Matlab 101
a.k.a. “The Matlabomicon”

David Weiss
a.k.a. “MC Matlab,” “The Matlab Maniac,”
“General Matlabissimo,” and “Matlab-sensei”

What you will learn

What you will learn

- MATLAB

The Matlab GUI

The Matlab GUI

- [Demo]

The Matlab GUI

- [Demo]
- Further details online on website

What is a Matlab program?

What is a Matlab program?

- Any text file with .m extension

What is a Matlab program?

- Any text file with .m extension
- Script: a saved series of Matlab commands that can be replayed

What is a Matlab program?

- Any text file with .m extension
- Script: a saved series of Matlab commands that can be replayed
- Function: Defines new Matlab commands

Why Matlab?

Why Matlab?

- Vast library of common scientific functions

Why Matlab?

- Vast library of common scientific functions
- Easy reading/writing data

Why Matlab?

- Vast library of common scientific functions
- Easy reading/writing data
- Easy visualization and plotting of data

Why Matlab?

- Vast library of common scientific functions
- Easy reading/writing data
- Easy visualization and plotting of data
- Easy debugging

Why Matlab?

- Vast library of common scientific functions
- Easy reading/writing data
- Easy visualization and plotting of data
- Easy debugging
- Kickbacks from Mathworks ($$)

Case study: YamSlam!

Case study: YamSlam!

- Given 3 chances to roll, how likely is it that you will roll 5 of a kind?

Case study: YamSlam!

- Given 3 chances to roll, how likely is it that you will roll 5 of a kind?
- Strategy: Pick the most common #, and re-roll dice that don't match

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

Sample from uniform distribution

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

In the range (0,6)

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

Compute for a 5 x 1 matrix

Rolling 5 dice

$$y = \boxed{\texttt{floor}}(\texttt{unifrnd}(0,6,5,1))+1;$$

Round down to get {0, 1, 2, 3, 4, 5}

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

Increment by 1 to match die #'s

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

Suppress output from command

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

What is the value of y?

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

5
6
1
6
4

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

1
2
4
6
6

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,5))+1;
```

y =

5	6	6	3	3
2	4	2	2	5
4	1	5	2	4
5	1	2	4	4
6	2	6	3	6

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

1
2
4
6
6

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

1
2
4
$\boxed{6}$
$\boxed{6}$

Keep these

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

Reroll these

$$\begin{matrix} \boxed{\begin{matrix} 1 \\ 2 \\ 4 \end{matrix}} \\ 6 \\ 6 \end{matrix}$$

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

$$\boxed{\begin{matrix}1\\2\\4\end{matrix}}$$

Reroll these

6

6

```
y(y ~= 6)= ...
```

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

$\boxed{\begin{matrix}1\\2\\4\end{matrix}}$

6

6

Reroll these

```
y(y ~= 6)= ...
  floor(unifrnd(...
             0,6,3,1))+1;
```

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

6

Reroll these 1

4

6

6

```
y(y ~= 6)= ...
  floor(unifrnd(...
            0,6,3,1))+1;
```

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

6
Reroll these 1
4
6
6

```
y(y ~= 6)= ...
  floor(unifrnd(...
          0,6,2,1))+1;
```

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

6
1
4
6
6

Reroll these

```
y([2 3])= ...
  floor(unifrnd(...
               0,6,2,1))+1;
```

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

6
Reroll these 1
4
6
6

```
y(2:3)= ...
  floor(unifrnd(...
              0,6,2,1))+1;
```

Rolling 5 dice

