CS 1112 Introduction to Computing Using MATLAB

Instructor: Dominic Diaz

Website: https://www.cs.cornell.edu/courses/cs1112/2022fa/

Today: Vectorized code + matrices

Agenda and announcements

- Last time
  - Linear interpolation
- Today
  - Vectorized computation
  - 2D arrays - matrix
  - Talk about prelim
- Announcements
  - Project 2 grades released (Regrade requests due by 10/7 at 11 PM)
  - Project 3 due Wednesday 10/5 (TOMORROW)
    - Late deadline (Thursday 10/6) will only be 5% off (not 10% like usual)
  - "Check your prelim 1 time/location" on CMS–read the "grading comment" to find exam time/location. **Any request for alternative arrangements (including conflicting exams) is due as a "regrade request" in CMS by 10/7 at 11 PM.**
  - Ex 07 will be due Thursday 10/13 at 9 PM (due to Fall break)
  - Prelim topics posted on website (will show on last slide)

Vectorized code

Vectorized code is code that performs element-by-element operations on arrays in one step.

Why is vectorized code useful:

- Appearance: looks cleaner and closer to math formulas
- Less error prone: less code ⇒ less errors
- Performance: vectorized code is typically faster

```
% add two vectors a and b, elementwise
a = [1, 3, 16, -2, 15,  6, -3];
b = [6, 5, 10, 11, -4, 10,  0];

c = zeros(1, length(a));
for i = 1:length(a)
     c(i) = a(i) + b(i);
end
```

Vectorized code

```
% add two vectors a and b, elementwise
a = [1, 3, 16, -2, 15,  6, -3];
b = [6, 5, 10, 11, -4, 10,  0];

c = a + b;
```

Both codes output the same vector:

c	7	8	26	9	11	16	-3

Vectorized element-wise operations between vectors

Works on each element individually

```
a = [1, 6, -2, 0, -6, 10];
b = [-1, 0, 8, 2, 50, -16];
```

Addition +

```
c = a + b;
% c = [0, 6, 6, 2, 44, -6]
```

Subtraction -

```
d = a - b;
% d = [2, 6, -10, -2, -56, 26]
```

Multiplication x

```
e = a.*b;
% e = [-1, 0, -16, 0, ...]
```

Division /

```
f = a./b;
% f = [-1, Inf, -0.25, ...]
```

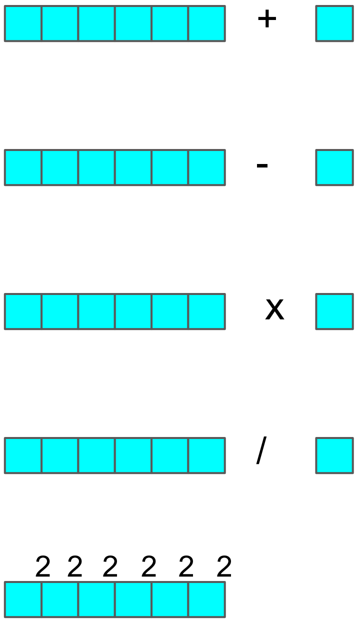
Power ^

```
g = a.^b;
% g = [1, 1, 256, 0, ...]
```

See a more comprehensive list of element-wise arithmetic operations in chapter 4.1

Vectorized element-wise operations between a vector and a scalar

```
a = [1, 6, -2, 0, -6, 10];
b = 2;
```



Addition

```
c = a + b;
% c = [3, 8, 0, 2, -4, 12]
```

Subtraction

```
d = a - b;
% d = [-1, 4, -4, ...]
```

Multiplication

```
e = a*b;    % or e = a.*b;
% e = [2, 12, -2, 0, ...]
```

Division

```
f = a/b;    % or f = a./b;
% f = [0.5, 3, -1, 0, ...]
```

Power

```
g = a.^b;
% g = [1, 36, 4, 0 ...]
```

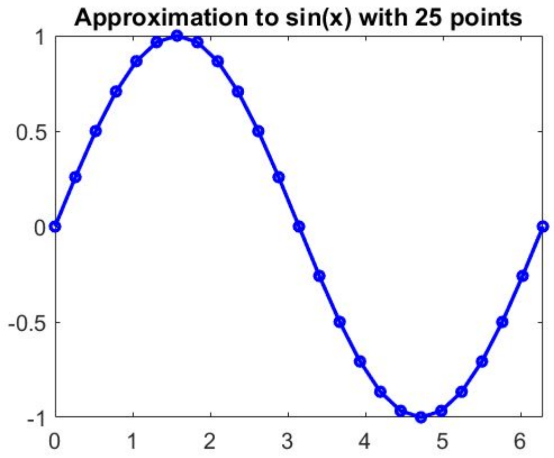
Simplified rule: for multiplication, division, and power, use `.*` `./` `.^`

