MATLAB Introduction Course: Lecture 4

Øivind K. Kjerstad

10. October 2014

TOC

1. Probability and Statistics
2. Data Structures
3. Images and Animation
4. Debugging

Statistics

- Whenever analyzing data, you have to compute statistics
  - `scores = 100*rand(1,100);`
- Built-in functions
  - mean, median, mode
- To group data into a histogram
  - `>> hist(scores,5:10:95);`
  - makes a histogram with bins centered at 5, 15, 25 ... 95
  - `>> N=histc(scores,0:10:100);`
  - returns the number of occurrences between the specified bin edges 0 to <10, 10 to <20..90 to <100, you can plot these manually:
  - `>> bar(0:10:100,N,'r')`

Random Numbers

- Many probabilistic processes rely on random numbers
- MATLAB contains the common distributions built in
  - `rand`
    - draws from the uniform distribution from 0 to 1
  - `randn`
    - draws from the standard normal distribution (Gaussian)
  - `random`
    - can give random numbers from many more distributions
    - see `doc random` for help
    - the docs also list other specific functions
- You can also seed the random number generators
  - `>> rand('state',0);`

Changing Mean and Variance

- We can alter the given distributions
  - `>> y=rand(1,100)*10+5;`
    - ★ gives 100 uniformly distributed numbers between 5 and 15
  - `>> y=floor(rand(1,100)*10+6);`
    - ★ gives 100 uniformly distributed integers between 10 and 15. `floor` or `ceil` is better to use here than `round`
  - `>> y=randn(1,1000)`
  - `>> y2=y*5+8`
    - ★ increases std to 5 and makes the mean 8-

Exercise 1

Exercise: Random walk

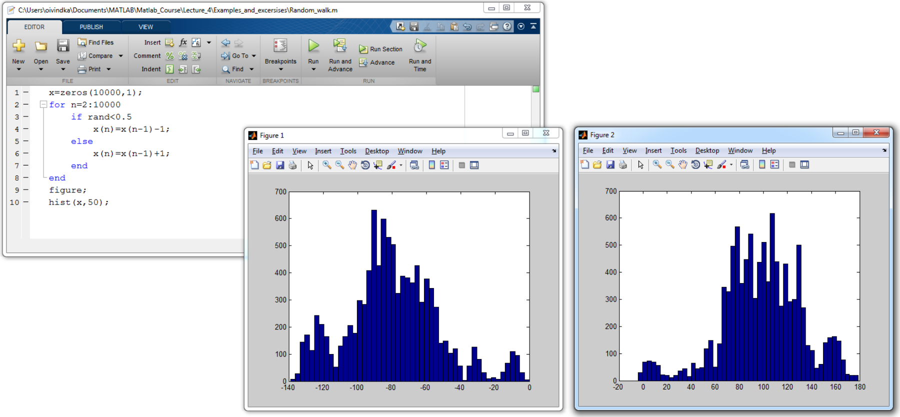
- Draw a random number in [0 1], if larger than 0.5 move 1 meter right, otherwise move 1 meter left
- Keep track of each new position in a vector
- Take 10.000 steps
- Plot the histogram of the positions

Proposed recipe

- 1 vector of size 1x10k
- 1 loop
- random numbers
- `hist`

- You may alter the recipe as you like

Solution



```
x=zeros(10000,1);
for n=2:10000
    if rand<0.5
        x(n)=x(n-1)-1;
    else
        x(n)=x(n-1)+1;
    end
end
figure;
hist(x,50);
```

- Notice that the histogram will be different each time

TOC

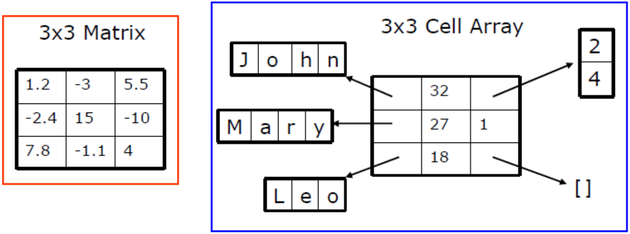
1. Probability and Statistics
2. Data Structures
3. Images and Animation
4. Debugging

