A First Look at Octave

GNU Octave (or Octave) is a software package for scientific computing. It is basically a clone of Matlab, and is available to download, modify and redistribute under the GNU General Public License (GPL) as published by the Free Software Foundation. As such, a good number of programs written to run in Matlab can run in Octave and vice-versa. When you find a Matlab program which does not run in Octave, the reason is often due to the fact that Octave is worked on largely by volunteers, so the introduction of new features lags behind that of the commercial package it mimics.

The official website for GNU Octave is `https://www.gnu.org/software/octave/`. From there you can download and install the software on your own computer. That is the most convenient way to try out the commands demonstrated in this introduction and, indeed, many sample codes displayed and used during the course. Once you have installed and started up Octave, here are some first (basic) operations.

Building a matrix

For me, more than any other feature of Matlab/Octave, it is the ease of building and using matrices that make it a software package of choice in linear algebra. Matrices are enclosed in '[' and ']' (square brackets), entries on a row are separated by at least one space (though entries can be expressions such as '3 - 2/5', so when a space is encountered, say, before or after a '+' or any mathematical operation, Octave will not interpret that space as ending the current entry), a semicolon is used to break between entries of one row and those of the next, and every row must have the same number of entries.

```
octave:19> [1 2 3; 4 2 + 3 6]
ans =


   1   2   3
   4   5   6
```

The result of most computations can be assigned a name. As no name was explicitly given to the 2-by-3 matrix above, it is called `ans`, at least until another computation reassigns a value to `ans`. Computations which do not end in a semicolon have values echoed to the screen.

```
octave:2> A = [1 2; 1 3; 1 4];   B = [1−i 2+i; 7 3*i ];
```

The line above contains two commands, assigning matrices to names `A` and `B`. As the assignment of `B` shows, an entry in a matrix can be a nonreal complex number. A variable name, when typed, reveals its contents, unless, of course, it is followed by a semicolon.

```
octave:24> A
```

```
A =

   1   2
   1   3
   1   4
```

The **Hermitian**, as well as the **transpose**, of an m-by-n matrix is n-by-m. When the matrix has real-number entries only, its Hermitian and its transpose are identical. The commands below show how the two are different when the matrix contains nonreal entries.

```
octave:34> transpose(B)
ans =

   1 − 1i   7 + 0i
   2 + 1i   0 + 3i

octave:35> B'
ans =

   1 + 1i   7 − 0i
   2 − 1i   0 − 3i
```

There are other convenient ways to generate special matrices. The `zeros()`, `ones()`, and `eye()` commands generate, respectively, a matrix full of zeros, a matrix full of ones, and an identity matrix. Only the `eye()` command is demonstrated below, so it would be good to investigate variants.

```
octave:41> eye(3,5)
ans =

Diagonal Matrix

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
```

Each of these commands offers the option of suppying just one number, as in `zeros(5)`, in which case one obtains a 5-by-5 (square) matrix full of zeros.

In this course we will take the word **vector** be mean an n-by-1 *column* vector. As vectors are simply special cases of matrices, the commands above apply equally well. If, for some reason, you want to build a vector with a patterned structure—containing all the whole numbers from 3 to 15, for instance, there is a special command using a colon to do just that:

```
octave:44> x = 3:15;
octave:45> x
x =
     3    4    5    6    7    8    9   10   11   12   13   14   15
octave:46> size(x)       # tells the dimensions of a matrix
ans =

     1   13
```

Notice, though, that the result of such a command has just one row with all the entries—i.e., it is a *column* vector. If it is a (column) vector one wants, then

```
octave:51> x = (3:15)';              # works
octave:53> x = 3:15;     x = x';     # works
octave:55> x = 3:15';                # does not work
```

You should play with this colon operator a bit, also trying these variants (output not provided here):

```
octave:58> 2:.5:4
octave:59> 3:-1
octave:60> 3:-2:-10
```

Another worthwhile method to know for building matrices is helpful when you want a diagonal matrix—i.e., one with entries only along the main diagonal. Of course, *eye(3)* will build the 3-by-3 identity matrix, having ones along the main diagonal and zeros elsewhere. These command places other values on the diagonal:

```
octave:62> diag(1:5)
octave:63> diag([3 1 -1])
octave:64> diag(ones(3,1), 1)
octave:65> diag(-ones(3,1), -1) + diag(ones(3,1), 1)    # like the matrix in Equation (16), p. 28
```

