**SOLVING SYSTEMS OF EQUATIONS AND MATRIX THEORY**

The syntax for solving systems of linear equations is very simple. Indeed, since solving systems of linear equations is usually a proceedure consisting of matrix operations, it should be easy in Matlab.

* In order to solve a system of linear equations, given in matrix form *Ax=b* where *x* and *b* are column vectors, you need only type *x=A\b*. Of course, if *A* is a square nonsingular matrix then we could also write (less efficiently from the numerical point of view) *x=inv(A)\*b* or *x=A^(-1)\*b*.
* Example:
* Solve the system of equations
* x\_1+2x\_2+3x\_3 = 366
* 4x\_1+5x\_2+6x\_3 = 804
* 7x\_1+8x\_2 = 351
* To carry this out in Matlab we type:
* A=[1 2 3;4 5 6;7 8 0];
* b=[366 804 351]';
* x=A\b

We could also check

det(A)

If this is not zero we could type

x=inv(A)\*b

x=A^(-1)\*b

* Lets consider an ill-posed problem: Find two numbers X1 and X2 whose average is 3.
* A=[1/2 1/2];
* b=3;
* x=A\b
* x\_ls =
* 6

0

This answer has the fewest nonzero entries.

* Here is another solution
* x=pinv(A)\*b % pinv gives the pseudo inverse solution,
* x\_pinv =
* 3.0000

3.0000

* There are, of course, infinitely many answers. The pseudo-inverse solution is always the smallest in norm
* [norm(x\_ls) norm(x\_pinv)]
* ans =
* 6.0000 4.2426
* Here is another example:
* A=[1 2 3;4 5 6;7 8 0;2 5 8]
* b=[366 804 351 514]';
* x=A\b %compute the least square solution

In the case of an overdetermined system (more equations than unknowns) the Matlab command *\* automatically finds the solution that minimizes the squared error in

* Ax-b. This solution is called the *least square solution*.
* res=A\*x-b % this residual has the smallest norm

**EXERCISES**

1. As you saw above, when there are fewer equations than unknowns (the underdetermined case) Matlab provides two easy ways to find solutions: *\* and *pinv*. Use these to solve the system (obtained from above) with
3. A=A' %create 3 equations in 4 unknowns
4. b=b(1:3) %build a new r.h.s

Check that the *\* solution has most zeros and that the *pinv* solution has the smallest norm.

1. A Matrix *A* is called *idempotent* if *A^2=A*. Show that
3. A=[4 -2;6 -3]

is idempotent. Show that

U=[2 -2 -4;-1 3 4;1 -2 -3]

V=[-1 2 4;1 -2 -4;-1 2 4]

are idempotent and find

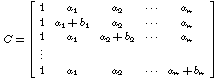
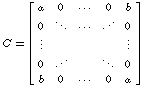
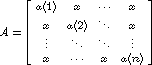
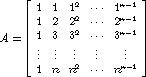
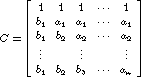
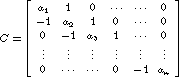
U\*V

Thus, unlike numbers, where only *1^2=1* and *0^2=0*, many matrices can be idempotent and *a\*b=0* does not mean that *a* or *b* is zero (Matrices do not form a division algebra).

1. A square matrix *A* is called *nilpotent* of order *r* if *A^r=0*. The number *r* is called the index.
3. A=[1 1;-1 -1]
4. A^2
5. Show that the following matrices are nilpotent:
7. A=[1 2 3;1 2 3;-1 -2 -3]
8. B=[-4 4 -4;1 -1 1;5 -5 5]
9. C=[0 1 0 0;0 0 1 0;0 0 0 1;0 0 0 0]
10. Write a function m-file as a function of *k* that allows you to verify the fact that the matrix *A* given by
12. A=[k 1+k;1-k -k]^2

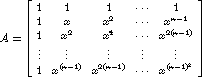
is the identity for any real or complex number *k*. Thus for matrices, a matrix can have infinitly many square roots. Along these lines note that the matrix *A=[0 1;0 0]* has no square roots. Some matrices have only a finite numer of square roots. Find the square roots of

A=[3 -4;1 -1]

