Creating and Using Arrays

Tae Eun Kim, Ph.D.

Monday, January 14, 2019

3.1. Creating and Using Arrays

Introduction to Arrays

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

Definition (Spaces of Column Vectors and Matrices)

Let $\mathbb{R}^m$ (or $\mathbb{C}^m$) denote the set of all real (or complex) column vectors with m elements. Let $\mathbb{R}^{m\times n}$ (or $\mathbb{C}^{m\times n}$) denote the set of all real (or complex) $m \times n$ matrices.

Notation

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 $\mathbf{v}$.

Similarly, 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 Vectors

- A *row vector* is created by
  » `x = [2 3 -6];`
  or
  » `x = [2, 3, -6];`
- A *column vector* is created by
  » `w = [2; 3; -6];`
- Alternately, one can *transpose* a row vector to obtain a column vector:
  » `w = [2 3 -6].';`
- The MATLAB expression "`x.'`" means $\mathbf{x}^{\mathrm{T}}$ while "`x'`" means $\mathbf{x}^{\mathrm{H}} = (\mathbf{x}^*)^{\mathrm{T}}$.

Creating Matrices

- A matrix is formed by
  `» A = [1 2 3; 4 5 6; 7 8 9];`
  or
  `» A = [1, 2, 3; 4, 5, 6; 7, 8, 9];`

Caution

You can form A by typing

```
» A = [1 2 3
4 5 6
7 8 9];
```

using ⟨ENTER⟩, but it is not recommended when working in the **Command Window**.

Accessing Elements of Arrays

- To access the i^{th} element of a vector $\mathbf{x}$:

  ```
  » x(i)
  ```

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

  ```
  » A(i,j)
  ```

- To assign values to a specific element:

  ```
  » x(i) = 7
  ```

  or

  ```
  » A(i,j) = -sqrt(5)
  ```

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

Question

Let `A` be a 3 by 3 matrix in MATLAB. What is the value of `A(4)`?

Operations Involving Arrays

- **Addition/subtraction:** "$+/-$"
  - a matrix and a matrix: $\mathbb{R}^{m\times n} + \mathbb{R}^{m\times n} = \mathbb{R}^{m\times n}$
  - a matrix and a scalar: $c + \mathbb{R}^{m\times n} = \mathbb{R}^{m\times n}$

- **Multiplication:** "$\star$"
  - a matrix by a matrix: $\mathbb{R}^{m\times n} \times \mathbb{R}^{n\times p} = \mathbb{R}^{m\times p}$
  - a matrix by a scalar: $c \cdot \mathbb{R}^{m\times n} = \mathbb{R}^{m\times n}$

- **Inner and outer products of vectors:** For a column vector $\mathtt{x}$,
  - inner product ($\mathtt{x'} \star \mathtt{x}$): $\mathbb{R}^{1\times m} \times \mathbb{R}^{m\times 1} = \mathbb{R}$
  - outer product ($\mathtt{x} \star \mathtt{x'}$): $\mathbb{R}^{m\times 1} \times \mathbb{R}^{1\times m} = \mathbb{R}^{m\times m}$

More Operations Involving Arrays

- **Elementwise operations:** "`.*`", "`./`", "`.^`"

  Example: For $\mathtt{v} = (v_1, v_2, \ldots, v_n)$ and $\mathtt{w} = (w_1, w_2, \ldots, w_n)$,

  $$\mathtt{v.*w} = (v_1 w_1, v_2 w_2, \ldots, v_n w_n).$$

- **Square matrix to a positive integral power:** "`^`"

  Example: For an m by m matrix `A`, the MATLAB commands

  `A^5` and `A*A*A*A*A`

  yield the same result.

- **Inversion of a square matrix**: "`inv`"

  Example: For an m by m matrix `A`,

  `inv(A)` and `A^(-1)`

  produce the same answer.

Arithmetic Progressions

Colon (:) operator

- `a:b` creates `[a,a+1,a+2,...,a+m]` where `m=fix(b-a)`.
- `a:d:b` creates `[a,a+d,a+2*d,...,a+m*d]` where `m=fix((b-a)/d)`.

`linspace` command

- `linspace(a,b,n)` creates the arithmetic progression of `n` terms starting at `a` and ending at `b`.

Note: The colon operator is more appropriate when the difference between successive elements is known; `linspace` is more suitable when the number of elements is known.

Do this: Compare `linspace(0,1,10)` and `linspace(0,1,11)`.

Arithmetic Progressions: Examples

Example: We can create an *periodic* arithmetic progressions using the colon operator and `mod` function. For instance, to create an array

$$(1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0)\,,$$

we simply type:

```
>> m = 5;
>> n = 15;
>> mod([1:n], m)
```

Exercise

Create the following vectors using ONE MATLAB statement.

