System Safety Matrix Methods

by Richard R. Zito, Ph.D.
Richard R. Zito Research LLC, Tucson, Arizona

The analysis of networks is a common feature in system safety analysis. These networks may range from electronic circuits to software flowcharts to maps of land, air, sea and communications traffic. Matrix methods are the natural tool for the analysis of these networks, and the object of this paper is to describe the basics of matrix methods in the context of three common problems encountered by systems safety engineers: the Bent Pin Problem, the Sneak Circuit Problem and the Analysis of Software Logic. Comparison of these analyses will reveal deep connections between these problems and suggest directions for future research.

Background and Problem Statement

The system safety engineer is often confronted with the analysis of networks. *The Bent Pin Problem*, the *Sneak Circuit Problem* and the *Analysis of Software Logic* are all examples of network problems.

The Bent Pin Problem may be described as follows: Given a multi-pin connector, how many short circuit paths can be formed if one or more pins are bent during insertion? The problem is particularly difficult for aerospace connectors that may have hundreds of long, thin pins. The last connector attached to a dangerous system is particularly problematic. How does the engineer make a continuity test for an umbilical connection used to launch a missile or detonate a warhead without igniting the missile or warhead itself?

The Sneak Circuit Problem involves detection of unintended electronic pathways in a complex circuit that may make that circuit behave in a way that was not envisioned by the designer. But how does an analyst capture the notion of *intent*? And how can the analyst be sure that this intent is accurately reflected in the circuitry that is being examined? Finally, how can the analyst be certain that his or her analysis is complete, and that nothing has been overlooked?

The problem of debugging software logic is closely related to the electronic Sneak Circuit Problem. In fact, software that behaves in an unintended manner can be thought of as containing sneak logic circuits.

All of these system safety problems can be formulated in terms of square matrices. A matrix is just a collection of numbers — in these problems, positive integers arranged in rows and columns. A square matrix is just a matrix, with an equal number (N) of rows and columns. Various types of mathematical operations — such as matrix multiplication of two matrices, raising a matrix to a power (the repeated matrix multiplication of a matrix by itself) and element by element multiplication, addition and subtraction of two matrices — are all well-defined operations for square matrices of equal size. In the next section, a particular type of square matrix, called the Adjacency Matrix, will be defined for a generalized network. Subsequent sections will apply the Adjacency Matrix to specific bent pin, sneak circuit and software analysis problems.

The Adjacency Matrix

The Adjacency Matrix of a graph, a schematic diagram composed of lines (called edges) and connection points represented as dots (called vertices), is simply a square matrix whose entries (or elements) A(i, j) are the number of edges that connect vertex i to vertex j. These connections are direct connections, not connections through another intermediate vertex. For example, consider the graph in Figure 1. Each vertex is labeled with a number, but letters or any other symbols could just as well be used.

It is clear that only one edge connects vertex 1 to vertex 2. There is one edge that connects vertex 1 to vertex 4. But vertex 1 has no direct connections to vertices 3, 5, 6, 7, 8 or 9. That is to say, in this case, vertex 1 is only directly connected to its two nearest neighbors. The Adjacency matrix is then constructed as shown in Figure 2.

Notice that the first row has a "1" in the second and fourth columns. All the other matrix elements in the first

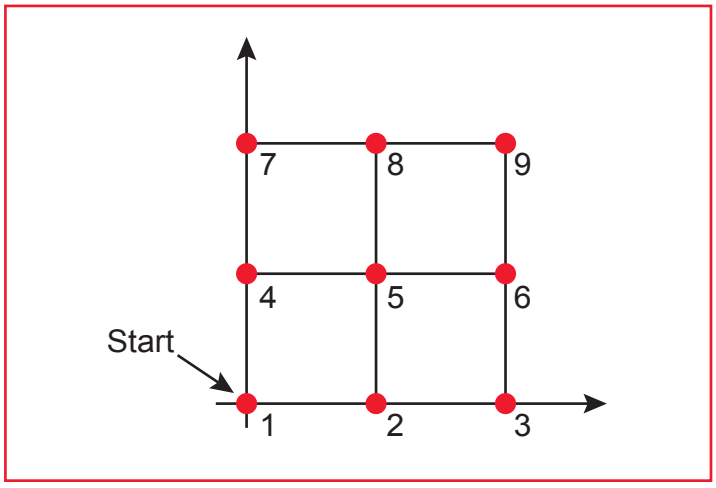


Figure 1 — A Simple Graph Containing Nine Vertices and 12 Edges.

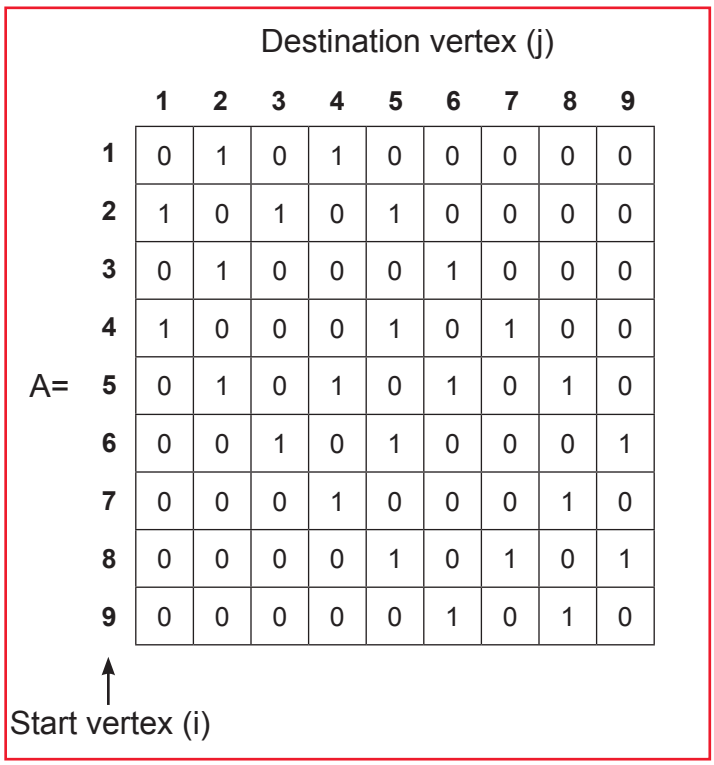


Figure 2 — The Adjacency Matrix corresponding to the network in Figure 1. Direct connections between the "Start" and "Destination" vertices are marked by a "1." If no direct connection exists between two vertices, then the corresponding matrix element is a "0."

