

# Matrices

Storing two-dimensional numerical data

---



**CS112 Scientific Computation**  
Department of Computer Science  
Wellesley College

## Our boys of summer<sup>1</sup>

Player Name	Player Number	Weight	At Bat	Home Runs	Batting Average	2011 Salary	RBI's	Runs	Stolen Bases
Jacoby Ellsbury	2	185	660	32	.321	2,400,000	105	119	39
Dustin Pedroia	15	180	635	21	.307	5,750,000	91	102	26
David Ortiz	34	230	525	29	.309	12,500,000	96	84	1
Adrian Gonzalez	28	225	630	27	.338	6,300,000	117	108	1

<sup>1</sup>Ellen's favorite Redsox players

## Medical imaging

Matrices are particularly good for storing data that is inherently two-dimensional

For example, the illustrated MRI slice is obtained from a two-dimensional grid of brightness measurements registered by an array of light sensitive elements



Matrices 6-3

## Matrices: The basics

- A matrix is a rectangular array of numbers
- We create a matrix of specific values with an assignment statement:

```
>> flowers = [1 3 2 7; 6 4 5 1; 2 8 3 7]
```

```
flowers =
```

```
1 3 2 7  
6 4 5 1  
2 8 3 7
```



	1	3	2	7
flowers	6	4	5	1
	2	8	3	7

Matrices 6-4

## Dimensions

- Each row must contain the same number of values!

`nums = [1 4 2; 6 8]` 

- **size** function returns the number of rows and columns in a matrix:

```
>> dims = size(flowers)
dims =
     3     4
>> rows = size(flowers, 1)
rows =
     3
>> cols = size(flowers, 2)
cols =
     4
```

flowers	1	3	2	7
	6	4	5	1
	2	8	3	7

How could you determine the total number of elements in a matrix?

Matrices 6-5

## Déjà vu all over again

Many computations can be performed on an entire matrix all at once

`flowers = 2 * flowers + 1`



flowers	1	3	2	7
	6	4	5	1
	2	8	3	7

flowers	3	7	5	15
	13	9		

Matrices 6-6

## Element-by-element matrix addition

**sumFlowers = flowers + addOns**

flowers	3	7	5	15
	13	9	11	3
	5	17	7	15

 + 

addOns	2	1	3	4
	2	4	3	1
	2	0	1	4

sumFlowers	5	8	8	19
	15	13		

How do you perform element-by-element multiplication?

Matrices 6-7

## Vectors are matrices... ... with one row or column

**rowNums = [1 2 3]**  
**rowNumsSize = size(rowNums)**  
**colNums = [1; 2; 3]**  
**colNumsSize = size(colNums)**

**length** function returns the largest dimension



rowNums	1	2	3
---------	---	---	---

rowNumsSize	1	3
-------------	---	---

colNums	1
	2
	3

colNumsSize	3	1
-------------	---	---

Matrices 6-8

## Now I know what you're thinking...

You probably think that we can use functions like **sum**, **prod**, **min**, **max** and **mean** in the same way they were used with vectors:

```
numbers = [1 3 2 4; 4 1 2 3]
totalSum = sum(numbers)
totalProd = prod(numbers)
minVal = min(numbers)
maxVal = max(numbers)
meanVal = mean(numbers)
```

numbers	1	3	2	4
	4	1	2	3



Matrices 6-9

## Hmmm... that's not what I expected...

```
numbers = [1 3 2 4; 4 1 2 3]
totalSum = sum(numbers)
totalProd = prod(numbers)
minVal = min(numbers)
maxVal = max(numbers)
meanVal = mean(numbers)
```



meanVal	2.5	2.0	2.0	3.5
---------	-----	-----	-----	-----

numbers	1	3	2	4
	4	1	2	3

totalSum	5	4	4	7
----------	---	---	---	---

totalProd	4	3	4	12
-----------	---	---	---	----

minVal	1	1	2	3
--------	---	---	---	---

maxVal	4	3	2	4
--------	---	---	---	---

Matrices 6-10

## Processing and displaying images

An **image** is a two-dimensional grid of measurements of brightness

We will start with images with brightness ranging from black (0.0) to white (1.0) with shades of gray in between ( $0.0 < b < 1.0$ )



