Lec 06: Arrays in MATLAB

Basics of Arrays in MATLAB

Introduction to Arrays

Vectors and matrices are often collectively called **arrays**.

Notation

- $\mathbb{R}^m$ (or $\mathbb{C}^m$): the set of all real (or complex) **column vectors** with m elements.
- $\mathbb{R}^{m\times n}$ (or $\mathbb{C}^{m\times n}$): the set of all real (or complex) $m \times n$ matrices.
- If $\mathbf{v} \in \mathbb{R}^m$ with $\mathbf{v} = (v_1, v_2, \ldots, v_m)^{\mathrm{T}}$, then for $1 \leq i \leq m$, $v_i \in \mathbb{R}$ is called the ith *element* or the ith *index* of $\mathbf{v}$.
- If $A \in \mathbb{R}^{m\times n}$ with $A = (a_{i,j})$, then for $1 \leq i \leq m$ and $1 \leq j \leq n$, $a_{i,j} \in \mathbb{R}$ is the element in the ith row and jth column of A.

Creating Arrays

- A *row vector* is created by

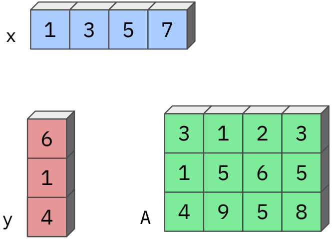
```
x = [1 3 5 7];
x = [1,3,5,7];
```

- A *column vector* is created by

```
y = [6; 1; 4];
y = [6 1 4].';
```

- A matrix is formed by

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



The MATLAB expression `x.'` means $\mathbf{x}^{\mathrm{T}}$ while `x'` means $\mathbf{x}^{\mathrm{H}} = (\mathbf{x}^*)^{\mathrm{T}}$.

Shape of Arrays

- To find the number of elements of a vector:

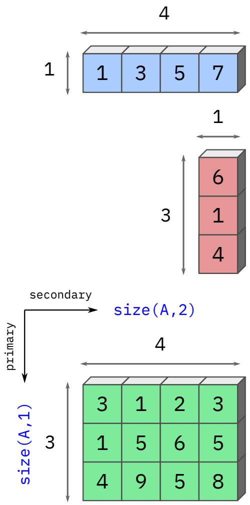
```
length(x)
length(y)
```

- To find the number of rows/columns of an array:

```
size(A,1) % # of rows
size(A,2) % # of cols
size(A)   % both
```

- To find the total number of elements of an array:

```
numel(A)
```



Shape of Arrays (Notes)

- For a matrix `A`, `length(A)` yields the larger of the two dimensions.
- The result of `size(A)` can be stored in two different ways:

```
szA = size(A)
[m, n] = size(A)
```

Q. How are they different?

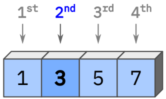
- All of the following generate *empty arrays*.

```
[]
[1:0]
[1:0].'
```

Q. What are their *sizes*? What are their `numel` values?

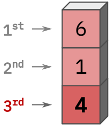
- To access the ith element of a vector:

```
x(2)
y(3)
```



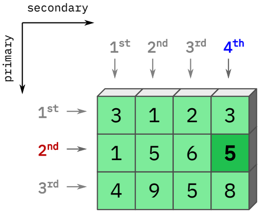
- To access the (i, j)-element of a matrix:

```
A(2,4)
```



- To assign values to a specific element:

```
x(2) = 2
A(2,4) = 0
```



- Indices start at **1** in MATLAB, not at 0!

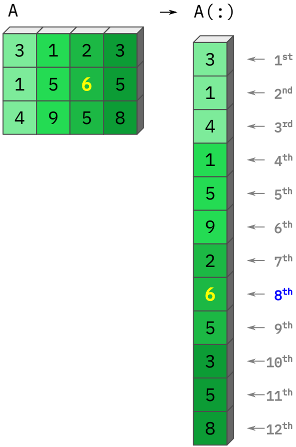
Linear Indexation and Straightening of Matrix

- MATLAB uses *column-major* layout by default, meaning that the elements of the columns are contiguous in memory.
- Consequently, one can get/set an element of a matrix using a single index.

```
A(8)
```