row are zero. That is to say, A(1,1) = 0 because Figure 1 has no loops that connect vertex 1 directly to itself. A(1,2) = A(1,4) = 1 because these elements represent direct connections from vertex 1 to vertices 2 and 4. And, A(1,3) = A(1,5) = A(1,6) = A(1,7) = A(1,8) = A(1,9) = 0 because vertex 1 has no direct connection to vertices 3, 5, 6, 7, 8 and 9. Similar arguments hold for the entries in row 2, where only A(2,1), A(2,3) and A(2,5) are non-zero; since vertex 2 only makes a direct connection to vertices 1, 3 and 5. The rest of the matrix elements displayed in Figure 2 can be filled in using similar reasoning. Therefore, the Adjacency Matrix captures, in numerical form, the connections within a graph. In fact, the Adjacency Matrix is the most powerful expression of network topology — as will be subsequently demonstrated.

The Bent Pin Problem

Suppose that an analyst is presented with a connector having a simple square array of pins, like that in Figure 1, with one pin at each vertex. Suppose the length of each pin is such that it can only make contact with its nearest neighbors. In that case, all possible short circuit connections are summarized by the edges of Figure 1. It is now possible to ask some interesting questions like, "How many ways are there to short pin 1 to pin 6 if 2 pins, 3 pins, 4 pins, etc. are allowed to be bent?" The problem is interesting in its own right, but its value is enhanced by its relationship to the Sneak Circuit Problem. The solution, known as Cayley's Theorem [Ref. 1], is to examine the matrix elements of A^2, A^3, A^4, etc. Raising a square matrix to a power is straightforward [Ref. 2] and can be easily accomplished by commercial off-the-shelf software like MATLAB [Ref. 3] or even EXCEL [Ref. 4]. It is clear from inspection of Figure 1 that there are no paths that are two or four bent pins long that connect pin 1 to pin 6. However, there are paths that are three bent pins long, so A^3 is a matrix that needs to be examined. It is displayed in Figure 3.

Examination of the element $A^3(1,6)$ shows that it has a value of 3, implying that three paths exist that can short pin 1 to pin 6. Inspection of Figure 1 shows that this is true. The paths are 1→2→3→6, 1→2→5→6 and 1→4→5→6. As previously stated, there are no paths that are four bent pins long that bridge the gap between pin 1 and pin 6. Therefore, the $A^4(1,6)$ element is zero, as expected. However, several short circuit paths five pins long can exist between pin 1 and pin 6. So, examination of A^5 is also very interesting (Figure 4).

$A^5(1,6) = 30$. But only four of these paths are relevant (viz., 1→4→5→8→9→6, 1→4→7→8→9→6, 1→4→7→8→5→6 and 1→2→5→8→9→6). All the other paths are called *redundant paths*. For example, one such redundant path is 1→2→5→2→3→6. This path is unphysical because once the pin rooted at position (vertex) 2 is bent toward position 5, it cannot later be re-bent toward position (vertex) 3. So, this redundant path can be safely eliminated from the count, as well as the similar paths

1→4→1→2→3→6,
1→4→1→4→5→6,
1→2→1→2→3→6,
1→2→1→2→5→6,
1→2→1→4→5→6,
1→2→3→2→3→6,
1→2→3→2→5→6,
1→4→7→4→5→6,
1→4→5→8→5→6,
1→4→5→2→5→6,
1→4→5→4→5→6,
1→2→5→4→5→6,
1→2→5→2→3→6,
1→2→5→8→5→6 and
1→2→5→2→5→6,

all for the same reason. What about a path like 1→2→3→6→9→6? This path is five edges (bent pins) long, and although vertex 6 has been visited twice, it was only bent once because the second visit is the last contact in the chain. However, a moment of reflection reminds the reader that bending *both* pin 1 *and* pin 6 means this short circuit path is removed from applied power (i.e., it is isolated) and is no longer a safety-critical issue. So, this path can be eliminated from safety analysis, as well as paths

$A^3 =$

0	5	0	5	0	3	0	3	0
5	0	5	0	8	0	3	0	3
0	5	0	3	0	5	0	3	0
5	0	3	0	8	0	5	0	3
0	8	0	8	0	8	0	8	0
3	0	5	0	8	0	3	0	5
0	3	0	5	0	3	0	5	0
3	0	3	0	8	0	5	0	5
0	3	0	3	0	5	0	5	0

Figure 3 — The matrix A^3. This matrix is symmetric about the main diagonal because bent pin short circuit paths extending from Pin i to Pin j are the same as those from Pin j to Pin i.

$A^5 =$

0	34	0	34	0	30	0	30	0
34	0	34	0	64	0	30	0	30
0	34	0	30	0	34	0	30	0
34	0	30	0	64	0	34	0	30
0	64	0	64	0	64	0	64	0
30	0	34	0	64	0	30	0	34
0	30	0	34	0	30	0	34	0
30	0	30	0	64	0	34	0	34
0	30	0	30	0	34	0	34	0

Figure 4 — A^5 for the Bent Pin Problem.

