Matrix and Vector Operations

Spring 2019
**length** and **size** functions

- **length** - returns the number of elements in a vector
- **size** - returns the number of rows and columns in a vector or matrix.
**length and size functions**

- **length** - returns the number of elements in a vector
- **size** - returns the number of rows and columns in a vector or matrix.

```
>> vec = -2:2
vec =
   -2  -1   0   1   2
>> length(vec)
an =
   5
>> size(vec)
an =
   1   5
```
length and size functions - Matrix case

```
1
2     >> M = [1:4;5:8]'
3     M =
4           1   5
5           2   6
6           3   7
7           4   8
8     >> [r,c]=size(M)
9     r =
10          4
11     c =
12          2
13     >> length(M)
14     ans =
15           4
```

- **size** - returns the number of rows and columns
- **length** - returns the number of rows or columns (whichever is the largest one).
Matrix and Array Operations

- Matrix operations follow the rules of linear algebra.
- Array operations execute element by element operations on elements of vectors, matrices or multi-dimensional arrays.
- The period character (.) distinguishes array operations from matrix operations.

<table>
<thead>
<tr>
<th>Op</th>
<th>Purpose</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>A+B adds A and B</td>
</tr>
<tr>
<td>+</td>
<td>Unitary plus</td>
<td>+A returns A</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>A−B subtracts B from A</td>
</tr>
<tr>
<td>-</td>
<td>Unitary minus</td>
<td>−A negates A</td>
</tr>
<tr>
<td>*</td>
<td>product</td>
<td>A*B is the usual matrix product</td>
</tr>
<tr>
<td>.*</td>
<td>Elmt-wise multiplication</td>
<td>A.*B is elmt-by-elmt product of A and B</td>
</tr>
<tr>
<td>.^</td>
<td>Elmt-wise multiplication</td>
<td>A.^ B has elements A(i,j) raised to B(i,j)</td>
</tr>
<tr>
<td>./</td>
<td>Right array division</td>
<td>A./B has elements A(i,j)/B(i,j)</td>
</tr>
</tbody>
</table>
Matrix operations

Define >>> a=[1 2 3]; b=[3 4 5]; c=[2;4;5]; d=[0,1];

- We can add vectors of the same dimension

```
1            >> a+b
2            ans =
3                4   6   8
```
Matrix operations

Define $\gg a = [1 \ 2 \ 3]; \ b = [3 \ 4 \ 5]; \ c = [2; 4; 5]; \ d = [0; 1];$

- We can add vectors of the same dimension

```
1   >> a+b
2      ans =
3          4   6   8
```

- If the dimensions are not the same, we get dimension errors

```
1   >> a+c
2      Error using +
3      Matrix dimensions must agree.
4   >> a+d
5      Error using +
6      Matrix dimensions must agree.
```
Matrix operations

Define

\[
\begin{align*}
\text{a} &= [1 \ 2 \ 3] ; \\
\text{b} &= [3 \ 4 \ 5] ; \\
\text{c} &= [2 ; 4 ; 5] ;
\end{align*}
\]

- We can scale vectors

\[
\begin{array}{c}
\text{1} \\
\text{2} \\
\text{3}
\end{array}
\begin{array}{c}
\gg \quad -2 \times \text{a} \\
\text{ans} = \\
-2 \quad -4 \quad -6
\end{array}
\]

1. We can scale vectors.
Matrix operations

Define >> a=[1 2 3]; b=[3 4 5]; c=[2;4;5];

- We can scale vectors

```
1 >> -2*a
2 ans =
3    -2   -4   -6
```

- We can multiply vectors of appropriate dimensions

```
1 >> a*c
2 ans =
3 25
```
Element-wise/ Component-wise operations

Define >> a=[1 2 3]; b=[3 4 5]; c=[2;4;5];

- Component-wise multiplication on vectors of the same dimension

<table>
<thead>
<tr>
<th></th>
<th>&gt;&gt; a.*b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ans =</td>
</tr>
<tr>
<td>3</td>
<td>3 8 15</td>
</tr>
</tbody>
</table>
**Element-wise/ Component-wise operations**

