Chapter 16 Numerical Linear Algebra

16.1 Sets of Linear Equations

MATLAB was developed to handle problems involving matrices and vectors in an efficient way. One of the most basic problems of this type involves the solution of a system of linear equations. The built-in matrix algebra functions of MATLAB facilitate the process of establishing whether a solution exists and finding it when it does. For example, the 4×3 system below

$$\begin{aligned} x + 4y + 7z &= -5 \\ x + y + z &= 1 \\ 2x + 3y + 4z &= 0 \\ x - y - 3z &= 0 \end{aligned}$$

is a system of equations which can be represented as $A\underline{x}=\underline{b}$ where

$$A=\begin{pmatrix} 1 & 4 & 7 \\ 1 & 1 & 1 \\ 2 & 3 & 4 \\ 1 & -1 & -3 \end{pmatrix}, \quad \underline{x}=\begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \underline{b}=\begin{pmatrix} -5 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

To check if a unique solution exists, the MATLAB function '`rank`' can be used with the coefficient matrix A and the augmented matrix $(A|\underline{b})$. If the ranks of both matrices are equal to the number of unknown variables, in this case 3, a unique solution exists. If the ranks are equal and less than 3, an infinite number of solutions exist, i.e. the system is underconstrained. Finally if the rank of A is one less than the rank of $(A|\underline{b})$, the equations are inconsistent, i.e the system is overconstrained and there is no solution.

Example 16.1.1

```
A=[1,4,7; 1,1,1; 2,3,4; 1 -1 -3];
b=[-5;1;0;0];
rA=rank(A) % Check rank of A
rAb=rank([A b]) % Check rank of (A|b)

rA =   2
rAb =  3
```

Since the ranks are unequal, the system of equations are inconsistent and no solution exists. Suppose the $A(4,3)$ element were +3 instead of -3.

Example 16.1.2

```
A=[1,4,7; 1,1,1; 2,3,4; 1 -1 3];
b=[-5;1;0;0];
rA=rank(A) % Check rank of A
rAb=rank([A b]) % Check rank of (A|b)

rA =   3
rAb =  3
```

In this case, the ranks are both equal to the number of unknowns so there is a unique solution to the system. The solution can be determined by transforming the augmented matrix into an echelon form using the '`rref`' function, which stands for row reduced echelon form, a Gauss elimination type of solution.

Example 16.1.3

```
format rat
rref([A b])

ans =
      1            0            0          13/6
      0            1            0          -1/3
      0            0            1          -5/6
      0            0            0            0
```

From the echelon form of $(A|\underline{b})$ the solution is given by

$$x = 13/6, \quad y = -1/3, \quad z=-5/6$$

When the coefficient matrix A is square (same number of equations as unknowns) the MATLAB function '`det`' will also reveal whether there is a unique solution. A nonzero value implies a unique solution and a zero determinant indicates either none or an infinite number of solutions. If the determinant is nonzero, the coefficient matrix A is nonsingular, and the matrix inverse can be used to obtain the unique solution. The following system of equations illustrates this point.

$$\begin{aligned} x_1 + 4x_2 + 2x_3 + 3x_4 &= 5 \\ 2x_1 - x_2 + 3x_3 + 8x_4 &= 2 \\ 3x_1 - x_2 + 4x_3 - x_4 &= 0 \\ 4x_1 - 6x_2 + 5x_3 + 2x_4 &= -3 \end{aligned}$$

Example 16.1.4

```
A=[1 4 2 3; 2 -1 3 8; 3 -1 4 -1; 4 -6 5 2];
b=[5;2;0;-3];
if  abs(det(A)) > 1e-5 % Check for unique solution
  D=det(A)
```

```
  disp('System is Consistent - Unique Solution Exists')
  disp(' ')
  x=inv(A)*b % Find unique solution to Ax=b
elseif rank([A b]) == rank(A) % Check if system is consistent
  disp('System is Consistent - Infinite Solutions Exist')
  disp(' ')
  exhelon_form=rref([A b])
else
  disp('System is Inconsistent - No Solutions Exist')
end % if abs(det) > 1e-5
```

```
D = -10
System is Consistent - Unique Solution Exists

x =
   -41/5
     1/5
    31/5
      0
```

If the A(4,4) element is changed to 4, the system is still consistent but is now indeterminate as shown in Example 16.1.5.

