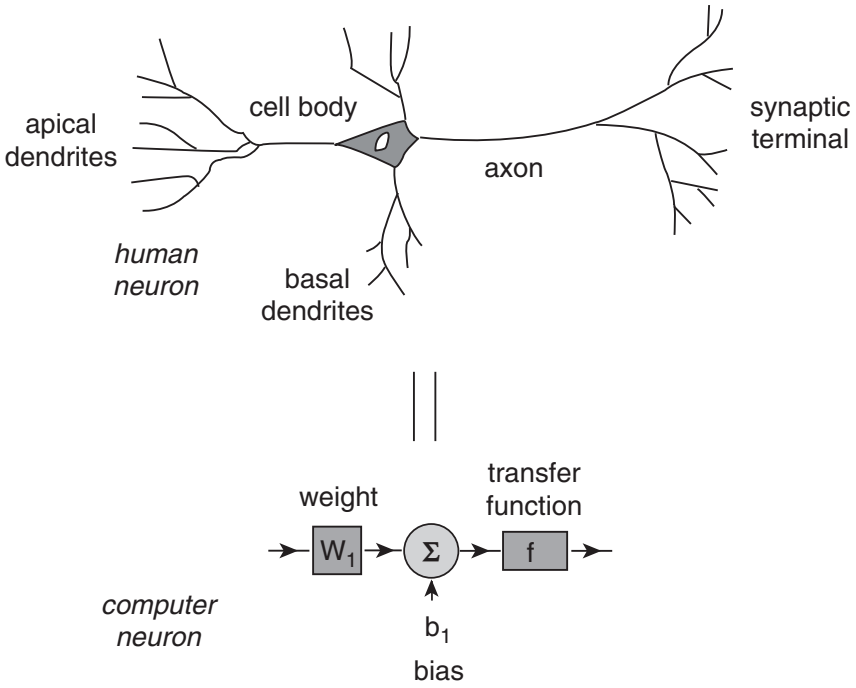

Figure 6.30 Diagram of a biological neuron (*top*) and schematic of the artificial neural network.

Then the processing element sums the weighted inputs. When the input becomes sufficiently large, the neuron transmits a single output that goes off to other neurons. The human-made neuron works by analogy. It takes an input, multiplies it by a weight, adds a bias, and then passes the result through a transfer function. Several neurons in parallel are known as a layer. Adding these layers together produces the neural network. The weights and bias values are optimized to produce the desired output. Although there are many ways to train the ANN, we are interested in coupling it with a GA to compute the optimum weights and biases. There are many good books on neural networks (e.g., Hagan et al., 1995; Fausett, 1994), so we will make no attempt to fully describe ANN. Instead, we just briefly explain how we use a GA to train one.

We wish to approximate the function

$$\mathrm{f}(x) = \frac{12}{x^2 \cos(x) + 1/x} \quad \text{for} \quad 1 \le \mathrm{x} \le 5 \tag{6.14}$$

To do this, we used the two-layer neural network shown in Figure 6.31 with log-sigmoid transfer functions. The transfer function determines the threshold and amount of signal being sent from a neuron. Although various transfer functions were tried, the log-sigmoid worked best for this problem. It has the form $1/(1 + e^{-n})$ and maps the input to the interval [0, 1].