- An array can be put into a column vector using

```
A(:)
```



Array Operations

Two Kinds of Transpose

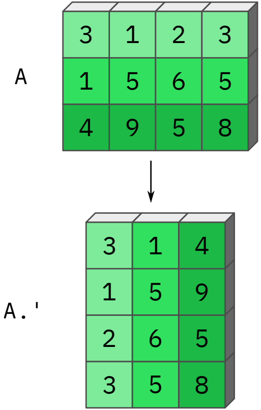
- The transpose of an array: A^{T}

```
A.'
```

- The conjugate transpose of an array: $A^{\mathrm{H}} = A^{*} = \overline{A}^{\mathrm{T}}$

```
A'
```

- If $A \in \mathbb{R}^{m\times n}$, $A^{\mathrm{H}} = A^{\mathrm{T}}$. So, if `A` is a real array, `A.'` and `A'` are equivalent.



Standard Arithmetic Operation

Standard arithmetic operations seen in linear algebra are executed using the familiar symbols.

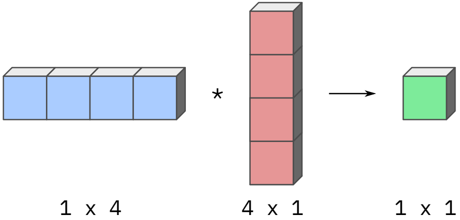
- Let `A,B` $\in \mathbb{R}^{m\times n}$ and `c` $\in \mathbb{R}$.
  - `A` $\pm$ `B`: elementwise addition/subtraction $(A \pm B)$
  - `A` $\pm$ `c`: *shifting* all elements of `A` by $\pm$ `c` $(A \pm c)$
- Let `A` $\in \mathbb{R}^{m\times p}$, `B` $\in \mathbb{R}^{p\times n}$, and `c` $\in \mathbb{R}$.
  - `A*B`: the $m \times n$ matrix obtained by the *linear algebraic* multiplication (AB)
  - `c*A`: scalar multiple of `A` (cA)
- Let `A` $\in \mathbb{R}^{m\times m}$ and `n` $\in \mathbb{N}$.
  - `A^n`: the `n`-th power of `A`; the same as `A*A*` $\cdots$ `*A` (`n` times) (A^n)

Standard Arithmetic Operation – Inner Products

Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$ be column vectors. The *inner product* of $\mathbf{x}$ and $\mathbf{y}$ is calculated by

$$\mathbf{x}^{\mathrm{T}}\mathbf{y} = x_1y_1 + x_2y_2 + \cdots + x_my_m = \sum_{j=1}^{m} x_jy_j \in \mathbb{R}.$$

In MATLAB, simply type `x'*y`.

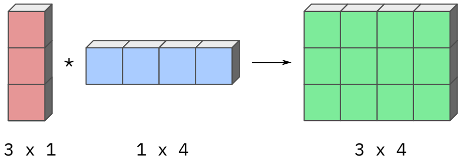


Standard Arithmetic Operation – Outer Products

Let $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$ be column vectors. The *outer product* of $\mathbf{x}$ and $\mathbf{y}$ is calculated by

$$\mathbf{x}\mathbf{y}^{\mathrm{T}} = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_my_1 & x_my_2 & \cdots & x_my_n \end{bmatrix} \in \mathbb{R}^{m\times n}.$$

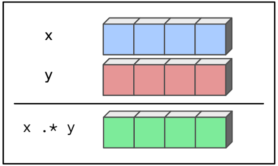
In MATLAB, simply type `x*y'`.



Elementwise Multiplication ($.*$)

- To multiply entries of two arrays of same size, element by element:

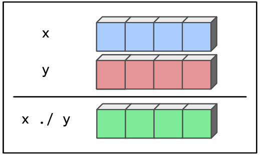
```
x .* y
```



Elementwise Division (./)

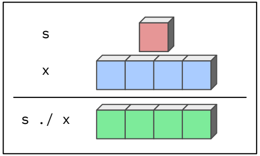
- To divide entries of an array by corresponding entries of another same-sized array:

```
x ./ y
```



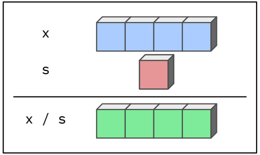
- To divide a number by multiple numbers (specified by entries of an array):

```
s ./ y
```



- To divide all entries of an array by a common number:

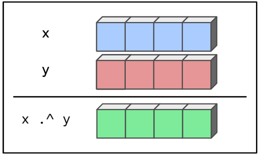
```
x / s
```



Elementwise Exponentiation (.^)

