MA519: Numerical Linear Algebra

Lecture 12: Matlab, Orthogonality, and the SVD

The goal of this lab is get a feel for how Matlab manipulates matrices and vectors, and to experiment with low-rank approximation.

Matlab is the standard tool for numerical computing in industry and research. It specialises in matrix computations (Matrix Laboratory), but includes functions for graphics, numerical integration and differentiation, solving differential equations, image and signal analysis, and much more.

Matlab is an *interpretive* environment – you type a command and it will execute it immediately. Nonetheless, one can group a set of commands together into a script or function file.

The details given below cover just enough of the fundamentals to get started with today's exercises. I suggest you get a book about Matlab from the library, or online. See the module website for links to the books by Driscoll, Higham×2, and Moler.

The Basics

1. In Matlab, everything is a *matrix*. A scalar variable is just a 1×1 matrix. To check this set, say, `t = 10`, and use the `size()` command to find the numbers of rows and columns of `t`.

2. To declare a row-vector array, try:
   ```
   >> x=[-4, -3, -2, -1, 0, 1, 2, 3, 4]
   ```
   Or, more simply,
   ```
   >>  x=-4:4
   ```
   To access, say, the 3rd entry
   ```
   >>  x(3)
   ```

3. We usually like to think of vectors as *column* vectors. To define one, try
   ```
   >>  x=[1;2;3]
   ```
   Or you can take the (hermitian) transpose of a row vector: `>>  x = [1,2,3]';`

   Verify that is the hermitian transpose by defining a complex-valued vector, and looking at its transpose:
   ```
   >>  i = sqrt(-1); x=[i, 1+i, 2]
   >>  x'
   ```

4. If you put a semicolon at the end of a line of Matlab, the line is executed, but the output is not shown. (This is useful if you are dealing with large vectors). If no semicolon is used, the output is shown in the command window.

5. We'll often want to run a collection of commands repeatedly. So, rather than type them individually, create a file with the following code
   ```
   x=-4:4
   for i=1:9
     y(i) = cos( x(i) );
   end
   plot(x,y);
   ```
   Save this as, say `class1.m`. To execute it, just type `>> class1` in the Matlab command window.

   Your file is an example of a Matlab *script file*.

6. The plot generated is not particularly good one. The points plotted are a unit apart. To get a better picture, try "easy plot" `>> ezplot(@cos,[-4,4])`

7. A row vector may be declared as follows:
   ```
   >>  x = a:h:b;
   ```
   This sets $x_1 = a$, $x_2 = a + h$, $x_3 = x + 2h$, ..., $x_n = b$. If `h` is omitted, it is assumed to be 1.

8. The script file from Part (5) is a little redundant. In Matlab, most functions can take a vector or matrix as an argument. So, in fact, we can just use
   ```
   >>  y = cos( x )
   ```
   which sets `y` to be a vector such that $y_i = \cos(x_i)$.

9. The `*` operator performs matrix-matrix multiplication. So, to compute the inner product of the (column) vector `x`, try `>> IP = x'*x;`

   For element-by-element multiplication use `.*` For example, `y = x.*x` sets $y_i = (x_i)^2$. So does `y = x.^2`.

10. Declare a Matrix as
    ```
    >> A = [3 -1 ; -2 3]
    ```

11. The entry of in row `i` and column `j` of a matrix is given by `A(i,j)`
    The i^{th} row of matrix A can be addressed as `A(i,:)`, and the `j`th column as `A(:, j)`.

12. To compute the inverse of a matrix (where possible)
    ```
    >> inv(A)
    ans =
       4.2857e-01   1.4286e-01
       2.8571e-01   4.2857e-01
    ```
    Other common linear algebra functions are also available, e.g., `det`, `trace`, `rank` .

