MATLAB

Introduction, Part II

CS101 Lecture #22

2016-12-12

Administrivia

Administrivia

- Last Lab session this Wednesday.
- Next Wednesday for Q&A.
- Homework #9 this Thursday.
- Grading policy for Labs/homeworks: drop the lowest grade; 25% for Homeworks; 30% for Final

Warmup Questions

Question #1

$$\begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix}$$

How can we produce this array?

A `ones(3,3) - 2*eye(3,3)`

B `ones(3,3) + 2*eye(3,3)`

C `2*ones(3,3) + eye(3,3)`

D `2*ones(3,3) - eye(3,3)`

Question #1

$$\begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix}$$

How can we produce this array?

A `ones(3,3) - 2*eye(3,3)`

B `ones(3,3) + 2*eye(3,3)`

C `2*ones(3,3) + eye(3,3)`

D `2*ones(3,3) - eye(3,3)` $\star$

Question #2

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

How do we access 6 in this array?

A `A(2,1)`
B `A(1,2)`
C `A(3,2)`
D `A(2,3)`

Question #2

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

How do we access 6 in this array?

A `A(2,1)`
B `A(1,2)`
C `A(3,2)` $\star$
D `A(2,3)`

MATLAB cont.d

Basics

```
a = [ 1 2 3 ]; %row vector
b = [ 1 2 3 ]'; %column vector
A = [ 1 2 3 ; 4 5 6 ]; %matrix
B = [ a ; b ];
```

Indexing arrays

- In more dimensions:

```
A = [ 1,2,3 ; 4,5,6 ; 7,8,9 ];
B = A( 1:2,1:2 );
C = A( :,1:2 );
D = A( :,1:2:end)    % start:interval:stop
```

Array Indexing

- We can slicing an array with an array of indices.

```
A = 0:10:100;
B = A( [ 5,9,2,2,5 ] );
```

Matrix–Vector Operations

- If A is an m × n matrix (i.e., with n columns), then the product A x is defined for n × 1 column vectors x . If we let A x = b , then b is an m × 1 column vector. In other words, the number of rows in A (which can be anything) determines the number of rows in the product b. `http://mathinsight.org/matrix_vector_multiplication`

Matrix–Vector Operations

- Identity matrix does not move things

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

```
[ 1 0 ; 0 1 ] * [ 2 3 ]'
```

Matrix–Vector Operations

- Identity matrix does not move things

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

```
[ 1 0 ; 0 1 ] * [ 2 3 ]'
```

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2 + 3 * 2 \\ 2 + 3 \end{pmatrix} = \begin{pmatrix} 8 \\ 5 \end{pmatrix}$$

```
[ 1 2 ; 1 1 ] * [ 2 3 ]'
```

Matrix–Vector Operations

- Identity matrix does not move things

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

```
[ 1 0 ; 0 1 ] * [ 2 3 ]'
```

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2+3*2 \\ 2+3 \end{pmatrix} = \begin{pmatrix} 8 \\ 5 \end{pmatrix}$$

```
[ 1 2 ; 1 1 ] * [ 2 3 ]'
```

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 2+3+0 \\ 0+3+1 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \end{pmatrix}$$

```
[ 1 1 0; 0 1 1] * [ 2 3 1]'
```

Matrix multiplication

$$\overset{A}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}} * \overset{B}{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}} = \overset{C}{\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}}$$

$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31} + a_{24} * b_{41}$$

Matrix multiplication

$$\overset{A}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}} * \overset{B}{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}} = \overset{C}{\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}}$$

$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31} + a_{24} * b_{41}$$

- Matrix multiplications are matrix–vector operations:

Matrix multiplication

$$\overset{A}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}} * \overset{B}{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}} = \overset{C}{\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}}$$

$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31} + a_{24} * b_{41}$$

- Matrix multiplications are matrix–vector operations:
  - `A*B(:,1) = C(:,1)`

Elementwise operations

- Elementwise operations are spreadsheet-like operations:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix}$$

Elementwise operations

- Elementwise operations are spreadsheet-like operations:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix}$$

```
[ 1 0 ; 0 1 ] .* [ 2 4 ; 3 5 ]
```