```
y = floor(unifrnd(0,6,5,1))+1;
```

y =

6
Reroll these 1
4
6
6

```
y([0 1 1 0 0 0]>0) = ...
    floor(unifrnd(...
                0,6,2,1))+1;
```

YamSlam! Simulator

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

Initialize y to 5 x 1 matrix

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

Roll dice indexed by `rollidx`

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

initialized to [1 2 3 4 5]

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

Updated to indices of non-matching dice

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

Count # of times each number occurs

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

Compute argmax (~ is maximum, discarded)

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

Record ‘1’ if all y are equal to y(1)

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

Why mean?

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

- Let’s run it!!

Why YamSlam!?

Why YamSlam!?

- Read the code we give you

Why YamSlam!?

- Read the code we give you
- Follow the online tutorials

Why YamSlam!?

- Read the code we give you
- Follow the online tutorials
- Try to figure it out

Why YamSlam!?

- Read the code we give you
- Follow the online tutorials
- Try to figure it out
- Ask questions on Piazza

YamSlam! Simulator

What is this called?

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        for i = 1:6
            c(i) = sum(y==i);
        end
        [~,maxi] = max(c);
        rollidx = find(y~=maxi);
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

YamSlam! Simulator

Simplifying through built-in commands

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        rollidx = find(y~=mode(y));
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

Key Matlab Concepts

Key Matlab Concepts

- All numeric types are matrices

Key Matlab Concepts

- All numeric types are matrices
- Vectorization instead of loops

Key Matlab Concepts

- All numeric types are matrices
- Vectorization instead of loops
- Indexing instead of control logic

All numeric data are matrices

All numeric data are matrices

```
A = 2; B = 3; A*B =
```

All numeric data are matrices

```
A = 2; B = 3; A*B =
```

[6]

All numeric data are matrices

```
A = 2; B = 3; A*B =
```

[6]

```
A = [1 2]; B = 3; A*B =
```

All numeric data are matrices

```
A = 2; B = 3; A*B =
```
[6]

```
A = [1 2]; B = 3; A*B =
```
[3 6]

All numeric data are matrices

```
A = 2; B = 3; A*B =
```
[6]

```
A = [1 2]; B = 3; A*B =
```
[3 6]

```
A = [1 2]; B = [3 4]; A*B =
```

All numeric data are matrices

```
A = 2; B = 3; A*B =
```
[6]

```
A = [1 2]; B = 3; A*B =
```
[3 6]

```
A = [1 2]; B = [3 4]; A*B =
```
ERROR

All numeric data are matrices

```
A = 2; B = 3; A*B =
```
[6]

```
A = [1 2]; B = 3; A*B =
```
[3 6]

```
A = [1 2]; B = [3 4]; A*B =
```
ERROR

```
A = [1 2]; B = [3 4]; A.*B =
```

All numeric data are matrices

```
A = 2; B = 3; A*B =
```
[6]

```
A = [1 2]; B = 3; A*B =
```
[3 6]

```
A = [1 2]; B = [3 4]; A*B =
```
ERROR

```
A = [1 2]; B = [3 4]; A.*B =
```
[3 8]

All numeric data are matrices

```
A = 2; B = 3; A*B =
```
[6]

```
A = [1 2]; B = 3; A*B =
```
[3 6]

```
A = [1 2]; B = [3 4]; A*B =
```
ERROR

```
A = [1 2]; B = [3 4]; A.*B =
```
[3 8]

```
A = [1 2]; B = [3 4]'; A*B =
```

All numeric data are matrices

```
A = 2; B = 3; A*B =
```
[6]

```
A = [1 2]; B = 3; A*B =
```
[3 6]

```
A = [1 2]; B = [3 4]; A*B =
```
ERROR

```
A = [1 2]; B = [3 4]; A.*B =
```
[3 8]

```
A = [1 2]; B = [3 4]'; A*B =
```
[11]

All numeric data are matrices

All numeric data are matrices

```
A = 2; size(A) =
```

All numeric data are matrices

```
A = 2; size(A) =   [1 1]
```

All numeric data are matrices

```
A = 2; size(A) =
```
[1 1]

```
A = [1 2;3 4]; size(A) =
```

All numeric data are matrices

`A = 2; size(A) =` [1 1]

`A = [1 2;3 4]; size(A) =` [2 2]

All numeric data are matrices

```
A = 2; size(A) =
```
[1 1]

```
A = [1 2;3 4]; size(A) =
```
[2 2]

```
A(2,:) =
```

All numeric data are matrices

```
A = 2; size(A) =
```
[1 1]

```
A = [1 2;3 4]; size(A) =
```
[2 2]

```
A(2,:) =
```
[3 4]

All numeric data are matrices

`A = 2; size(A) =` [1 1]

`A = [1 2;3 4]; size(A) =` [2 2]

`A(2,:) =` [3 4] `A(:,2) =`

All numeric data are matrices