Define

```
>> a=[1 2 3]; b=[3 4 5]; c=[2;4;5];
```

- Component-wise multiplication on vectors of the same dimension

```
1
>> a.*b
2
ans =
3
     3  8  15
```

- If the dimensions are not equal

```
1
>> a.*c
2
Error using .*
3
Matrix dimensions must agree.
```
Component-wise operations

Define >> \( a=[1\ 2\ 3] \); \( b=[3\ 4\ 5] \); \( c=[2;4;5] \);

- Square every component of a vector

\[
\begin{array}{c}
1 \\
2 \\
3 \\
\end{array}
\]
\[
\begin{array}{c}
\gg a.^{\text{\textasciitilde}2} \\
\text{ans} = \\
1\quad 4\quad 9 \\
\end{array}
\]
Component-wise operations

Define
\[
\begin{align*}
&>> \quad a = [1 \ 2 \ 3]; \quad b = [3 \ 4 \ 5]; \quad c = [2; 4; 5]; \\
&\textbf{Square every component of a vector} \\
&1 \quad >> \quad a.\hat{\text{2}} \\
&\quad \text{ans} = \\
&\quad 1 \quad 4 \quad 9
\end{align*}
\]

- \(a(i)^b(i)\)

\[
\begin{align*}
&1 \quad >> \quad a.\hat{\text{b}} \\
&\quad \text{ans} = \\
&\quad 1 \quad 16 \quad 243
\end{align*}
\]
Matrix square & Component-wise square

Define >> \( A = [1, 2; 3, 4] ; \ B = [0, 1; 1, 0] ; \)

- The square of a matrix i.e \( A^2 \).

\[
\begin{array}{ll}
1 & \gg A^2 \\
2 & ans = \\
3 & 7 & 10 \\
4 & 15 & 22 \\
\end{array}
\]

- Component-wise square of \( A \) i.e \( A(i,j)^2 \)

\[
\begin{array}{ll}
1 & \gg A. ^2 \\
2 & ans = \\
3 & 1 & 4 \\
4 & 9 & 16 \\
\end{array}
\]
Component-wise operations

Define $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$; $B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$; $A(i,j) = B(i,j)$

$$
\begin{array}{l}
1 \\
2 \\
3 \\
4 \\
\end{array}
\begin{array}{l}
\text{ans} = \\
\text{ans} = \\
\text{ans} = \\
\text{ans} = \\
\end{array}
\begin{array}{ll}
1 & 2 \\
3 & 1 \\
\end{array}
$$

$$
\text{A} \cdot \text{B} = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}
$$
Functions acting on matrices or vectors

- All actions are automatically done component-wise, e.g., given a matrix with random entries on \([0, 1]\)

```
>> M = rand(4)
M =
0.8722 0.9585 0.0591 0.4272
0.0522 0.7900 0.7409 0.1687
0.2197 0.4519 0.5068 0.7517
0.4596 0.3334 0.1999 0.3684
```

- We can round off each entry to create a random binary matrix:

```
>> round(M)
an =
1 1 0 0
0 1 1 0
0 0 1 1
0 0 0 0
```
Functions acting on Matrices or vectors

- Define $M$ as

```matlab
>> M = rand(4)
M =
0.8722 0.9585 0.0591 0.4272
0.0522 0.7900 0.7409 0.1687
0.2197 0.4519 0.5068 0.7517
0.4596 0.3334 0.1999 0.3684
```

- Compute $e^M$

```matlab
>> exp(M)
ans =
2.3923 2.6079 1.0609 1.5329
1.0536 2.2035 2.0978 1.1838
1.2457 1.5713 1.6600 2.1206
1.5835 1.3957 1.2213 1.4453
```
Functions acting on Matrices or vectors: Exercise

- Create a vector of alternating 1s and 0s

```matlab
vec = 1:10
mod(vec,2)
```

$\begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{bmatrix}$
Create a vector of alternating 1s and 0s

```
>> vec = 1:10
vec =  
   1   2   3   4   5   6   7   8   9  10
>> mod(vec,2)
an =  
   1   0   1   0   1   0   1   0   1   0
```
Functions acting on Matrices or vectors: Exercise