1→2→3→6→5→6,
1→2→3→6→3→6,
1→2→3→6→9→6,
1→2→5→6→5→6,
1→2→5→6→3→6,
1→2→5→6→9→6,
1→4→5→6→5→6,
1→4→5→6→9→6 and
1→4→5→6→3→6.

In fact, each chain must begin with a 1, and end in a 6 with no two vertices in the bent pin chain the same and, of course as a corollary, no two consecutive vertex numbers can be the same because that would mean that a pin was bent back on itself (a peculiarity that will not be addressed in this analysis). Therefore, all 30 paths have been accounted for. Again, only four of these paths are a concern to the analyst; the other 26 can be discarded.

It would be helpful, and sometimes necessary, if redundant paths could be eliminated from the count *a priori*, instead of *a posteriori*. In fact, there is a mathematical way to remove these redundant paths from the count by subtracting the *Redundancy Matrix* from powers of A. The Redundancy Matrix may be computed directly from A, but this involved calculation will not be presented here other than by citation [Ref. 1].

Since $A^6(1,6) = 0$, there cannot exist any conducting paths that are six bent pins long. However, there are two other short circuit paths that are seven pins long (1→4→7→8→5→2→3→6 and 1→2→5→4→7→8→9→6). $A^7(1,6) = 256$, and all but two of these paths are redundant. It is not necessary to search longer paths, since there are none that can short pin 1 to pin 6 without revisiting some pin or other — which is not allowed, as stated above. So, the total number of possible short circuit paths is 3 + 4 +2 = 8. A surprisingly large number for such a simple problem! The path count is complete and certain. There are no other paths. Of course, not all these short circuit paths are equally probable. The shortest paths are most likely to form, while the probability of forming longer paths drops off dramatically by orders of magnitude. These calculations can be found in the literature [Ref. 5] and will not be repeated here. All that is of interest in this publication is the use of matrix methods for counting conducting short circuit paths. Once this counting procedure is understood for the bent pin problem, it can be reapplied to other system safety problems of even greater interest.

Finally, it should be noted that for the bent pin problem, the analyst is interested only in the off-diagonal elements of the powers of A. This is because diagonal elements represent the number of paths that short pin i to itself, but the bent pin analysis focuses on shorting pin i to another pin j. Therefore, only off-diagonal elements are of interest. In the next section, the value of the diagonal elements of $A^n(i,i)$ will become clear, where n is any positive integer power and $1 \le i \le N$ where N, as defined above, is the number of rows or columns of the square Adjacency Matrix.

The simple example presented here involves only nine pins, and each identical pin is short enough to involve only a nearest neighbor contact should it be bent. The results of most calculations can be verified easily by inspection. But what if a connector had 300 pins, each long enough to contact a second- or third-nearest neighbor? Then matters would be much more complex, and the analyst could not be *certain* that all possible short circuit possibilities were taken into account by inspection. The advantage of computational bent pin analysis is that it brings certainty to the process of identifying short circuit possibilities.

The Sneak Circuit Problem

A "Sneak Circuit" is any conducting loop of an electronic network that can, under certain conditions, initiate an undesirable function, or inhibit a desired function. The purpose of sneak circuit analysis is to identify such conducting loops.

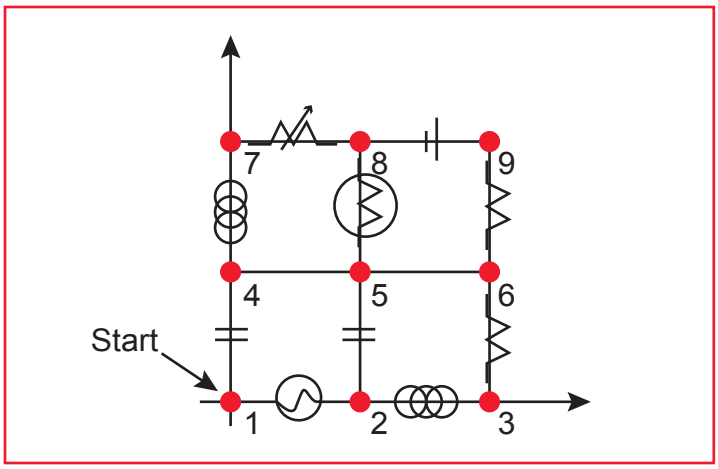


Figure 5 — A Fictitious "Bridge Circuit" Composed of Standard Electrical Symbols for Wires, Resistors, a Variable Resistor, Inductors, Capacitors, and a Battery and Signal Generator, all superimposed on Figure 1.

There are many well-known software products on the market that can help an analyst detect sneak circuits with some level of fidelity. Here, matrix methods will be used to detect sneak circuit loops with certainty. To understand how this goal is accomplished, consider the *bridge circuit* above. Does Figure 5 contain any sneak circuits? That is the question that will be answered.

All the components in Figure 5 are conducting in both directions. Therefore, in terms of conducting paths, Figure 5 can be reduced to the graph in Figure 1. The Adjacency Matrix is the same as that displayed in Figure 2. A good place to begin a sneak circuit analysis is at vertex 5 because it has the highest *incidence*. That means that more edges have vertex 5 as an end point than any other vertex. Next, count the number of loops that pass through vertex 5. In other words, count the number of loops that start from vertex 5 and return to vertex 5. Inspection of Figure 1 (or Figure 5) clearly indicates that loops may be four, six or eight edges (components) long. Therefore, the matrices A^4, A^6 and A^8 are of interest [Figure 6]. In particular, the (5,5) element of these matrices is of interest because it counts the number of circuit paths that start at vertex 5 and return to vertex 5. However, current can flow in two directions in Figure 5 (i.e., there are no diodes in place). This bi-directionality must be taken into account when interpreting the value of the (5,5) element and determining the number of physical loops to which it corresponds.