1887 Crew Team  
Wellesley College

Matrices 6-11

## Creating a tiny image

```
>> tinyImage = [ 0.0 0.0 0.0 0.0 0.0 0.0; ...  
                 0.0 0.5 0.5 0.5 0.5 0.0; ...  
                 0.0 0.5 1.0 1.0 0.5 0.0; ...  
                 0.0 0.5 1.0 1.0 0.5 0.0; ...  
                 0.0 0.5 0.5 0.5 0.5 0.0; ...  
                 0.0 0.0 0.0 0.0 0.0 0.0]
```

	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.5	0.5	0.5	0.5	0.0
	0.0	0.5	1.0	1.0	0.5	0.0
	0.0	0.5	1.0	1.0	0.5	0.0
	0.0	0.5	0.5	0.5	0.5	0.0
	0.0	0.0	0.0	0.0	0.0	0.0
<b>tinyImage</b>						

```
>> imshow(tinyImage)
```

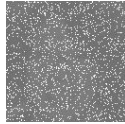


(not to scale)

Matrices 6-12

## A mystery: Who am I?

This very corrupted image was received  
by anonymous courier late last night



Let's figure out what's in it using the  
Image Processing Toolbox

```
>> imtool(image)
```



Matrices 6-13

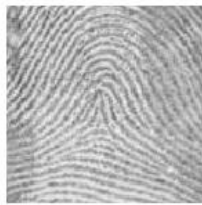
## Whodunit??



**Suspect**



**Scott**



**Randy**

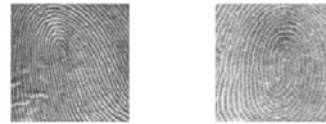


**Sohie**

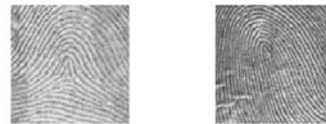
Matrices 6-14

## Our strategy

**Step 1.** Calculate the difference between two images



**Step 2.** Use the **abs** function to calculate the magnitude of the difference between two images



**Step 3.** Calculate the average difference across the entire image

Matrices 6-15

## Creating matrices with constant values

To create a matrix of all ones:

`nums1 = ones(2,3)`

`nums2 = ones(1,5)`

nums1	1	1	1
	1	1	1

To create a matrix of all zeros:

`nums3 = zeros(3,1)`

`nums4 = zeros(4,3)`

nums2	1	1	1	1	1
-------	---	---	---	---	---

nums3	0
	0
	0

nums4	0	0	0
	0	0	0
	0	0	0
	0	0	0

Matrices 6-16

## Indexing with matrices

Each row and column in a matrix is specified by an index

```
nums = [1 3 7 4; 8 5 2 6]
```

nums	1	2	3	4
1	1	3	7	4
2	8	5	2	6

We can use the indices to **read or change** the contents of a location

```
val = nums(2,3)  
nums(1,4) = 9  
nums(1,end) = 9
```

} Similar to vectors

Matrices 6-17

## Time-out exercise

Starting with a fresh copy of **nums**

```
nums = [1 3 7 4; 8 5 2 6]
```

nums	1	2	3	4
1	1	3	7	4
2	8	5	2	6

what would the contents of **nums** and **val** be after executing the following statements?

```
nums(2,3) = nums(1,2) + nums(2,4)  
nums(1,3) = nums(1,4) + nums(2,1)  
val = nums(4,3)
```

Matrices 6-18

## Auto expansion of matrices

```
>> nums = [1 3 7 4; 8 5 2 6]
```

nums	1	2	3	4
1	1	3	7	4
2	8	5	2	6



```
>> nums(4, 7) = 3
```

nums	1	2	3	4	5	6	7
1	1	3	7	4	0	0	0
2	8	5	2	6	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	3

Matrices 6-19