MATLAB Basics

9.20.2019

Mengkai Ma

Use Less For Loop

Why?

- Indeed more intuitive, straightforward and easy to start, but……
- For loop is single-threading (less efficient)
- Not elegant (plenty of indentations)

We are using MATLAB (matrix laboratory)

- Use matrix operations well
- Whenever encounter problems, try finding built-in functions first

Basic Array Operations

Array and matrix assignment

```
A = [1 2 3 4]
A = 1:4
A = [1, 2, 3, 4]
A = [[1, 2], [3, 4]] % concatenate
% A = [1, 2, 3, 4]

A = [1, 2; 3, 4]
A = [[1, 2]; [3, 4]]
% A = [[1, 2];
%       [3, 4]]
```

Basic Array Operations - Indexing

1-D Array

```
A = 2:10
% A     = [2 3 4 5 6 7 8 9 10]
% index =  1 2 3 4 5 6 7 8 9
A(3)
% ans = 4
A(2:4)
% ans = [3 4 5]
A([1 3 5])
% ans = [2 4 6]
A([1 1 3 3 5 5])
% ans = [2 2 4 4 6 6]
indices = find(rem(A, 2) == 0)
% indices = [1 3 5 7 9]
A(indices)
% ans = [2 4 6 8 10]
```

Basic Array Operations - Indexing

1-D Array

```
A = 2:10
% A     = [2 3 4 5 6 7 8 9 10]
% index =  1 2 3 4 5 6 7 8 9
A([3, 5, 8]) = []
% A = [2 3 5 7 8 10]
A(3:7) = []
% A = [2 3 9 10]
A(find(A>5)) = []
% A = [2 3 4]
A(A<6) = []
% A = [6 7 8 9 10]
```

Basic Matrix Operations - Indexing

2-D Matrix

```
A = [1 2 3 4;5 6 7 8;9 10 11 12]
% index | 1  2  3  4
%-------+------------
%   1   | 1  2  3  4
%   2   | 5  6  7  8
%   3   | 9 10 11 12
A(2,2)
% ans = 6
A(8)
% ans = 7
A(2,1:3)
% ans = 5 6 7
A(3,:)
% ans = 9 10 11 12
```

Basic Matrix Operations - Indexing

2-D Matrix

```
B =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4

>> B(2:3,:)=[]
B =
     1     2     3     4
     1     2     3     4

% Try it yourself
B(find(B>2)) % index array
B(B>2)       % logical array
```

Basic Matrix Operations - Arithmetic Operators

2-D Matrix

```
B =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4

B.^2 =
     1     4     9    16
     1     4     9    16
     1     4     9    16
     1     4     9    16
```

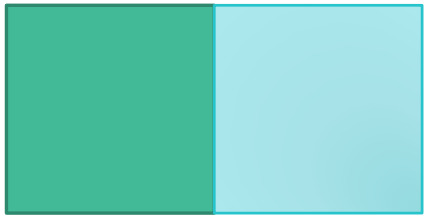
Basic Matrix Operations - Reshaping

```
B =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> reshape(B, 2, 6)

ans =
     1     9     6     3    11     8
     5     2    10     7     4    12
```

Basic Matrix Operations - Concatenation

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

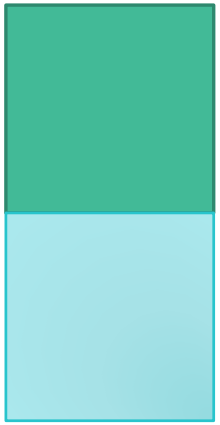
```
>> [A, B]

ans =
     1     1     1     1     2     3
     2     2     2     1     2     3
     3     3     3     1     2     3
```



Basic Matrix Operations - Concatenation

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

```
>> [A; B]

ans =
     1     1     1
     2     2     2
     3     3     3
     1     2     3
     1     2     3
     1     2     3
```



Basic Matrix Operations - Concatenation

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

```
>> cat(3, A, B)

ans(:,:,1) =
     1     1     1
     2     2     2
     3     3     3

ans(:,:,2) =
     1     2     3
     1     2     3
     1     2     3
```

```
[A, B] == cat(2, A, B)

[A; B] == cat(1, A, B)
```

Basic Matrix Operations - Other functions

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

`sum(A)`	$\begin{bmatrix} 6 & 6 & 6 \end{bmatrix}$
`sum(A, 2)`	$\begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix}$
`sum(A, 'all')`	18
`sum(A(:))`	18

Similarly, `mean(A)`

Basic Matrix Operations

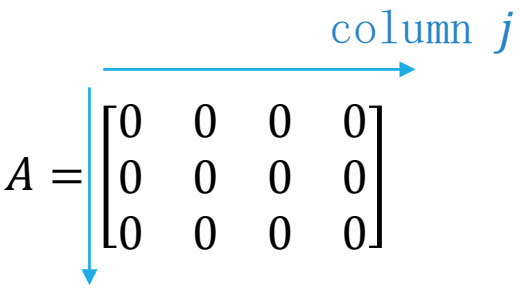
$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

No more inefficient for loops!

```
s = 0;
for i = 1:4
    for j = 1:4
        s = s + A(i, j);
    end
end
```

Generate and Initialize New Matrices

Matrix A: 3x4, all 0




```
for i = 1 : 3
    for j = 1 : 4
        A(i, j) = 0;
    end
end
```

```
% all 0
A = zeros(3, 4);
% all 1
A = zeros(3, 4) + 1;
A = ones(3, 4);
% all 3
A = ones(3, 4) * 3;
A = 3 - zeros(3, 4);
```

