MATLAB's Magical Mystery Tour

Ancient matrices with mystical powers

Magic squares? Why does MATLAB have magic squares? Even though their origins lie in centuries old recreational mathematics, they turn out to be very useful today for explaining features of MATLAB and for illustrating concepts from linear algebra.

An *n-by-n* magic square is an array containing the integers from 1 to n^2, arranged so that all the rows and columns have the same sum. For each $n > 2$, there are lots of magic squares of order n, but MATLAB's function `magic(n)` generates a particular one. Here is the 4-by-4 magic square

```
A = magic(4)
    16    2    3   13
     5   11   10    8
     9    7    6   12
     4   14   15    1
```

The function `sum(A)` forms the column sums. Here it produces

```
sum(A) =
    34   34   34   34
```

This shows that all the columns have the same sum. But why is it 34? Because `sum(1:16)/4` is 34. For a 4-by-4 magic square, the magic sum has to be 34.

The row sums can be obtained by transposing the matrix with `A'`, summing the columns of `A'`, and then transposing the result.

```
sum(A')'
    34   34   34   34
```

MATLAB's magic squares are among the "special" magic squares whose diagonals also have the magic sum. The principal diagonal goes from the upper right to the lower left. Its sum is

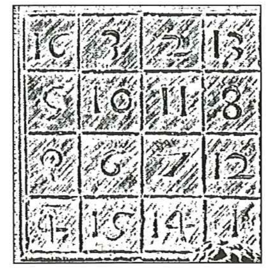
```
sum(diag(A))
    34
```

To get the other, "antidiagonal," we use a function originally intended for reorienting graphics arrays.

```
B = fliplr(A)
        13    3    2   16
         8   10   11    5
         2    6    7    9
         1   15   14    4
sum(diag(B))
        34
```

Row and columns permutations preserve the magic property. Let's swap the second and third columns.

```
A(:,[1 3 2 4])
    16    3    2   13
     5   10   11    8
     9    6    7   12
     4   15   14    1
```



This particular permutation appears in the Renaissance engraving *Melancolia II* by the German artist and amateur mathematician Albrecht Durer. It allowed him to slip in the date, 1514, when he did the work.

For a magic square of order n, the magic sum is `sum(1:n^2)/n`, which turns out to be $(n^3 + n)/2$. Let's call this value μ_n. So $\mu_3 = 15$, $\mu_4 = 34$, $\mu_5 = 65$, etc. Here is our first exercise for the reader: explain why $n^3 + n$ is always divisible by 2.

Many other MATLAB functions return μ_n when applied to `magic(n)`. The function `norm(A)` measures the maximum possible magnification of the linear transformation represented by `A`. The one and infinity norms, `norm(A,1)` and `norm(A,inf)`, are simply the largest row and column sums. So they are obviously equal to the magic sum. But the Euclidean norm, denoted by `norm(A,2)` or simply `norm(A)`, also turns out to be μ_n. We leave the explanation as another exercise, although this one is nontrivial.

Eigenvalues and eigenvalues are at the heart of many of MATLAB's matrix operations. What is the largest eigenvalue of `A = magic(n)`? You are well advised to guess that it's the magic sum, $\mu = \mu_n$. The fact that μ is *some* eigenvalue is easy to see. It's because the vector of all ones, `e = ones(n,1)`, is an eigenvector. The matrix-vector multiplication, `A*e`, is simply another way of computing the row sums and

```
A*e = μ*e
```

The fact that μ_n is actually the *largest* eigenvalue is an illustration of the Perron-Froebenius theorem, which is a deep theorem about eigenvalues of matrices with positive elements.

Singular values are another powerful tool in MATLAB's matrix arsenal. Sure enough, the largest singular value, `max(svd(magic(n)))`, is also μ_n. Explaining why that is true is another exercise.

What is known about the eigenvalues and singular values other than the largest, the Perron root? Well I, for one, don't know very much. If anybody can provide interesting characterizations of these values, I'd like to hear about it.

Now for a bit of a surprise. What is the matrix inverse of `magic(n)`? It

Melancolia II by Albrecht Durer, 1514

depends upon whether *n* is odd or even. For odd *n*, the matrices `magic(n)` are well conditioned. The matrices

```
X = inv(magic(n))
```

do not have positive, integer entries, but they do have equal row and column sums.

But, for even *n*, the determinant, `det(magic(n))`, is zero, and the inverse does not exist. If `A = magic(4)`, trying to compute `inv(A)` produces an error message.

