# Lecture 8

# Matrices and Matrix Operations in Matlab

You should review the vector operations in Lecture 1.

### Matrix operations

Recall how to multiply a matrix $A$ times a vector $\mathbf{v}$:

$$A\mathbf{v} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \cdot (-1) + 2 \cdot 2 \\ 3 \cdot (-1) + 4 \cdot 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}.$$

This is a special case of matrix multiplication. To multiply two matrices, $A$ and $B$ you proceed as follows:

$$AB = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} -1 & -2 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} -1+4 & -2+2 \\ -3+8 & -6+4 \end{pmatrix} = \begin{pmatrix} 3 & 0 \\ 5 & -2 \end{pmatrix}.$$

Here both $A$ and $B$ are $2 \times 2$ matrices. Matrices can be multiplied together in this way provided that the number of columns of $A$ match the number of rows of $B$. We always list the size of a matrix by rows, then columns, so a $3 \times 5$ matrix would have 3 rows and 5 columns. So, if $A$ is $m \times n$ and $B$ is $p \times q$, then we can multiply $AB$ if and only if $n = p$. A column vector can be thought of as a $p \times 1$ matrix and a row vector as a $1 \times q$ matrix. Unless otherwise specified we will assume a vector $\mathbf{v}$ to be a column vector and so $A\mathbf{v}$ makes sense as long as the number of columns of $A$ matches the number of entries in $\mathbf{v}$.

Printing matrices on the screen takes up a lot of space, so you may want to use

```
≫ format compact
```

Enter a matrix into Matlab either as

```
≫ A = [ 1   3 -2 5 ;  -1   -1 5 4 ; 0 1 -9   0]
```

or

```
≫ A = [1,3,-2,5; -1,-1,5,4; 0,1,-9,0]
```

Also enter a vector $\mathbf{u}$:

```
≫ u = [ 1   2   3   4]'
```

To multiply a matrix times a vector $A\mathbf{u}$ use `*`:

```
≫ A*u
```

Since $A$ is 3 by 4 and $\mathbf{u}$ is 4 by 1 this multiplication is valid and the result is a 3 by 1 vector.

Now enter another matrix $B$ using

```
≫ B = [3 2 1; 7 6 5; 4 3 2]
```

You can multiply $B$ times $A$ with

```
≫ B*A
```

but $A$ times $B$ is not defined and

```
≫ A*B
```

will result in an error message.

You can multiply a matrix by a scalar:

```
≫ 2*A
```

Adding matrices $A + A$ will give the same result:

```
≫ A + A
```

You can even add a number to a matrix:

```
≫ A + 3    % add 3 to every entry of A
```

### Component-wise operations

Just as for vectors, adding a '`.`' before '`*`', '`/`', or '`^`' produces entry-wise multiplication, division and exponentiation. If you enter

```
≫ B*B
```

the result will be $BB$, i.e. matrix multiplication of $B$ times itself. But, if you enter

```
≫ B.*B
```

the entries of the resulting matrix will contain the squares of the same entries of $B$. Similarly if you want $B$ multiplied by itself 3 times then enter

```
≫ B^3
```

but, if you want to cube all the entries of B then enter

```
≫ B.^3
```

Note that `B*B` and `B^3` only make sense if $B$ is square, but `B.*B` and `B.^3` make sense for any size matrix.

## The identity matrix and the inverse of a matrix

The $n \times n$ *identity matrix* is a square matrix with ones on the diagonal and zeros everywhere else. It is called the identity because it plays the same role that 1 plays in multiplication, i.e.

$$AI = A, \qquad IA = A, \qquad I\mathbf{v} = \mathbf{v}$$

for any matrix $A$ or vector $\mathbf{v}$ where the sizes match. An identity matrix in MATLAB is produced by the command

```
≫ I = eye(3)
```

