Fondamenti di Informatica (Elettronici)

Vettori e matrici in Julia – Lezione 34

8 gennaio 2021

Vettori in Julia

1

[1] Tratto da: Pathak, Go, Zeng, and Boyd, EE108: Introduction to Matrix Methods, Stanford University, 2016.

Section 1

Vectors in Julia

Vectors: main topics

- how to create and manipulate vectors in Julia

Vectors: main topics

- how to create and manipulate vectors in Julia
- how Julia notation differs from math notation

Vectors

- Vector operations

Vectors

- Vector operations
- Norm and distance

Vectors

- vectors are represented by arrays in Julia

```
x = [8, -4, 3.5]    # (x = [8;-4;3.5] also works)
```

Vectors

- vectors are represented by arrays in Julia
- to create the 3-vector

$$x = (8, -4, 3.5) = \begin{bmatrix} 8 \\ -4 \\ 3.5 \end{bmatrix}$$

use

```
x = [8, -4, 3.5]    # (x = [8;-4;3.5] also works)
```

Vectors

- vectors are represented by arrays in Julia
- to create the 3-vector

$$x = (8, -4, 3.5) = \begin{bmatrix} 8 \\ -4 \\ 3.5 \end{bmatrix}$$

use

```
x = [8, -4, 3.5]    # (x = [8;-4;3.5] also works)
```

- watch out for similar looking expressions:

Vectors

- vectors are represented by arrays in Julia
- to create the 3-vector

$$x = (8, -4, 3.5) = \begin{bmatrix} 8 \\ -4 \\ 3.5 \end{bmatrix}$$

use

```
x = [8, -4, 3.5]    # (x = [8;-4;3.5] also works)
```

- watch out for similar looking expressions:
  - `(8,-4,3.5)` and `[8 -4 3.5]` are not equivalent in Julia

Vectors

- vectors are represented by arrays in Julia
- to create the 3-vector

$$x = (8, -4, 3.5) = \begin{bmatrix} 8 \\ -4 \\ 3.5 \end{bmatrix}$$

use

```
x = [8, -4, 3.5]    # (x = [8;-4;3.5] also works)
```

- watch out for similar looking expressions:
  - `(8,-4,3.5)` and `[8 -4 3.5]` are not equivalent in Julia
- length of a vector: `length(x)`

Ranges

- to get a range from `i` to `j` (`for i`$\leq$`j`), use a colon (`:`)

Ranges

- to get a range from `i` to `j` (`for i`$\leq$`j`), use a colon (`:`)
  - the range from `1` to `10` is `1:10`

Ranges

- to get a range from `i` to `j` (`for i`$\leq$`j`), use a colon (`:`)
  - the range from `1` to `10` is `1:10`
  - `collect(1:10)` returns the array

Ranges

- to get a range from `i` to `j` (`for i≤j`), use a colon (`:`)
  - the range from `1` to `10` is `1:10`
  - `collect(1:10)` returns the array
- the default increment between values is `1`. (`1:3` is `1, 2, 3`)

Ranges

- to get a range from `i` to `j` (`for i`$\leq$`j`), use a colon (`:`)
  - the range from `1` to `10` is `1:10`
  - `collect(1:10)` returns the array
- the default increment between values is `1`. (`1:3` is `1, 2, 3`)
- to specify an increment size add an additional argument:

Ranges

- to get a range from `i` to `j` (`for i≤j`), use a colon (`:`)
  - the range from `1` to `10` is `1:10`
  - `collect(1:10)` returns the array
- the default increment between values is `1`. (`1:3` is `1, 2, 3`)
- to specify an increment size add an additional argument:
  - the range from `1` to `10` with a step size of `0.1` is `1:0.1:10`

Indexing and slicing

- indexes run from `1` to `n`: x_2 is `x[2]`

Indexing and slicing

- indexes run from `1` to `n`: x_2 is `x[2]`
- can also set an element, e.g., `x[3] = 10.5`

Indexing and slicing

- indexes run from `1` to `n`: x_2 is `x[2]`
- can also set an element, e.g., `x[3] = 10.5`
- use a range to select more than one element

Indexing and slicing

- indexes run from `1` to `n`: x_2 is `x[2]`
- can also set an element, e.g., `x[3] = 10.5`
- use a range to select more than one element
  - `x[2:3]` selects the second and third elements