The value of $A^4(5,5) = 32$. Of these 32 conducting loops, each four electrical elements (edges) long, 24 are redundant loops and are of no interest to the analyst because they all demand unphysical electron motions. These redundant loops can be divided into three groups, as follows: First, there are eight *palindromic* redundant paths; these are

5→6→3→6→5,
5→6→9→6→5,
5→2→3→2→5,
5→2→1→2→5,
5→8→7→8→5,
5→8→9→8→5,
5→4→7→4→5 and
5→4→1→4→5.

"Palindromic" means that the sequence of vertices that defines each of these paths is the same in the forward and reverse direction. So even if electrons could move along such paths (they cannot), the mathematics

$$A^4 = \begin{pmatrix} 10 & 0 & 8 & 0 & 16 & 0 & 8 & 0 & 6 \\ 0 & 18 & 0 & 16 & 0 & 16 & 0 & 14 & 0 \\ 8 & 0 & 10 & 0 & 16 & 0 & 6 & 0 & 8 \\ 0 & 16 & 0 & 18 & 0 & 14 & 0 & 16 & 0 \\ 16 & 0 & 16 & 0 & \boxed{32} & 0 & 16 & 0 & 16 \\ 0 & 16 & 0 & 14 & 0 & 18 & 0 & 16 & 0 \\ 8 & 0 & 6 & 0 & 16 & 0 & 10 & 0 & 8 \\ 0 & 14 & 0 & 16 & 0 & 16 & 0 & 18 & 0 \\ 6 & 0 & 8 & 0 & 16 & 0 & 8 & 0 & 10 \end{pmatrix}$$

$$A^6 = \begin{pmatrix} 68 & 0 & 64 & 0 & 128 & 0 & 64 & 0 & 60 \\ 0 & 132 & 0 & 128 & 0 & 128 & 0 & 124 & 0 \\ 64 & 0 & 68 & 0 & 128 & 0 & 60 & 0 & 64 \\ 0 & 128 & 0 & 132 & 0 & 124 & 0 & 128 & 0 \\ 128 & 0 & 128 & 0 & \boxed{256} & 0 & 128 & 0 & 128 \\ 0 & 128 & 0 & 128 & 0 & 132 & 0 & 128 & 0 \\ 64 & 0 & 60 & 0 & 128 & 0 & 68 & 0 & 64 \\ 0 & 124 & 0 & 128 & 0 & 128 & 0 & 132 & 0 \\ 60 & 0 & 64 & 0 & 128 & 0 & 64 & 0 & 68 \end{pmatrix}$$

$$A^8 = \begin{pmatrix} 520 & 0 & 512 & 0 & 1024 & 0 & 512 & 0 & 504 \\ 0 & 1032 & 0 & 1024 & 0 & 1024 & 0 & 1016 & 0 \\ 512 & 0 & 520 & 0 & 1024 & 0 & 504 & 0 & 512 \\ 0 & 1024 & 0 & 1032 & 0 & 1016 & 0 & 1024 & 0 \\ 1024 & 0 & 1024 & 0 & \boxed{2048} & 0 & 1024 & 0 & 1024 \\ 0 & 1024 & 0 & 1016 & 0 & 1032 & 0 & 1024 & 0 \\ 512 & 0 & 504 & 0 & 1024 & 0 & 520 & 0 & 512 \\ 0 & 1016 & 0 & 1024 & 0 & 1024 & 0 & 1032 & 0 \\ 504 & 0 & 512 & 0 & 1024 & 0 & 512 & 0 & 520 \end{pmatrix}$$

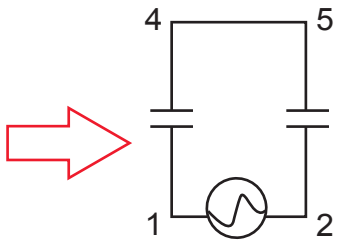
Figure 6 — A^4, A^6 and A^8 Corresponding to Figure 5.

has no way to identify the forward and reverse directions. Therefore, each of these paths will only be counted once in $A^4(5,5)$. Next, there are four palindromic redundant sequences that are repeats of two edge sequences counted by the $A^2(5,5)$ element; these are 5→4→5→4→5, 5→6→5→6→5, 5→2→5→2→5 and 5→8→5→8→5. Again, these each will make a single contribution to $A^4(5,5)$. The third (and last) group of redundant paths is those that are non-palindromic. There are six of these:

5→4→5→8→5,
5→4→5→2→5,
5→4→5→6→5,
5→8→5→6→5,
5→8→5→2→5 and
5→6→5→2→5.

Each will be counted twice, once in the forward direction and once in the reverse direction, for a contribution of 12 to the value of $A^4(5,5)$. The total count of redundant paths is, therefore, 8 + 4 + 2(6) = 24, as stated above — and all are to be ignored. The following are four non-redundant (and non-palindromic) physical paths of interest to the analyst:

5→4→7→8→5
5→8→9→6→5
5→6→3→2→5
5→2→1→4→5



Each of these physical loops is counted twice in $A^4(5,5)$, once in the forward direction and once in the reverse direction, for a total loop count value of 24 + 2(4) = 32. Therefore, it is certain that all conducting loops that are four edges (components) long and contain vertex 5 have been detected. Once all the simple physical conducting loops are known, each one can be easily checked to see if it is a sneak circuit. There is a science to this as well. However, since the focus of this publication is matrix methods, the solution to this second part of the sneak circuit problem will not be discussed in detail. For now, it is only necessary to note that the very last of the non-redundant loops above (5→2→1→4→5) is a sneak circuit. Why? It is because the impedance of a capacitor is given by $1/\omega C$, where C is capacitance, $\omega = 2\pi f$ and f is the frequency of the signal generator. Under the special conditions of high f or large C, the signal generator will be shorted and fail.