Example 16.1.5

```
A=[1 4 2 3; 2 -1 3 8; 3 -1 4 -1; 4 -6 5 4];
b=[5;2;0;-3];
if  abs(det(A)) > 1e-5 % Check for unique solution
  D=det(A)
  disp('System is Consistent - Unique Solution Exists')
  disp(' ')
  x=inv(A)*b % Find unique solution to Ax=b
elseif rank([A b]) == rank(A) % Check if system is consistent
  disp('System is Consistent - Infinite Solutions Exist')
  disp(' ')
  Echelon_form=rref([A b])
else
  disp('System is Inconsistent - No Solutions Exist')
end % if abs(det) > 1e-5
```

```
System is Consistent - Infinite Solutions Exist

Echelon_form =
     1     0     0  -161/5   -41/5
     0     1     0   -14/5     1/5
     0     0     1   116/5    31/5
     0     0     0       0       0
```

From the row reduced echelon form, the solution is

$$x_1 = -\frac{41}{5} + \frac{161}{5}x_4, \qquad x_2 = \frac{1}{5} + \frac{14}{5}x_4, \qquad x_3 = \frac{31}{5} - \frac{116}{5}x_4$$

Finally, the system is inconsistent if $A(4,4) = 4$ and the element $\underline{b}(4)$ is other than -3 as shown below.

Example 16.1.6

```
A=[1 4 2 3; 2 -1 3 8; 3 -1 4 -1; 4 -6 5 4];
b=[5;2;0;1];
if  abs(det(A)) > 1e-5 % Check for unique solution
  D=det(A)
  disp('System is Consistent - Unique Solution Exists')
  dsip(' ')
  x=inv(A)*b % Find unique solution to Ax=b
elseif rank([A b]) == rank(A) % Check if system is consistent
  disp('System is Consistent - Infinite Solutions Exist')
  disp(' ')
  exhelon_form=rref([A b])
else
  disp('System is Inconsistent - No Solutions Exist')
end % if abs(det) > 1e-5
```

```
System is Inconsistent - No Solutions Exist
```

16.2 Matrix Functions

MATLAB does way more than simply allow one to determine the nature of solutions to a system of linear equations. Numerous built-in functions with matrix arguments reduce problems in linear algebra to a few simple commands. A complete listing of matrix functions can be accessed by typing `help matfun'` or `helpwin matfun'`. The first command results in the following

```
» help matfun

 Matrix functions - numerical linear algebra.

 Matrix analysis.
   norm          - Matrix or vector norm.
   normest       - Estimate the matrix 2-norm.
   rank          - Matrix rank.
   det           - Determinant.
   trace         - Sum of diagonal elements.
   null          - Null space.
   orth          - Orthogonalization.
   rref          - Reduced row echelon form.
   subspace      - Angle between two subspaces.

 Linear equations.
   \ and /       - Linear equation solution; use "help slash".
   inv           - Matrix inverse.
   cond          - Condition number with respect to inversion.
   condest       - 1-norm condition number estimate.
   chol          - Cholesky factorization.
   cholinc       - Incomplete Cholesky factorization.
   lu            - LU factorization.
   luinc         - Incomplete LU factorization.
   qr            - Orthogonal-triangular decomposition.
   lsqnonneg     - Linear least squares with nonnegativity constraints.
   pinv          - Pseudoinverse.
   lscov         - Least squares with known covariance.

 Eigenvalues and singular values.
   eig           - Eigenvalues and eigenvectors.
   svd           - Singular value decomposition.
   gsvd          - Generalized ingular value decomposition.
   eigs          - A few eigenvalues.
   svds          - A few singular values.
   poly          - Characteristic polynomial.
   polyeig       - Polynomial eigenvalue problem.
   condeig       - Condition number with respect to eigenvalues.
```

hess - Hessenberg form.
qz - QZ factorization for generalized eigenvalues.
schur - Schur decomposition.

Matrix functions.
expm - Matrix exponential.
logm - Matrix logarithm.
sqrtm - Matrix square root.
funm - Evaluate general matrix function.

Factorization utilities
qrdelete - Delete column from QR factorization.
qrinsert - Insert column in QR factorization.
rsf2csf - Real block diagonal form to complex diagonal form.
cdf2rdf - Complex diagonal form to real block diagonal form.
balance - Diagonal scaling to improve eigenvalue accuracy.
planerot - Given's plane rotation.
cholupdate - rank 1 update to Cholesky factorization.
qrupdate - rank 1 update to QR factorization.

The problem of finding the eigenvalues and eigenvectors of a matrix is a common one. The MATLAB function '`eig`' will return the eigenvalues of a square matrix.

