Given the circuit topology in Figure 1, find the resistor and NTC thermistor values that generate the Voltage vs. Temperature curve shown in Figure 2. (Note: Only standard values may be used.)
While this question may not appear in a textbook, it is a common practical problem that you may encounter when you try to apply temperature compensation to an active device using a thermistor network. Some of us may be able to look at the circuit in and determine the solution using a scratch pad and numerous equations. The rest of us would probably just enter the values into a simulator or spreadsheet and slowly try to adjust the values until we were able to match the curves as best as possible.

Finding the optimal solution to this circuit using either of the methods above can prove challenging given the number of variables and the temperature dependence involved in its solution. Being constrained to only standard resistor values increases the difficulty. While solving this circuit for one set of conditions may be acceptable, it can be tedious to solve for multiple conditions, networks or desired responses. As the networks become more complicated, finding an optimal solution can often times be too difficult or too time-consuming to solve symbolically or empirically. In contrast, a numerical method, such as the genetic algorithm, can solve these problems quickly and easily.

**Darwinian design?**

The words genetics and circuits rarely find themselves in the same context, but the genetic process can be applied to solve component values in circuits such as that shown in Figure 1. This article discusses applications of thermistors for temperature compensation and gives a brief description of the genetic algorithm. An example of the algorithm being used to quickly solve Figure 1 will then be illustrated.

Thermistor networks are commonly used to apply temperature compensation to a wide variety of
circuit applications. For example, thermistor networks are used to adjust power supply circuits, PWM outputs, as well as provide temperature-compensated bias currents to transistors, amplifier circuits, and laser diodes. Given the many applications for thermistor networks, the required thermistor values will vary greatly.

Figure 3 is an example of the compensation needed for a laser diode used in a fiber optic communication link application. To maintain the quality of the communication link, the bias current to the laser diode must be adjusted as temperature changes. The temperature-compensated bias current can be applied to the laser using the circuit shown in Figure 3. The optimal values must be determined to match the desired voltage vs. temperature curve of Figure 2. A near ideal solution can be found in a matter of seconds when you use the genetic algorithm.

Figure 3 The compensation needed for a laser diode used in a fiber optic communication link application

The genetic algorithm is an optimization algorithm that mimics the genetic process to find optimal solutions to multi-variable problems. The genetic algorithm was first used in the 1950s (Reference 1) but has only been widely used in the last 15 to 20 years due to advances in computer technology and processor speed.

The algorithm starts by creating a random population or a population of predetermined starting points. Each population element (individual) in a population (Figure 4) is composed of an array of variables that defines its construction (analogous to its genetic code). Once each population element is initialized, its performance is measured using a set of criteria and then compared against all other elements. The population elements are then sorted according to their performance. The elements with a poor ranking are eliminated (as by natural selection) and a new generation of elements is created by mixing the variables (genetic code) of the best population elements from the previous generation (Figure 5).
Figure 4 Each population element (individual) in a population is composed of an array of variables that defines its construction.
Figure 5 The elements with a poor ranking are eliminated (as by natural selection) and a new generation of elements is created by mixing the variables (genetic code) of the best population elements from the previous generation.

If the process were allowed to loop after completing the steps to this point, the population would quickly saturate to a common identical genetic code. A minimum will be found but it may not be the global minimum. To keep the population from saturating or converging on a local minimum, the computer program can introduce mutations (random changes) to a population element’s variables as well as new random elements into the population (Figure 6).
The frequency and amount of randomness that the program should introduce will vary from one application to another. If too much randomness is introduced the algorithm will not have time to converge on an ideal solution. If too little is introduced, finding the global minimum could take an extremely long time. You should tune the amount of randomness by trial and error or dynamically increase the randomness as the population saturation increases.

There are other optimization algorithms that can be faster than the genetic algorithm. They are often more complicated to program for multi-variable problems and they will often converge on a local minimum. The genetic algorithm overcomes these problems.

The genetic algorithm is based on a random process so the time needed to converge on an absolute optimal solution will be random. The average convergence time will increase as the number of variables in each population element increases. Determining and calculating the performance criteria for a given application can also be difficult for some applications. In other words, it is sometimes difficult to write equations that express what the optimal solution should look like. For simple passive circuits such as the thermistor circuit shown in Figure 1, you can easily write and define the performance criteria. Given the low number of variables, the computer program can also find the solution in a reasonable amount of time. The genetic algorithm is, therefore, well suited to solve these types of circuits.

**Code example**

Using a free visual basic compiler (Reference 2), a simple routine was written to solve the thermistor network values (Figure 1) to match the desired response (Figure 2). Download the source code for this example and the executable file.
The algorithm is implemented in software with the following basic steps:

1. Define the available resistor and thermistor values that can be used as possible variables in each population element. An array is declared for available resistors and another for available thermistors.
2. Generate a random population of elements. (Population composed of 100 elements for this example). Each population element is an array that contains an entry for each component in the circuit (Figure 1).
3. Equate the desired voltage or resistance of the network over temperature and graph its response. The desired voltage/resistance is then imported into the code using a polynomial equation.
4. Calculate the mean square error of each population element.
5. Sort the population from lowest error to highest error and graph the best solution.
6. Eliminate the worst solution and randomly or methodically combine the remaining elements to create a second generation of population elements.
7. Introduce randomness by introducing new random population elements or by randomly changing one or more variables in one or more of the existing population elements.
8. Return to step 4 and repeat until the mean square error is sufficiently low and the user decides to stop the process.

![Figure 7](image)

The software's graphical user interface is shown in Figure 7. From this interface, you can select the thermistor network configuration, input the desired response using a polynomial expression, select to solve for total resistance or a voltage, the temperature range to optimize and you can also select if 1% or 5% resistor values should be used (Figure 8). Pressing the “Run” button starts the algorithm. The mean square error, iteration count and the component values for the current best solution are

Just as with genetics and evolution the solution "grows" from one state to another. The optimal solution is essentially the best genetic code (circuit element values) to the environment (desired circuit response).

The software's graphical user interface is shown in Figure 7. From this interface, you can select the thermistor network configuration, input the desired response using a polynomial expression, select to solve for total resistance or a voltage, the temperature range to optimize and you can also select if 1% or 5% resistor values should be used (Figure 8). Pressing the “Run” button starts the algorithm. The mean square error, iteration count and the component values for the current best solution are
then updated as the program runs.

A graph is also drawn showing the desired response (in green) as well as the current best solution (shown in red). The graphical output after the computer has run 30 iterations is shown in Figure 8. As seen in this figure, the response is non-ideal. After 500 iterations, which takes approximately 10 seconds, a very good response has been generated (Figure 9).
Figure 9 After 500 iterations, a very good response has been generated.

Letting the program run longer reveals the results shown in Figure 10, which is almost an exact match.
Letting the program run longer reveals an almost exact match.

Using **Figure 1** as a reference this solution is composed of the following component values:

R1 = 8.2k, R2 = 3.6k, R3 = 2.2k, R4 =1.1k,

TH1 = 47k Beta = 4700, TH2 = 40 Beta = 2750

The speed of convergence will vary from one test to another. The program will converge much faster with a simpler network and will take longer to converge with a more complicated network or if 1% resistor values are used because there are more variables to choose from.

Putting this all into perspective, a network composed of two thermistors and four resistors was closely matched to the desired temperature response in less than 30 seconds using only standard component values.

**References**
1. [www.wikipedia.com](http://www.wikipedia.com), Genetic Algorithm