Similar reasoning can be applied to paths that are six and eight edges long — but none of those paths are sneak circuits. The large values of the (5,5) element of A^6 and A^8, 256 and 2,048 respectively, highlight the need to remove redundant conducting loops from the path count via the Redundancy Matrix. Although the solution to this sneak circuit problem might have been found by careful inspection of the simple bridge circuit in Figure 5, the object of these calculations has been to demonstrate a fail-safe *certain* matrix method of decomposing an electrical network of any complexity into all its simple loops, each of which may be checked to see if it is a sneak circuit or could be a sneak circuit under appropriate conditions. In practice, electrical networks may be hundreds of times more complex than the network in Figure 5.

As a second example, to emphasize some of these points, suppose the resistor between vertex 8 and vertex 5 (enclosed by a circle) was the detonator of a warhead. Suppose, also, that detonation depended on the current flow through this device as controlled by the variable resistor between vertices 7 and 8. This variable resistance might be low if a target is close, and high when it is far away, thereby controlling the current in loop 5→2→1→4→7→8→5 containing the signal generator, variable resistor and electrical detonator. But loop analysis indicates that an eight-edge path also exists that does not contain the variable resistor, viz., 5→4→1→2→3→6→9→8→5. So, there are potentially two ways that this hypothetical detonator could be activated. Is this what the designer intended? If not, one of these loops is a sneak circuit. In summary, the loop-counting procedure exposes all possible loops that pass through a safety-critical vertex (i.e., a vertex connected to a safety-critical part).

Finally, it is important to note that only the diagonal matrix elements are of concern in the loop-counting procedure — whereas in the Bent Pin Problem, only the off-diagonal elements were of interest. This is to say that the Bent Pin Problem and the Sneak Circuit Problem are, in a matrix sense, complementary to one another. In fact, *to each Bent Pin Problem (BPP) with Adjacency Matrix A, there corresponds a Sneak Circuit Problem (SCP) also with Adjacency Matrix A, such that the off-diagonal matrix elements of A are the "solution" (short circuit path counts) of the BPP and the diagonal matrix elements are the "solution" (conducting loop counts) of the SCP. Therefore, the BPP and the SCP are mathematical "Duals" of one another.* This, then, is the hidden relationship between the BPP and the SCP.

The Analysis of Software Logic

Suppose an analyst is confronted with the missile software flowchart in Figure 7. Each instruction will be labeled with a vertex number, 1 through 9. The flow of logic is denoted by an arrow (i.e., a directional edge). The instruction at vertex 2 is safety critical (a self-destruct decision). The analyst intends the self-destruct command to be activated if, and only if, two conditions are fulfilled: A) the missile is off-course and B) midcourse correction fails (i.e., A and B <=> self-destruct activated). It is easier to understand the shortcomings of the logic in Figure 7,

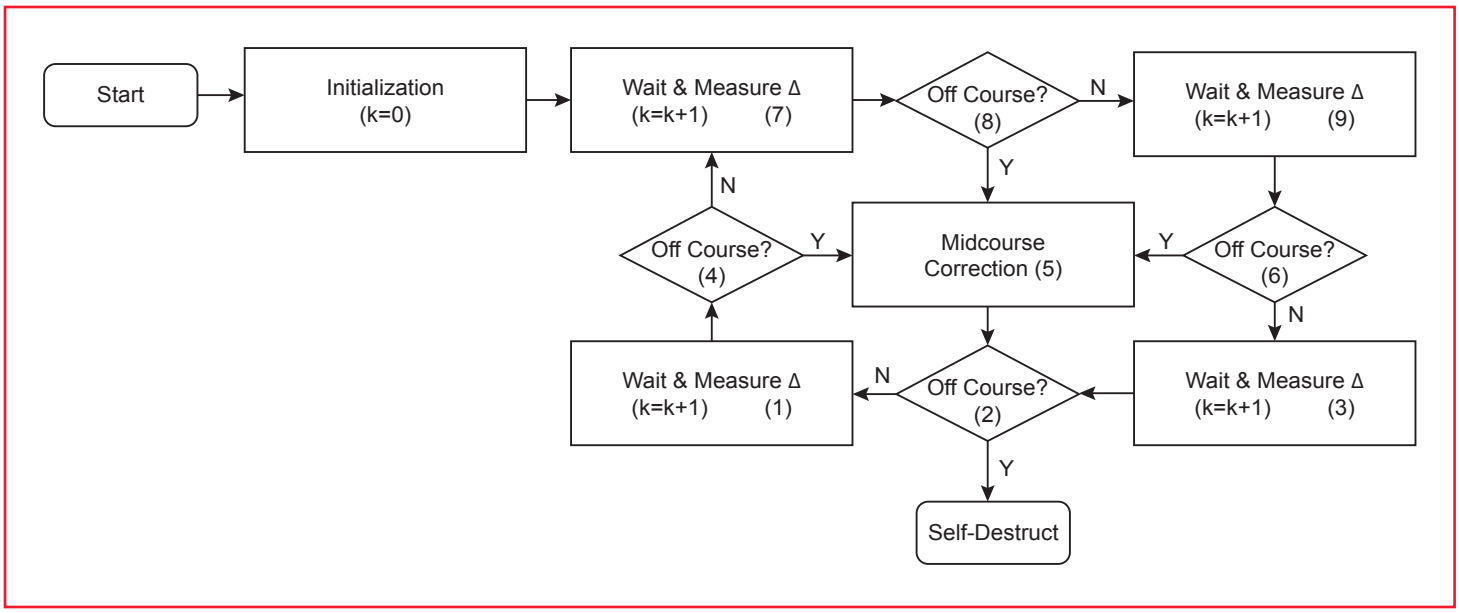


Figure 7— Flowchart for a missile guidance and self-destruct system. Δ is a measure of how off-course a missile is. The counter k is updated by 1 each time the system has to "wait," and is proportional to the missile time-of-flight.

and its repair, if the analyst's intentions are also phrased as a logically equivalent contrapositive statement and De Morgan's Law is applied: *Self-destruct will be suppressed if the missile remains on course or midcourse correction is successful* (i.e., in symbolic logic, self-destruct suppressed <=> ~A or ~B). In other words, logic will continue to flow as long as the missile remains on course or, if off course, a midcourse correction promptly returns the missile to its intended trajectory. Therefore, only two logic loops should satisfy the requirements of this code, one for ~A and one for ~B. And, only the ~B loop ("not B" loop), should be capable of suppressing self-destruct through safety-critical statement 2. But how many logical paths actually exist that are capable of suppressing the self-destruct instruction in Figure 7?

