Very Basic MATLAB

Peter J. Olver
October, 2005

Matrices: Type your matrix as follows:

Use **,** or **space** to separate entries, and **;** or **return** after each row.

```
>> A = [4 5 6 -9;5 0 -3 6;7 8 5 0; -1 4 5 1]
```

or

```
>> A = [4,5,6,-9;5,0,-3,6;7,8,5,0;-1,4,5,1]
```

or

```
>> A = [ 4 5 6 -9
         5 0 -3 6
         7 8 5 0
        -1 4 5 1 ]
```

The output will be:

```
A =
     4     5     6    -9
     5     0    -3     6
     7     8     5     0
    -1     4     5     1
```

You can identify an entry of a matrix by

```
>> A(2,3)
ans =
      -3
```

A colon **:** indicates all entries in a row or column

```
>> A(2,:)
ans =
     5     0    -3     6
>> A(:,3)
ans =
     6
    -3
     5
     5
```

You can use these to modify entries

```
>> A(2,3) = 10
A =
     4     5     6    -9
     5     0    10     6
     7     8     5     0
    -1     4     5     1
```

or to add in rows or columns

```
>> A(5,:)  = [0 1 0 -1]
A =
     4     5     6    -9
     5     0    10     6
     7     8     5     0
    -1     4     5     1
     0     1     0    -1
```

or to delete them

```
>> A(:,2) = []
A =
     4     6    -9
     5    10     6
     7     5     0
    -1     5     1
     0     0    -1
```

Accessing Part of a Matrix:

```
>> A = [4,5,6,-9;5,0,-3,6;7,8,5,0;-1,4,5,1]
A =
     4     5     6    -9
     5     0    -3     6
     7     8     5     0
    -1     4     5     1
>> A([1 3],:)
ans =
     4     5     6    -9
     7     8     5     0
>> A(:,2:4)
ans =
     5     6    -9
     0    -3     6
     8     5     0
     4     5     1
>> A(2:3,1:3)
ans =
     5     0    -3
     7     8     5
```

Switching two rows in a matrix:

```
>> A([3 1],:)  = A([1 3],:)
A =
     7     8     5     0
     5     0    -3     6
     4     5     6    -9
    -1     4     5     1
```

The Zero matrix:

```
>> zeros(2,3)
ans =
     0     0     0
     0     0     0
>> zeros(3)
ans =
     0     0     0
     0     0     0
     0     0     0
```

Identity Matrix:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

Matrix of Ones:

```
>> ones(2,3)
ans =
     1     1     1
     1     1     1
```

Random Matrix:

```
>> A = rand(2,3)
A =
     0.9501    0.4860    0.4565
     0.2311    0.8913    0.0185
```

Note that the random entries all lie between 0 and 1.

Transpose of a Matrix:

```
>> A = [4,5,6,-9;5,0,-3,6;7,8,5,0;-1,4,5,1]
A =
     4     5     6    -9
     5     0    -3     6
     7     8     5     0
    -1     4     5     1

>> transpose(A)
ans =
     4     5     7    -1
     5     0     8     4
     6    -3     5     5
    -9     6     0     1

>> A'
ans =
     4     5     7    -1
     5     0     8     4
     6    -3     5     5
    -9     6     0     1
```

Diagonal of a Matrix:

```
>> diag(A)
ans =
     4
     0
     5
     1
```

Row vector:

```
>> v = [1 2 3 4 5]
v =
     1    2    3    4    5
```

Column vector:

```
>> v = [1;2;3;4;5]
v =
     1
     2
     3
     4
     5
```

or use transpose operation ’

```
>> v = [1 2 3 4 5]'
v =
      1
      2
      3
      4
      5
```

Forming Other Vectors:

```
>> v = [1:5]
v =
      1      2      3      4      5
>> v = [10:-2:0]
v =
     10      8      6      4      2      0
>> v = linspace(0,1,6)
v =
       0      0.2000      0.4000      0.6000      0.8000      1.0000
```

Important: to avoid output, particularly of large matrices, use a semicolon **;** at the end of the line:

```
>> v = linspace(0,1,100);
```

gives a row vector whose entries are 100 equally spaced points from 0 to 1.