- $\mathbf{v} = (1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0)$
- $\mathbf{w} = (1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)$

Geometric and Other Progressions

Elementwise operations such as "`.*`", "`./`", and "`.^`" are useful in generating progressions other than arthmetic ones. By way of examples:

Exercise

Create

- $\mathbf{v} = (1, 2, 4, 8, \ldots, 1024)$
- $\mathbf{w} = (1, 4, 9, 16, \ldots, 100)$

using ONE MatLab statement.

Answer:

```
>> v = 2.^[0:10]
>> w = [1:10].^2
```

Alternately,

```
>> v = 2.^linspace(0, 10, 11)
>> w = linspace(1, 10, 10).^2
```

Geometric and Other Progressions (cont'd)

When mathematical functions such as `sin`, `sind`, `log`, `exp` are applied to arrays, they perform elementwise operations.

Exercise

Create the following vectors using ONE MATLAB statement.

- $\mathbf{v} = (\sin 0°, \sin 30°, \sin 60°, \ldots, \sin 180°)$
- $\mathbf{w} = (\mathrm{e}^1, \mathrm{e}^4, \mathrm{e}^9, \ldots, \mathrm{e}^{64})$

Answer:

```
>> v = sind(0:30:180)
>> w = exp([1:8].^2)
```

Building an Array out of Arrays

`reshape` and `repmat` commands

- `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.
- If the arrays `A` and `B` have comparable sizes, then we can concatenate them:
  - vertically by `[A; B]`
  - horizontally by `[A B]`

Example: See what happens:

- `reshape(1:30, 6, 5)`
- `repmat([1 2; 3 4], 2, 3)`

DIY: Type `help flip` to learn about `flip` command. Do the same with `flipud` and `flilr`.

The Size of Vectors and Matrices

It is easy to determine the size of a vector or a matrix.

`length`, `size`, and `numel` commands

Let $\mathtt{v} \in \mathbb{R}^m$ and $\mathtt{A} \in \mathbb{R}^{m\times n}$. Then

- `length(v)` gives the number of elements of `v`, which is m.
- `size(A, 1)` gives the number of rows of `A`, which is m.
- `size(A, 2)` gives the number of colums of `A`, which is n.
- `size(A)` outputs a two-vector (m, n).
- `numel(A)` returns the total number of elements in `A`, which is mn.

Note: The result of `size(A)` can be stored in two different ways:

`szA = size(A)` or `[mA, nA] = size(A)`.

How are they different?

Example (Empty arrays): What are the sizes of `[]`, `[1:0]`, and `[1:0]'`? What are their `numel` values?

Creating an Entire Matrix in One Statement

The following commands come in handy in many situations.

`zeros`, `ones`, and `eye` commands

Let $m, n \in \mathbb{N}$. Then

- `zeros(m)` creates an $m \times m$ zero matrix; `zeros(m,n)` an $m \times n$ zero matrix.
- `ones` behaves similarly creating a matrix consisting of all ones.
- `eye(m)` creates an $m \times m$ identity matrix.

Notes:

- `zeros([m,n])` yields the same result as `zeros(m,n)`.
- `zeros(size(A))` creates a zero matrix of the same size as `A`.
- The same applies to `ones`.

Creating Matrices Using Diagonals

We can create a matrix matrix by specifying the diagonal elements; the diagonal entries of a certain matrix can be extracted as a vector as well.

`diag` command: insertion

Let $\mathtt{v} = (v_1, v_2, \ldots, v_n)$. Then the statement `D = diag(v)` creates the diagonal matrix

$$D = \begin{pmatrix} v_1 & 0 & 0 & \cdots & 0 \\ 0 & v_2 & 0 & \cdots & 0 \\ 0 & 0 & v_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & v_n \end{pmatrix}.$$

For $\mathtt{k} \in \mathbb{N}$,

- `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.

Creating Matrices Using Diagonals (cont'd)

We can create a matrix matrix by specifying the diagonal elements; the diagonal entries of a certain matrix can be extracted as a vector as well.

`diag`, `triu`, and `tril` commands: extraction

Given a (full) matrix $\mathtt{A} \in \mathbb{R}^{m\times m}$ and $-m < \mathtt{k} < m$,

- `diag(A,k)` extracts the `k`th diagonal of `A`.
- `triu(A,k)` extracts the upper triangular entries above and including the `k`th diagonal of `A`.
- `tril(A,k)` extracts the lower triangular entries below and including the `k`th diagonal of `A`.

Note:

- The commands also work for non-square matrices.
- To create a block diagonal structure, use `blkdiag`.

Indices as Vectors

We can access multiple elements of an array using vectors. Let `v` be a vector and `A` be a matrix.

- `v(i:j)` returns ith through jth entries.
- `A(i:j, k:l)` returns the intersection of rows form i to j and columns from k to l.
- `v(end)` outputs the last element of `v`, which is the same as `v(length(v))`.
- We may assign values to the elements specified by indicial-vectors.
- `A(:)` turns `A` into a column vector.