The goal is to compute the optimum weights and biases of the ANN using

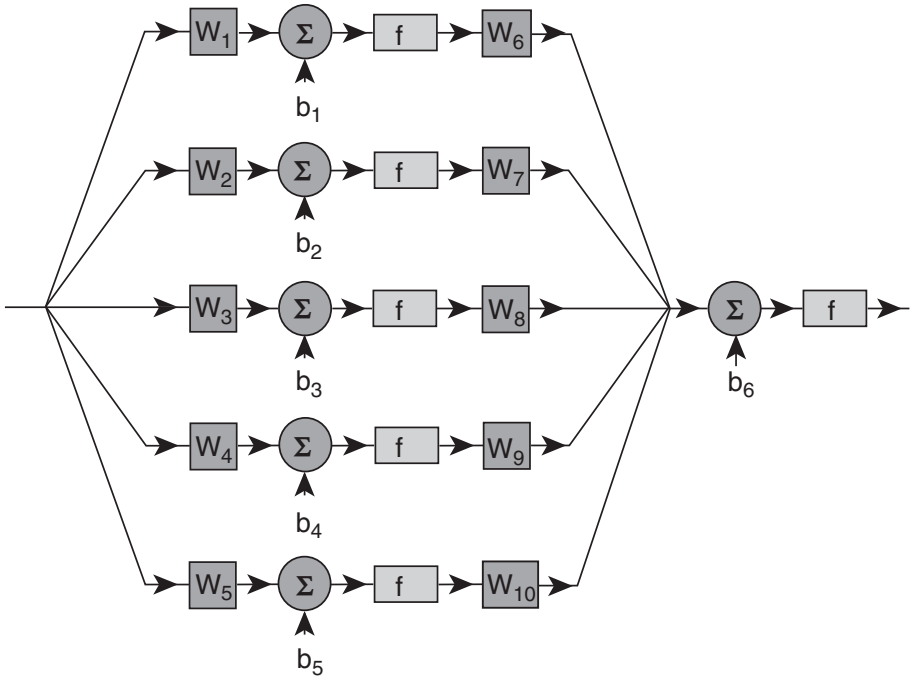

Figure 6.31 Two-layer neural network used to compute fit to (6.14).

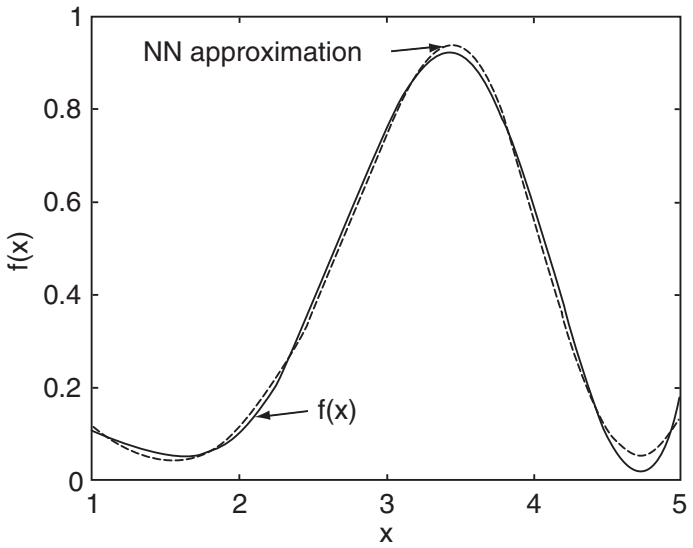

Figure 6.32 Comparison of the exact function of (6.14) with the ANN/hybrid GA approximation.

a GA. The GA chromosome is made up of potential weights and biases. The GA cost function computes the mean square difference between the current guess of the function and the exact function evaluated at specific points in x. The function was sampled at intervals of 0.1 for training. We used a hybrid GA having $N_{pop} = 8$ and $\mu = 0.1$. The local optimizer was a Nelder-Mead algorithm. The resulting approximation to (6.14) is shown in Figure 6.32. Note that the function computed from the neural network with GA hybrid training matches the known curve quite well.

6.10 SOLVING HIGH-ORDER NONLINEAR PARTIAL DIFFERENTIAL EQUATIONS

Two mathematical tools of scientists and engineers are ordinary and partial differential equations (ODEs and PDEs). Normally we don't think of these equations as minimization problems. However, if we want to find values where a differential equation is zero (a form in which we can always cast the system), we can look for the minimum of its absolute value. Koza (1992) demonstrated that a GA could solve a simple differential equation by minimizing the value of the solution at 200 points. To do this, he numerically differentiated at each point and fit the appropriate solution using a GA. Karr et al. (2001) used GAs to solve inverse initial boundary value problems and found a large improvement in matching measured values. That technique was demonstrated on elliptic, parabolic, and hyperbolic PDEs.

We demonstrate here that a GA can be a useful technique for solving a highly nonlinear differential equation that is formally nonintegrable. For comparison, we do know its solitary wave approximate solution. Solitary waves, or solitons, are permanent-form waves for which the nonlinearity balances the dispersion to produce a coherent structure. We examine the super Korteweg-de Vries equation (SKDV), a fifth-order nonlinear partial differential equation:

$$u_t + \alpha u u_x + \mu u_{xxx} - \nu u_{xxx} = 0 \tag{6.15}$$

The functional form is denoted by u; time derivative by the t subscript; spatial derivative by the x subscript; and α, μ, and ν are variables of the problem. We wish to solve for waves that are steadily translating, so we write the t variation using a Galilean tranformation, $X = x - ct$, where c is the phase speed of the wave. Thus our SKDV becomes a fifth-order, nonlinear ordinary differential equation:

$$(\alpha u - c)u_X + \mu u_{XXX} - \nu u_{XXXX} = 0 \tag{6.16}$$

Boyd (1986) extensively studied methods of solving this equation. He expanded the solution in terms of Fourier series to find periodic cnoidal wave solutions (solitons that are repeated periodically). Among the methods used are the analytical Stokes's expansion, which intrinsically assumes small amplitude waves, and the numerical Newton-Kantorovich iterative method, which can go beyond the small amplitude regime if care is taken to provide a very good first guess. Haupt and Boyd (1988a) were able to extend these methods to deal with resonance conditions. These methods were generalized to two dimensions to find double-cnoidal waves (two waves of differing wave number on each period) for the integrable Korteweg-de Vries equation (1988b) and the nonintegrable regularized long wave equation (Haupt, 1988). However, these methods require careful analytics and programming that is very problem specific. Here we are able to add a simple modification to the cost function of our GA to obtain a similar result.