With `null(A)` or `rref(A)`, we can find that the vector `v = [1 3 -3 -1]'` satisfies `A*v = 0`. So there is a linear dependence among the columns of A

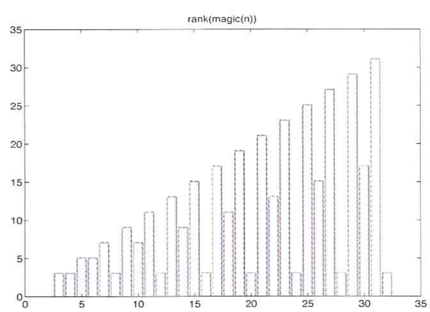
```
A(:,1) + 3*A(:,2) = 3*A(:,3) + A(:,4)
```

The rank of a matrix is the number of linearly independent rows and columns. A *n*-by-*n* matrix is singular if its rank, *r*, is not equal to its order. Here is a table of the rank of the magic squares up to order 20, generated with

```
for n = 3:20, r(n) = rank(magic(n)); end
```

n =	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
r =	3	3	5	5	7	3	9	7	11	3	13	9	15	3	17	11	19	3

Do you see the pattern? Maybe a bar graph will help.



Here is the pattern.

n	rank
odd	n
even, not divisible by 4	n/2+2
divisible by 4	3

The explanation for this intriguing behavior lies in the algorithms MATLAB uses to generate magic squares. I learned about the algorithm for odd order when I was in junior high school and I spent many idle hours in band practice generating magic squares.

Maybe you already know the algorithm, or can see it in `magic(5)`.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

The integers from 1 to 25 are inserted along diagonals, starting in the middle of the first row and heading in a northeasternly direction. When you go off an edge of the matrix, which you do at the very first step, continue from the opposite edge. When you bump into a cell that is already occupied, drop down one row and continue. Programming this algorithm is a good exercise, although it is tricky to vectorize. Those of you who are spreadsheet whizzes might try it there.

I find it plausible, but I don't have a formal proof, that the algorithm for odd *n* generates a nonsingular matrix. It just seems highly unlikely that any linear dependencies among the rows or columns would be created by this process.

The algorithms for even order are another matter. There are two, one for "singly even"— *n* is divisible by 2, but not by 4 — and one for "doubly even." I didn't learn about these algorithms until I wanted to include magic squares in the first MATLAB.

If *n* is singly even, then *n/2* is odd and `magic(n)` can be constructed from four copies of `magic(n/2)`. For example, `magic(10)` is obtained from `A = magic(5)` by forming

```
[ A      A+50
  A+75   A+25]
```

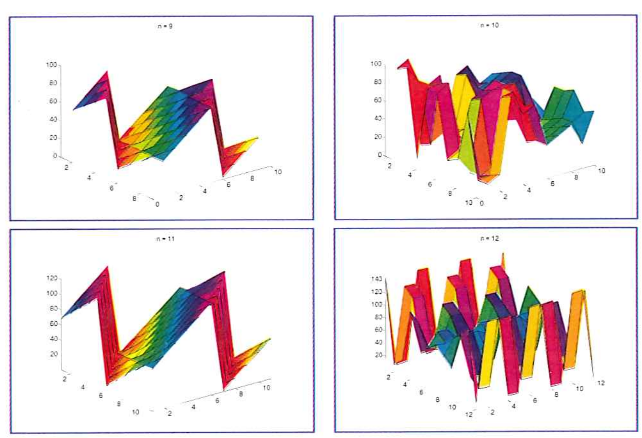
This might be called a "Kronecker sum" of `A` and `25*M` where `M+1 = [1 3; 4 2]` is the closest we can get to a magic square of order 2. The column sums are all OK because `sum(A) + sum(A+75)` equals `sum(A+50) + sum(A+25)`. But the rows sums are not quite right. The algorithm must finish by doing a few swaps of pieces of rows to clean up the row sums.

The Kronecker sum is singular. It's rank is only *n/2*. The partial row operations boost this to *n/2+2*, but do not scramble the elements enough to generate an invertible matrix.

Here is your last assignment for today. Investigate `magic(n)` when *n* is divisible by 4, so both *n* and *n/2* are even and neither of the above algorithms is applicable. Since `magic()` is a built-in function, you'll have to reverse engineer MATLAB's algorithm. Then explain why the rank is always 3.

It's time for a little graphic relief. Shown below are

```
surf(magic(9))     surf(magic(10))
surf(magic(11))    surf(magic(12))
```



You can see the three different cases — on the left, the upper right, and the lower right. If you increase of each of the orders by 4, you get more cells, but the global shapes remain the same. The odd *n* case on the left reminds me of Origami.

Well, there's more, but we'll leave that for another place. Just remember this. Durer and his friends in the 16th century were right. These squares really are magic.