Matlab in Math 461, part two

More Matlab operations

Matlab allows you to do various matrix operations easily. We have already seen:

- `rref(A)` is the reduced row echelon form of A.
- `x = A\b` solves $Ax = b$.
- `A*x` is the matrix product.

Some new operations are:

- `A'` is the transpose of A (actually the conjugate transpose if A is complex, see below).
- `inv(A)` is the inverse of A.
- `A*B` is the matrix product of two appropriately sized matrices.
- `real(A)` is the real part of a complex matrix A.
- `imag(A)` is the imaginary part of a complex matrix A, so `A` is the same as `real(A)+i*imag(A)`.
- `conj(A)` is the complex conjugate of a complex matrix A, see Lay, Appendix B page A4. So `conj(A)` is the same as `real(A)-i*imag(A)`.

Even more operations on matrices

If A is a matrix then `max(A)` gives a row vector whose entries are the maxima of the entries in each column of A. Likewise, `sum(A)` gives a row vector whose entries are the sum of the entries in each column of A.[1] You could sum the row entries of A by using the transpose, `(sum(A'))'` will produce a column vector whose entries are the sum of each row of A. Some other operations are `prod(A)` for the product of the column entries and `min(A)` for the minimum.

Transpose and complex matrices

If A is a complex matrix then A' is not the transpose, but the conjugate transpose. The conjugate transpose is obtained from the transpose by taking the complex conjugate of each entry, which means you change the sign of the imaginary part. For complex matrices, the conjugate transpose is generally more useful than the transpose but if you actually want the transpose of a complex matrix A then `conj(A')` or `A.'` will calculate it. Of course for a real matrix, the transpose and the conjugate transpose are the same. The conjugate transpose of a matrix A is written A^* and calculated in matlab as `A'`. The conjugate of a matrix A is written $\bar{A}$ and calculated in matlab as `conj(A)`. So $A^* = \bar{A}^T$.

Supressing unwanted output

Often you do not need or want to see the results of a Matlab operation, for example when you generate a large random matrix. You may suppress output by following a Matlab command with a semicolon. I encourage you to do this when appropriate.

Seeing whether two large matrices are equal

If you need to see whether two large matrices D and E are equal it is tedious and error prone to compare each entry. A better way is to look at the difference $D - E$ and see whether it is zero or very small in comparison to D and E. This sure beats comparing each of the 100 entries in two 10×10 matrices. Note that because of roundoff error two matrices might not be quite equal even though theoretically they should be equal and in fact would be equal if the computer could do exact arithmetic. For example, `D-inv(inv(D))` will generally not evaluate to the 0 matrix even though it should.

However, if D and E are large, printing out $D - E$ can take up a great deal of paper. So here is a good way to check whether two large matrices D and E are very nearly equal. The command

```
>> max(max(abs(D-E)))
```

[1] But as a convenience, if A is a row vector then `sum(A)` and `max(A)` find the sum and maximum of its entries. This is why `max(max(abs(D-E)))` works and we don't need to write `max(max(abs(D-E))')`.

will print out the entry of $D - E$ with largest absolute value. If this is quite small compared with the entries of D then D and E are equal for all practical purposes. (The reason two max are needed above is that the first max gives you a vector with the maximum entry in each column, and the second max finds the maximum of those numbers.) There are other ways to check whether a matrix is small. The shortest is `norm(D-E)` but we will not know what this number is until section 7.4. Another is `norm(D-E,1)` which calculates `max(sum(abs(D-E)))`. The results of all these commands are single numbers so it is easy to tell at a glance that D and E are very close or not.

Matlab problems due Feb. 26

In problems 2-4 you need to compare two matrices to see if they are equal. Do **not** do this by printing them out and inspecting them. Do not even print out their difference. Instead use the method above with `max(max(abs(D-E)))` or `norm(D-E,1)`. Remember that because of roundoff errors it is possible that the difference of two matrices which should be equal is nonzero but very small, in which case you can say that they are essentially equal. Be sure to start your session with the command `rand('state',sum(100*clock));` so that your answers are random and differ from other groups.

Problem 1: Generate a random 3×4 complex matrix C with small integer entries, for example with the commands `C = randint(3,4,10)+i*randint(3,4,10)` or `C = round(10*rand(3,4)+10i*rand(3,4))`. Calculate `C'`, `real(C)`, `imag(C)`, and `conj(C)` and look at the output and understand your answers. (You need not print out this problem, just tell me you did it.)

Problem 2: Generate random complex 7×7 matrices A and B. For each part below, calculate the matrices $A1, A2, A3$, and $A4$ and determine which are equal.

a) $A1 = AB, A2 = BA, A3 = (B^*A^*)^*, A4 = (A^*B^*)^*$.
b) $A1 = A^*B^*, A2 = B^*A^*, A3 = (BA)^*, A4 = (AB)^*$.
c) $A1 = A^{-1}B^{-1}, A2 = B^{-1}A^{-1}, A3 = (AB)^{-1}, A4 = (BA)^{-1}$.
d) $A1 = \bar{A}\bar{B}, A2 = \bar{B}\bar{A}, A3 = \overline{AB}, A4 = \overline{BA}$.

Problem 3: With the A above, check whether or not A equals $(A^{-1})^{-1}$. Now check whether A equals $(A^*)^*$. Now check whether $(A^*)^{-1}$ equals $(A^{-1})^*$. Finally, check whether $\bar{A}^{-1}$ essentially equals $\overline{A^{-1}}$.

Problem 4: With the A above, reduce $[A \quad I_7]$ to reduced echelon form and extract from this A^{-1}. Compare the result with `inv(A)`. You can generate the 7×7 identity matrix by `eye(7)` so in matlab you could write $[A \quad I_7]$ as `[A eye(7)]`. Note you can extract the eighth through 14th columns of a matrix D with the command `D(:,8:14);`.