At first inspection, it might seem that the flowchart of Figure 7 has the same topology as Figures 1 or 5. In fact, this is not the case. Software logic, unlike the circuit in Figure 5, is directional, and that will change the appearance of the Adjacency Matrix in a profound way. First, how many ways can information move from vertex 1 to vertex 2? The analyst might be tempted to say "1." But that would be a mistake because the arrow on the connecting line (edge) is pointing in the wrong direction! However, there is a properly oriented directed line connecting vertex 1 to vertex 4. Therefore, the first row of the Adjacency Matrix is as shown in Figure 8, at right.

Similarly, vertex 2 has only one properly oriented arrow pointing to vertex "1," but no other vertices. Therefore, the second row of A has a "1" in position (i,j) = (2,1), but zeros everywhere else. In this manner, all elements of A may be assigned. Unlike the Adjacency Matrix in the Sneak Circuit Problem, this matrix is asymmetrical about the main diagonal running from the upper left down to the lower right of the matrix A. The Sneak Circuit Problem Adjacency Matrix would have been asymmetrical also if that network had contained

Destination vertex (j)

A=

	1	2	3	4	5	6	7	8	9
1	0	0	0	1	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0
4	0	0	0	0	1	0	1	0	0
5	0	1	0	0	0	0	0	0	0
6	0	0	1	0	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	1	0	0	0

Start vertex (i)

Figure 8 — The Adjacency Matrix for the flow chart in Figure 7. The asymmetry is due to the fact that Figure 7 forms a directed graph. This Adjacency Matrix is also more sparse than its predecessor in Figure 2 due to a reduction of the number of connections caused by directedness.

$A^4 =$

1	0	0	0	1	0	0	0	1
0	1	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0
0	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	0
0	0	0	2	0	0	0	0	0
1	0	1	0	1	0	0	0	0
0	2	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0

$A^6 =$

1	0	1	0	2	0	1	0	0
0	1	0	1	0	1	0	0	0
1	0	0	0	1	0	0	0	1
0	3	0	1	0	0	0	1	0
1	0	0	0	1	0	0	0	1
0	2	0	0	0	0	0	2	0
2	0	0	0	1	0	1	0	0
0	1	0	2	0	0	0	1	0
0	0	0	0	2	0	2	0	0

$A^8 =$

3	0	0	0	2	0	1	0	1
0	3	0	1	0	0	0	1	0
1	0	1	0	2	0	1	0	0
0	2	0	3	0	1	0	1	0
1	0	1	0	2	0	1	0	0
0	2	0	2	0	2	0	0	0
1	0	0	0	3	0	2	0	1
0	3	0	1	0	1	0	2	0
2	0	0	0	2	0	0	0	2

Figure 9 — Powers of A For the Flowchart of Figure 7.

a directional part, like a diode. But all the parts in the sneak circuit example allowed current to flow in both directions. The asymmetry of A means that there exists an arrow of time for the system it describes (i.e., electric current or logic current must flow in a given direction as

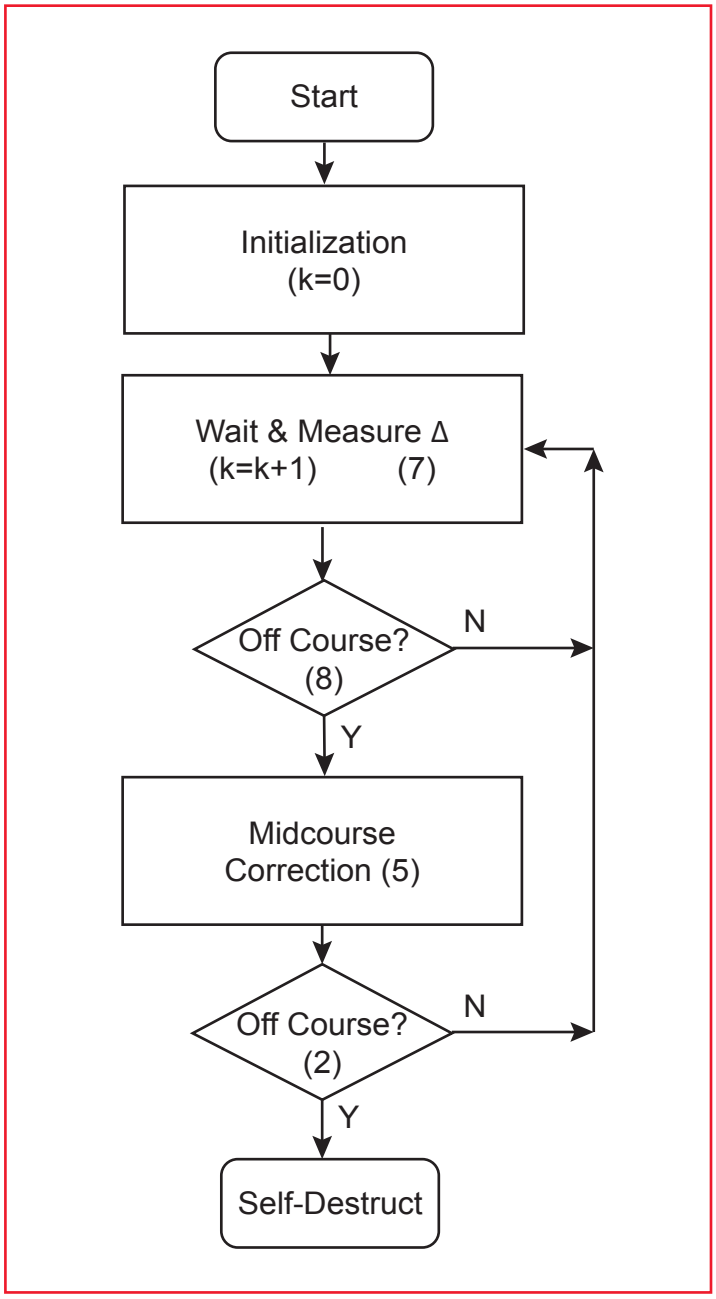


Figure 10 — Corrected Flowchart for a Missile Guidance and Self-destruct System.

time progresses). More will be said about this phenomenon in the conclusion of this paper.

