1 Linear system of equations

Try following commands on MATLAB:

```
A = ones(4, 4)
b = randn(4, 1)
x = A \b
```

Does it have a solution? If not, what is the reason? Now, try:

```
b = ones(4, 1)
x = [1 0 0 0]'
A * x
```

So, the system $Ax = b$ does have a solution. Still unfortunately, that is not the only solution. Try,

```
x = [0 1 0 0]'
A * x
```

This x is also a solution. Next, try:

```
x = [-4 5 2.27 -2.27]'
A*x
```

This x is a solution! It is not hard to see that there are endless possibilities for solutions of this equation.

1.1 Theorems

1.1.1 Theorem 1

A linear system $Ax = b$ may have 0, 1 or infinitely many solutions.

Obviously, in most engineering applications we would want to have exactly one solution. The following two theorems tell us exactly when we can and cannot expect this.

1.1.2 Theorem 2

Suppose A is a square matrix (n x n). The following are equivalent:

1. The equation $Ax = b$ has exactly one solution for any b.
2. $det(A) \neq 0$.
3. A has an inverse.
4. The only solution of $Ax = 0$ is $x = 0$.
5. The columns of A are linearly independent (as vectors)

6. The rows of A are linearly independent.

If A has these properties, then it is called non-singular. On the other hand, a matrix that does not have these properties is called singular.

1.1.3 Theorem 3

Suppose A is square matrix, the following are equivalent:

1. The equation $Ax = b$ has 0 or ∞ many solutions depending on b.
2. $det(A) = 0$.
3. A does not has an inverse.
4. The equation $Ax = 0$ has solutions other than $x = 0$.
5. The columns of A are linearly dependent as vectors.
6. The rows of A are linearly dependent.

To see how the two theorems work, define two matrices (type in A1 then scroll up and modify to make A2):

$$\mathrm{A1} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \mathrm{A2} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{bmatrix}$$

and two vectors

$$\mathrm{b1} = \begin{bmatrix} 0 \\ 3 \\ 6 \end{bmatrix}, \mathrm{A2} = \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix}$$

First, calculate the determinants of the matrices:

det(A1)
det(A2)

Then, attempt to find the inverse:

inv(A1)
inv(A2)

Which matrix is singular and which is non-singular? Finally, attempt to solve all the possible equations:

Ax = b
x = A1 \ b1
x = A1 \ b2
x = A2 \ b1

x = A2 \ b2

As you can see, equations involving the non-singular matrix have one and only one solution, but equation involving a singular matrix are more complicated.

1.2 The Residual

Recall that the residual for an approximate solution x of an equation $f(x) = 0$ is defined as $r = ||f(x)||$. It is a measure of how close the equation is to being satisfied. For a linear system of equations, we define the residual vector of an approximate solution x by

$$r = Ax - b \tag{1}$$

If the solution vector x were exactly correct, then r would be exactly the zero vector. The size (norm) of r is an indication of how close we have come to solving $Ax = b$. We will refer to this number as the scalar residual or just Residual of the approximate solution.

$$r = ||Ax - b|| \tag{2}$$

2 Accuracy, Condition Numbers and Pivoting

In this lecture, we will discuss two separate issues of accuracy is solving linear systems. The first, pivoting, is a method that ensures that Gaussian elimination proceeds as accurately as possible. The second, condition number, is a measure of how bad a matrix is. We will see that if a matrix has a bad condition number, the solutions are unstable with respect to small changes in data.

2.1 The effect of rounding

All computers store numbers as finite strings of binary floating point digits (bits). This limits numbers to a fixed number of significant digits and implies that after even the most basic calculations, rounding must happen.
Consider, the following exaggerated example. Suppose, that our computer can only store 2 significant digits and it is asked to do Gaussian elimination on

$$\begin{bmatrix} .001 & 1 & | & 3 \\ 1 & 2 & | & 5 \end{bmatrix} \tag{3}$$

Doing the elimination exactly would produce:

$$\begin{bmatrix} .001 & 1 & | & 3 \\ 0 & -998 & | & -2995 \end{bmatrix} \tag{4}$$

but rounding to 2 digits, our computer would store this as

$$\begin{bmatrix} .001 & 1 & | & 3 \\ 0 & -1000 & | & -3000 \end{bmatrix} \tag{5}$$

