PREVIEW

The power that MATLAB brings to digital image processing is the extensive set of functions that are available in MATLAB for processing multidimensional arrays, of which matrices (two-dimensional numerical arrays) have a one-to-one correspondence with digital images. In this brief tutorial we discuss and illustrate a number of ways used to manipulate matrices. Usually, this is the first step in learning how to apply MATLAB tools to image-processing applications.

MATRIX NOTATION

Matrices in MATLAB can be represented by any symbol or string of symbols, such as A, a, MAT, RGB, real_mat, and so on. For the most part, we will use uppercase italic notation for matrices (e.g., A, B), and lowercase italics for scalar variables, (e.g., a, b, c) when writing equations or using these symbols in a line of text. When writing a command line, we will use monospaced characters, and make use of the prompt "$>>$" at the beginning of a line, which is the way the reader would see it the screen in the MATLAB workspace. For example, the addition of two matrices will be represented in the form $A = B + C$ in a line of text, and by

```
>> A = B + C;
```

when referring to a command line, where the semicolon at the end of a line is MATLAB's standard symbol for *suppressing* output. If the semicolon is not present at the end of a command line, MATLAB outputs the contents of the operation. In the preceding line, this means that the contents of A would be output on the screen if the semicolon were omitted. We also use monospace notation for the names of MATLAB functions, such as `read`, `write`, `plot`, and so on. When a matrix specifically refers to an image, we will use symbols such as f and g, in an attempt to be as consistent as possible with the notation used in the book *Digital Image Processing*.

The matrix symbol, A, is understood to mean

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \cdots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{bmatrix}.$$

In other words, a matrix is an array of symbols, consisting of M rows and N columns. We denote the size of such an array by the notation $M \times N$. If $M = 1$, we call the

$1 \times N$ matrix A a *row vector*. If N=1, we call the $M \times 1$ matrix A a *column vector*. If $M = N = 1$, then A is a *scalar*.

The notation $A(i, j)$ denotes the element located in row i and column j of A. For example, $A(6, 2)$ is the element located in sixth row and second column of the array. Observe that the first element of the tuple (i, j) refers to a *row* of the array and the second to a *column*, a relationship that is used also to denote *pixel location* in all images in this book.

MATRIX INDICES

Matrices can be represented conveniently in MATLAB by writing individual rows, separated by semicolons (this use of the semicolon is different of the use of the same character at the end of a line). For example, typing

```
>> A = [1 2 3;4 5 6;7 8 9];
```

creates a 3×3 matrix in MATLAB in which the first row has elements 1, 2, 3, the second row has elements 4, 5, 6, and so on (note the use of the brackets [ ]). If, instead, we write the same line without the final semicolon, we would get

```
>> A = [1 2 3;4 5 6;7 8 9]
A =
     1   2   3
     4   5   6
     7   8   9
```

One of MATLAB's most powerful index operators is the `colon` operator ":". It is used in a number of different forms. For example, writing

```
>> 1:5
```

produces the row of integers

```
1 2 3 4 5
```

If we wanted to create an $1 \times N$ matrix, X, containing the sequence $1, 2, \ldots, N$ for an integer value of N, we would write

```
>> X = 1:N;
```

Alternatively, we could use the fact that `a:b` is the same as `colon(a,b)` and write

```
>> X = colon(1,N);
```

Nonuniform spacing can be obtained by specifying an increment. For example,

```
>> 100:-10:50
```

produces the sequence

```
100 90 80 70 60 50
```

The same sequence would be produced by the statement `colon(100,-10,50)`. When used in matrix indexing, the colon operator selects parts of a matrix. For example, the following statement extracts the third column of A:

```
>> C3 = A(:,3)
C3 =
     3
     6
     9
```

Similarly, we extract the second row as follows:

```
>> R2 = A(2,:)
R2 =
     4  5 6
```

Finally, the following statement extracts the top two rows:

```
>> T2 = A(1:2,1:3)
T2 =
     1   2   3
     4   5   6
```

Note in the first example how using the colon in `A(:,3)` signified all the rows of the array. In other words, `A(:,3)` is the same as `A(1:3,3)`.

The colon notation can be used also in combination with other operators. For example,

```
>> S = sum(B(1:k,j));
```

computes the sum of the first k elements of the jth column of B and stores it as s.

The colon operator is an important tool used in a number of different applications throughout the rest of the book.

GENERATING MATRICES

MATLAB provides four functions that generate the following basic matrices:

- `zeros(M,N)` generates an $M \times N$ matrix of zeros.
- `ones(M,N)` generates an $M \times N$ matrix of ones.
- `rand(M,N)` generates an $M \times N$ matrix whose entries are uniformly-distributed random numbers in the interval $[0.0, 1.0]$.
- `randn(M,N)` generates an $M \times N$ matrix whose numbers are normally-distributed (i.e., Gaussian) numbers with mean 0 and variance 1.

