Magic Squares

Generating a classic array by computer

BY ROBERT T. KUROSAKA

A magic square is an array of numbers, usually consecutive, arranged so that the sum of each row, column, and main diagonal is a constant. A 4-by-4 (or order-4) magic square is shown in figure 1; its constant sum is 34.

How are magic squares constructed? If we were to try to create a 4-by-4 magic square by "brute force," we could program a computer to examine all possible 4-by-4 arrays of the numbers 1 through 16. Eventually, this would produce all possible order-4 magic squares. Unfortunately, the number of possible arrays is $16! = 1\times2\times3\times \ldots \times14\times15\times16 > 2\times10^{13}$. If the computer could examine one array per microsecond, it would take eight months to complete the project.

Perhaps we'd be better off using a little mathematical intuition and some trial and error. Let us construct an order-3 magic square without electronic aids. The numbers 1 through 9 are placed in a 3-by-3 array. Wherever they are placed, the *average* value of each entry is 5, the middle number of the series. Any particular row, column, or diagonal contains three numbers, each with an average value of 5. Therefore, their sum will be 15, the constant sum for an order-3 magic square.

In general, a magic square of order N contains N^2 numbers whose average is $(1+N^2)/2$, the average of the smallest and largest numbers. Any row, column, or diagonal has N such numbers; their sum will be $N(1+N^2)/2$.

In our order-3 magic square, it is fairly obvious that the middle number, 5, should be in the center cell (figure 2a) and that the other eight numbers should be paired so that their sum is 10: (1,9), (2,8), (3,7), and (4,6). These pairs will be located in diametrically opposite cells.

Suppose the "1" were placed in a corner cell, say *a*. The "9" will then be in cell *i* (figure 2b). We find that the "8" cannot be entered. It cannot be placed in cells *c*, *f*, *g*, or *h*, making the sum of the right column or bottom row exceed 15. If the "8" is in cell *b* (or *d*), cell *c* must be "6," and the right column exceeds 15. This means the "1" must not occupy a corner cell.

Place the "1" in cell *b* and the "9" in cell *h* (figure 2c). The "7" cannot be placed in *g* or *i*, making the bottom row exceed 15. Nor can it be in *a* since we will be compelled to place another "7" in cell *c* because 7+1+*c* must equal 15. The "7" must therefore be in the second row, say at *d*. Inspecting figure 2d, we can easily fill the remaining cells: the "8" must be at *c*, the "2" at *g*, and so on. The completed order-3 magic square is shown in figure 2e.

Magic squares of larger order will require even more trial and error, making a more orderly procedure desirable. For example, the order-4 magic square can be constructed as follows: recite the numbers from 1 through 16, reading from left to right in the array, and enter numbers only in the cells lying on the two main diagonals (see figure 3). Starting at the last (lower right) cell, count from 1 through 16 again, moving *backward* through the array, and enter numbers in the empty cells only. The result will be the order-4 magic square shown in figure 1. Unfortunately, this procedure works only for order-4 magic squares.

Is there a more general procedure that works for as many different orders of magic squares as we please? In this column, I'll present some algorithms for constructing any odd-order magic square. The approach won't work for even-order squares, which lack a center square and therefore behave very differently.

The odd-order magic-square algorithms are elementary and easily programmed. The outline for the procedure follows:

1. Select a starting cell and enter the "1."
2. Select the move, a repeated maneuver used to enter the "2," the "3," and so on.
3. Write an edge-guard routine to prevent moving off the array.

(continued)

Robert T. Kurosaka teaches mathematics in the Massachusetts State College system. He invites your correspondence, c/o BYTE, POB 372, Hancock, NH 03449.

4. Determine the break-move, a second maneuver used when progress is blocked by an already-filled cell.

In choosing the starting cell, only the center cell is forbidden. For our first example, use the middle cell of the top row. For a magic square of order *N*, this position would be ROW = 1, COLUMN = (*N*+1)/2 in an *N*-by-*N* array. Our example will be an order-5 magic square.

For the move, we will use the northeast diagonal move. In figure 4a, the "1" is in the starting cell. Moving in a northeasterly direction, we enter the other numbers into the empty cells. When we move off the edge of the array, we reenter at the opposite edge, using a wraparound effect as if the array were printed on a cylinder. In the program, the edge-wrapping statements see to this task.

We continue in this manner until we encounter an already-filled cell. When this occurs, the break-move is performed, interrupting our diagonal progress briefly. For our present example, the break-move is down-one. That is, place the next number in the cell directly below the last cell that was filled. In figure 4b, the numbers "1" through "5" have been entered; the "6" is blocked by the "1" in the next diagonal cell. Using the break-move, we enter the "6" directly below the "5" (the last cell filled, not below the "1"). We continue diagonally until we are blocked again and so on. The completed order-5 magic square is shown in figure 4c.

