1 Linear equations and matrix manipulation

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

$$A\mathbf{v} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1.(-1) + 2.2 \\ 3.(-1) + 4.2 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

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

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

Here both A and B are 2x2 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 $3x5$ matrix would have 3 rows and 5 columns. So, if A is mxn and B is pxq, then we can multiply AB if and only if n = p. A column vector can be thought of as a $px1$ matrix and a row vector as a $1xq$ 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 following commands.

```
>>A = [ 1 3 -2 5; -1 -1 5 4; 0 1 -9 0]
>>u = [1 2 3 4]'
>>A*u
>>B = [3 2 1; 7 6 5; 4 3 2]
>>B * A
>>A * B
```

Does A * B give you error message ? Why?

Similarly, try following commands

```
>>A + A
>>A + B
>>A + 3
```

We have also observed in past labs that adding . before *, ÷ or ˆ produces entry-wise multiplication, division and exponentiation. Following commands will show the difference.

>>$B * B$
>>$B. * B$
>>$B\hat{}3$
>>$B.\hat{}3$

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 of matrix.

1.1 Identity matrix and inverse

The nxn 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 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
$AA^{-1} = I, \qquad A^{-1}A = I$

In theory, an inverse is very important because if you have an equation
$Ax = b$
where A and b are known and x is unknown (such problems are very common) then the theoretical solution is
$x = A^{-1}b$

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 matrixs inverse very conveniently:
>>C = randn(5,5)
>>inv(C)

However, not all square matrices have inverse:

>>D = ones(5, 5)
>>inv(D)

1.2 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 mxn matrix. The norm of A is

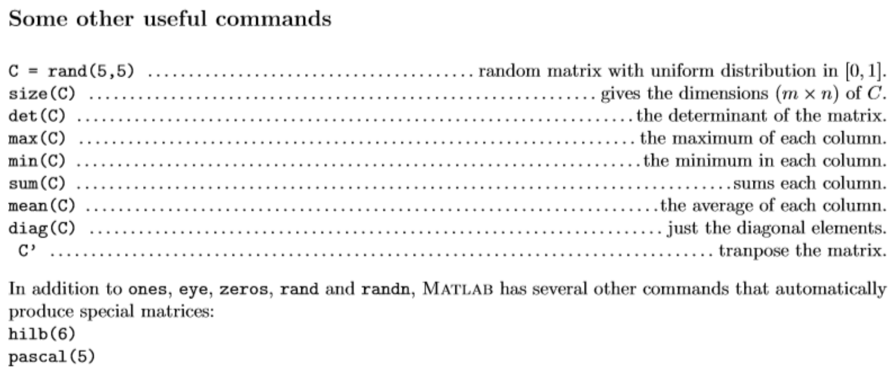
The maximum in the definition is taken over all vectors with length 1 (unit vectors), so the definition means the largest factor that the matrix stretches (or shrinks) a unit vector. This definition seems cumbersome at first, but it turns out to be the best one. For example, with this definition we have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

In MATLAB, the norm of a matrix is obtained by the commands

>>norm(A)

For instance, the norm of an indentiry matrix is :
>>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.