Indexing and slicing

- indexes run from `1` to `n`: x_2 is `x[2]`
- can also set an element, e.g., `x[3] = 10.5`
- use a range to select more than one element
  - `x[2:3]` selects the second and third elements
- `x[end]` selects the last element

Indexing and slicing

- indexes run from `1` to `n`: x_2 is `x[2]`
- can also set an element, e.g., `x[3] = 10.5`
- use a range to select more than one element
  - `x[2:3]` selects the second and third elements
- `x[end]` selects the last element
- to select every other element use `x[1:2:end]`

Block vectors

- to form a stacked vector like

$$a = (b, c) = \begin{bmatrix} b \\ c \end{bmatrix}$$

(with b and c vectors)

```
a = [b; c]
```

(`a = [b, c]` does NOT work)

```
a = [b; 2; c; -6]
```

Block vectors

- to form a stacked vector like

$$a = (b, c) = \begin{bmatrix} b \\ c \end{bmatrix}$$

(with b and c vectors)

```
a = [b; c]
```

(`a = [b, c]` does NOT work)

- can mix vectors and scalars:

```
a = [b; 2; c; -6]
```

Basic functions for arrays

- sum of (the entries of) a vector: `sum(x)`

Basic functions for arrays

- sum of (the entries of) a vector: `sum(x)`
- mean of the entries (**avg**(x)): `mean(x)`

Basic functions for arrays

- sum of (the entries of) a vector: `sum(x)`
- mean of the entries (**avg**(x)): `mean(x)`
- $\mathbf{0}_n$ is `zeros(n)`

Basic functions for arrays

- sum of (the entries of) a vector: `sum(x)`
- mean of the entries (**avg**(x)): `mean(x)`
- $\mathbf{0}_n$ is `zeros(n)`
- $\mathbf{1}_n$ is `ones(n)`

Creating unit vectors

- form e_3 with length 10

Creating unit vectors

- form e_3 with length 10
- create a zero vector of size 10 then set the third element to 1:

Creating unit vectors

- form e_3 with length 10
- create a zero vector of size 10 then set the third element to 1:

Creating unit vectors

- form e_3 with length 10
- create a zero vector of size 10 then set the third element to 1:

```
julia> e_3 = zeros(10);

julia> e_3[3] = 1;

julia> e_3'
1×10 LinearAlgebra.Adjoint{Float64,Array{Float64,1}}:
 0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

List of vectors

- to form a list with vectors a, b, and c:

```
vector_list = [a,b,c]
```

List of vectors

- to form a list with vectors a, b, and c:

```
vector_list = [a,b,c]
```

List of vectors

- to form a list with vectors a, b, and c:

```
vector_list = [a,b,c]
```

- the second vector in this list is

```
vector_list[2]
```

List of vectors

- to form a list with vectors a, b, and c:

```
vector_list = [a,b,c]
```

- the second vector in this list is

```
vector_list[2]
```

List of vectors

- to form a list with vectors a, b, and c:

```
vector_list = [a,b,c]
```

- the second vector in this list is

```
vector_list[2]
```

- to access an element in a vector:

```
vector_list[2][3]
```

Section 2

Vector operations

Vector addition and subtraction

- vector addition uses $+$, for example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

is written

```
julia> [1, 2, 3] + [4, 5, 6]
3-element Array{Int64,1}:
 5
 7
 9
```

Vector addition and subtraction

- vector addition uses +, for example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

  is written

```
julia> [1, 2, 3] + [4, 5, 6]
3-element Array{Int64,1}:
 5
 7
 9
```

- subtraction uses -

Vector addition and subtraction

- vector addition uses +, for example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

  is written

```
julia> [1, 2, 3] + [4, 5, 6]
3-element Array{Int64,1}:
 5
 7
 9
```

- subtraction uses -
- the arrays must have the same length (unless one is scalar)

Scalar-vector addition

- in Julia, a scalar and a vector can be added

Scalar-vector addition

- in Julia, a scalar and a vector can be added
- the scalar is added to each entry of the vector