Back-solving this reduced system gives:
$x_1 = 0$ and $x_2 = 3$

This seems fine until you realize that back-solving the un-rounded system gives:
$x_1 = -1$ and $x_2 = 3.001$

2.2 Row pivoting

A way to fix the problem is to use pivoting, which means to switch rows of the matrix. Since, switching rows of the augmented matrix just corresponds to switching the order of the equations, no harm is done:

$$\begin{bmatrix} 1 & 2 & | & 5 \\ .001 & 1 & | & 3 \end{bmatrix} \tag{6}$$

Exact elimination would produce

$$\begin{bmatrix} 1 & 2 & | & 5 \\ 0 & .998 & | & 2.995 \end{bmatrix} \tag{7}$$

Storing this result with only 2 significant digits gives

$$\begin{bmatrix} 1 & 2 & | & 5 \\ 0 & 1 & | & 3 \end{bmatrix} \tag{8}$$

Now back-solving produces:
$x_1 = -1$ and $x_2 = 3$.
which is the true solution (rounded to 2 significant digits).

The reason this worked is because 1 is bigger than 0.001. To pivot we switch rows so that the largest entry in a column is the one used to eliminate the others. In bigger matrices, after each column is completed, compare the diagonal element of the next column with all the entries below it. Switch it (and the entire row) with the one with greatest absolute value. For example in the following matrix, the first column is finished and before doing the second column, pivoting should occur since $|-2| > |1|$.

$$\begin{bmatrix} 1 & -2 & 3 & | & 4 \\ 0 & 1 & 6 & | & 7 \\ 0 & -2 & -10 & | & -10 \end{bmatrix} \tag{9}$$

Pivoting the 2nd and 3rd rows would produce:

$$\begin{bmatrix} 1 & -2 & 3 & | & 4 \\ 0 & -2 & -10 & | & -10 \\ 0 & 1 & 6 & | & 7 \end{bmatrix} \tag{10}$$

2.3 Condition number

In some systems, problems occur even without rounding. Consider the following augmented matrices:

$$\begin{bmatrix} 1 & 1/2 & | & 3/2 \\ 1/2 & 1/3 & | & 1 \end{bmatrix} \tag{11}$$

and

$$\begin{bmatrix} 1 & 1/2 & | & 3/2 \\ 1/2 & 1/3 & | & 5/6 \end{bmatrix} \tag{12}$$

Here, we have the same A, but two different input vectors:
$b_1 = (3/2, 1)'$ and $b_2 = (3/2, 5/6)'$

which are pretty close to one another. You would expect then that the solutions x_1 and x_2 would also be close. Notice, that this matrix does not need pivoting. Eliminating exactly, we get:

$$\begin{bmatrix} 1 & 1/2 & | & 3/2 \\ 0 & 1/12 & | & 1/4 \end{bmatrix} \tag{13}$$

and

$$\begin{bmatrix} 1 & 1/2 & | & 3/2 \\ 0 & 1/12 & | & 1/12 \end{bmatrix} \tag{14}$$

Now, solving we find:
$x_1 = (0, 3)'$ and $x_2 = (1, 1)'$ which are not close at all despite the fact that we did the calculations exactly. This poses a new problem: some matrices are very sensitive to small changes in input data. The extent of this sensitivity is measured by the condition number. The definition of condition number is: consider all small changes δA and δb in A and b and the resulting change, δx, in the solution x. Then,

$$cond(A) = max\left[\frac{||\delta x||/||x||}{\frac{\delta A}{A} + \frac{\delta b}{b}}\right] = max\left[\frac{Relativeerrorofoutput}{Relativeerrorofinputs}\right] \tag{15}$$

Put another way, changes in the input data get multiplied by the condition number to produce changes in the outputs. Thus, a high condition number is bad. It implies that small errors in the input can cause large errors in the output.

In MATLAB, enter:
H = hilb(2)

which should result in the matrix above. MATLAB produces condition number of a matrix with the command:
cond(H)

Thus, for this matrix small errors in the input can get magnified by 19 in the output. Next, try the matrix

A = [1.2969 0.8648; .2161 .1441]
cond(A)

For this matrix, small errors in the input can get magnified by 2.5 x $10^(8)$ in the output! (We will see this in the exercise) This is obviously not very good for engineering where all measurements, constants and inputs are approximate.