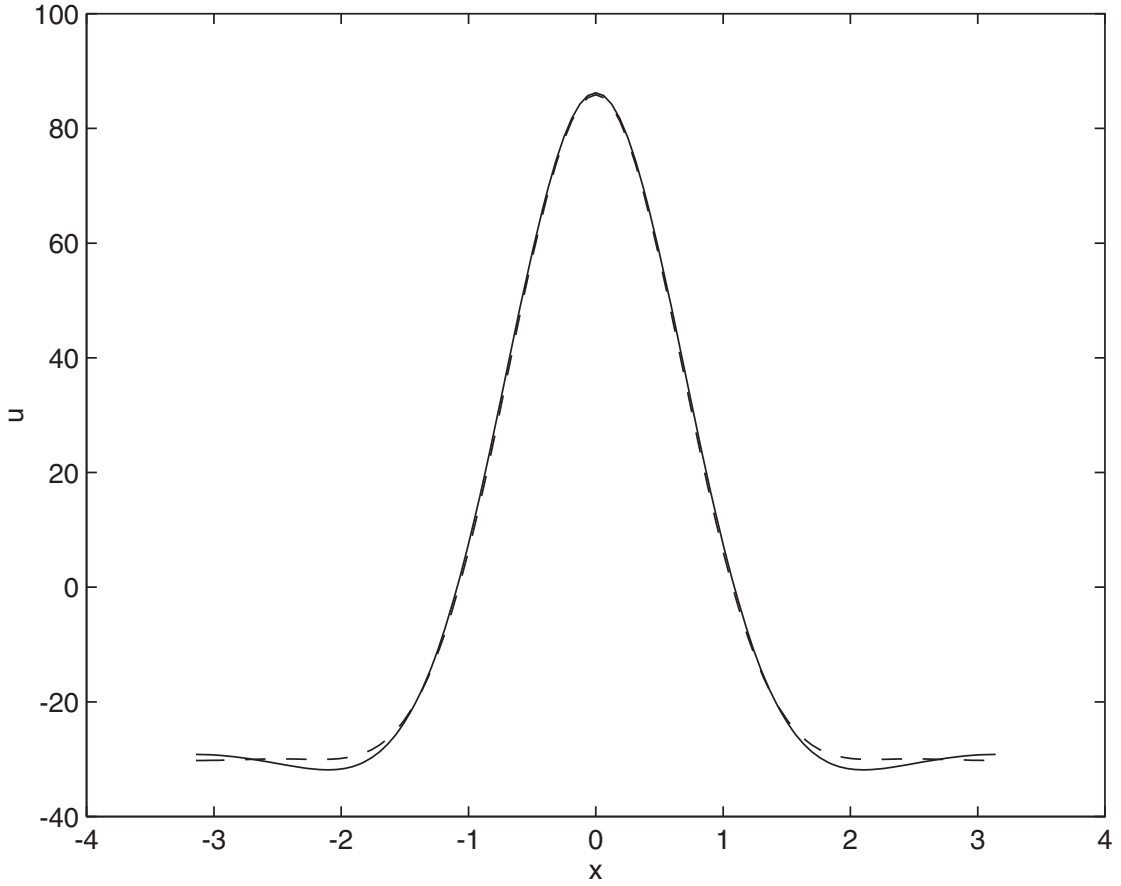

Figure 6.33 Cnoidal wave of the super Korteweg de Vries equation. *Solid line*: exact solution; *dashed line*: genetic algorithm solution.

To find the solution of equation (6.16), we expand the function u in terms of a Fourier cosine series to K terms to obtain the approximation, u_K:

$$u(X) \simeq u_k(X) = \sum_{k=1}^{K} a_k \cos(kX) \tag{6.17}$$

The cosine series assumes that the function is symmetric about the X-axis (without loss of generality). In addition we use the "cnoidal convention" by assuming that the constant term a_0 is 0. Now we can easily take derivatives as powers of the wave numbers to write the cost that we wish to minimize as

$$\text{cost}(u_k) = \sum_{k=1}^{K} [-k(\alpha u - c) + k^3\mu + k^5 v] a_k \sin(kx) \tag{6.18}$$

This is reasonably easy to put into the cost function of a GA where we want to find the coefficients of the series, a_k. The only minor complication is computing u to insert into the cost function, (6.18). However, this is merely one extra line of code.