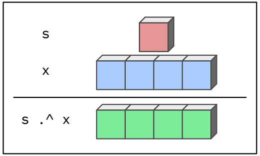
- To raise all entries of an array to (different) powers:

```
x .^ y
```



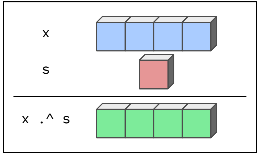
- To raise a number to multiple powers (specified by entries of an array):

```
s .^ x
```



- To raise all entries of an array to a common power:

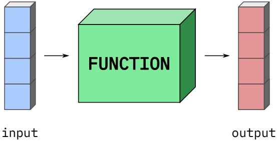
```
x .^ y
```



Mathematical Functions

- Built-in mathematical functions accept array inputs and return arrays of function evaluation, *e.g.*,

```
sqrt(A)
sin(A)
mod(A)
...
```



Array Constructors

Colon Operator

Suppose `a < b`.

- To create an arithmetic progression from `a` to `b` (increment by 1):

```
a:b
```

  The result is a row vector `[a, a+1, a+2, ..., a+m]`, where

$$\texttt{m} = \lfloor \texttt{b-a} \rfloor.$$

- To create an arithmetic progression from `a` to `b` with steps of size `d > 0`:

```
a:d:b
```

  The result is a row vector `[a, a+d, a+2*d, ..., a+m*d]`, where

$$\texttt{m} = \lfloor \texttt{(b-a)/d} \rfloor.$$

LINSPACE and LOG SPACE

- To create a row vector of `n` numbers evenly spaced between `a` and `b`:

```
linspace(a, b, n)
```

The result is `[a, a+d, a+2*d, ..., b]`, where

```
d = (b-a)/(n-1).
```

- To create a row vector of `n` numbers that are logarithmically evenly spaced between 10^a and 10^b:

```
logspace(a, b, n)
```

The result is $[10^a, 10^{a+d}, 10^{a+2d}, \ldots, 10^b]$, where

```
d = (b-a)/(n-1).
```

ZEROS, ONES, and EYE

- To create an $(m \times n)$ zero matrix:

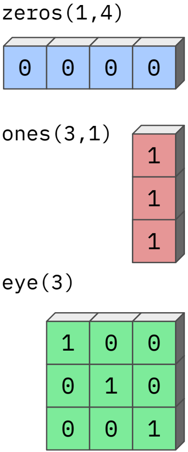
```
zeros(m, n)
```

- To create an $(m \times n)$ matrix all whose entries are one:

```
ones(m, n)
```

- To create the $(m \times m)$ identity matrix:

```
eye(m)
```



Random Arrays

Each of the following generates an $(m \times n)$ array of random numbers:

- `rand(m,n)`: uniform random numbers in $(0, 1)$
- `randi(k,m,n)`: uniform random integers in $[1, k]$
- `randn(m,n)`: Gaussian random numbers with mean 0 and standard deviation 1

Random Arrays (Application)

To generate an $(m \times n)$ array of

- uniform random numbers in (a, b):

```
a + (b - a)*rand(m, n)
```

- uniform random integers in $[k_1, k_2]$:

```
randi([k1, k2], m, n)
```

- Gaussian random numbers with mean μ and standard deviation σ:

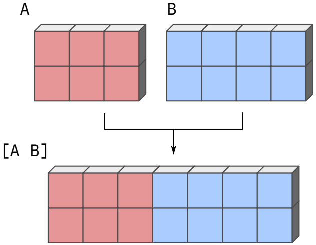
```
mu + sig*randn(m, n)
```

Building Arrays Out Of Arrays

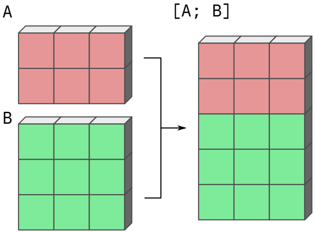
Concatenation

If two arrays `A` and `B` have *comparable* sizes, we can concatenate them.

- horizontally by `[A B]`

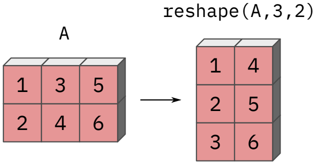


- vertically by `[A; B]`

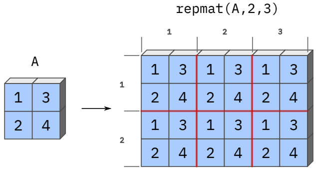


RESHAPE and REPMAT

- `reshape(A, m, n)` reshapes the array `A` into an $m \times n$ matrix whose elements are taken *columnwise* from `A`.