Figure 1: Useful commands



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)
```

2 Linear systems

Linear systems of equations naturally occur in many places in engineering, such as structural analysis, dynamics and electric circuits. Computers have made it possible to quickly and accurately solve larger and larger systems of equations. Not only has this allowed engineers to handle more and more complex problems where linear systems naturally occur, but has also prompted engineers to use linear systems to solve problems where they do not naturally occur such as thermodynamics, internal stress-strain analysis, fluids and chemical processes. It has become standard practice in many areas to analyze a problem by transforming it into a linear systems of equations and then solving those equation by computer. In this way, computers have made linear systems of equations the most frequently used tool in modern engineering.

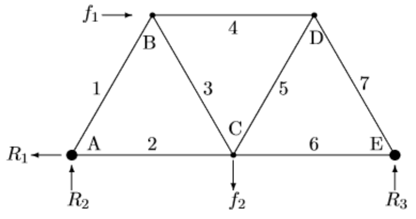
In Figure 2 we show a truss with equilateral triangles. In Statics you may use the method of joints to write equations for each node of the truss1. This set of equations is an example of a linear system. Making the approximation

$\sqrt{3}/2 \approx .8660$, the equations for this truss are

$.5T_1 + T_2 = R_1 = f_1$
$.866T_1 = -R_2 = -.433f_1 - .5f_2$
$-.5T_1 + .5T_3 + T_4 = -f_1$
$.866T_1 + .866T_3 = 0$
$-T_2 - .5T_3 + .5T_5 + T_6 = 0$
$.866T_3 + .866T_5 = f_2$
$-T_4 - .5T_5 + .5T_7 = 0,$

where T_i represents the tension in the i-th member of the truss.

Figure 2: An equilateral truss. Joints or nodes are labeled alphabetically, A, B, ...and Members (edges) are labeled numerically: 1, 2, ... The forces f1 and f2 are applied loads and R1, R2 and R3 are reaction forces applied by the supports.



You could solve this system by hand with a little time and patience; systematically eliminating variables and substituting. Obviously, it would be a lot better to put the equations on a computer and let the computer solve it. In the next few lectures we will learn how to use a computer effectively to solve linear systems. The first key to dealing with linear systems is to realize that they are equivalent to matrices, which contain numbers, not variables.
As we discuss various aspects of matrices, we wish to keep in mind that the matrices that come up in engineering systems are really large. It is not unusual in real engineering to use matrices whose dimensions are in the thousands! It is frequently the case that a method that is fine for a 2x2 or 3x3 matrix is totally inappropriate for a 20002000 matrix. We thus want to emphasize methods that work for large matrices.

2.1 Linear systems and matrices

The system of linear equation
$x_1 - 2x_2 + 3x_3 = 4$
$2x_1 - 5x_2 + 12x_3 = 15$
$2x_2 - 10x_3 = -10$

is equivalent to the matrix equation

$$\begin{bmatrix} 1 & -2 & 3 \\ 2 & -5 & 12 \\ 0 & 2 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 15 \\ -10 \end{bmatrix}$$

which is equivalent to the augmented matrix

$$\left[\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 2 & -5 & 12 & 15 \\ 0 & 2 & -10 & -10 \end{array}\right] \tag{1}$$

The advantage of the augmented matrix, is that it contains only numbers, not variables. The reason this is better is because computers are much better in dealing with numbers than variables. To solve this system, the main steps are called Gaussian elimination and back substitution.
The augmented matrix for the equilateral truss equations is given by

$$\left[\begin{array}{ccccccc|c} .5 & 1 & 0 & 0 & 0 & 0 & 0 & f_1 \\ .866 & 0 & 0 & 0 & 0 & 0 & 0 & -4.33f_1 - .5f_2 \\ -.5 & 0 & .5 & 1 & 0 & 0 & 0 & -f_1 \\ -.866 & 0 & .866 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -.5 & 0 & .5 & 1 & 0 & 0 \\ 0 & 0 & .866 & 0 & .866 & 0 & 0 & f_2 \\ 0 & 0 & 0 & -1 & -.5 & 0 & .5 & 0 \end{array}\right] \tag{2}$$

Notice that a lot of the entries are 0. Matrices like this, called sparse, are common in applications and there are methods specifically designed to efficiently handle sparse matrices.

2.2 Triangular matrices and back substitution

Consider a linear system whose augmented matrix happens to be

$$\left[\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 0 & -1 & 6 & 7 \\ 0 & 0 & 2 & 4 \end{array}\right] \tag{3}$$

Recall that each row represents an equation and each column a variable. The last row represents the equation $2x_3 = 4$. The equation is easily solved, i.e. $x_3 = 2$. The second row represents the equation $x_2 + 6x_3 = 7$, but since we know $x_3 = 2$, this simplifies to: $x_2 + 12 = 7$. This is easily solved, giving $x_2 = 5$. Finally, since we know x_2 and x_3, the first row simplifies to: $x_1 10 + 6 = 4$. Thus, we have $x_1 = 8$ and so we know the whole solution vector: $x = 8, 5, 2$. The process we just did is called back substitution, which is both efficient and easily programmed. The property that made it possible to solve the system so easily is that A in this case is upper triangular. In the next section, we show an efficient way to transform an augmented matrix into an upper triangular matrix.

2.3 Gaussian Elimination

Consider the matrix

$$A = \left[\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 2 & -5 & 12 & 15 \\ 0 & 2 & -10 & -10 \end{array}\right] \tag{4}$$

The first step of Gaussian elimination is to get rid of the 2 in the (2,1) position by subtracting 2 times the first row from the second row, i.e. (new 2nd = old 2nd - (2) 1st). We can do this because it is essentially the same as adding equations, which is a valid algebraic operation. This leads to

$$A = \left[\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 0 & -1 & 6 & 7 \\ 0 & 2 & -10 & -10 \end{array}\right] \tag{5}$$

There is already a zero in the lower left corner, so we dont need to eliminate anything there. To eliminate the third row, second column, we need to subtract 2 times the second row from the third row, (new 3rd = old 3rd - (-2) 2nd), to obtain

$$A = \left[\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 0 & -1 & 6 & 7 \\ 0 & 0 & 2 & 4 \end{array}\right] \tag{6}$$

We can now solve it by back substitution (as did before)

2.4 MATLAB's matrix solve command

In MATLAB, the standard way to solve a system $A\mathbf{x} = \mathbf{b}$ is by the command
$>>x = A\backslash b$

This command carries out Gaussian elimination and back substitution. We can do the above computations as follows:

```
>>A = [ 1 -2 3; 2 -5 12; 0 2 -10]
>>b = [4 15 -10]'
>>x = A\b
```

Next, use the MATLAB commands described above to solve $A\mathbf{x} = \mathbf{b}$ when the augmented matrix for the system is

$$A = \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{array}\right] \tag{7}$$

by entering
$>>x1 = A\backslash b$
Check the result by entering
>>A*x1 - b

You will see that the resulting answer satisfies the equation exactly. Next, try solving using the inverse of A:
>>x2 = inv(A) * b
This answer can be seen to be inaccurate by checking
>>A*x2 - b
Thus, we see one of the reasons why the inverse is never used for actual computation, only for theory.

3 Practice Questions - To be completed in lab

1. Write a well-commented Matlab function program **myinvcheck** that

(a) makes a nxn random matrix (normally distributed, A = randn(n,n)),
(b) calculates its inverse (B = inv(A)),
(c) multiplies the two back together,
(d) calculates the residual (difference from the desired nxn identity matrix eye(n)), and
(e) returns the norm of the residual.

2. Write a well-commented Matlab script program that calls **myinvcheck** for $n = 10, 20, 40, ..., 2^i 10$ for some moderate i, records the results of each trial, and plots the error versus n using a log plot. (See help loglog.) What happens to error as n gets big? Turn in a printout of the programs, the plot, and a very brief report on the results of your experiments.

3. Set $f_1 = 1000N$ and $f_2 = 5000N$ in the equations for the equilateral truss. Input the coefficient matrix A and the right hand side vector b in augmented matrix of truss into MATLAB. Solve the system using the command \ to find the tension in each member of the truss. Save the matrix A as Aequiltruss and keep it for later use. (Enter save Aequiltruss A.) Print out and turn in A, b and the solution x.

4. Write each system of equations as an augmented matrix, then find the solutions using Gaussian elimination and back substitution (the algorithm in this chapter). Check your solutions using MATLAB.

$$x_1 + x_2 = 2, 4x_1 + 5x_2 = 10 \tag{8}$$

$$x_1 + 2x_2 + 3x_3 = -1, 4x_1 + 7x_2 + 14x_3 = 3, x_1 + 4x_2 + 4x_3 = 1 \tag{9}$$