1. Solve the systems of equations
3. x\_1+2x\_2-x\_3=3
4. 3x\_1-x\_2+2x\_3=1
5. 2x\_1-2x\_2+3x\_3=2
6. x\_1-x\_2+x\_3=-1
8. 5x\_1+3x\_2+7x\_3-4=0
9. 3x\_1+26x\_2-2x\_3-9=0
10. 7x\_1+2x\_2+10x\_3-5=0
11. Write a program to input two numbers *a* and *b* and build the an *n* by *n* matrix *A* (for any desired *n*) with the number *a* on the main diagonal and the number *b* everwhere else. Then find the determinant of *A* and find a value for *x* in the formula *d=(a-b)^n+x(a-b)^(n-1)* so that *d* is the determinant of *A*. After some fooling around or using a bit of math you find: *ans: x=nb*.
12. Write a program that allows you to input two vectors *x* and *y* of length *n* and build a matrix *G* with entries *G(i,j)=x(i)^(y(j))*. Try a few different vectors *x* and *y* of lengths *n* (for a few different *n*'s) and compute the determinants of the resulting matrices. Can you give necessary and sufficient conditions for this determinant to be zero? I can't find a formula for the determinant but, I think it is not zero if none of the x(i) are equal and none of the y(j) are equal It is zero if any of these are equal.
13. Consider the special case of the previous problem when *y=0:(n-1)*. For this case, compare your answer for the determinant with the answer you get by writing a program to compute *d=\prod\_{1<= i< j<= n}(x\_j-x\_i)*.
14. Write a program that inputs vectors *a* and *b* of the same length *n* and builds the matrix . Find the determinant. Look at some simple cases. You will see that the answer only depends on the vector *b*. Can you guess how?
15. Write a program that inputs two numbers *a* and *b* and builds the *2k* by *2k* matrix (you must input *k*) . (i.e., a is on the main diagonal, *b* is on the backwards diagonal and there are zeros everywhere else.) Find the determinant. Take some simple cases. A formula for the determinant looks like *(a^x-b^y)^z* for some *x*, *y* and *z*. Find *x*, *y*, and *z*.
16. Write a program that inputs a vector *a* of any length *n* and a number *x* and builds the *n* by *n* matrix Then define the function *f(x)= prod(a-x)*, compute the derivative of *f* and compare the determinant of *A* with *d=f(x)-xf'(x)* (i.e., write a series of matlab statements that computes *f* and *f'* and *d* -- you might want to determine the derivative of f by hand first so you have a formula for it.)
17. For any positive integer *n* build the matrix Compute the determinant and compare with *d=1!2!3!... (n-1)!*.
18. Write a program that inputs two vectors *a* and *b* of length *n* and then builds the matrix (Note the size of *C* is *(n+1)* by *(n+1)*.) Find the determinant. Can you guess a formula for the determinant in terms of *a* and *b*?
19. Write a program that inputs a vector *a* of length *n* and builds the matrix Let *Ck* denote the determinant of the submatrix of *C* given by *C(1:k,1:k)* for *k* from *3* to *n*. Show that *Ck=a\_kC(k-1)+C(k-2)*
20. For any *n* build the *n* by *n* matrix I:\ARCHIVOS MATLAB\Archivos HTML\matlab-5_archivos\Untitled7.gifand find the inverse.
21. Repeat the last problem but with the lower diagonal having all *-1* instead of *1*.
22. Recall, from college algebra or trig, that the equation *x^n=1* has *n* solutions called the *n*th-roots of unity. These roots can be found using DeMoivre's formula which uses Euler's formula   
      
    *exp(i\*theta)=cos (theta )+i\*sin (theta )*.   
      
    The result is   
      
    *omega(j)= exp(2\*j\*pi\* i/n))*, for *j=0, ..., (n-1)*.  
      
    In matlab this vector of numbers can be written very easily. Write a program that inputs *n* and finds *omega*. Do a little experimenting, for example,

Check that omega(j)^n=1,

* 1. What are the values of  
     i) sum(omega),  
     ii) sum(omega.^2),  
     iii) prod(omega).

Now build a matrix in the form Compute the determinant with *x* replaced by any *omega(j)*. Can you figure out a formula for the determinant?

1. Let us consider an exercise intended to convince you not to play the lottery. Write a program that lets you input six integers from 1 to 50 and also an integer *n* corresponding to playing the lottery *n* times. Next the program whould generate six distinct integers from 1 to 50 and compare with your six numbers to see if you have matched 3, 4, 5 or 6 numbers. The program should do this process *n* times keeping track of how many times you had a winner and what type of winner it was, i.e., a 3, 4, 5, or 6 number winner. You might want to write it as a "function" m-file with variable *n* (the number of times to run the lottery) and a vector *v* which contains your lottery numbers (e.g. *[1 23 32 27 19 48]*. (Hint: You might find the following useful:
2. test=zeros(1,5);
3. while all(test~=0)~=1 % this while statement helps to obtain a unique vector
4. lottst=floor(50\*rand(1,6))+1 % After you run the program a few times put a
5. % semicolon at the end to suppress prining lottst
6. lot1=sort(lottst);
7. test=lot1(2:6)-lot1(1:5);
8. end
9. disp('the lottery numbers are')
10. disp(lottst);

[RETURN TO MAIN DOC](http://texas.math.ttu.edu/%7Egilliam/mlhtml_1/m4330_ml_1.html)