***[Margin Note: See Chapter 5 for additional discussions on random numbers].

For example:

```
>> A = 5*ones(3,3)
A =
     5   5   5
     5   5   5
     5   5   5
>> B = rand(2,4)
B =
     0.2311   0.4860   0.7621   0.0185
     0.6068   0.8913   0.4565   0.8214
```

MATRIX CONCATENATION

Matrix concatenation is the process of joining small matrices to create larger matrices. The concatenation operator is the pair of square brackets, [ ]. Earlier in this section, when we wrote

```
>> A = [1 2 3;4 5 6;7 8 9]
```

all we did was concatenate three rows (separated by semicolons) to create a 3×3 matrix. Note that individual elements are separated by a space. Separating the elements by

commas would yield the same result. Note also the order in which the elements were concatenated. The group of three elements before the first semicolon formed the first row, the next group formed the second row, and so on. Clearly the number of elements between semicolons have to be equal. This concept is applicable also when the elements themselves are matrices. For example, consider the 2×2 matrix

```
>> B=[1 2;3 4];
```

The statement

```
>> C = [B B;B+4 B-1]
```

in which $B + 4$ and $B - 1$ indicate adding 4 and subtracting 1 from all elements of B, respectively, yields the result

```
C =

     1     2     1     2

     3     4     3     4

     5     6     0     1

     7     8     2     3
```

Note how matrix B was duplicated twice in the top half of C, corresponding to the two occurrences of B before the first semicolon in the statement `C = [B B;B+4 B-1].`

DELETING ROWS AND COLUMNS

Rows and/or columns in a matrix can be deleted by using the "empty" concatenation operator, which consists of the two brackets with no content between them (blank spaces are acceptable). For example, to delete the second column of the matrix C just discussed, we write

```
>> C(:,2)=[]
C =

     1     1     2

     3     3     4

     5     0     1

     7     2     3
```

The 2×2 center of the original matrix C can be extracted by deleting the first and last rows and first and last columns of the array. This is accomplished in two steps:

```
>> C(1:3:4,:)=[]
```

which yields

```
C =

     3    4    3    4

     5    6    0    1
```

Then, the statement

```
>> C(:,1:3:4)=[]
```

gives the desired result:

```
C =

     4    3

     6    0
```

OBTAINING INFORMATION ABOUT MATRIX PROPERTIES

When working with matrices, it often is necessary to obtain information such as size, and range of values of a given array. In this section we introduce several functions that are useful for this purpose. We assume a general matrix A of size $M \times N$. For numerical examples we use the matrix `B = [5 1 2;3 9 4;7 6 8]`

$$B = \begin{bmatrix} 5 & 1 & 2 \\ 3 & 9 & 4 \\ 7 & 6 & 8 \end{bmatrix}$$

Functions `max` and `min`

If A is a vector (i.e., either $M = 1$ or $N = 1$), `max(A)` returns the value of the largest element in the vector. If A is a matrix, `max(A)` returns a *row* vector containing the maximum element from each column. This means that, if A is a matrix, we use

```
>> max(max(A))
```

to obtain the value of the largest element in A. For example,

```
>> max(B)
```

```
ans =

     7   9   8

>> max(max(B))

ans =

     9
```

The function `[V,R]=max(A)` returns in V the maximum value in each column of A, and in R the corresponding row indices of those values. For example,

```
>> [V,R] = max(B)

V =

     7   9   8

R =

     3   2   3
```

The `min` function behaves in the same way, except that the function looks for minimum values. Use the same matrices as in the preceding examples,

```
>> min(B)

ans =

     3   1   2

>> min(min(B))

ans =

     1

>> [V,R] = min(B)

V =

     3   1   2

R =

     2   1   1
```

Functions `size`, `length`, **and** `ndims`

If A is an $M \times N$ matrix, the function `size(A)` returns a row vector with components

M and N. If D is a row or column vector, `length(D)` returns the number of elements in the vector. Note that `length(D)` is the same as `max(size(D))`. Finally, the function `ndims(A)` returns the number of dimension of the array A (i.e., 2 for a matrix and 1 for a vector). In the following examples we use the matrix B defined previously:

```
>> size(B)
ans =
     3   3
>> S = size(B)
S =
     3   3
>> D = B(2,:)
D =
     3   9   4
>> size(D)
ans =
     1   3
>> length(D)
ans =
     3
>> ndims(B)
ans =
     2
```

Function `whos`

Another way to obtain information about matrix properties is to use the `whos` function. For example, applying this function to matrix B in the preceding paragraph gives

```
>> whos B
Name    Size    Bytes    Class
B         3x3      72         double array
```

```
Grand total is 9 elements using 72 bytes
```

CONCLUSION

The previous review, although brief, is a good foundation for learning the basics of how to manipulate matrices (images) in MATLAB. The on-line help environment in MATLAB is excellent, and gives detailed syntax for all the functions in the package. It is recommended that the reader become familiar with the various ways to search MATLAB's on-line help database. This is time well spent.