Multiple return values

- Functions can return several values.

```
function [ a,b ] = func( x )
    a = x ^ 2;
    b = x ^ 3;
end

[ q r ] = func( 3 )
```

for statement

- The `for` loop iterates over a set of possible values.
- We create a `for` loop as follows:
  - start with `for var = range`, where you create `var` and provide `range`
  - a **block** of code
  - closing statement `end`

for statement

- The `for` loop iterates over a set of possible values.
- We create a `for` loop as follows:
  - start with `for var = range`, where you create `var` and provide `range`
  - a **block** of code
  - closing statement `end`
- Also have `continue` and `break` available.
- No colons

```
sum = 0;
for i = 1:100
    sum = sum + i^2;
end
```

Example: Finite difference

```
%% set parameters
a = -9.8;
tmax = 0.5;     % maximum time (s)
dt = 0.01;      % time step (s)

%% data initialization
t = 0:dt:tmax;                % (s)
v = zeros(size(t));           % (m/s)
y = zeros(size(t));           % (m)
y(1) = 1;
```

Example: Finite difference

```
%% loop through time steps
for i = 2:length(t)   %or numel
     v(i) = v(i-1) + a*dt;
     y(i) = y(i-1) + v(i-1)*dt;
end
```

if/else statement

- We create an `if` statement as follows:
  - the keyword `if`
  - a logical comparison (more on this)
  - a **block** of code
  - the keyword `end`
- Also have `else` and `elseif` available.
- No colons

Example: absolute.m

```
function [ y ] = absolute( x )
    y = 0;
    if x >= 0
        y = x;
    else
        y = -x;
    end
end
```

while statement

- We create an `while` statement as follows:
  - the keyword `while`
  - a logical comparison
  - a **block** of code
  - the keyword `end`
- Also have `continue` and `break` available.
- No colons

The Art of MATLAB programming

- Rewrite for/while loops as built-in Matrix operations
  - for/while loops are slow in matlab
  - Matlab as a high-level language is overall slower than C/C++
  - However, its built-in matrix/vector operations are highly optimized and very fast!
- How?

The Art of MATLAB programming

- Example: compute the inner product of two vectors a and b

```
ans = 0;
for i = 1:length(a)
    ans = ans + a(i)*b(i);
end
```

- What other ways to do this?

The Art of MATLAB programming

- Exercise: find the closest number in `a` for each number in `b`

```
a = [51  47  53   2  21  39  57  20  31   7];
b = [56  75  13  30  35   8  30  28  90  93];
```

Logical statements

- MATLAB uses the `logical` type for boolean.

Logical statements

- MATLAB uses the `logical` type for boolean.
- A `logical` type is 1-byte long, has values 0/1:
  - 0 means `False`
  - 1 means `True`

Logical statements

- MATLAB uses the `logical` type for boolean.
- A `logical` type is 1-byte long, has values 0/1:
  - 0 means `False`
  - 1 means `True`
- Available logical operators include:
  - <, >, <=, >=, ==,˜=
  - `&&`, `&` for AND
  - `||`, `|` for OR

Logical statements

- MATLAB uses the `logical` type for boolean.
- A `logical` type is 1-byte long, has values $0/1$:
  - 0 means `False`
  - 1 means `True`
- Available logical operators include:
  - <, >, <=, >=, ==,˜=
  - `&&`, `&` for AND
  - `||`, `|` for OR
  - `&&`, `||` are called *short-circuit* logical operator

Logical statements

- MATLAB uses the `logical` type for boolean.
- A `logical` type is 1-byte long, has values 0/1:
  - 0 means `False`
  - 1 means `True`
- Available logical operators include:
  - <, >, <=, >=, ==,~=
  - `&&, &` for AND
  - `||, |` for OR
  - `&&, ||` are called *short-circuit* logical operator
  - Can use logical operators for indexing!

Slicing an array with logical operators

- Slicing an array with logical operators.

```
A = rand(10,1) - rand(10,1);
B = A( A < 0 ); %select the negative values from A
A( A<0 ) = 0;  %set negative values in A to 0
```

File I/O

File I/O

- Saving data uses `'save'`:
  ```
  A = [ 1 2 3 ; 4 5 6 ];
  B = 5;
  save( 'myvariables', 'A', 'B' );
  ```
- Note that the *string* version of the variable name is required.