The central question for the analyst now is, "How many logic loops pass through safety-critical statement 2?" To answer this question, the analyst must look at A^4, A^6 and A^8 as before. These are displayed in Figure 9.

The (2,2) element of these three matrices is $A^4(2,2) = 1$, $A^6(2,2) = 1$ and $A^8(2,2) = 3$. Notice that the entries of these three matrices are much smaller than those of the corresponding matrices for the Sneak Circuit problem. That is because redundant paths containing edge retracements in forbidden directions are no longer being counted. Also, since Figure 7 is directional and since direction has been explicitly taken into account in the

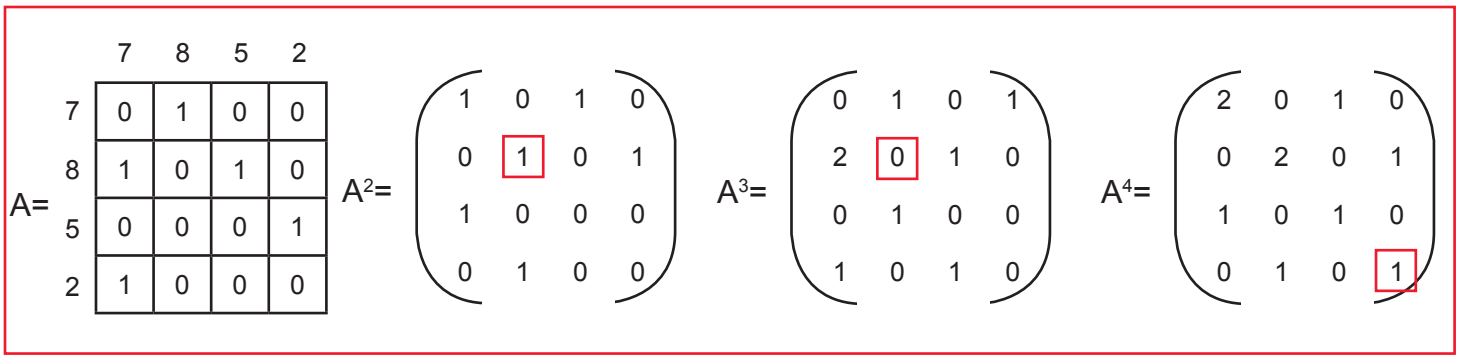


Figure 11 — The Adjacency Matrix and its lower powers for the redesigned flowchart in Figure 10.

corresponding Adjacency Matrix, it is not necessary to divide the (2,2) entries by 2 to get the number of (non-palindromic) software loops (physical loops) passing through safety-critical statement 2. The element $A^4(2,2)$ has a value of 1 because logic can only flow in the four-edge loop 2→1→4→5→2 (see Figure 7). Although many loops with six edges were allowed in Figure 5, here only one such loop is allowed (2→1→4→7→8→5→2) — directionality considerations forbid any others. Therefore, $A^6(2,2) = 1$.

Finally, there are three logic loops that are eight edges long. They are

2→1→4→7→8→9→6→3→2,
2→1→4→7→8→9→6→5→2 and
2→1→4→5→2→1→4→5→2;

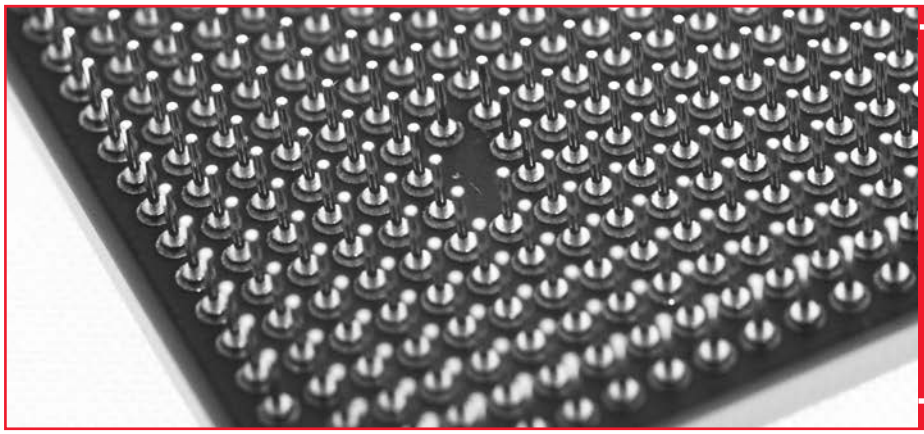
the last being the earlier-cited four-edge loop repeated to yield eight edges. Therefore, the total number of loops is five. But the analyst has only one "if-and-only-if" compound condition in mind for self-destruct. *Therefore, this program contains unnecessary code. All that is necessary are two loops total, only one of which contains safety-critical statement 2 that is necessary to activate/suppress self-destruct. All the others can be eliminated.* The result of this simplification might look like Figure 10.

