More Matlab

Some Matlab operations

Matlab allows you to do various matrix operations easily. For example:

- `rref(A)` is the reduced row echelon form of A.
- `A*x` is the matrix product.
- `A'` is the transpose of A (actually the conjugate transpose if A is complex).
- `inv(A)` is the inverse of A.
- If $k = 1, 2, 3, \ldots$ then `A^k` is A^k.
- `x = A\b` solves $Ax = b$
- `A*B` is the matrix product AB of two appropriately sized matrices.
- If c is a scalar then `c*B` is the scalar multiple cB.

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.

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. 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)))`

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.

[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))')`.

Matlab problems due Feb. 27

In some problems below, you need to compare two matrices to see if they are equal. Use the method above with `max(max(abs(D-E)))` or `norm(D-E,1)`. You don't need to exhibit the matrices (although you can if you're curious). 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.

Your completed project should be in the format specified for the earlier homework. The commands for your homework should be developed in an m-file as before.

Problem 1: Generate and exhibit random 5×5 matrices A and B. Then test equality for each of the five equations below, as follows.

(i) If the left side is a matrix D and the right side is E, have MATLAB compute max(max(abs(D-E))). For example, you can check $(A^*)^{-1} = (A^{-1})^*$ by calculating max(max(abs(inv(A')-(inv(A))'))) and seeing whether or not it is 0 or very small in comparison with the entries of A.

(ii) For parts b and e, comment as to whether the equation is true for all 5×5 matrices A and B. For parts a, c and d, comment as whether the equation is true for all 5×5 invertible matrices A and B.

Here are the five equations.

(a) $(A^*)^{-1} = (A^{-1})^*$
(b) $(A+B)^2 = A^2 + 2AB + B^2$.
(c) $(A+B)^{-1} = A^{-1} + B^{-1}$.
(d) $(A^{-1})^3 = (A^3)^{-1}$.
(e) $(A+B)^* = A^* + B^*$

Problem 2: With your A above, reduce $[A \quad I_5]$ to reduced echelon form and extract from this A^{-1}. Compare the result with `inv(A)`. You can generate the 5×5 identity matrix by `eye(5)` so in matlab you could write $[A \quad I_5]$ as `[A eye(5)]`. Note you can extract the sixth through 10th columns of a matrix D with the command `D(:,6:10)`.

Problem 3: Let H be the matrix

[1,1/2,1/3,1/4,1/5; 1/2,1/3,1/4,1/5,1/6; 1/3,1/4,1/5,1/6,1/7; 1/4,1/5,1/6,1/7,1/8; 1/5,1/6,1/7,1/8,1/9].

Compute inv(H) and max(max(abs(inv(H)))). What is notable here?

Problem 4: Do Problem 41a, page 133 of Lay.