Advanced Data Structures

- We have used 2D matrices
  - Can have n-dimensions
  - Every element must be the same type (ex. integers, doubles, characters...)
  - Matrices are space-efficient and convenient for calculation
  - Large matrices with many zeros can be made sparse
  - `a=zeros(100); a(1,3)=10; a(21,5)=pi; b=sparse(a)`
- Sometimes, more complex data structures are more appropriate
  - ★ **Cell array:** it's like an array, but elements don't have to be the same type
  - ★ **Structs:** can bundle variable names and values into one structure (Like object oriented programming in MATLAB)

Cell arrays

- A cell is just like a matrix, but each field can contain anything (even other matrices):



1.2	-3	5.5
-2.4	15	-10
7.8	-1.1	4

- One cell can contain people's names, ages, and the ages of their children
- To do the same with matrices, you would need 3 variables and padding

Cell arrays

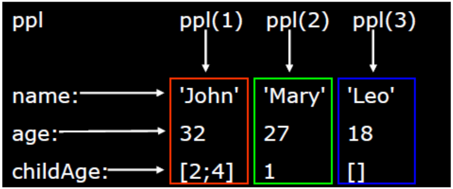
- To initialize a cell, specify the size
  - `>> a=cell(3,10);`
- Or do it manually, with curly braces {}
  - `>> c={'hello world',[1 5 6 2],rand(3,2)};`
  - c is a cell with 1 row and 3 columns
- Each element of a cell can be anything
- To access a cell element, use curly braces {}
  - `>> a{1,1}=[1 3 4 -10];`
  - `>> a{2,1}='hello world 2';`
  - `>> a{1,2}=c{3};`

Structs

- Structs allow you to name and bundle relevant variables
  - Like C-structs, which are objects with fields
- To initialize an empty struct:
  - `>> s=struct([]);`
    - ★ size(s) will be 1x1
    - ★ initialization is optional but is recommended when using large structs
- To add fields
  - `>> s.name = 'Jack Bauer';`
  - `>> s.scores = [95 98 67];`
  - `>> s.year = 'G3';`
    - ★ Fields can be anything: matrix, cell, even struct
    - ★ Useful for keeping variables together
- For more information, see `doc struct`

Struct Arrays

- To initialize a struct array, give field, values pairs
  - `>> ppl=struct('name',{'John','Mary','Leo'}...`
    `,'age',{32,27,18},'childAge',{[2;4],1,[]});`
    - size(s2)=1x3
    - every cell must have the same size
  - `>> person=ppl(2);`
    - person is now a struct with fields name, age, children
    - the values of the fields are the second index into each cell
- `>> person.name`
  - Returns 'Mary'
- `>> ppl(1).age`
  - Returns 32



Struct access

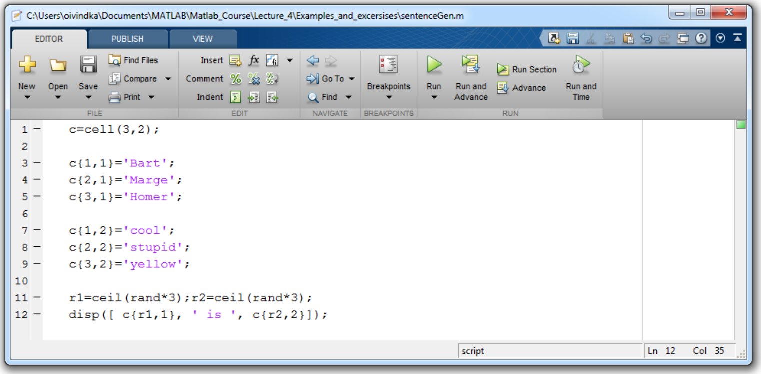
- To access 1x1 struct fields, give name of the field
  - `>> stu=s.name;`
  - `>> scor=s.scores;`
    - 1x1 structs are useful when passing many variables to a function. put them all in a struct, and pass the struct
- To access nx1 struct arrays, use indices
  - `>> person=ppl(2);`
    - person is a struct with name, age, and child age
  - `>> personName=ppl(2).name;`
    - personName is 'Mary'
  - `>> a=[ppl.age];`
    - a is a 1x3 vector of the ages; this may not always work, the vectors must be able to be concatenated