- What about random bits?

```matlab
vec = round(rand(1,10)*10)
```

```matlab
vec =
8 9 1 9 6 1 3 5 10 10
```

```matlab
mod(vec,2)
```

```matlab
ans =
0 1 1 1 0 1 1 1 0 0
```
Functions acting on Matrices or vectors: Exercise

What about random bits?

```
1  >> vec = round(rand(1,10)*10)  
2  vec = 
3  8  9  1  9  6  1  3  5  10  10  
4  >> mod(vec,2)  
5  ans = 
6  0  1  1  1  0  1  1  1  1  0  0  
```
The MATLAB functions `reshape`, `fliplr`, `flipud` and `rot90` can change the dimensions or configuration of matrices.

- e.g define $M$ - a matrix of 12 random integers on $[0, 100]$.

```
>> M=randi(100,3,4)
M =
     95    63    73     2
     2    54    10    30
     83    66    88    18
```

- Reshape to $2 \times 6$ (`reshape` - iterates through $M$ column-wise)

```
>> reshape(M,2,6)
ans =
    95    83    54    73    88    30
     2    63    66    10     2    18
```
**fliplr**

- e.g define $M$ - a matrix of 12 random integers on [0, 100].

```
>> M=randi(100,3,4)
M =
    95  63  73   2
    2  54  10  30
    83  66  88  18
```

- *fliplr* - “flips” the matrix from left to right

```
>> fliplr(M)
ans =
     2   73   63   95
    30   10   54    2
    18   88   66   83
```
flipup

- e.g define $M$ - a matrix of 12 random integers on $[0, 100]$.  

\begin{verbatim}
>> M=randi([0,100],3,4)
M =
95  63  73   2
2   54  10  30
83  66  88  18
\end{verbatim}

- \texttt{flipup} - “flips” the matrix from up to down

\begin{verbatim}
>> flipud(M)
ans =
83  66  88  18
2   54  10  30
95  63  73   2
\end{verbatim}
rot90

- e.g define $M$ - a matrix of 12 random integers on $[0, 100]$.

```
>> M=randi(100,3,4)
M =
    95   63   73    2
    2   54   10   30
    83   66   88   18
```

- rot90 - counterclockwise rotation of 90 degrees

```
>> rot90 (M)
ans =
    2   30   18
   73   10   88
   63   54   66
   95    2   83
```
repmat

- e.g. define $M$ - a matrix of 12 random integers on $[0, 100]$.

```matlab
>> M = randi(100,3,4)
M =
   95   63   73    2
   2   54   10   30
  83   66   88   18
```

- `repmat` will duplicate a matrix, e.g.

```matlab
>> repmat(M,2,2)
ans =
   95   63   73    2   95   63   73    2
   2   54   10   30   2   54   10   30
  83   66   88   18  83   66   88   18
```
Three-dimensional matrices

Think about printing 2D matrices on sheets of paper and stacking them.

- Create $M$ with entries 1 – 20 as

  
  \[
  \begin{bmatrix}
  1 & 5 & 9 & 13 & 17 \\
  2 & 6 & 10 & 14 & 18 \\
  3 & 7 & 11 & 15 & 19 \\
  4 & 8 & 12 & 16 & 20 \\
  \end{bmatrix}
  \]

- Add a second matrix on top on $M$ as

  \[
  \begin{bmatrix}
  \end{bmatrix}
  \]

- Check the size of $M$

  \[
  \begin{bmatrix}
  \end{bmatrix}
  \]
Three-dimensional matrices

- Check the contents of the matrix \( M \):

```
1   >> M
2
M(:, :, 1) =
3     1   5   9   13  17
4     2   6  10  14  18
5     3   7  11  15  19
6     4   8  12  16  20
7
M(:, :, 2) =
8     17  13   9   5   1
9     18  14  10   6   2
10    19  15  11   7   3
11    20  16  12   8   4
```
3D matrices and images – RGB color model

RGB model

- Create an array of colors by combining various ratios of red, green and blue.
- The RGB values are integers in [0, 255].
- We can store the R, B, G values in the form of a 3D matrix where each layer corresponds to a color channel.