Is there a solution to the problem of bad condition numbers? Usually, bad conditions numbers in engineering contexts result from poor design. So, the engineering solution to bad conditioning is redesign.

Finally, find the determinant of the matrix A above:
det(A)
which will be small. If det(A) = 0, then the matrix is singular, which is bad because it implies there will not be unique solution. The case here, det(A) $\approx$ 0, is also bad, because it means the matrix is almost singular. Although, det(A) $\approx$ 0 generally indicates that the condition number will be large, they are actually independent things. To see this, find the determinant and condition number of the matrix [1e-10,0;0,1e-10] and the matrix [1e+10,0;0,1e-10].

3 LU Decomposition

In many applications where linear systems appear, one needs to solve Ax = b for many different vectors b. For instance, a structure must be tested under several different loads, not just one. As in the example of a truss (discussed in past labs), the loading in such a problem is usually represented by the vector **b**. Gaussian elimination with pivoting is the most efficient and accurate way to solve a linear system. Most of the work in this method is spent on the matrix A itself. If we need to solve several different systems with the same A, and A is big, then we would like to avoid repeating the steps of Gaussian elimination on A for every different b. This can be accomplished by the LU decomposition, which in effect records the steps of Gaussian elimination.

3.1 LU Decomposition

The main idea of the LU decomposition is to record the steps used in Gaussian elimination on A in the places where the zero is produced. Consider the matrix:

$$A = \begin{bmatrix} 1 & -2 & 3 \\ 2 & -5 & 12 \\ 0 & 2 & -10 \end{bmatrix} \tag{16}$$

The first step of Gaussian elimination is to subtract 2 times the first row from the second row. In order to record what we have done, we will put the multiplier, 2, into the place it was used to make a zero, i.e. the second row, first column.

In order to make it clear that it is a record of the step and not an element of A, we will put it in parentheses. This leads to

$$\begin{bmatrix} 1 & -2 & 3 \\ (2) & -1 & 6 \\ 0 & 2 & -10 \end{bmatrix} \tag{17}$$

There is already a zero in the lower left corner, so we dont need to eliminate anything there. We record this fact with a (0). To eliminate the third row, second column, we need to subtract 2 times the second row from the third row. Recording the 2 in the spot it was used we have

$$\begin{bmatrix} 1 & -2 & 3 \\ (2) & -1 & 6 \\ (0) & (-2) & 2 \end{bmatrix} \tag{18}$$

Let U be the upper triangular matrix produced, and let L be the lower triangular matrix with the records and ones on the diagonal, i.e.

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \tag{19}$$

and

$$\begin{bmatrix} 1 & -2 & 3 \\ 0 & -1 & 6 \\ 0 & 0 & 2 \end{bmatrix} \tag{20}$$

then, we have the following wonderful property:

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 3 \\ 0 & -1 & 6 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 3 \\ 2 & -5 & 12 \\ 0 & 2 & -10 \end{bmatrix} = A \tag{21}$$

Thus, we see that A is actually the product of L and U. Here, L is lower triangular and U is upper triangular. When a matrix can be written as a product of simpler matrices, we call that a decomposition of A and this one we call the LU decomposition.

3.2 Using LU to solve equations

If we also include pivoting, then an LU decomposition for A consists of three matrices P, L and U such that

$$PA = LU \tag{22}$$

The pivot matrix P is the identity matrix, with the same rows switched as the rows of A are switched in the pivoting. For instance,

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{23}$$

would be the pivot matrix, if the second and third rows of A are switched by pivoting. MATLAB will produce an LU decomposition with pivoting for a matrix A with the command.
$> [LUP] = lu(A)$
where P is the pivot matrix. To use this information to solve Ax = b, we first pivot both sides by multiplying by the pivot matrix:

$$PAx = Pb = d \tag{24}$$

Substituting LU for PA we get

$$LUx = d \tag{25}$$

Then, we define y = Ux, which is unknown since x is unknown. Using forward-substitution, we can (easily) solve

$$Ly = d \tag{26}$$

for y and then using back-substitution we can (easily) solve

$$Ux = y \tag{27}$$

for x. In MATLAB, this would work as follows:
A = rand(5,5)
$[LUP] = lu(A)$
b = rand(5, 1)
d = P * b
$y = L\backslash d$
$x = U\backslash y$
rnorm = norm(A*x - b)