The parameters that we used here are $v = 1, \mu = 0, \alpha = 1$, and a phase speed

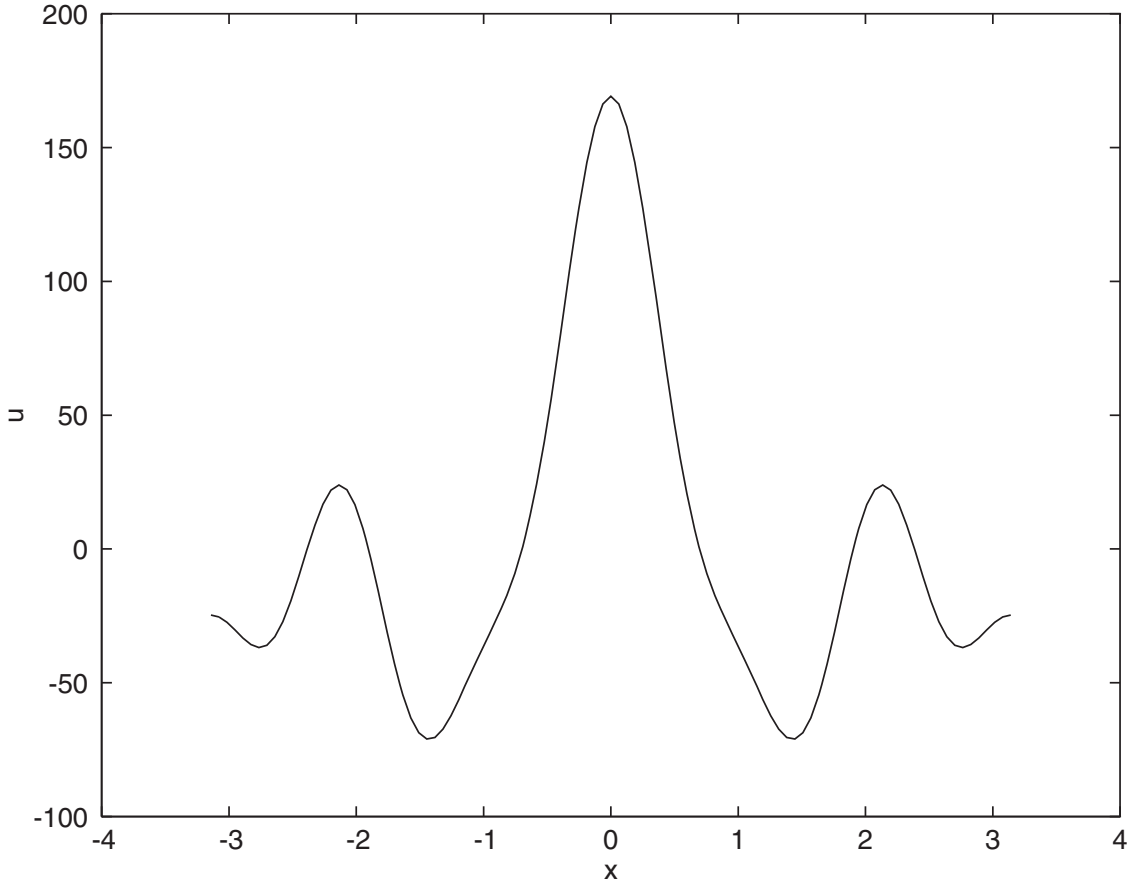

Figure 6.34 Double cnoidal wave of the super Korteweg de Vries equation as computed by the genetic algorithm.

of $c = 14.683$ to match with a well-known nonlinear solution. The phase speed and amplitude of solitary-type waves are interdependent. We could instead have specified the amplitude and solved for the phase speed. It is equivalent. We computed the coefficients, a_k, to find the best cnoidal wave solution for $K = 6$. We used $N_{pop} = 100$, $\mu = 0.2$, and 70 iterations. We evaluated the cost function at grid points and summed their absolute value. The results appear in Figure 6.33. The solid line is the "exact" solution reported by Boyd (1986) and the dashed line is the GA's approximation to it. They are barely distinguishable. In addition we show a GA solution that converged to a double cnoidal wave as Figure 6.34. Such double cnoidal waves are very difficult to compute using other methods (Haupt and Boyd, 1988b).