File I/O

- Saving data uses `'save'`:
  ```
  A = [ 1 2 3 ; 4 5 6 ];
  B = 5;
  save( 'myvariables', 'A', 'B' );
  ```
- Note that the *string* version of the variable name is required.
- `'load'` to load the variables from saved file:
  ```
  all = load('myvariables')
  load( 'myvariables', 'A' );
  ```

File I/O

- load (write) matrix data from (to) txt file:

```
M = dlmread( 'rawmatdata.txt' );
dlmwrite(filename, M);
```

File I/O

- load (write) matrix data from (to) txt file:

  ```
  M = dlmread( 'rawmatdata.txt' );
  dlmwrite(filename, M);
  ```

- ASCII-delimited *numeric* data.
- Automatically detect delimiter in file, or user specified

File I/O

- load (write) matrix data from (to) txt file:

```
M = dlmread( 'rawmatdata.txt' );
dlmwrite(filename, M);
```

- ASCII-delimited *numeric* data.
- Automatically detect delimiter in file, or user specified
- Another tool to use: `importdata`

File I/O

- load (write) matrix data from (to) txt file:

  ```
  M = dlmread( 'rawmatdata.txt' );
  dlmwrite(filename, M);
  ```
- ASCII-delimited *numeric* data.
- Automatically detect delimiter in file, or user specified
- Another tool to use: `importdata`
- Old process using `fopen`, `fprintf`, `fclose` also common.

Images!

- Images can also be opened as files.

```
A = imread( 'duck-color.jpg' );
imshow(A);
```

Images!

- Images can also be opened as files.

```
A = imread( 'duck-color.jpg' );
imshow(A);
```

- A (raster) image is a grid of pixels, the size of the grid is called the *resolution* (number of samples in X/Y).
- Gray images uses a single value to denote the *grayscale* for each pixel.
- Color images usually use 3 values (R,G,B) for each pixel. (Why?)

Images!

- Images can also be opened as files.

```
A = imread( 'duck-color.jpg' );
imshow(A);
```

- A (raster) image is a grid of pixels, the size of the grid is called the *resolution* (number of samples in X/Y).
- Gray images uses a single value to denote the *grayscale* for each pixel.
- Color images usually use 3 values (R,G,B) for each pixel. (Why?)
- Methods to display an image: `imshow`, `imagesc`, `pcolor`.

Applications: Image processing

- Example 1: Adjust brightness
- Example 2: Resize an image

Adjust brightness

```
A = imread( 'duck-color.jpg' );
A = im2double(A);   %convert value to 0 - 1
B = A.^0.5;
C = A.^2;
figure; imshow(A);
figure; imshow(B);
figure; imshow(C);
```

Resize an image

- Reduce image resolution (make it smaller):

Resize an image

- Reduce image resolution (make it smaller):

```
A = imread( 'duck-color.jpg' );
B = A(1:2:end, 1:2:end, :);
figure; imshow(A);
figure; imshow(B);
```

Resize an image

- Reduce image resolution (make it smaller):

```
A = imread( 'duck-color.jpg' );
B = A(1:2:end, 1:2:end, :);
figure; imshow(A);
figure; imshow(B);
```

- How to increase the image resolution (add pixels to it)?

Resize an image

- Reduce image resolution (make it smaller):

  ```
  A = imread( 'duck-color.jpg' );
  B = A(1:2:end, 1:2:end, :);
  figure; imshow(A);
  figure; imshow(B);
  ```

- How to increase the image resolution (add pixels to it)?

  See 'img_upsample.m'

Plot

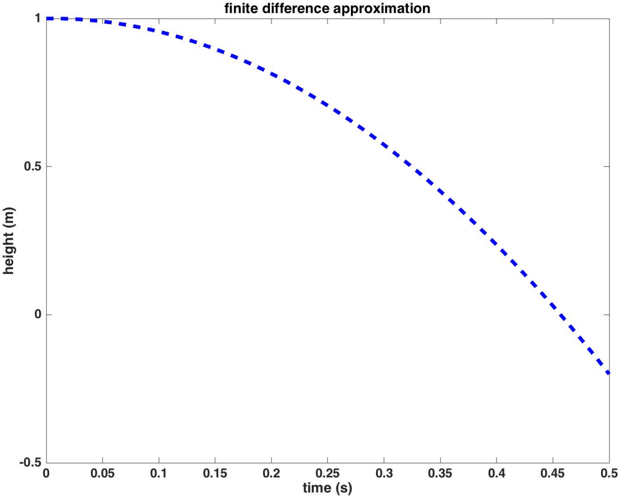
Plot in Matlab

- Open a figure: `h = figure(i)`
- Plot in a figure: `plot`
- Set Title: `title`
- Set x/y label: `xlabel`, `ylabel`
- Set range of plot: `axis([x_min, x_max, y_min, y_max])`
- Hold on for multiple plots: `hold on`
- Set a figure to current active window: `figure(h)`

