Linear Algebra and Solving Linear Systems using MATLAB

By Gary Hill

An Introduction to Linear Algebra

In Algebra I you learned to solve the equation $2xX = 4$ for variable X by dividing both sides of the equation by 2 and recognizing that $2/2 = 1$

$X = 4/2$

You have also learned how to solve more complex equations like this second order polynomial $X^2 + X - 1 = 0$ using the quadratic equation.

$$X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The first example is a *linear* system while the second is *nonlinear* system. In linear algebra we only consider the first class of equations (i.e., they are all linear). That is the good news. The bad news is that we allow each term (*even the numbers 2 and 4*) to be matrices of any dimension $A(m, n)$, which may be real or complex (having a **real** and an **imag**inary part). It is just for the solution of these kinds of problems that Matlab was created.

Linear Systems (Section 9.2)

Let's see if we can make things a little simpler. In engineering if you want to describe a system with n independent variables you will need m equations, where $n = m$. This is known as a square system (Section 9.2.1) You also learned this in your Algebra I class where your instructor would always give you two equations to solve for two unknown variables (you see they were nice to you). If they had given you m equations to solve for n variable where $m < n$ you would have been in trouble because the system was *underdetermined* (Section 9.2.3). Conversely, if they had given you m equations to solve for n variable where $m > n$ you would also not know what to do with all that extra information, the system was *overdetermined* (Section 9.2.2). So let's limit our discussion to square systems.

Converting a Set of Linear Equations to Matrix Form

Source: Getting Started with Matlab by Rudra Pratap

Up to this point I have assumed a matrix form can be obtained from a set of linear equations. In this section I will take you through the process step-by-step to solve the following set of equations.

$5x = 3y - 2z + 10$
$8y + 4z = 3x + 20$
$2x + 4y - 9z = 9$

Step 1: Rearrange equations: Write each equation with all unknown quantaties on the left-hand and all known quantities on the right-hand side. Thus, for our example, we rearrange them so all terms involving x, y, and z are on the left side of the equals sign

$+5x - 3y + 2z = 10$
$-3x + 8y + 4z = 20$
$+2x + 4y - 9z = 9$

Step 2: Write the equations in matrix form: To write the equations in the matrix form $[A]\{X\} = \{B\}$ where {X} is the vertical (column) vector of unknowns, you have to arrange the unknowns in vector x, the coefficients of the unknowns in matrix A, and the constants on the right-hand side of the equations in vertical vector B. In this example, the unknown vector is

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix},$$

The coefficient matrix is

$$A = \begin{bmatrix} 5 & -3 & 2 \\ -3 & 8 & 4 \\ 2 & 4 & -9 \end{bmatrix},$$

And the known constant vector is

$$B = \begin{bmatrix} 10 \\ 20 \\ 9 \end{bmatrix}.$$

Note that the columns of A are simply the coefficients of each unknown from all three equations. Now you are ready to solve the system in Matlab.

Solving Linear System

Let's go back to the Introduction and our Algebra I class equation $2xX = 4$. As mentioned before to solve this problem you divided both sides by 2. You may not remember, but the key here was that the inverse of 2 is $\frac{1}{2}$, so $2\ x\ \frac{1}{2} = 1$ and $1xX = X$, the identity rule. In linear algebra we also want to solve for X, but now X is a matrix[1].

Using matrix notation, a system of simultaneous linear equations is written as

$$AX = B$$

Where A is a square matrix which describes our system (in our simple example the number 2), X is our unknown variables, and B is the output of our system in state A (for our simple example with $X = 2$, then for an input A of 2 you get an output of 4). To solve for X we use a methodology completely analogous to our simple solution. Specifically, we need to find the inverse of A^{-1} (`inv(A)`) such that $A^{-1}xA = I$, where I is known as the identify matrix (`eye(n)`). So instead of $\frac{1}{2}$ we have A^{-1} and in the place of 1 we have the identity matrix I.

Putting it all together…

Starting with our system described by the equation:

[1] The terms array and matrix are often used interchangeably in engineering. However, technically, an array is an orderly grouping of information, whereas a matrix is a two-dimensional numeric array used in linear algebra. [Matlab for Engineers, by Holly Moore Chapter 10 Matrix Algebra]

$$AX = B$$

Multiplying both sides of the equation by the inverse of A we get

$$A^{-1}AX = A^{-1}B$$

We know that $A^{-1}xA = I$ and that under the laws of matrix multiplication that any square matrix multiplied by the identity matrix is simply the matrix we have

$$X = A^{-1}B$$

As in all matrix multiplication, the order of multiplication is important. For example is A is a 3x3 matrix, its inverse must also be a 3x3 matrix. Since the inside dimensions must match we also know that for this example B must be a 3x1 vertical vector.

In Matlab the inverse is found with the `inv()` function. Here are a few other ways of expressing the above equation in Matlab.

```
X = inv(A)*B                % Section 2.2
X = A^-1*B
X = A\B                     % Section 9.2.1
X = linsolve(A,B,OPTS)      % Section 9.2.1 also see help linsolve
```

The expression `X = A\B`, known as matrix left division, uses a technique known as Gaussian elimination. Using this approach allows Matlab to also solve systems which are overdetermined and underdetermined. These are also known as overdefined and underdefined respectively. For the overdefined case the equations must be consistent (define the same system). For the underdefined case Matlab returns one of an infinite number of possible solutions.

Up to this point I have been assuming that `A` is square and not singular, If `A` is not square, or is square and singular, then `inv(A)` does not exist. In this case you can use the pseudoinverse function to get a least square solution. In other words A*x-b will not return a zero vector, so x is not an exact solution.

```
X = pinv(A)*b               %Section 9.3
```