```
julia> [2, 4, 8] .+ 3
3-element Array{Int64,1}:
     5
     7
 11
```

gives (in mathematical notation)

$$\begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} + 3 = \begin{bmatrix} 5 \\ 7 \\ 11 \end{bmatrix}$$

Scalar-vector multiplication

- scalar-vector multiplication uses $*$

Scalar-vector multiplication

- scalar-vector multiplication uses *
- for example,

$$(-2)\begin{bmatrix} 1 \\ 9 \\ 6 \end{bmatrix}$$

is written:

```
-2 * [1, 9, 6]
```

Scalar-vector multiplication

- scalar-vector multiplication uses *
- for example,

$$(-2)\begin{bmatrix} 1 \\ 9 \\ 6 \end{bmatrix}$$

is written:

```
-2 * [1, 9, 6]
```

- the other order gives the same result:

```
[1, 9, 6] * -2
```

https://stackoverflow.com/questions/52334857/adding-scalar-to-an-array-in-julia

In Julia syntax (like in MATLAB), + and * operate on congruent arrays.

For element-wise operation, you should use `.+` and `.*`. Strangely, this does not seem to matter for * but it does for +.

https://stackoverflow.com/questions/52334857/adding-scalar-to-an-array-in-julia

In Julia syntax (like in MATLAB), + and * operate on congruent arrays.

For element-wise operation, you should use `.+` and `.*`. Strangely, this does not seem to matter for * but it does for +.

It does not matter for * because multiplication of a matrix by scalar is a well defined operation (in mathematical sense), but you can also broadcast it to get the same result.

On the other hand addition of a scalar and a matrix is not a standard operation so you have to broadcast it. There is a difference though if you wanted to use broadcast fusion as then you would need to use `.*` everywhere.

– Bogumił Kamiński

Example

```
julia> m = 1.1
1.1

julia> c = 0.11
0.11

julia> x = rand(1,2)
1×2 Array{Float64,2}:
 0.543101  0.335506

julia> y = m*x .+ c
1×2 Array{Float64,2}:
 0.707411  0.479056

julia> y = m * x + c
ERROR: MethodError: no method matching +(::Array{Float64,2}, ::Float64)
For element-wise addition, use broadcasting with dot syntax: array .+ scalar

julia> y = m .* x .+ c
1×2 Array{Float64,2}:
 0.707411  0.479056
```

Inner product

- inner product $a^\top b$ is written as

  dot(a,b)

Inner product

- inner product $a^\top b$ is written as

  dot(a,b)

- a and b must have the same length

Section 3

Norm and distance

Norm and distance

- the norm $||x|| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ is written

  `norm(x)`

Norm and distance

- the norm $||x|| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ is written

  `norm(x)`

- dist(x, y) = ||y - x|| is written

  `norm(y-x)`

RMS (Root Mean Square – radice della media dei quadrati)

- **rms**(x) is defined as[2]

$$\textbf{rms}(x) = \sqrt{\frac{1}{n}(x_1^2 + \cdots + x_n^2)} = \frac{||x||}{\sqrt{n}}$$

.

[2] In teoria dei segnali il valore efficace di una funzione periodica è il valore che avrebbe un segnale costante di pari potenza media

RMS (Root Mean Square – radice della media dei quadrati)

- **rms**(x) is defined as[2]

$$\textbf{rms}(x) = \sqrt{\frac{1}{n}(x_1^2 + \cdots + x_n^2)} = \frac{||x||}{\sqrt{n}}$$

.

[2] In teoria dei segnali il valore efficace di una funzione periodica è il valore che avrebbe un segnale costante di pari potenza media

RMS (Root Mean Square – radice della media dei quadrati)

- **rms**(x) is defined as[2]

$$\textbf{rms}(x) = \sqrt{\frac{1}{n}(x_1^2 + \cdots + x_n^2)} = \frac{||x||}{\sqrt{n}}$$

.

- can be expressed as

```
julia> rms(x) = norm(x)/sqrt(length(x))
rms (generic function with 1 method)

julia> x
1×2 Array{Float64,2}:
 0.543101  0.335506

julia> rms(x)
0.4513993124036708 7
```

[2] In teoria dei segnali il valore efficace di una funzione periodica è il valore che avrebbe un segnale costante di pari potenza media

Standard deviation[3]

- standard deviation is defined as

$$\mathbf{std}(x) = \frac{||x - \mathbf{avg}(x)\mathbf{1}||}{\sqrt{n}}$$

[3] Functions std and mean are in Statistics module.

Standard deviation[3]

- standard deviation is defined as

$$\mathbf{std}(x) = \frac{||x - \mathbf{avg}(x)\mathbf{1}||}{\sqrt{n}}$$

- which can be expressed as