Size of a Matrix:

```
>> A = [4 5 6 -9 7;5 0 -3 6 -2;7 8 5 0 5 ; -1 4 5 1 -9 ]
A =
      4      5      6     -9      7
      5      0     -3      6     -2
      7      8      5      0      5
     -1      4      5      1     -9
>> size(A)
ans =
      4      5
>> [m,n] = size(A)
m =
      4
n =
      5
```

Output Formats

The command `format` is used to change output format. The default is

```
>> format short
>> pi
ans =
    3.1416
>> format long
>> pi
ans =
   3.14159265358979
>> format rat
>> pi
ans =
  355/113
```

This allows you to work in rational arithmetic and gives the "best" rational approximation to the answer. Let's return to the default.

```
>> format short
>> pi
ans =
    3.1416
```

Arithmetic operators

+ Matrix addition.

A + B adds matrices A and B. The matrices A and B must have the same dimensions unless one is a scalar (1×1 matrix). A scalar can be added to anything.

```
>> A = [4,5,6,-9;5,0,-3,6;7,8,5,0;-1,4,5,1]
A =
     4     5     6    -9
     5     0    -3     6
     7     8     5     0
    -1     4     5     1
>> B = [9 2 4 -9;1 4 -2 -6;8 1 7 0; -3 -4 5 9 ]
B =
     9     2     4    -9
     1     4    -2    -6
     8     1     7     0
    -3    -4     5     9
 >> A + B
ans =
    13     7    10   -18
     6     4    -5     0
    15     9    12     0
    -4     0    10    10
```

- Matrix subtraction.

A - B subtracts matrix A from B. Note that A and B must have the same dimensions unless one is a scalar.

```
>> A - B
ans =
    -5     3     2     0
     4    -4    -1    12
    -1     7    -2     0
     2     8     0    -8
```

*** Scalar multiplication**

```
>> 3*A - 4*B
ans =
   -24     7     2     9
    11   -16    -1    42
   -11    20   -13     0
     9    28    -5   -33
```

* Matrix multiplication.

A*B is the matrix product of A and B. A scalar (a 1-by-1 matrix) may multiply anything. Otherwise, the number of columns of A must equal the number of rows of B.

```
>> A * B
ans =
   116    70     3  -147
     3   -17    29     9
   111    51    47  -111
    32    15    28    -6
```

Note that two matrices must be compatible before we can multiply them.

The order of multiplication is important!

```
>> v = [1 2 3 4]
v =
     1     2     3     4
>> w = [1;2;3;4]
w =
     1
     2
     3
     4
>> v * w
ans =
     30
>> w * v
ans =
     1     2     3     4
     2     4     6     8
     3     6     9    12
     4     8    12    16
```

.* Array multiplication

A.*B denotes element-by-element multiplication. A and B must have the same dimensions unless one is a scalar.

A scalar can be multiplied into anything.

```
>> a = [3 4 5 6 7 8 9]
a =
     3     4     5     6     7     8     9
>> b = [8 6 2 4 5 6 -1]
b =
     8     6     2     4     5     6    -1
```

```
>> a .* b
ans =
    24    24    10    24    35    48    -9
```

^ **Matrix power.**

C = A ^n is A to the n-th power if n is a scalar and A is square. If n is an integer greater than one, the power is computed by repeated multiplication.

```
>> A = [4 5 6 -9;5 0 -3 6;7 8 5 0; -1 4 5 1 ]
A =
     4     5     6    -9
     5     0    -3     6
     7     8     5     0
    -1     4     5     1

>> A^3
ans =
          501          352          351         -651
          451          169          -87          174
         1103          799          533         -492
          445          482          413         -182
```

.^ **Array power.**

C = A .^ B denotes element-by-element powers. A and B must have the same dimensions unless one is a scalar. A scalar can go in either position.

```
>> A = [8 6 2 4 5 6 -1 ]
A =
     8     6     2     4     5     6    -1
>> A.^3
ans =
   512   216     8    64   125   216    -1
```