Some MATLAB built-in functions can take vectors as inputs

```
% plot the sine function
numPts = 25;
x = linspace(0, 2*pi, numPts);

y = zeros(1, length(x));
for i = 1:length(x)
   y(i) = sin(x(i));
end

plot(x,y, "-ob")
```

These codes do the same thing!

```
% plot the sine function
numPts = 25;
x = linspace(0, 2*pi, numPts);

y = sin(x) % creates a vector

plot(x,y, "-ob")
```



Vectorized code in action

Can we plot the following function? Yes!

$$f(x) = \frac{\sin(5x)\exp(-x/2)}{1+x^2} \text{ for } -2 \leq x \leq 3$$

```
x = linspace(-2, 3, 100);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
plot(x,y)
```

Vectorized code in action

Can we plot the following function? Yes!

$$f(x) = \frac{\sin(5x)\exp(-x/2)}{1+x^2} \text{ for } -2 \leq x \leq 3$$

scalar * vector

vector / scalar

vector ^ scalar

```
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
```

vector * vector

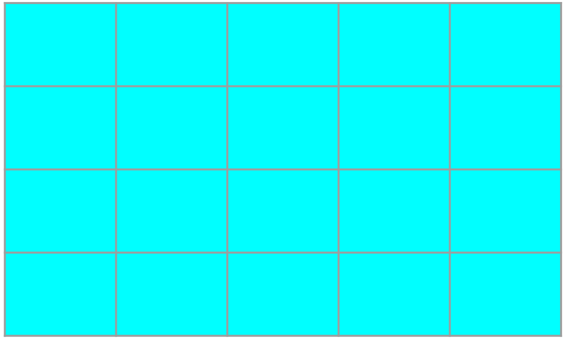
Exercise: try to recreate this same result using non-vectorized code (use for loop).

End of
prelim 1 material!

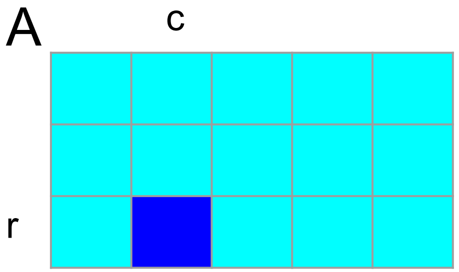
1D arrays to 2D arrays

Previously we've looked at 1D arrays (representing object positions, colors, die rolls):

Now we'll looked at 2D arrays (also called matrices):



2D array: matrix

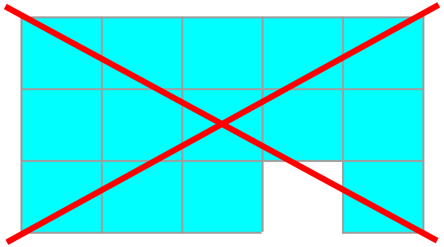


- An array is a named collection of like data organized into rows and columns
- A 2D array is like a table, and is also called a matrix
- Two indices identify the position of a value in a matrix

`A(r,c)`

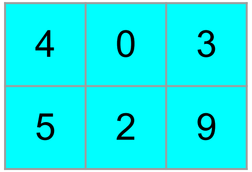
First index: row index
Second index: column index

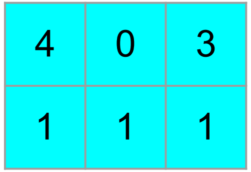
- Both indices start at 1
- 2D arrays must be rectangular: all rows must have the same number of columns

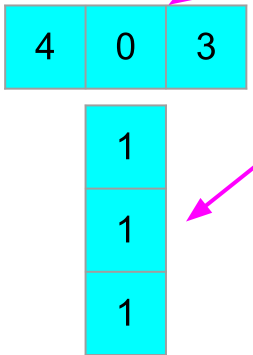


Creating a matrix

- Built-in functions: ones, zeros, rand
  - `zeros(2,3)` will create a 2-by-3 matrix of all 0s
  - `zeros(2)` will create a 2-by-2 matrix of all 0s
- Build a matrix using square brackets, [ ], but the dimensions must match up
  - `X = [4 0 3; 5 2 9];`
  - `Y = [4 0 3; ones(1,3)];`
  - `Z = [4 0 3; ones(3,1)]; % error!`





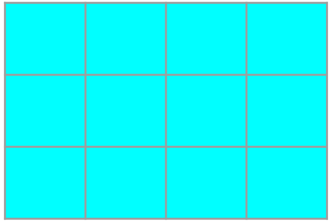


No way to combine these two into a matrix while preserving orientation.