Some observations: every cell is filled. Each row, column, and diagonal has the same sum of 65. The final entry, 25, is diametrically opposite the first entry, 1. The middle number, 13, occupies the center cell. These conditions are necessary for a magic square of odd order and should be checked after each construction.

You may wish to run the sample program (see listing 1) before reading these detailed comments. [*Editor's note: The listing for Magic Square is available for downloading via* BYTEnet Listings. *The telephone number is* (603) 924-9820.]

Line 200 creates the northeast move. The computer is told to move up-one, right-one from its present position in the array.

Lines 210 and 220 are the edge-guard statements for this move, preventing off-the-array movement at the top and right edges.

Line 240 is a retreat-move that is required just before the break-move in line 250. Recall that the break-move (down-one) is performed when an already-filled cell is encountered, and it is performed from the *last-filled cell*. In figure 4b, the "5" is in the cell at (2,2), and the computer performs line 200 (moves northeast) and considers the cell at (1,3). That is, the computer is "at" (1,3) already. This cell fails the test at line 230 since it does not contain zero, so the "6" must be entered below the "5." The computer is told in line 240 to retreat to the last-filled cell (down-one, left-one) and then to perform the break-move (down-one) in line 250. Of course, these two lines can be combined into one statement (down-two, left-one). However, as you modify and expand the program to include a variety of moves, starting cells, and break-moves, you may find it convenient to separate the retreat-move

(*continued*)

1	15	14	4
12	6	7	9
8	10	11	5
13	3	2	16

Figure 1: *A 4-by-4 magic square. Each row, column, and main diagonal adds up to 34.*

a)	a	b	c	b)	1	b	c
	d	5	f		d	5	f
	g	h	i		g	h	9
c)	a	1	c	d)	a	1	c
	7	5	3		d	5	f
	g	9	i		g	9	i
e)	6	1	8				
	7	5	3				
	2	9	4				

Figure 2: *Steps in the construction of a 3-by-3 magic square by trial and error.*

1	•	•	4
•	6	7	•
•	10	11	•
13	•	•	16

Figure 3: *The counting-algorithm construction technique for a 4-by-4 magic square. The figure shows the square after finishing the counting-forward step.*

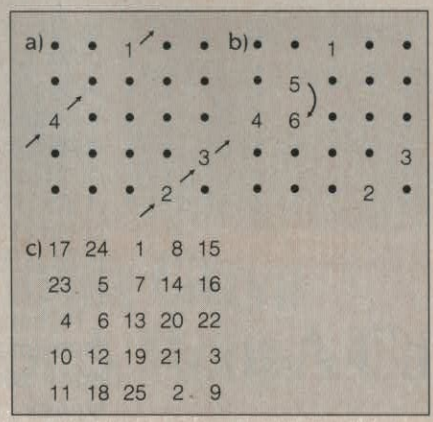

Figure 4: *(a) The northeast diagonal move for constructing odd-parity magic squares from the top-row, middle-column position. (b) The break-move for this square's construction. (c) The completed magic square.*