```
A = 2; size(A) =
```
[1 1]

```
A = [1 2;3 4]; size(A) =
```
[2 2]

```
A(2,:) =
```
[3 4]

```
A(:,2) =
```
[2; 4]

All numeric data are matrices

```
A = 2; size(A) =
```
[1 1]

```
A = [1 2;3 4]; size(A) =
```
[2 2]

```
A(2,:) =
```
[3 4]

```
A(:,2) =
```
[2; 4]

```
length(A) =
```

All numeric data are matrices

`A = 2; size(A) =` [1 1]

`A = [1 2;3 4]; size(A) =` [2 2]

`A(2,:) =` [3 4]   `A(:,2) =` [2; 4]

`length(A) =` [2]

All numeric data are matrices

`A = 2; size(A) =` [1 1]

`A = [1 2;3 4]; size(A) =` [2 2]

`A(2,:) =` [3 4] `A(:,2) =` [2; 4]

`length(A) =` [2] `numel(A) =`

All numeric data are matrices

```
A = 2; size(A) =
```
[1 1]

```
A = [1 2;3 4]; size(A) =
```
[2 2]

```
A(2,:) =
```
[3 4]

```
A(:,2) =
```
[2; 4]

```
length(A) =
```
[2]

```
numel(A) =
```
[4]

All numeric data are matrices

`A = 2; size(A) =` [1 1]

`A = [1 2;3 4]; size(A) =` [2 2]

`A(2,:) =` [3 4]   `A(:,2) =` [2; 4]

`length(A) =` [2]   `numel(A) =` [4]

`A(1:4) =`

All numeric data are matrices

`A = 2; size(A) =` [1 1]

`A = [1 2;3 4]; size(A) =` [2 2]

`A(2,:) =` [3 4] `A(:,2) =` [2; 4]

`length(A) =` [2] `numel(A) =` [4]

`A(1:4) =` [1 3 2 4]

Matlab is column-major!

$$\begin{matrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{matrix} \qquad \xcancel{\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}}$$

Vectorization

Vectorization

```
sum_v = 0
for i = 1:10
   sum_v = sum_v + v(i)
end
```

Vectorization

```
sum_v = 0
for i = 1:10
    sum_v = sum_v + v(i)
end
```

NEVER DO THIS

Vectorization

```
sum_v = 0
for i = 1:10
    sum_v = sum_v + v(i)
end
```

NEVER DO THIS

<u>EVER</u>

Vectorization

```
sum_v = sum(v)
```

Vectorization

```
sum_v = sum(v)
```

What if min(size(v)) > 1?

Vectorization

```
sum_v = sum(v)
```

What if min(size(v)) > 1?

```
1 4 7
2 5 8
3 6 9
```

```
sum(v, 1)
```

Vectorization

```
sum_v = sum(v)
```

What if min(size(v)) > 1?

[6 15 24]

1	4	7
2	5	8
3	6	9

```
sum(v, 1)
```

Vectorization

`sum_v = sum(v)`

What if min(size(v)) > 1?

[6 15 24]

1	4	7
2	5	8
3	6	9

`sum(v, 1)`

1	4	7
2	5	8
3	6	9

`sum(v, 2)`

Vectorization

```
sum_v = sum(v)
```

What if min(size(v)) > 1?

[6 15 24]

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
sum(v, 1)
```

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

[12;
15;
17]

```
sum(v, 2)
```

Vectorization Doesn't Always Work

Vectorization Doesn't Always Work

Vectorization Doesn't Always Work

```
for i = 1:numel(dishes)
  wash_dish(dishes(i))
end
```

Vectorization Doesn't Always Work

```
for i = 1:numel(dishes)
  wash_dish(dishes(i))
end
```

Vectorization Doesn't Always Work

```
for i = 1:numel(dishes)
  wash_dish(dishes(i))
end
```

```
wash(dishes)
```

Vectorization Doesn't Always Work

```
for i = 1:numel(dishes)
  wash_dish(dishes(i))
end
```

```
wash(dishes)
```

Ask Piazza #matlab if you have questions!