Example 16.2.1

```
A=100*rand(5)
x=eig(A)
```

```
A =
    31.6728   53.2132    0.9171   14.4678   81.8699
    12.6724   76.2404   64.4713   20.7478   34.8959
    11.5800   41.7496   11.2353   85.0641   16.2547
    16.4358   83.5530   21.5702   71.3109   53.9751
     9.3948   80.4729   90.9287    8.0425   92.3327

x =
    1.0e+002 *
    2.2146
    0.2224 + 0.1368i
    0.2224 - 0.1368i
    0.0843 + 0.4418i
    0.0843 - 0.4418i
```

It can also be used with two outputs when the eigenvectors are required as well.

Example 16.2.2

```
A=10-5*randn(3)
[v,d]=eig(A)
```

```
A =
     4.0454    8.3635    6.3710
     4.0542    9.1268   12.9416
    10.1882   10.9335   -0.9159

v =
     0.6963   -0.4983   -0.0292
    -0.6533   -0.6799   -0.5765
     0.2973   -0.5380    0.8166

d =
    -1.0822    0         0
     0        22.3377    0
     0         0        -8.9992
```

Another common application in linear algebra involves finding an orthonormal set of vectors or basis in the range of a matrix. The vectors in the basis are all unit vectors which are mutually orthogonal. If the matrix is singular, the number of vectors in the basis is less than the dimension of the matrix. The MATLAB command is '`orth`'. It is illustrated below on two 4×4 matrices; the first is nonsingular and the second is singular.

Example 16.2.3

```
A1=[1 4 2 3; 2 -1 3 8; 3 -1 4 -1; 4 -6 5 2] % rank(A1)=4
v1=orth(A1) % Find basis vectors for range of A1
A2=[1 4 2 3; 2 -1 3 8; 3 -1 4 -1; 4 -6 5 4] % rank(A2)=3
v2=orth(A2) % Find basis vectors for range of A2

A1 =
     1     4     2     3
     2    -1     3     8
     3    -1     4    -1
     4    -6     5     2

v1 =
     0.1532    0.5691    0.6288    0.5072
     0.6453    0.5578   -0.3487   -0.3884
     0.2936   -0.3173    0.6868   -0.5843
     0.6884   -0.5142   -0.1061    0.5005

A2 =
     1     4     2     3
     2    -1     3     8
     3    -1     4    -1
     4    -6     5     4

v2 =
     0.1564    0.6287    0.5747
     0.6320    0.5122   -0.2972
     0.2434   -0.3514    0.7532
     0.7190   -0.4679   -0.1188
```

It is easy to verify the unit size and orthogonality of the basis vectors. The length of the basis vectors can determined using the '`norm`' function or computed in a straightforward manner using the definition of the length of a vector. The dot product of two vectors is zero when the vectors are orthogonal. The following example demonstrates this for the basis vectors obtained in Example 16.2.3.

Example 16.2.4

```
x1=v2(:,1) % Create vector x1 from 1st column of basis v2
x2=v2(:,2) % Create vector x2 from 2nd column of basis v2
x3=v2(:,3) % Create vector x3 from 3rd column of basis v2
L1=(norm(x1)) % Find length of vector x1
dp12=sum(x1.*x2) % Compute dot product of vectors x1 and x2
dp13=sum(x1.*x3) % Compute dot product of vectors x1 and x32
L2=sqrt(sum(x2.*x2)) % Calculate length of vector x2
dp23=sum(x2.*x3) % Compute dot product of vectors x2 and x3
L3=sqrt(sum(x3.*x3)) % Calculate length of vector x2

x1 =
     0.1564
     0.6320
     0.2434
     0.7190

x2 =
     0.6287
     0.5122
    -0.3514
    -0.4679

x3 =
     0.5747
    -0.2972
     0.7532
    -0.1188

L1 =    1.0000
dp12 = -5.5511e-017
dp13 = -2.7756e-017
L2 =    1.0000
dp23 =  9.7145e-017
L3 =    1.0000
```

The '`polyvalm`' function evaluates a polynomial function of a square matrix. For example, if A is the square matrix below,

$$A=\begin{pmatrix}1 & 0 & 2\\ 3 & -4 & 5\\ 0 & -1 & 0\end{pmatrix}$$

then $f(A)=3A^2+2A+4I$ is obtained as shown in Example 16.2.5 below.

Example 16.2.5

```
A=[1 0 2; 3 -4 5; 0 -1 0]
v=[3 2 4];
f_A=polyvalm(v,A)

A =
      1            0            2
      3           -4            5
      0           -1            0

f_A =
      9           -6           10
    -21           29          -32
     -9           10          -11
```