Exercise 2

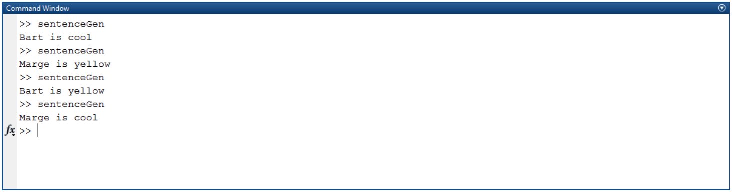
Cells

- Create a script called `sentenceGen`
- Make a 3x2 cell, and put three **names** into the first column, and **adjectives** into the second column
- Pick two random integers in [1 2 3]
- Display/print a sentence of the form '*name* is *adjectives*.'
- Run the script a few times

Exercise 2 solution



```
c=cell(3,2);

c{1,1}='Bart';
c{2,1}='Marge';
c{3,1}='Homer';

c{1,2}='cool';
c{2,2}='stupid';
c{3,2}='yellow';

r1=ceil(rand*3);r2=ceil(rand*3);
disp([ c{r1,1}, ' is ', c{r2,2}]);
```



```
>> sentenceGen
Bart is cool
>> sentenceGen
Marge is yellow
>> sentenceGen
Bart is yellow
>> sentenceGen
Marge is cool
>> 
```

TOC

1. Probability and Statistics
2. Data Structures
3. Images and Animation
4. Debugging

Reading/Writing Images

- Load/Read images into MATLAB with `imread`
  - it supports most image formats
  - jpeg, tiff, gif, bmp, png, hdf, pcx, xwd, ico, cur, ras, pbm, pgm, ppm
  - see `doc imread`
- Create/Write images with `imwrite`
  - see `doc imwrite`

Animations

- MATLAB makes it easy to capture movie frames and play them back automatically
- The most common movie formats are
  - avi
  - gif
- Avi
  - good when you have 'natural' frames with lots of colors and few clearly defined edges
- gif
  - Good for making movies of plots or text where only a few colors exist (limited to 256) and there are well-defined lines

Creating animations

Display in figure

```
for t=1:30
    imagesc(rand(200));
    colormap(gray);
    pause(.5);
end
```

Save as avi movie

```
for t=1:30
    imagesc(rand(200));
    colormap(gray);
    M(t) = getframe;
end

movie2avi(M,'myMov.avi');
```

- To create gifs use `imwrite`
- Movies and animations are useful when dealing with dynamic systems

TOC

1. Probability and Statistics
2. Data Structures
3. Images and Animation
4. Debugging

Display data in the command window

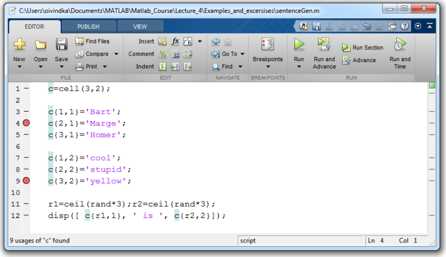
- When debugging scripts or functions, use `disp` (or `fprintf`) to print messages to the command window
  - `>> disp('starting loop')`
  - `>> disp('loop is over')`
    - ⋆ `disp` prints the given string to the command window
- It's also helpful to show variable values
  - `>> disp(strcat(['loop iteration ',num2str(n)]))`
  - `strcat` concatenates the given strings

Sometimes it's easier to remove some semicolons to print to the command window!

Debugging

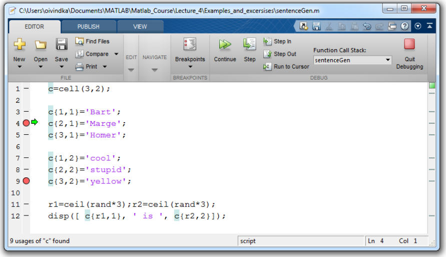
- To use the debugger, set breakpoints
  - Click on - next to line numbers in MATLAB files
  - Each red dot that appears is a breakpoint
  - Run the program
  - The program pauses when it reaches a breakpoint
  - Use the command window to probe variables
  - Use the debugging buttons to control debugger

Example



```
c=cell(3,2);

c{1,1}='Bart';
c{2,1}='Marge';
c{3,1}='Homer';

c{1,2}='cool';
c{2,2}='stupid';
c{3,2}='yellow';

r1=ceil(rand*3);r2=ceil(rand*3);
disp([ c{r1,1}, ' is ', c{r2,2}]);
```