Finally, matrices can be built in blocks. In what the example here, note that we still follow the conventions that a semicolon marks the end of a row, and rows must contain the same number of entries. A new convention for block building is that blocks which go side by side, between successive semicolons, must also have the same number of rows.

```
octave:76> A = rand(2,3);       # a 2-by-3 matrix with real entries taken from a uniform distribution on (0,1)
octave:77> [[A' diag([5 4 3]); 2.^(1:4) 2] (5:-1:2)']
ans =

    0.49328    0.71849    5.00000     0.00000    0.00000    5.00000
    0.09233    0.87590    0.00000     4.00000    0.00000    4.00000
    0.21702    0.88283    0.00000     0.00000    3.00000    3.00000
    2.00000    4.00000    8.00000    16.00000    2.00000    2.00000
```

Matrix arithmetic

I will presume that you are current (or will quickly make yourself so) on how sums, differences and products involving matrices are defined. The usual operations of addition, subtraction, and multiplication, either multiplication between two matrices, or multiplication of a scalar times a matrix, are performed in Octave using the usual symbols. These operations are designed to obey the usual rules, so that

```
octave:78> ones(2,3) + diag([1 1 2])
error: operator +: nonconformant arguments (op1 is 2x3, op2 is 3x3)
```

does not make sense (Why?), but

```
octave:81> subD = diag([1 1 1],−1);
octave:82> subD + subD' − 2*eye(4)
ans =

  −2   1   0   0
   1  −2   1   0
   0   1  −2   1
   0   0   1  −2
```

does. The product `2*eye(4)` in the above is understood to be scalar multiplication. But below, the asterisk is for matrix multiplication, as should be clear from context.

```
octave:86> A = [4 3 −1; 1 0 2];   B = [3 1; −2 5; 1 −1];
octave:87> A*B
ans =

    5   20
    5   −1

octave:88> B*A
ans =

   13    9   −1
   −3   −6   12
    3    3   −3
```

What about $\mathbf{B}^2$, using the same matrix $\mathbf{B}$ above? If you have no idea what to expect, try it out.

Gil Strang makes the point, on p. 33 of our text, that it is useful to be equally comfortable in the world of "linear combinations of vectors" and in the world of "matrix-vector products". As an illustration, he gives us vectors

```
octave:89> u = [1; 3],     v = [−2; 2]
u =
   1
   3

v =

  -2
   2
```

The linear combination of **u** and **v**, giving weight 3 to **u** and weight 1 to **v** is

```
octave:90> 3*u + v
ans =

    1
   11
```

If we place these weights in a vector **x**, and build a coefficient matrix **A** whose columns, in sequence, are **u** and **v**, then the product **Ax** is identical to the previous linear combination:

```
octave:91> A = [u v]
A =

   1  -2
   3   2

octave:92> x = [3; 1]
x =

   3
   1

octave:93> A*x
ans =

    1
   11
```

Referencing entries of a matrix

In your previous exposure to linear algebra, you may have seen the notation $\mathbf{A} = (a_{ij})$ and learned that this meant a_{ij} might be used to refer to the entry of matrix **A** found in the i^{th} row, j^{th} column. Octave follows a similar convention. If you have an existing matrix **A**, you can use `A(i,j)` to reference the entry in row i, column j:

```
octave:101> A = reshape([1 1 2 3 5 8 13 21 34 55 89 144], 4, 3)
A =
     1     5    34
     1     8    55
     2    13    89
     3    21   144

octave:102> A(3,2)
ans =   13
```

The command `A(3,2)` produced the entry in row 3, column 2 of the matrix **A**. Perhaps the actions of the command `reshape()` are self explanatory, given the result. Should you want to more about it (or, indeed, other Octave commands), you can type

```
octave:99> help reshape
```

Octave referencing is not limited to producing only one element of a matrix. Here are some other examples to try out:

```
octave:103> A(3,:)          # produces a matrix containing the 3rd row of A
octave:104> A([2 3],2)      # produces a matrix containing the 2nd and 3rd entries of column 2
octave:105> A (:,:)         # produces all of A
octave:106> A(:,[1 3])      # produces a matrix containing the 1st and 3rd columns of A
octave:107> A(:)            # produces a vector containing all entries of A, stacked by column
```