Length of a Vector, Norm of a Vector, Dot Product

```
>> u = [8 -7 6 5 4 -3 2 1 9]
u =
     8    -7     6     5     4    -3     2     1     9
>> length(u)
ans =
     9
```

```
>> norm(u)
ans =
   16.8819
>> v = [9 -8 7 6 -4 5 0 2 -4]
v =
     9    -8     7     6    -4     5     0     2    -4
>> dot(u,v)
ans =
   135
>> u'*v
ans =
   135
```

Complex vectors:

```
>> u = [2-3i, 4+6i,-3,+2i]
u =
   2.0000- 3.0000i   4.0000+ 6.0000i  -3.0000        0+ 2.0000i
>> conj(u)
ans =
   2.0000+ 3.0000i   4.0000- 6.0000i  -3.0000        0- 2.0000i
```

Hermitian transpose:

```
>> u'
ans =
   2.0000+ 3.0000i
   4.0000- 6.0000i
  -3.0000
        0- 2.0000i
>> norm(u)
ans =
    8.8318
>> dot(u,u)
ans =
    78
>> sqrt(ans)
ans =
    8.8318
>> u'*u
ans =
    78
```

Solving Systems of Linear Equations

The best way of solving a system of linear equations

$$A\mathbf{x} = \mathbf{b}$$

in MATLAB is to use the backslash operation \ (backwards division)

```
>> A = [1 2 3;-1 0 2;1 3 1]
A =
     1     2     3
    -1     0     2
     1     3     1

>> b = [1; 0; 0]
b =
     1
     0
     0

>> x = A\b
x =
     0.6667
    -0.3333
     0.3333
```

The backslash is implemented by using Gaussian elimination with partial pivoting. An alternative, but less accurate, method is to compute inverses:

```
>> B = inv(A)
B =
     0.6667    -0.7778    -0.4444
    -0.3333     0.2222     0.5556
     0.3333     0.1111    -0.2222
```

or

```
>> B = A∧(-1)
B =
     0.6667    -0.7778    -0.4444
    -0.3333     0.2222     0.5556
     0.3333     0.1111    -0.2222
>> x = B * b
x =
     0.6667
    -0.3333
     0.3333
```

Another method is to use the command rref:

To solve the following system of linear equations:

$$\begin{aligned} x_1 + 4x_2 - 2x_3 + x_4 &= 2 \\ 2x_1 + 9x_2 - 3x_3 - 2x_4 &= 5 \\ x_1 + 5x_2 - x_4 &= 3 \\ 3x_1 + 14x_2 + 7x_3 - 2x_4 &= 6 \end{aligned}$$

we form the augmented matrix:

```
>> A = [1,4,-2,3,2; 2,9,-3,-2,5; 1,5,0,-1,3; 3,14,7,-2,6]
A =
     1     4    -2     3     2
     2     9    -3    -2     5
     1     5     0    -1     3
     3    14     7    -2     6
>> rref(A)
ans =
    1.0000         0         0         0   -5.0256
         0    1.0000         0         0    1.6154
         0         0    1.0000         0   -0.2051
         0         0         0    1.0000    0.0513
```

The solution is : $x_1 = -5.0256,\ x_2 = 1.6154,\ x_3 = -0.2051,\ x_4 = 0.0513.$

Case 1: Infinitely many solutions:

```
>> A = [-2 2 -2;1 -1 1; 2 -2 2]
A =
    -2     2    -2
     1    -1     1
     2    -2     2
>> b = [-8; 4; 8]
b =
    -8
     4
     8
>> A\b
Warning:  Matrix is singular to working precision.
ans =
     ∞
     ∞
     ∞
```

MATLABis unable to find the solutions;

In this case, we can apply `rref` to the augmented matrix.

```
>> C = [A b]
C =
    -2          2          -2          -8
     1         -1           1           4
     2         -2           2           8
>> rref(C)
ans =
     1         -1           1           4
     0          0           0           0
     0          0           0           0
```