We can then solve for any other b without redoing the LU step. Repeat the sequence for a new right hand side: c = randn(5,1); you can start at the third line. While this may not seem like a big savings, it would be if A were a large matrix from an actual application.
The LU decomposition is an example of Matrix Decomposition which means taking a general matrix A and breaking it down into components with simpler properties. Here L and U are simpler because they are lower and upper triangular. There are many other matrix decomposition that are useful in various contexts. Some of the most useful of these are the QR decomposition, the Singular Value decomposition and Cholesky decomposition

4 Exercise

1. By hand, find all the solutions (if any) of the following linear system using the augmented matrix and Gaussian elimination:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 4 \\ 4x_1 + 5x_2 + 6x_3 &= 10 \\ 7x_1 + 8x_2 + 9x_3 &= 14 \end{aligned} \tag{28}$$

2. Write a well-commented MATLAB function program **mysolvecheck** with input a number n that makes a random n n matrix A and a random vector b, solves the linear system Ax = b, calculates the scalar residual $r = ||Ax - b||$, and outputs that number as r.

3. Write a well-commented MATLAB script program that calls **mysolvecheck** 10 times each for n = 10, 50, 100, 500, and 1000, records and averages the results and makes a log-log plot of the average e vs. n. Increase the maximum n until the program stops running within 5 minutes. Turn in the plot and the two programs.

4. Let,
$$A = \begin{bmatrix} 1.2969 & .8648 \\ .2161 & .1441 \end{bmatrix} \tag{29}$$
(a) Find the determinant and inverse of A (using MATLAB).
(b) Let B be the matrix obtained from A by rounding off to three decimal places (1.2969 → 1.297). Find the determinant and inverse of B. How do A^{-1} and B^{-1} differ? Explain how this happened.
(c) Set b1 = [1.2969; 0.2161] and do $x = A\backslash b1$. Repeat the process but with vector b2 obtained from b1 by rounding off to three decimal places. Explain exactly what happened. Why was the first answer so simple? Why do two answers differ by so much?

5. Try,
B = [sin(sym(1)) sin(sym(2)); sin(sym(3)) sin(sym(4))]
c = [1; 2]
x = B \ c
pretty(x)
Next, input the matrix
$$Cs = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \tag{30}$$
symbolically as above by wrapping each number in **sym**. Create a numerical version via Cn = double(Cs) and define the two vectors d1 = [4; 8] and d2 = [1; 1]. Solve the systems Cs*x = d1, Cn*x = d1, Cs*x = d2, and Cn*x = d2. Explain the results. Does the symbolic or non-symbolic way give more information?

6. Recall the matrix A that you saved using Aequiltruss in previous labs. (Enter > load Aequiltruss or click on the file Aequiltruss.mat in the folder where you saved it; this should reproduce the matrix A.) Find the condition number for this matrix. Write a well-commented script program to calculate the mean and median condition number of 1000 randomly generated 7 by 7 matrices. How does the condition number of the truss matrix compare to randomly generated matrices? Turn in your script with the answers to these questions.

7. Solve the systems below by hand using Gaussian elimination and back substitution on the augmented matrix. As a by-product, find the LU decomposition of A. Pivot wherever appropriate. Check by hand that LU = PA and Ax = b.
   (a)

$$A = \begin{bmatrix} 2 & 4 \\ .5 & 4 \end{bmatrix} \tag{31}$$

$$b = \begin{bmatrix} 0 \\ 3 \end{bmatrix} \tag{32}$$

   (b)

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 5 \end{bmatrix} \tag{33}$$

$$b = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \tag{34}$$

8. Finish the following MATLAB function program:

Figure 1: Program to write

```
function [x1, r1, x2, r2] = mysolve(A,b)
    % Solves linear systems using the LU decomposition with pivoting
    % and also with the built-in solve function A\b.
    % Inputs: A -- the matrix
    %         b -- the right-hand vector
    % Outputs: x1 -- the solution using the LU method
    %          r1 -- the norm of the residual using the LU method
    %          x2 -- the solution using the built-in method
    %          r2 -- the norm of the residual using the
    %                built-in method
```

Using format long, test the program on both random matrices (randn(n,n)) and Hilbert matrices (hilb(n)) with n large (as big as you can make it and the program still runs). Print your program and summarize your observations. (Do not print any random matrices or vectors.)