Generate and Initialize New Matrices

Matrix B: 3x4, random integers in range $[1,10]$

$$B = \begin{bmatrix} 1 & 9 & 7 & 2 \\ 4 & 10 & 3 & 7 \\ 6 & 5 & 5 & 8 \end{bmatrix}$$

```
>> B = randi([1, 10], 3, 4)

B =

     8     7     2     4
     8     7     5     6
     3     2    10     3
```

Generate and Initialize New Matrices

Matrix C: 3x4

$$C = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
% memory preallocation
A = zeros(3, 4);
for i = 1 : 3
    for j = 1 : 4
        A(i, j) = j;
    end
end
```

```
% repeat a pattern
C = repmat([1 2 3 4], [3, 1])

% matrix multiplication trick
C = [1;1;1;1] * [1 2 3 4]
```

Matrix multiplication - Wikipedia

Generate and Initialize New Matrices

Matrix C: 3x4

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{bmatrix}$$

```
% memory preallocation
C = zeros(3, 4);
for i = 1 : 3
    for j = 1 : 4
        C(i, j) = i;
    end
end
```

```
% repeat a pattern
C = repmat([1;2;3], [1, 4])

% matrix multiplication trick
C = [1;2;3] * [1 1 1 1]
```

Generate and Initialize New Matrices

Matrix D: 4x4

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

```
D = zeros(4, 4);
for i = 1 : 4
    for j = 1 : 4
        D(i, j) = i + j;
    end
end
```

Generate and Initialize New Matrices

Matrix D: 4x4

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

```
[X, Y] = meshgrid(1:4, 1:4)
```

```
X =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4

Y =
     1     1     1     1
     2     2     2     2
     3     3     3     3
     4     4     4     4
```

Generate and Initialize New Matrices

Matrix D: 4x4

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

```
X =                                   Y =
     1     2     3     4                 1     1     1     1
     1     2     3     4                 2     2     2     2
     1     2     3     4                 3     3     3     3
     1     2     3     4                 4     4     4     4
```

```
X + Y =
     2     3     4     5
     3     4     5     6
     4     5     6     7
     5     6     7     8
```

Generate and Initialize New Matrices

Matrix F: 4x4

$$E = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

```
X =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4

Y =
     1     1     1     1
     2     2     2     2
     3     3     3     3
     4     4     4     4

X + Y =
     2     3     4     5
     3     4     5     6
     4     5     6     7
     5     6     7     8

rem(X + Y, 2) == 0
     1     0     1     0
     0     1     0     1
     1     0     1     0
     0     1     0     1
```

Generate and Initialize New Matrices

Matrix E: 4x4

$$E = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 2 & 3 & 4 \\ 3 & 3 & 3 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

```
X =                                   Y =
     1     2     3     4                 1     1     1     1
     1     2     3     4                 2     2     2     2
     1     2     3     4                 3     3     3     3
     1     2     3     4                 4     4     4     4
```

```
max(X, Y) =
     1     2     3     4
     2     2     3     4
     3     3     3     4
     4     4     4     4
```

Stand on Your Own

1. Use doc / help + function name (e.g. doc imread)
2. Google your question (e.g. matlab imwrite parameters)
3. Wikipedia (e.g. bilinear interpolation)
4. MATLAB Examples
5. Array vs. Matrix Operations

Optional: Block Processing

blockproc: apply a function to every blocks inside a matrix

```
B = randi(10, 4, 4)
foo = @(mat) mean(mat.data, 'all');
blockproc(B, [2, 2], foo)

B =
     3    10    10     4
     5     5     4     3
     1     5     2     5
    10     4     8     1
```

Function handle

```
function result = foo(mat)
result = mean(mat.data, 'all')
```

Optional: Block Processing

blockproc: apply a function to every blocks inside a matrix

```
B = randi(10, 4, 4)
foo = @(mat) mean(mat.data, 'all');
blockproc(B, [2, 2], foo)

B =
     3    10    10     4
     5     5     4     3
     1     5     2     5
    10     4     8     1

ans =
    5.7500    5.2500
    5.0000    4.0000
```

Optional: Block Processing

blockproc: apply a function to every blocks inside a matrix

```
B = randi(10, 4, 4)
foo = @(mat) mean(mat.data, 'all');
blockproc(B, [2, 2], foo)
```

```
B =
     3    10    10     4
     5     5     4     3
     1     5     2     5
    10     4     8     1

ans =
    5.7500    5.2500
    5.0000    4.0000
```

Optional: Block Processing

blockproc: apply a function to every blocks inside a matrix

```
B = randi(10, 4, 4)
foo = @(mat) mean(mat.data, 'all');
blockproc(B, [2, 2], foo)

B =
     3    10    10     4
     5     5     4     3
     1     5     2     5
    10     4     8     1

ans =
    5.7500    5.2500
    5.0000    4.0000
```

Optional: Block Processing

blockproc: apply a function to every blocks inside a matrix

```
B = randi(10, 4, 4)
foo = @(mat) mean(mat.data, 'all');
blockproc(B, [2, 2], foo)
```

```
B =
     3    10    10     4
     5     5     4     3
     1     5     2     5
    10     4     8     1

ans =
    5.7500    5.2500
    5.0000    4.0000
```