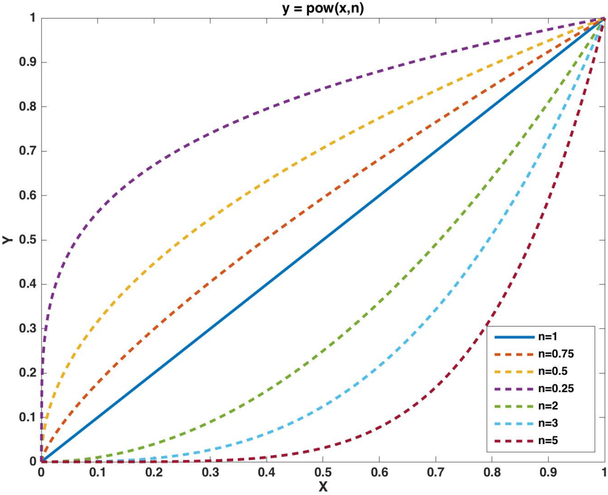
Example: plot trajectory

```
>> finite_difference
%t,y computed for object falling trajectory

h = figure;
plot(t,y, 'b--', 'linewidth', 4);
title('finite difference approximation');
xlabel('time (s)');
ylabel('height (m)');
axis([0, 0.5, -0.5, 1]);
```

Example: plot trajectory



Example: plot the color mapping curves



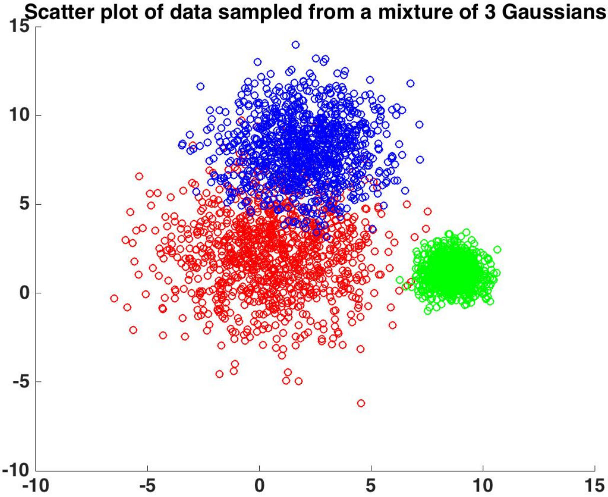
See `'plot_xpow_n.m'`

scatter plot

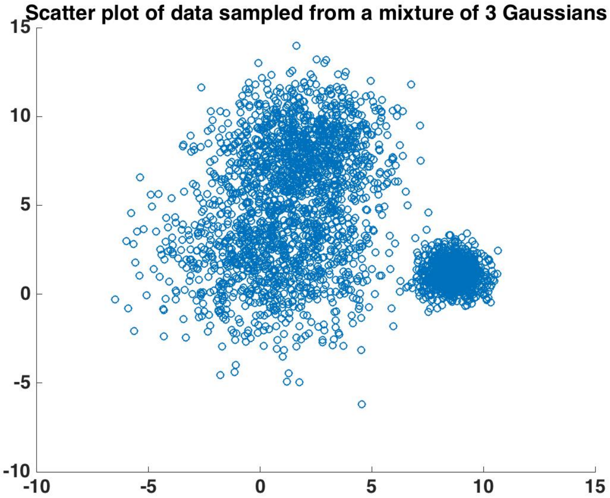
- draw data from a Mixture of Gaussians

```
C = 3;
center = rand(C,2)*10;
sigma = abs(randn(C,1))

Xs = [];
Ys = [];
for i = 1:C
    c = center(i,:);
    s = sigma(i);

    Xs = [Xs; c(1) + s*randn(1000,1)];
    Ys = [Ys; c(2) + s*randn(1000,1)];
end
scatter(Xs(1:1000), Ys(1:1000), 'r');
scatter(Xs(1001:2000), Ys(1001:2000), 'g');
scatter(Xs(2001:3000), Ys(2001:3000), 'b');
```

Result


Scatter plot of data sampled from a mixture of 3 Gaussians
15
10
5
0
-5
-10
-10
-5
0
5
10
15

Result


Scatter plot of data sampled from a mixture of 3 Gaussians
15
10
5
0
-5
-10
-10
-5
0
5
10
15