9 usages of "c" found script Ln 4 Col 1

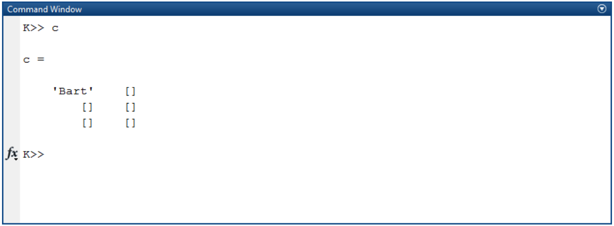


```
c=cell(3,2);

c{1,1}='Bart';
c{2,1}='Marge';
c{3,1}='Homer';

c{1,2}='cool';
c{2,2}='stupid';
c{3,2}='yellow';

r1=ceil(rand*3);r2=ceil(rand*3);
disp([ c{r1,1}, ' is ', c{r2,2}]);
```

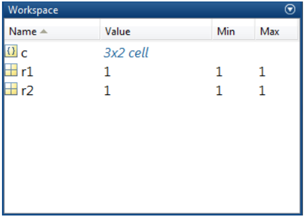
9 usages of "c" found script Ln 4 Col 1

- Insert breakpoints
- Run the code

- Notice that the program stops



```
K>> c

c =

    'Bart'     []
        []     []
        []     []

fx K>>
```



Name ▲	Value	Min	Max
c	3x2 cell		
r1	1	1	1
r2	1	1	1

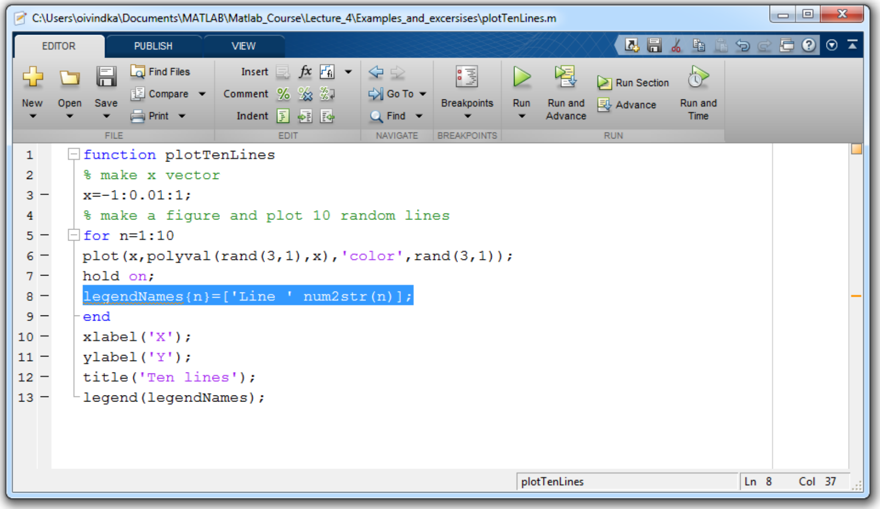
- Inspect variables in the workspace

Exercise

Debug the function

```
function plotTenLines

% make x vector
x=-1:0.01:1;

% make a figure and plot 10 random lines
for n=1:10
   plot(x,polyval(rand(3,1),x),’color’,rand(3,1));
   hold on;
   legendNames(n,:)=[’Line ’ num2str(n)];
end

xlabel(’X’);
ylabel(’Y’);
title(’Ten lines’);
legend(legendNames);
```

Solution



```
function plotTenLines
% make x vector
x=-1:0.01:1;
% make a figure and plot 10 random lines
for n=1:10
plot(x,polyval(rand(3,1),x),'color',rand(3,1));
hold on;
legendNames{n}=['Line ' num2str(n)];
end
xlabel('X');
ylabel('Y');
title('Ten lines');
legend(legendNames);
```

Performance Measures

- It can be useful to know how long your code takes to run
  - To predict how long a loop will take
  - To pinpoint inefficient code
- You can time operations using tic/toc:

```
>> tic;
>> CommandBlock1
>> a=toc;
>> CommandBlock2
>> b=toc;
```

  - `tic` resets the timer
  - Each `toc` returns the current value in seconds
  - You may have multiple tocs per tic
- You may also use the MATLAB profiler

```
>> THE END
```

Next week there is no lecture, the final lecture will be announced on email