Clearly, by inspection, there is only one logic loop that passes through safety-critical statement 2 and satisfies the ~B condition. But how can these notions be expressed numerically? As usual, construction of the Adjacency Matrix is the first step. Notice that the numerical labels on necessary instructions held over from Figure 7 have been kept as is. This will not cause any problems, since vertices can be labeled with any numbers or symbols that the analyst desires. Vertex (instruction) 7 is not connected to itself, but it is connected to vertex (instruction) 8 by an arrow with the proper directionality. Also, vertex (instruction) 7 is not directly connected to vertices (instructions) 5 or 2. Therefore, the first row of A is 0 1 0 0. Similarly, the other rows of A can be filled in to yield Figure 11 (far left). Examination of $A^2(8,8)$ yields a value of 1, meaning that logic flows in one independent loop two edges long if the missile is in an "on course" (i.e., ~A) condition (8→7→8). If the missile is off course, but midcourse correction has been successful (~B), matrix element $A^4(2,2)$ tells the analyst that logic will flow in only one loop four edges long that passes through safety-critical statement 2 (i.e., 2→7→8→5→2). If the missile is still off course after midcourse correction (statement 5), it will self-destruct via safety-critical statement 2. Note that the loop 8→5→2→8 is forbidden (i.e., $A^3(8,8) = 0$) as an independent loop via directionality considerations that are a natural consequence of the matrix calculations and form a kind of tacit "selection rule" on possible loops. Therefore, matrix analysis indicates that the code logic, represented by the flowchart in Figure 10, is now behaving as it should and has been purged of extraneous code.

Therefore, the number of logic loops passing through a safety-critical statement must equal the number of conditions in the analyst's mind for activating that statement. "Activation," in this context, can also mean suppression of a function. If there are fewer logic loops passing through the safety-critical statement than there are analyst's conditions, then some functionality in the analyst's mind will never be realized by the code. If there are more loops than conditions, then there are extra conditions unknown to the analyst that may trigger activation. These "sneak logic circuits" may make the code behave in unexpected or unwanted ways. Extra conditions can also manifest themselves as superfluous or unnecessary code, as in this last example.

Conclusion

The three problems that were selected for this paper were chosen not for complexity (although they are non-trivial), but primarily for their pedagogic value. They were also chosen to display certain features of the Adjacency Matrix A. For the Bent Pin Problem, A is always symmetric about its main diagonal. The reason for this is that current can flow in two directions along a bent pin because it is basically a non-rectifying conducting wire. Also, it is the off-diagonal elements of powers of A that are of interest in the Bent Pin Problem.



> "...matrix methods are one of the tools of that unifying mathematical theory. Intuitive methods are already obsolete, and it is only a matter of time before they are completely replaced by precise mathematical methods."

To every Bent Pin Problem there corresponds a complementary Sneak Circuit Problem, with an identical Adjacency Matrix. However, it is the diagonal matrix elements of A that are of interest now. And A for this "dual" Sneak Circuit Problem will be symmetric. If, however, the analyst is confronted with a Sneak Circuit Problem in which the electrical network under analysis contains rectifying components, then the Matrix A will be asymmetric. In general, the symmetry or asymmetry of A will depend on the details of any given Sneak Circuit Problem.

Finally, the matrix A for the Analysis of Software Logic will always be asymmetric about the main diagonal because software has a direction of flow. Therefore, the asymmetry of A defines an arrow of time for the system that it describes. Software starts from an initial instruction and, as time progresses, more and more instructions are involved. Software logic spreads with time, like gas molecules spreading into an evacuated box. So, software has an entropy change associated with it. The idea of entropy for a network is not new. It is part of complexity theory [Ref. 1], and it may be defined in several ways depending on the problem. Here, the most useful definition of entropy or, more properly, the change in entropy by physical analogy, ΔS, seems to be

$$\Delta S = \tfrac{1}{2} \sum_{\substack{\text{All } i,\, j \\ i \neq j}} | A_{ij} - A_{ji} | \quad .$$

where the pre-factor of ½ prevents double counting when the values of i and j are interchanged. The larger and the more complex the code, the larger will be ΔS. Entropy may sound like an abstract idea that is too theoretical to be useful. In fact, it is down to earth and practical. The higher the entropy of a piece of code, the more prone it will be to unexpected or undesirable behavior (such as timing errors, for example). Given the choice between several pieces of code or several software designs, the one with the lowest entropy should be preferred, since it will be the most robust. Entropy considerations aside, it should also be noted that it is the diagonal matrix elements of powers of A that are of interest for debugging logic. In that sense, the Analysis of Software Logic and the Sneak Circuit Problem are "similar" to each other (not "duals").

It has been tradition, in practice, to consider system safety problems as heuristic, disconnected and independent problems whose solutions involve methods that are primarily intuitive. In fact, there is an underlying and unifying theory to system safety. And, as shown in this publication, matrix methods are one of the tools of that unifying mathematical theory. Intuitive methods are already obsolete, and it is only a matter of time before they are completely replaced by precise mathematical methods.

About the Author

Dr. Richard R. Zito is the Manager of Richard R. Zito Research LLC, located in Tucson, Arizona. He is the author of 106 publications, the inventor on 14 patents and the winner of four Small Business Innovation Research (SBIR) awards.

References

1. Marshall, C.W. *Applied Graph Theory*, pp. 178-179, 258-260, 235-240, John Wiley, New York, 1971.
2. Selby, S.M. *Standard Mathematical Tables, 16th ed.*, p. 108, CRC, Cleveland, Ohio, 1968.
3. Hanselman, D. and B. Littlefield. *Mastering MATLAB 5*, Prentice-Hall, Upper Saddle River, New Jersey, 1998.
4. Person, R. *Using Microsoft EXCEL 97*, Que, Indianapolis, Indiana, 1997.
5. Zito, R.R. "The Curious Bent Pin Problem – I," *Proceedings of the 29th International Systems Safety Conference*, Las Vegas, Nevada, August 8 – 12, 2011.