So we see that GAs show promise for finding solutions of differential and partial differential equations, even when these equations are highly nonlinear and have high-order derivatives.

BIBLIOGRAPHY

Beychok, M. R. 1994. *Fundamentals of Stack Gas Dispersion*, 3rd ed. Irvine, CA: Milton Beychok.

Boyd, J. P. 1986. Solitons from sine waves: Analytical and numerical methods for non-integrable solitary and cnoidal waves. *Physica* **21D**:227–246.

Branstator, G., and S. E. Haupt. 1998. An empirical model of barotropic atmospheric dynamics and its response to forcing. *J. Climate* **11**:2645–2667.

Cartwright, H. M., and S. P. Harris. 1993. Analysis of the distribution of airborne pollution using GAs. *Atmos. Environ* **27A**:1783–1791.

Chambers, L. (ed.). 1995. *GAs, Applications*, Vol. 1. New York: CRC Press.

Davidor, Y. 1991. *GAs and Robotics*. River Edge, NJ: World Scientific.

Davis, L. 1991. *Handbook of GAs*. New York: Van Nostrand Reinhold.

Fausett, L. 1994. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Upper Saddle River, NJ: Prentice Hall.

Hagan, M. T., H. B. Demuth, and M. Beale. 1995. *Neural Network Design*. Boston: PWS.

Hasselmann, K. 1976. Stochastic climate models. Part I: Theory. *Tellus* **28**:473–485.

Haupt, R. L. 1995. An introduction to GAs for electromagnetics. *IEEE Ant. Propagat. Mag.* **37**:7–15.

Haupt, S. E. 1988. Solving nonlinear wave problems with spectral boundary value techniques. Ph.D. dissertation. University of Michigan, Ann Arbor.

Haupt, S. E., and J. P. Boyd. 1988a. Modeling nonlinear resonance: A modification to the Stokes' perturbation expansion. *Wave Motion* **10**:83–98.

Haupt, S. E., and J. P. Boyd. 1988b. Double cnoidal waves of the Korteweg De Vries equation: Solution by the spectral boundary value approach. *Physica* **50D**:117–134.

Holland, J. H. 1992. Genetic Algorithms. *Sci. Am.* **267**:66–72.

Karr, C. L. 2003. Minimization of sonic boom using an evolutionary algorithm. Paper AIAA 2003-0463. 40st AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV.

Karr, C. L., I. Yakushin, and K. Nicolosi. 2001. Solving inverse initial-value, boundary-value problems via GA. *Eng. Appl. Art. Intell.* **13**:625–633.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* **220**:671–680.

Koza, J. R. 1992. The Genetic Programming Paradigm: Genetically Breeding Populations of Computer Programs to Solve Problems. In B. Soucek (ed.), *Dynamic, Genetic, and Chaotic Programming: The Sixth Generation*. New York: J. Wiley, pp. 203–321.

Loughlin, D. H., S. R. Ranjithan, J. W. Baugh, Jr., and E. D. Brill Jr. 2000. Application of GAs for the design of ozone control strategies. *J. Air Waste Manage. Assoc.* **50**:1050–1063.

Michalewicz, Z. 1992. *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag.

Obayashi, S., D. Sasaki, Y. Takeguchi, and N. Hirose. 2000. Multiobjective evolutionary computation for supersonic wing-shape optimization. *IEEE Trans. Evol. Comput.* **4**:182–187.

Pack, D., G. Toussaint, and R. Haupt. 1996. Robot trajectory planning using a GA. Int. Symp. on Optical Science, Engineering, and Instrumentation. SPIE's Annual Meeting, Denver, CO.

Penland, C. 1989. Random forcing and forecasting using principal oscillation pattern analysis. *Mon. Weather Rev.* **117**:2165–2185.

Penland, C. 1996. A stochasic model of IndoPacific sea surface temperature anomalies. *Physica* **98D**:534–558.

Penland, C., and M. Ghil. 1993. Forecasting northern hemisphere 700 mb geopotential height anomalies using empirical normal modes. *Mon. Weather Rev.* **121**:2355.

Penland, C., and T. Magorian. 1993. Prediction of NINO3 sea-surface temperatures using linear inverse modeling. *J. Climate* **6**:1067.

Whitley, D., T. Starkweather, and D. Shaner. 1991. The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In L. Davis (ed.), *Handbook of GAs*. New York: Van Nostrand Reinhold.

Widrow, B., and S. D. Sterns, 1985. *Adaptive Signal Processing*. Upper Saddle River, NJ: Prentice-Hall.

Yao, X. (ed.). 1995. *Progress in Evolutionary Computation*. New York: Springer-Verlag.