13. Of obvious interest to us is `svd`. It can be used in two ways:
`>> sigma = svd(A)` returns the singular values of A and stores them in the vector `sigma`.
`>> [U,Sigma,V] = svd(A)` computes that factorisation (in Matlab notation) `A=U*Sigma*V'`

14. Miscellaneous other useful functions include

    - `>> A = rand(m,n)` – creates a matrix with (uniformly distributed) random entries. Use `randn` to get normally distributed entries. The functions `zeros(m,n)` and `ones(m,n)` return decidedly nonrandom matrices.
    - `>> I = eye(n)` – identity matrix
    - `>> Z = complex(A,B)` – where A and B are real matrices of the same size, sets $Z = A + iB$.
    - `>> E = eig(A)` – (tries) to return the eigenvalues of A.
    - `>> norm(x)` computes the 2-norm of the vector (or Matrix) x. `norm(x,p)` computes the p-norm, and `norm(x, inf)` returns $\|x\|_\infty$.
      `>> norm(A, 'fro')` computes the Frobenius norm of the matrix A.

.....

And now for some exercises.

(a) Suppose that $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$ is an orthonormal set of vectors. Then, for any vector $\boldsymbol{\nu}$,

$$\mathbf{r} = \boldsymbol{\nu} - (\mathbf{q}_1^\star \boldsymbol{\nu})\mathbf{q}_1 - (\mathbf{q}_2^\star \boldsymbol{\nu})\mathbf{q}_2 - \cdots - (\mathbf{q}_n^\star \boldsymbol{\nu})\mathbf{q}_n,$$

is orthogonal to $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$.

Form any pair of linearly independent (column) vectors $\mathbf{q}_1, \boldsymbol{\nu} \in \mathbb{R}^4$ in Matlab. Rescale $\mathbf{q}_1$ so that is has norm 1: `>> q1 = q1/norm(q1)` .

Set $\mathbf{r} = \boldsymbol{\nu} - \mathbf{q}_1(\mathbf{q}_1^\star \boldsymbol{\nu})$ and verify that it is orthogonal to $\mathbf{q}_1$. Let $\mathbf{q}_2 = \mathbf{r}/\|\mathbf{r}\|$.

Now choose another $\boldsymbol{\nu}$ and construct $\mathbf{r}$ that is orthogonal to both $\mathbf{q}_1$ and $\mathbf{q}_2$. Again rescale $\mathbf{r}$ to get the unit vector $\mathbf{q}_3$ so that $\{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3\}$ forms an orthonormal set.

Repeat the procedure to get $\{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4\}$, an orthonormal basis for $\mathbb{R}^4$.

What happens it you try this algorithm again to get $\mathbf{q}_5$?

(b) Put the vectors you got in (a) to form that matrix

$$Q = (q_1|q_2|q_3|q_4).$$

In Matlab, this can be done as:
`>> Q=[q1,q2,q3,q4];`
Verify that this is a unitary matrix.

(c) Work out how to create a random symmetric matrix with real entries. Verify that its eigenvalues are real. Now try with a random hermitian matrix with complex entries. Again verify that its eigenvalues are real.

(d) We saw in class that, if λ is an eigenvalue of a unitary matrix, then $|\lambda| = 1$. Verify this for the matrix you formed in (a).

Make a larger unitary matrix, for example by computing the U and V in the SVD of a large(ish) matrix. Again check its eigenvalues. Plot them on the argand plane:
`>> E = eig(Q); plot(real(E), imag(E), 'o');`

(e) In our final example, we'll experiment with some low-rank approximations. The following code loads a standard test image, of a clown, and then displays it.
We then compute the SVD, and construct a new matrix and plot the singular values, using a semilogarithmic plot. Then we make a new matrix from the first $p = 20$ columns of U, entries of S and rows of V'. How does this compare with the original image? Experiment with the choice of p.

```
load clown
figure(1);
imagesc(X); colormap('gray');

[U,S,V]=svd(X);
figure(2);
semilogy(diag(S), '--o');

p=1:20;
New=U(:,p)*S(p,p)*(V(:,p))';
figure(5);
imagesc(New); colormap('gray');
```

Next download the `board.mat` file from the module website. This is of a much simpler image: a chess board. Now what is the smallest value of p that you can choose to get a reasonable low-rank approximation? Why?