A square matrix $A$ can have an *inverse* which is denoted by $A^{-1}$. The definition of the inverse is that

$$AA^{-1} = I \qquad \text{and} \qquad A^{-1}A = I.$$

In theory an inverse is very important, because if you have an equation

$$A\mathbf{x} = \mathbf{b}$$

where $A$ and $\mathbf{b}$ are known and $\mathbf{x}$ is unknown (as we will see, such problems are very common and important) then the theoretical solution is

$$\mathbf{x} = A^{-1}\mathbf{b}.$$

We will see later that this is not a practical way to solve an equation, and $A^{-1}$ is only important for the purpose of derivations. In MATLAB we can calculate a matrix's inverse very conveniently:

```
≫ C = randn(5,5)
≫ inv(C)
```

However, not all square matrices have inverses:

```
≫ D = ones(5,5)
≫ inv(D)
```

## The "Norm" of a matrix

For a vector, the "norm" means the same thing as the length (geometrically, not the number of entries). Another way to think of it is how far the vector is from being the zero vector. We want to measure a matrix in much the same way and the *norm* is such a quantity. The usual definition of the norm of a matrix is

**Definition 1** *Suppose $A$ is a $m \times n$ matrix. The norm of $A$ is*

$$\|A\| \equiv \max_{\|\mathbf{v}\|=1} \|A\mathbf{v}\|.$$

The maximum in the definition is taken over all vectors with length 1 (unit vectors), so the definition means the largest factor that the matrix stretches (or shrinks) a unit vector. This definition seems cumbersome at first, but it turns out to be the best one. For example, with this definition we have the following inequality for any vector $\mathbf{v}$:

$$\|A\mathbf{v}\| \leq \|A\|\|\mathbf{v}\|.$$

In MATLAB the norm of a matrix is obtained by the command

```
≫ norm(A)
```

For instance the norm of an identity matrix is 1:

```
≫ norm(eye(100))
```

and the norm of a zero matrix is 0:

```
≫ norm(zeros(50,50))
```

For a matrix the norm defined above and calculated by MATLAB is not the square root of the sum of the square of its entries. That quantity is called the *Froebenius norm*, which is also sometimes useful, but we will not need it.

### Some other useful commands

`C = rand(5,5)` ..... random matrix with uniform distribution in $[0,1]$.
`size(C)` ..... gives the dimensions $(m \times n)$ of $C$.
`det(C)` ..... the determinant of the matrix.
`max(C)` ..... the maximum of each column.
`min(C)` ..... the minimum in each column.
`sum(C)` ..... sums each column.
`mean(C)` ..... the average of each column.
`diag(C)` ..... just the diagonal elements.
`C'` ..... tranpose the matrix.

In addition to `ones`, `eye`, `zeros`, `rand` and `randn`, MATLAB has several other commands that automatically produce special matrices:
`hilb(6)`
`pascal(5)`

## Exercises

8.1 Enter the matrix $A$ by

```
≫ A = [[3 2 1; 6 5 4; 9 8 7]
```

and also the matrix

$$B = \begin{bmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}.$$

Find (a) `A*A`, (b) `A^2`, (c) `A.^2`, (d) `A.*B`, (e) `A*B`. Turn in the output.

8.2 By hand, calculate $A\mathbf{v}$, $AB$, and $BA$ for:

$$A = \begin{pmatrix} 2 & 4 & -1 \\ -2 & 1 & 9 \\ -1 & -1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ -1 & -2 & 0 \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix}.$$

Check the results using MATLAB. Think about how fast computers are. Turn in your hand work.

8.3 Write a well-commented MATLAB **function** program `myinvcheck` that

- makes a $n \times n$ random matrix (normally distributed, `A = randn(n,n)`),
- calculates its inverse (`B = inv(A)`),
- multiplies the two back together,
- calculates the residual (difference between $AB$ and `eye(n)`), and
- returns the scalar residual (norm of the difference).

Turn in your program.

8.4 Write a well-commented MATLAB **script** program `myinvcheckplot` that calls `myinvcheck` for $n = 10, 20, 40, \ldots, 2^i 10$ for some moderate $i$, records the results of each trial, and plots the scalar residual versus `n` using a log plot. (See `help loglog`.)

What happens to the scalar residual as $n$ gets big? Turn in the program, the plot, and a very brief report on the results of your experiments. (Do not print any large random matrices.)