Logical indexing

Logical indexing

```
for i = 1:num_animals
    if weight(i) > 30
        is_dog(i) = true
    else
        is_cat(i) = true
    end
end
```

Logical indexing

```
for i = 1:num_animals
    if weight(i) > 30
        is_dog(i) = true
    else
        is_cat(i) = true
    end
end
```

NEVER DO THIS

Logical indexing

```
for i = 1:num_animals
    if weight(i) > 30
        is_dog(i) = true
    else
        is_cat(i) = true
    end
end
```

NEVER DO THIS

EVER

Logical indexing

```
is_dog = weight > 30;
is_cat = ~is_dog;
```

Logical indexing

```
is_dog = weight > 30;
is_cat = ~is_dog;

A = mean(weight(is_dog));
```

Logical indexing

```
is_dog = weight > 30;
is_cat = ~is_dog;

A = mean(weight(is_dog));
```

- computes mean dog weight

Logical indexing

```
is_dog = weight > 30;
is_cat = ~is_dog;

A = mean(weight(is_dog));
```

- computes mean dog weight

```
weight(is_dog) = [];
```

Logical indexing

```
is_dog = weight > 30;
is_cat = ~is_dog;

A = mean(weight(is_dog));
```
- computes mean dog weight

```
weight(is_dog) = [];
```
- erases all dog weights

Logical Indexing Pitfalls

Logical Indexing Pitfalls

- `[0 1 1 0 0 1]` is NOT `logical`

Logical Indexing Pitfalls

- `[0 1 1 0 0 1]` is NOT `logical`
    - `[0 1 1 0 0 1]>0` IS `logical`

Logical Indexing Pitfalls

- `[0 1 1 0 0 1]` is NOT `logical`
  - `[0 1 1 0 0 1]>0` IS `logical`
- Size of indexing matrix and target matrix must be exactly equal

Logical Indexing Pitfalls

- `[0 1 1 0 0 1]` is NOT `logical`
  - `[0 1 1 0 0 1]>0` IS `logical`
- Size of indexing matrix and target matrix must be exactly equal
- Size of assignment target must be [0 0], [1 1] or equal

Logical Indexing Pitfalls

- `[0 1 1 0 0 1]` is NOT `logical`
  - `[0 1 1 0 0 1]>0` IS `logical`
- Size of indexing matrix and target matrix must be exactly equal
- Size of assignment target must be [0 0], [1 1] or equal
- Useful functions: `any()`, `all()`

Putting it all together

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        rollidx = find(y~=mode(y));
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

Putting it all together

logical indexing

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        rollidx = find(y~=mode(y));
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

Putting it all together

logical indexing

vectorization

```
tic
for r = 1:1000
    rollidx = 1:5;
    y = zeros(5,1);
    for t = 1:3
        y(rollidx) = floor(unifrnd( ...
            0,6,numel(rollidx),1))+1;
        rollidx = find(y~=mode(y));
    end
    yamslam(r) = all(y==y(1));
end
toc
disp(['Probability of yamslam: ' ...
    num2str(mean(yamslam))])
```

How we use Matlab in CIS520

How we use Matlab in CIS520

- **Homeworks will consist of 2 parts:**

How we use Matlab in CIS520

- **Homeworks will consist of 2 parts:**
  - Implement/test an ML algorithm on data

How we use Matlab in CIS520

- **Homeworks will consist of 2 parts:**
  - Implement/test an ML algorithm on data
  - Analysis your results in a **brief** report

How we use Matlab in CIS520

- **Homeworks will consist of 2 parts:**
  - Implement/test an ML algorithm on data
  - Analysis your results in a **brief** report
- **Final project will be a competition**

How we use Matlab in CIS520

How we use Matlab in CIS520

- We will cover submitting / testing Matlab assignments when first HW goes out

How we use Matlab in CIS520

- We will cover submitting / testing Matlab assignments when first HW goes out
- Knowing Matlab basics BEFORE starting assignments will make things easier

How we use Matlab in CIS520

- We will cover submitting / testing Matlab assignments when first HW goes out
- Knowing Matlab basics BEFORE starting assignments will make things easier
- Expect to spend >10 hrs per assignment

What you should do next

- Things not covered today: plotting, data input/output, debugging, etc...
- Follow the online Matlab materials at: https://alliance.seas.upenn.edu/~cis520/wiki/index.php?n=Recitations.MatlabTutorial