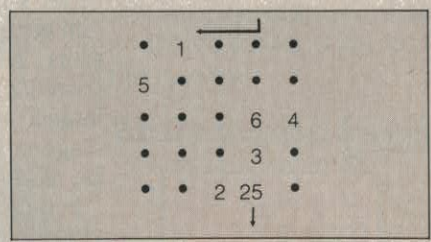

Figure 5: *The shortest path for an off-center starting position.*

Listing 1: *The Magic Square program.*

```
10 '*****************************************************************************************************
20 '*                                                                                                   *
30 '*                                    MAGIC SQUARE PROGRAM                                           *
40 '*                                                                                                   *
50 '*                                       by Bob Kurosaka                                             *
60 '*                                                                                                   *
70 '*****************************************************************************************************
80 REM
90 REM
100 DEFINT A-Z:CLS
110 INPUT "Enter the size of the square's side (odd number, 3 or larger)";SIDE
120 IF SIDE/2=INT(SIDE/2) THEN PRINT "SIDE MUST BE AN ODD NUMBER":GOTO 110
130 IF SIDE<3 THEN PRINT "NUMBER MUST BE 3 OR LARGER":GOTO 110
140 DIM ARRAY(SIDE,SIDE)
150 CLS
160 ROW=1: COLUMN=(SIDE+1)/2                    'LOCATE STARTING CELL
170 ARRAY(ROW,COLUMN)=1                         'INITIALIZE FIRST VALUE
180 REM ARRAY-FILLING ROUTINE
190 FOR I=2 TO SIDE^2
200       ROW=ROW-1:COLUMN=COLUMN+1             'NORTHEAST MOVE
210       IF ROW<1 THEN ROW=ROW+SIDE            'WRAP AROUND THE EDGES OF THE ARRAY
220       IF COLUMN>SIDE THEN COLUMN=COLUMN-SIDE
230       IF ARRAY(ROW,COLUMN)=0 THEN 280       'IF CELL IS EMPTY, FILL IT
240       ROW=ROW+1:COLUMN=COLUMN-1             'OTHERWISE, RETREAT
250       ROW=ROW+1                             'AND MAKE THE BREAK-MOVE
260       IF ROW>SIDE THEN ROW=ROW-SIDE         'CHECK FOR EDGE-WRAPPING CONDITIONS
270       IF COLUMN<1 THEN COLUMN=COLUMN+SIDE
280       ARRAY(ROW,COLUMN)=I                   'FILL THE CELL
290 NEXT I
300 REM PRINT THE SQUARE
310 PRINT "MAGIC SQUARE, ORDER";SIDE
320 PRINT "EACH ROW, COLUMN, AND DIAGONAL ADD UP TO";SIDE*(SIDE^2+1)/2
330 PRINT
340 FOR ROW=1 TO SIDE
350       FOR COLUMN=1 TO SIDE
360             PRINT USING "####";ARRAY(ROW,COLUMN);
370       NEXT COLUMN
380       PRINT
390 NEXT ROW
400 END
```

Listing 2: *Changes to the program to use a northwest move.*

```
200 ROW=ROW-1: COLUMN=COLUMN-1                  (northwest move)
220 IF COLUMN<1 THEN COLUMN=COLUMN+SIDE         (new edge-guard statement)
240 ROW=ROW+1: COLUMN=COLUMN+1                  (new retreat-move)
270 IF COLUMN>SIDE THEN COLUMN=COLUMN-SIDE      (new edge-guard statement for the new retreat-move)
```

Listing 3: A *complete edge-guard subroutine.*

```
410 IF ROW<1 THEN ROW=ROW+SIDE
420 IF ROW>SIDE THEN ROW=ROW-SIDE
430 IF COLUMN<1 THEN COLUMN=COLUMN+SIDE
440 IF COLUMN>SIDE THEN COLUMN=COLUMN-SIDE
450 RETURN
```

(continued)

and the break-move for clarity.

For variety, the northwest move may be used with a few changes in the program (see listing 2).

For even more variety, a different starting cell may be chosen. This, however, dictates a new break-move in line 250, and we must alter its edge-guard statements accordingly.

Here is the method for determining the break-move for any permitted starting cell. Once the "1" is placed in the starting cell, the final cell is also determined; it is always diametrically opposite the "1." The shortest path from the final cell to the starting cell will be the break-move.

In figure 4c, one path from the "25" back to the "1" may be called up-four, but using the wraparound effect, the shortest path is down-one (250 ROW =ROW+1). In figure 5, the "1" is left of center in the top row. The "25" will be right of center in the bottom row. The shortest path from "25" to "1" is down-one, left-two (ROW=ROW+1: COLUMN=COLUMN−2). After entering "1" through "5" with a northeast move, the "6" is blocked. The break-move causes the "6" to be entered down-one, left-two from the "5."

Try experimenting with other starting cells and other diagonal moves to discover which starting cells permit which diagonal moves. Hints: The four corner cells may not be used as starting cells. The center cell is always forbidden. If the starting cell is adjacent to the center cell (to its left or right, above or below), any of the four diagonal moves may be used.

With this large choice of moves and break-moves, a complete edge-guard subroutine may be in order (see listing 3). Then lines 210 and 220 could be replaced by GOSUB 410, as could lines 260 and 270.

If you require a challenge beyond diagonal moves, try the Knight's move, the L-shaped move used in chess. Move two cells in any direction, turn 90 degrees, and move one cell. For example, one Knight's move is right-two, up-one, or its equivalent, up-one, right-two (ROW=ROW−1: COLUMN=COLUMN+2). Variations of the Knight's move (such as up-one, right-*three*) may also be used.

Knight's moves offer both challenges in programming and variety in the results. We may use nearly any starting cell, even the corners, and choose from up to eight different Knight's moves. (You will learn, however, that most starting cells will not permit a choice of all eight Knight's moves.)

I hope this brief lesson has demystified the magic square for you and that you feel inspired to experiment further. In the meantime, I welcome your responses: questions, comments, criticisms, improvements, insights, conjectures, and suggestions for future columns. ■