You can use `rrefmovie` to see each step of Gaussian elimination.

```
>> rrefmovie(C)
  Original matrix
C =
    -2          2          -2          -8
     1         -1           1           4
     2         -2           2           8
Press any key to continue.  .  .
  pivot = C(1,1)
C =
     1         -1           1           4
     1         -1           1           4
     2         -2           2           8
Press any key to continue.  .  .
  eliminate in column 1
C =
     1         -1           1           4
     1         -1           1           4
     2         -2           2           8
Press any key to continue.  .  .
C =
     1         -1           1           4
     0          0           0           0
     2         -2           2           8
Press any key to continue.  .  .
C =
     1         -1           1           4
     0          0           0           0
     0          0           0           0
Press any key to continue.  .  .
  column 2 is negligible
```

```
C =
      1            -1             1             4
      0             0             0             0
      0             0             0             0
Press any key to continue.  .  .
  column 3 is negligible
C =
      1            -1             1             4
      0             0             0             0
      0             0             0             0
Press any key to continue.  .  .
  column 4 is negligible
C =
      1            -1             1             4
      0             0             0             0
      0             0             0             0
```

Conclusion: There are infinitely many solutions since row 2 and row 3 are all zeros.

Case 2: No solutions:

```
>> A = [-2 1; 4 -2]
A =
    -2     1
     4    -2
>> b = [5; -1]
b =
     5
    -1
>> A\b
Warning:  Matrix is singular to working precision.
ans =
     ∞
     ∞
>> C = [A b]
C =
    -2     1     5      4    -2    -1
>> rref(C)
ans =
    1.0000   -0.5000          0
         0         0     1.0000
```

Conclusion: Row 2 is not all zeros, and the system is incompatible.

Important: If the coefficient matrix A is rectangular (not square) then `A\b` gives the least squares solution (relative to the Euclidean norm) to the system $A\mathbf{x} = \mathbf{b}$. If the solution is not unique, it gives the least squares solution $\mathbf{x}$ with minimal Euclidean norm.

```
>> A = [1 1;2 1;-5, -1]
A =
     1     1
     2     1
    -5    -1
>> b = [1;1;1]
b =
      1
      1
      1
>> A\b
ans =
   -0.5385
    1.7692
```

If you want the least squares solution in the square case, one trick is to add an extra equation $0 = 0$ to make the coefficient matrix rectangular:

```
>> A = [-2 2 -2;1 -1 1; 2 -2 2]
A =
    -2     2    -2
     1    -1     1
     2    -2     2
>> b=[-8; 4; 8]
b =
    -8
     4
     8
>> A\b
Warning:  Matrix is singular to working precision.
ans =
     ∞
     ∞
     ∞
>> A(4,:)  = 0
A =
     -2             2            -2
      1            -1             1
      2            -2             2
      0             0             0
```

```
>> b(4) = 0
b =
    -8
     4
     8
     0

>> A\b
Warning:  Rank deficient, rank = 1  tol =    2.6645e-15.
ans =
    4.0000
         0
         0
```

Functions

Functions are vectors! Namely, a vector `x` and a vector `y` of the same length correspond to the sampled function values (x_i, y_i).

To plot the function $y = x^2 - .5\,x$ first enter an array of independent variables:

```
>> x = linspace(0,1,25)
>> y = x.^2 - .5 *x;
>> plot(x,y)
```

The plot shows up in a new window. To plot in a different color, use

```
>> plot(x,y,'r')
```

where the character string `'r'` means red. Use the helpwindow to see other options.

To plot graphs on top of each other, use `hold on`.

```
>> hold on
>> z = exp(x);
>> plot(x,z)
>> plot(x,z,'g')
```

`hold off` will stop simultaneous plotting. Alternatively, use

```
>> plot(x,y,'r',x,z,'g')
```

Surface Plots

Here `x` and `y` must give a regtangular array, and `z` is a matrix whose entries are the values of the function at the array points.

```
>> x =linspace(-1,1,40); y = x;
>> z = x' * (y.^2);
>> surf(x,y,z)
```

Typing the command

```
>> rotate3d
```

will allow you to use the mouse interactively to rotate the graph to view it from other angles.