Length of a vector, size of a matrix

v

M



```
% To determine how many elements in a
% vector v, use length function

v = [1, 4, 10, -4, 16];
n = length(v);

% n stores length of v
```

```
% To determine how many rows/columns in
% a matrix M, use size function

M = [1, 2, 5, 7;
     3, 0, 0, 6;
     4, 3, 2, 0];
[nr, nc] = size(M);

% nr stores number of rows of M
% nc stores number of columns of M
```

Changing values in a matrix

```
% Create a matrix of size(2,3) with zeros
M = zeros(2, 3);
```

M

0	0	0
0	0	0

```
% Change the element in row 1, column 1
M(1,1) = 9;
```

Row number (row index)

Column number (column index)

M

9	0	0
0	0	0

```
% Change the element in row 2, column 1
M(2,1) = 7;
```

M

9	0	0
7	0	0

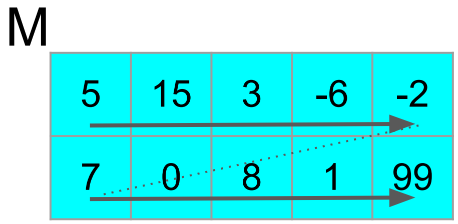
```
% Change the element in row 2, column 3
M(2,3) = M(2,1);
```

M

9	0	0
7	0	7

Poll Everywhere

Example: display all values in a matrix



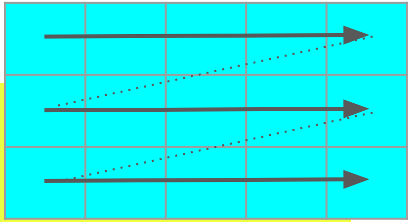

```
% Given some matrix M
[nr, nc] = size(M);
for r = 1:nr
    for c = 1:nc

        disp(M(r,c));

    end
end
```

What does the computer do?

nr = 2, nc = 5
r = 1, c = 1 display M(1,1) = 5
r = 1, c = 2 display M(1,2) = 15
r = 1, c = 3 display M(1,3) = 3
r = 1, c = 4 display M(1,4) = -6
r = 1, c = 5 display M(1,5) = -2
r = 2, c = 1 display M(2,1) = 7
r = 2, c = 2 display M(2,1) = 0
r = 2, c = 3 display M(2,1) = 8
r = 2, c = 4 display M(2,1) = 1
r = 2, c = 5 display M(2,1) = 99

Code for traversing a matrix

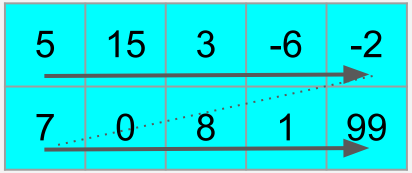
M



```
[nr, nc] = size(M);

% r will loop through row index
for r = 1:nr
   % c will loop through column index
   for c = 1:nc

      % Do something with M(r,c)

   end
end
```

Example: compute the minimum value in a matrix

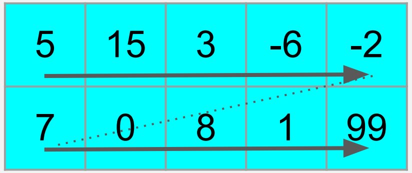
```
function minVal = minInMatrix(M)
% compute the smallest value (minVal) in matrix M, not empty

[nr, nc] = size(M);

for r = 1:nr
    for c = 1:nc


        % see if M(r,c) is the minimum value



    end
end
```



Minimum: -6

Example: compute the minimum value in a matrix

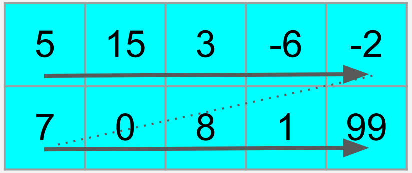
```
function minVal = minInMatrix(M)
% compute the smallest value (minVal) in matrix M, not empty

[nr, nc] = size(M);

for r = 1:nr
    for c = 1:nc

        if M(r,c) < minVal

        end

    end
end
```



Minimum: -6

Example: compute the minimum value in a matrix

```
function minVal = minInMatrix(M)
% compute the smallest value (minVal) in matrix M, not empty

[nr, nc] = size(M);
minVal = M(1,1);
for r = 1:nr
    for c = 1:nc

        if M(r,c) < minVal
            minVal = M(r,c);
        end


    end
end
```



Minimum: -6

How to study for the prelim in this class

1. Write your own solutions to examples from lecture
2. Redo exercise problems un-aided
3. Do review questions (posted on prelim 1 page)
4. Do one old exam, using notes as needed
5. Do the second old exam un-aided–this is your best diagnostic
6. Review specific topics further as necessary
7. Do third old exam like you are taking the real exam

Just reading code and solutions will **not** help!

Check out https://www.cs.cornell.edu/courses/cs1112/2022fa/ -> Exams -> prelim1 for list of topics!