- `repmat(A, m, n)` replicates the array `A`, m times vertically and n times horizontally.



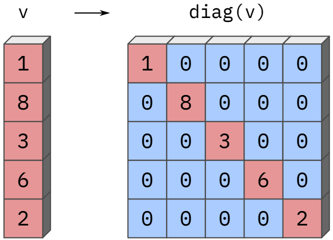
`FLIP`

- Type `help flip` on the Command Window and learn about `flip` function.
- Do the same with its two variants, `flipud` and `fliplr`

Creating Diagonal Matrices

- To create a diagonal matrix

$$\begin{bmatrix} v_1 & 0 & 0 & \cdots & 0 \\ 0 & v_2 & 0 & \cdots & 0 \\ 0 & 0 & v_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & v_n \end{bmatrix}:$$




```
diag(v)
```

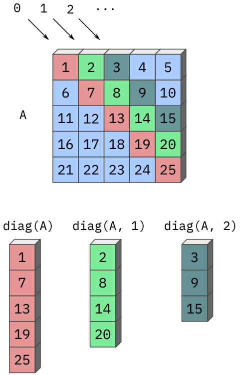
Note.

- `diag(v,k)` puts the elements of `v` on the `k`-th super-diagonal.
- `diag(v,-k)` puts the elements of `v` on the `k`-th sub-diagonal.

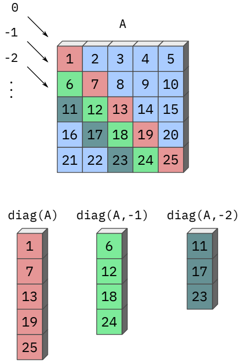
Extracting Diagonal Elements

Use `diag(A,k)` to extract the `k`-th diagonal of `A`.[1]

- `k > 0` for super-diagonals:



- `k < 0` for sub-diagonals:



[1] diag(A) short for diag(A,0) .

Slicing Arrays

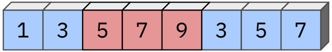
Using Vectors as Indices

To get/set multiple elements of an array at once, use vector indices.

- To grab 3rd, 4th, and 5th elements of `x`:

```
x(3:5)   % or x([3 4 5])
```

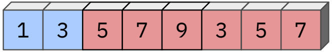
x(3:5)



- To grab 3rd to 8th elements of `x`:

```
x(3:8)
x(3:end)
```

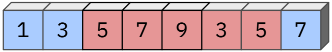
x(3:8) or x(3:end)



- To grab 3rd to 7th elements of `x`:

```
x(3:7)
x(3:end-1)
```

x(3:7) or x(3:end-1)



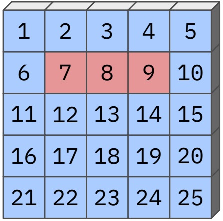
Using Vectors as Indices – Example

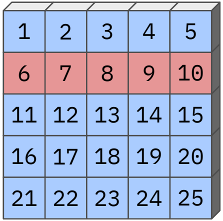
- To extract 2nd, 3rd, and 4th columns of the 2nd row of `A`:

```
A(2,2:4)   % or A(2,[2 3 4])
```

- To extract the entire 2nd row of `A`:

```
A(2,1:5)
A(2,1:end)
A(2,:)
```

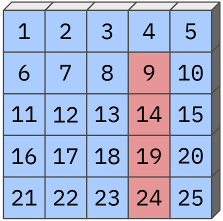




Using Vectors as Indices – Example

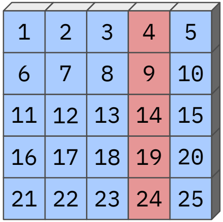
- To extract 2nd through 5th elements of the 4th column of `A`:

```
A([2 3 4 5],4)
A(2:5,4)
A(2:end,4)
```



- To extract the entire 4th column of `A`:

```
A(1:5,4)
A(1:end,4)
A(:,4)
```



Using Vectors as Indices – Example

- To grab the *interior block* of `A`:

```
A(2:4,2:4)
A(2:end-1,2:end-1)
```

- To extract every other elements on every other rows as shown:

```
A(1:2:5,1:2:5)
A(1:2:end,1:2:end)
```