Memory Efficient Algorithm and Closed Form Formulas for Computing Odd Magic Squares of Higher Order

P.C. Perera

Department of Engineering Mathematics, Faculty of Engineering
University of Peradeniya

Introduction

Magic squares have been studied for at least three thousand years, the earliest recorded appearance dating to 2200 BC, in China. In the 9th century, Arab astrologers used them in calculating horoscopes, and by 1300 AD, magic squares had spread to the West. An engraving by the German artist Albrecht Dürer included a magic square in which the artist embedded the date, 1514, in the form of two consecutive numbers in the bottom row. Because the concept of a magic square is so easily understood, magic squares have been particularly attractive to puzzlers and amateur mathematicians. In modern mathematics, the notions pertaining to magic squares are utilized to characterize magic graphs as well as magic cubes. (Jezny and Trenkler et al. 1983 and Adler, Robert Li et al. 1978)

The n^{th} order magic square is an $n \times n$ array consisting of the first n natural numbers such that the sum along rows, along columns as well as along two diagonals are all equal to $\frac{n(n^2+1)}{2}$. Of course, given any magic square, a rotation or reflection will produce another magic square. Not counting these as distinct, it is known that there is only one 3^{rd} order normal magic square, and there are 880 normal 4^{th} order magic squares. The number of distinct normal magic squares increases dramatically with its size. For instance, there are over 13 million normal magic squares of 5^{th} order. The algorithm devised in this endeavor produces only one such normal magic square for a given odd natural number.

Even though the computational aspects of lower order magic squares are fairly simple and straightforward, those of the higher order counterparts demand memory and time efficient algorithms. The computational time as well as the memory requirement for the computation of $n \times n$ magic square are both of order n^2. The unavailability of a closed form expression for the n^{th} order magic square results lengthy source codes. Moreover, as n becomes very large, high memory requirement makes the computation of n^{th} order magic square impossible. In this endeavor, it is attempted to devise a memory efficient algorithm and formulate a closed-form expression to compute odd higher order magic squares by formulating de la Loubere type magic square.

The objectives of this work are to develop memory efficient algorithms in the context of n^{th} ($n \bmod 2 = 1$) order magic square to determine,

(a) the cell $(i,j) \in \mathcal{N} \times \mathcal{N}$ for a given k satisfying $1 \leq k \leq n^2$, and

(b) the entry k with $1 \leq k \leq n^2$ when the cell $(i,j) \in \mathcal{N} \times \mathcal{N}$ is specified,

where $\mathcal{N} = \{1, 2, ..., n\}$.

Preliminaries

Letting the entry of the i^{th} row and the j^{th} column of the magic square be x_{ij}, the problem of magic square can be formulated as follows. For the sake of notational simplicity,

henceforth, unless specified otherwise, the entry of the cell corresponding to the i^{th} row and the j^{th} column of the n^{th} order magic square is denoted by $M_n(i,j)$. Then,

$$\sum_{j=1}^{n} x_{ij} = \frac{n(n^2+1)}{2} \quad \text{for } i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} = \frac{n(n^2+1)}{2} \quad \text{for } j = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ii} = \frac{n(n^2+1)}{2}$$

$$\sum_{j=1}^{n} x_{n+1-j,j} = \frac{n(n^2+1)}{2}$$

where $x_{ij} \in \{1, 2, ..., n^2\}$.

It should be noted that the solution of the problem is not unique. The optimum algorithm available at present is used in MATLAB. In MATLAB, two $n \times n$ arrays are declared by the function MESHGRID in the process of obtaining the n^{th} order magic square. Then those two $n \times n$ arrays are manipulated to obtain the n^{th} order magic square. Thus, it is obvious that the demand for the memory is an issue as n becomes large. To comprehend the demand for the memory and the associated complexity of other existing algorithms, consider the following algorithm used in MATLAB.

```
function M = magic(n)

n = floor
 ...(real(double(n(1))));

if mod(n,2) == 1
   [J,I] = meshgrid(1:n);
   A = mod(I+J-(n+3)/2,n);
   B = mod(I+2*J-2,n);
   M = n*A + B + 1;
end
```

In the case of the above MATLAB routine, **A**, **B**, **I** and **J** are of n^{th} order square matrices. Thus, the memory requirement and the computational time pertaining to the above algorithm may be very high as n becomes very large. The case is the same for the other existing algorithms used for computation of magic squares. The algorithm devised in this work requires much less computer memory as well as computational time.

The main result

The following results are conjectured and proved in this work. For the sake of brevity, their proofs are not presented here.

Theorems

Theorem 1. *For a given n and k satisfying $n \bmod 2 = 1$ and $1 \leq k \leq n^2$,*

$$M_n(a, b) = k,$$

where

$$a = \left(\frac{(n+2k-3)}{2} - \left\lfloor \frac{k-1}{n} \right\rfloor \right) \bmod n + 1$$

and

$$b = \left(n + k - 2 - 2\left\lfloor \frac{k-1}{n} \right\rfloor \right) \bmod n + 1$$

Example 1. *If $n = 7$, using the above algorithm, the MATLAB routine*

```
n=7;
for k=1:n^2,
a=mod((n+2k-3)/2+
    ...floor((k-1)/n),n)+1;
b=mod((n-k-2+
    ...2*floor((k-1)/n),n)+1
M(a,b)=k;
end;
```

produces 7^{th} order magic square as

30	39	48	1	10	19	28
38	47	7	9	18	27	29
46	6	8	17	26	35	37
5	14	16	25	34	36	45
13	15	24	33	42	44	4
21	23	32	41	43	3	12
22	31	40	49	2	11	20

Remark 1. *In the light of Example 1, it is obvious that this algorithm converts a problem which demands the manipulation of two $n \times n$ arrays, to a problem of simplifying a simple expression involved two variables, namely, n and k.*

Theorem 2. *Denoting $\{1, 2, ..., n\}$ by $\mathcal{N}$, for a given cell $(i, j) \in \mathcal{N} \times \mathcal{N}$, the corresponding entry $M_n(i, j)$ of the n^{th} order magic square is given by*

$$M_n(i,j) = n\left[i - j + \frac{(n-1)}{2}\right] \bmod n + (2i - j - 1) \bmod n + 1.$$

Example 2. *If $n = 11$, using the above algorithm, the MATLAB routine*

```
n=11;
for j=1:n,
M(2,j)=n*mod( i-j+(n-1)/2,n)+
...mod(2i-j-1,n)+1;
end;
```

produces the set of entries in the second row of the 11^{th} order magic square as

80	93	106	119	11	13	26

... | 39 | 52 | 65 | 67 |

Conclusions

Unlike the other algorithms associated with magic squares, this algorithm can be converted to any high level language very easily due to the absence of manipulations of arrays. Nevertheless, it should be noted that the above algorithms are not applicable for the even order magic squares. Moreover, the improvement of the time efficiency associated with the algorithms developed in this endeavor are quite insignificant. Hence, the above facts should be considered in the process of generalizing and extending the aforementioned algorithms to make them applicable for the computation of a magic square of any order.

References

Jezny, S. and Trenkler, M. (1983) Characterization of magic graphs, *Czechoslovak Math. Journal*, 33(2), 435-438.

Adler, A. and Robert Li, S.Y. (1978) Magic N-cubes and prouhet sequences, *American Mathematical Monthly*, 84(1), 618-627.