Random Vectors and Matrices

`rand`, `randi`, and `randn` commands

To generate an $m \times n$ matrix of:

- `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 Vectors and Matrices (cont'd)

- To generate uniform random numbers in (a, b), use
$$\mathtt{a} + (\mathtt{b} - \mathtt{a}) * \mathtt{rand}(\mathtt{m}, \mathtt{n}).$$
- To generate uniform random integers in $[k_1, k_2]$, use
$$\mathtt{randi}([\mathtt{k1}, \mathtt{k2}], \mathtt{m}, \mathtt{n}).$$
- To generate Gaussian random numbers with mean μ and standard deviation σ, use
$$\mathtt{mu} + \mathtt{sig} * \mathtt{randn}(\mathtt{m}, \mathtt{n}).$$

Question: What does MATLAB return when you type, say, `rand(5)`?

The find Function

In case we want to locate all the elements of a certain array satisfying simple conditions, we can use find command.

Basic Usage of find

Let v be an array of numbers. Then find(<condition>) returns the (linear) indices of v satisfying <condition>. In case v is a matrix, the indices correspond to those of the column vector v(:).

Some examples of <condition> are

- v > k or v >= k
- v < k or v <= k
- v == k or v ~= k

In order to combine more than two conditions, use

- "&" for **and**
- "|" for **or**

Vector Norms

The "length" of a vector $\mathbf{v}$ can be measured by its **norm**.

Definition (p-norm of a vector)

Let $p \in [1, \infty)$. The p-norm of $\mathbf{v} \in \mathbb{R}^m$ is denoted by $\|\mathbf{v}\|_p$ and is defined by

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{m} |v_i|^p \right)^{1/p}.$$

When $p = \infty$,

$$\|\mathbf{v}\|_\infty = \max_{1 \le i \le m} |v_i|.$$

The most commonly used p values are 1, 2, and ∞ and these norms can be calculated by the following MATLAB commands:

- `norm(v, 1)` for $\|\mathbf{v}\|_1$
- `norm(v, 2)` for $\|\mathbf{v}\|_2$ (Alternately, one can use `norm(v)`.)
- `norm(v, inf)` for $\|\mathbf{v}\|_\infty$

Matrix Norms

Definition (p-norm of a matrix)

Let $p \in [1, \infty]$. The p-norm of $A \in \mathbb{R}^{m \times n}$ is given by

$$\|A\|_p = \max_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|_p}{\|\mathbf{x}\|_p} = \max_{\|\mathbf{x}\|_p = 1} \|A\mathbf{x}\|_p \,.$$

(The following details can be skipped at the first reading.) The commonly used p-norms (for $p = 1, 2, \infty$) can also be calculated by

$$\begin{aligned}
\texttt{norm(A, 1)}: &\qquad \|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}| \\
\texttt{norm(A, 2)}: &\qquad \|A\|_2 = \sqrt{\lambda_{\max}(A^* A)} = \sigma_{\max}(A) \\
\texttt{norm(A, inf)}: &\qquad \|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}| \,.
\end{aligned}$$

Frobenius Norm of a Matrix

Another commonly used matrix norm, other than p-norms, is the Frobenius norm.

Definition (Frobenius norm of a matrix)

The Frobenius norm of $A \in \mathbb{R}^{m \times n}$ is given by

$$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2}.$$

Notes:

- It can be shown that $\|A\|_2 \leq \|A\|_F$.
- `norm(A, 'fro')` calculates the Frobenius norm of A in MATLAB.
- Alternately, we can use `norm(A(:))`. Think about why.

Tables Using Matrices in MATLAB

It is easy to store lots of data in MATLAB using matrices viewing them as a way of organizing a two-dimensional list of numbers in a tabular form.

Example: Stirling's formula revisited

```
1  %% script m-file: stirn
2  n = 20;
3  n_vec = (2:2:n)';
4  fact = factorial(n_vec);
5  stir = sqrt(2*pi*n_vec).*(n_vec/exp(1)).^(n_vec);
6  abs_err = stir - fact;
7  rel_err = stir./fact - 1;
8  T = [n_vec, fact, stir, abs_err, rel_err];
9  disp('            n            n!       stirling     abs_err      rel_err');
10 format short g;
11 disp(T);
```

Note: The table header positions (line 9) are obtained by trial and error.

Tables Using Matrices in MATLAB (cont'd)

The output of `stirn.m` on the COMMAND WINDOW:

```
>> stirn

         n            n!      stirling       abs_err       rel_err

         2             2         1.919     -0.080996     -0.040498
         4            24        23.506      -0.49382     -0.020576
         6           720        710.08       -9.9218      -0.01378
         8         40320         39902        -417.6     -0.010357
        10    3.6288e+06    3.5987e+06        -30104     -0.008296
        12      4.79e+08    4.7569e+08   -3.3141e+06    -0.0069188
        14    8.7178e+10    8.6661e+10   -5.1729e+08    -0.0059337
        16    2.0923e+13    2.0814e+13   -1.0868e+11    -0.0051941
        18    6.4024e+15    6.3728e+15   -2.9569e+13    -0.0046185
        20    2.4329e+18    2.4228e+18   -1.0115e+16    -0.0041577

>> diary off
```