```
std(x) = norm(x - mean(x))/sqrt(length(x))
```

[3] Functions std and mean are in Statistics module.

Standard deviation[3]

- standard deviation is defined as

$$\mathbf{std}(x) = \frac{||x - \mathbf{avg}(x)\mathbf{1}||}{\sqrt{n}}$$

- which can be expressed as

```
std(x) = norm(x - mean(x))/sqrt(length(x))
```

- warning: the Julia function `std` does not use this definition

[3] Functions std and mean are in Statistics module.

Population standard deviation of grades of eight students

Suppose that the entire population of interest is eight students in a particular class. For a finite set of numbers, the population standard deviation is found by taking the square root of the average of the squared deviations of the values subtracted from their average value. The marks of a class of eight students (that is, a statistical population) are the following eight values:

$$2,\ 4,\ 4,\ 4,\ 5,\ 5,\ 7,\ 9.$$

These eight data points have the mean (average) of 5:

$$\mu = \frac{2+4+4+4+5+5+7+9}{8} = \frac{40}{8} = 5.$$

Population standard deviation of grades of eight students

First, calculate the deviations of each data point from the mean, and square the result of each:

$$\begin{aligned}(2-5)^2 &= (-3)^2 = 9 & \qquad (5-5)^2 &= 0^2 = 0\\ (4-5)^2 &= (-1)^2 = 1 & \qquad (5-5)^2 &= 0^2 = 0\\ (4-5)^2 &= (-1)^2 = 1 & \qquad (7-5)^2 &= 2^2 = 4\\ (4-5)^2 &= (-1)^2 = 1 & \qquad (9-5)^2 &= 4^2 = 16.\end{aligned}$$

The variance is the mean of these values:

$$\sigma^2 = \frac{9+1+1+1+0+0+4+16}{8} = \frac{32}{8} = 4.$$

and the population standard deviation is equal to the square root of the variance:

$$\sigma = \sqrt{4} = 2.$$

Population standard deviation of grades of eight students

```
julia> using Statistics

julia> mean([2, 4, 4, 4, 5, 5, 7, 9.])
5.0

julia> std
std (generic function with 2 methods)

julia> std([2, 4, 4, 4, 5, 5, 7, 9.])
2.138089935299395

julia> std([2, 4, 4, 4, 5, 5, 7, 9.], corrected=false)
2.0
```

The default above is known as Bessel's correction.

Section 4

Angle

Angle between vectors

- the angle between two vectors a and b is

$$\angle(a, b) = \arccos\left(\frac{a^\top b}{\|a\|\,\|b\|}\right)$$

Angle between vectors

- the angle between two vectors a and b is

$$\angle(a, b) = \arccos\left(\frac{a^\top b}{\|a\|\,\|b\|}\right)$$

Angle between vectors

- the angle between two vectors a and b is

$$\angle(a, b) = \arccos\left(\frac{a^\top b}{\|a\|\,\|b\|}\right)$$

- can be expressed as

```
julia> angle(a,b) = acos(dot(a,b)/(norm(a)*norm(b)))
angle (generic function with 1 method)

julia> angle([1,0],[0,1])
1.5707963267948966

julia> angle([1,0],[-1,0])
3.141592653589793

julia> angle([1,0],[0,-1])
1.5707963267948966
```

Angle between vectors

`atan(y)` and `atan(y, x)`

Compute the inverse tangent of `y` or `y/x`, respectively.

For one argument, this is the angle in radians between the positive x-axis and the point `(1, y)`, returning a value in the interval `[-\pi/2, \pi/2]`.

For two arguments, this is the angle in radians between the positive x-axis and the point `(x, y)`, returning a value in the interval `[-\pi, \pi]`.

This corresponds to a standard `atan2` function.

```
julia> atan(1,-1)
2.356194490192345

julia> atan(1,-1)/pi*180
135.0
```