Octave and MATLAB

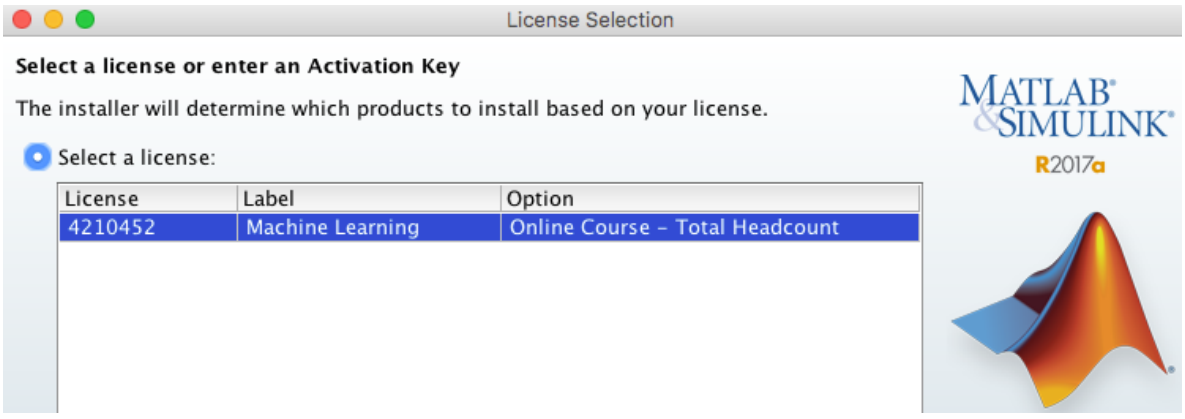
Octave installation

- GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically.
- Octave is a free alternative to MATLAB
- For more info http://octave.org
- Octave is a good way to easily apply ML concepts
- Download and install Octave with GUI 4.0.3 from

https://sourceforge.net/projects/octave/files/Octave%20MacOSX%20Binary/2016-07-11-binary-octave-4.0.3/octave_gui_403_appleblas.dmg/download

Matlab installation

- The MATLAB platform is optimized for solving engineering and scientific problems. The matrix-based MATLAB language is the world's most natural way to express computational mathematics.
- Matlab is a good way to easily apply ML concepts
- Download and install 12 months licensed version
- Run the installer and log in as g.filosofi@me.com and select license id 4210452



- Products installed: MATLAB 9.2 (R2017a), Optimization Toolbox 7.6, Statistics and Machine Learning Toolbox 11.1

Octave/MATLAB misc commands

Many commands are similar in Octave and MATLAB. Here is a sample

- to clear existing <variablename> use

clear <variablename>

- to clean the terminal screen

clc

- to suppress the printout add a semicolon. Example

A = ones(3,4);

- disp() prints results. sprintf() allows formatted string

```
>> a = pi;
>> disp(a)
 3.1416
>> disp(sprintf('2 decimals: %0.2f', a))
2 decimals: 3.14
```

- 1 and 0 stands also for true and false

```
>> disp(3==3)
 1
```

- format keyword allows to switch from single to double precision

```
>> disp(a)
 3.1416
>> format long
>> disp(a)
 3.14159265358979
>> format short
>> disp(a)
 3.1416
```

- Enter a matrix

```
>> A = [1,2;3,4;5,6]
A =

   1   2
   3   4
   5   6
```

- In Octave/MATLAB the first index of a vector or matrix is 1, not 0.
- Create 1xn and nx1 matrices (row and column vectors)

```
>> v = [1,2,3]
v =

   1   2   3

>> w = [1;2;3]
w =

   1
   2
   3
```

- Create a sequence of numbers from 1 to 2, equally spaced by 0.1

```
>> v = 1:0.1:2
```

- Create matrices with all 0s, all 1s or random

```
>> zeros(1,3)
ans =

   0   0   0

>> ones(1,3)
ans =

   1   1   1

>> rand(1,3)
ans =

   0.288444   0.907745   0.013835
```

- Use ... to continue the statement on the next line

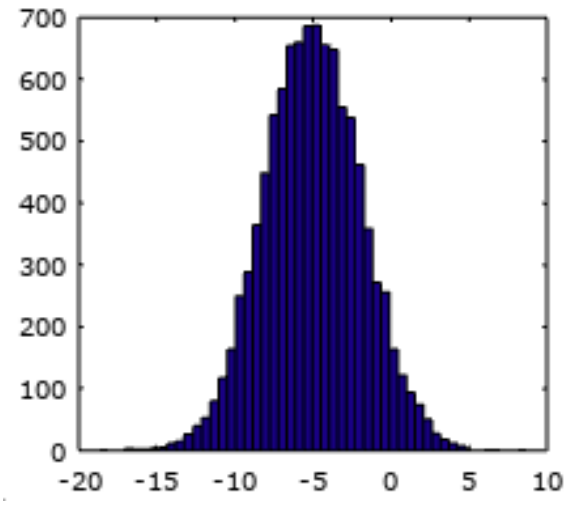
```
>> S = 3*ones(3,3) ...
- 2*eye(3);
>> S
S =

   1   3   3
   3   1   3
   3   3   1
```

- To create a random matrix with normal distribution (mean 0 and variance 1)

```
>> randn(1,3)
ans =

  -0.61154   0.79205  -1.01969
```

- To plot an histogram of 10000 values using 50 bins

```
>> w = -5 + sqrt(10)*randn(1,10000);
>> hist(w,50)
```



- The identity matrix

```
>> eye(4)
ans =

Diagonal Matrix

   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```

- To get the size of a matrix

```
>> size(A)
ans =

   3   2

>> size(A,1)
ans =  3
>> size(A,2)
ans =  2
```

for vectors we use the keyword *length* instead

- To generate a vector with random permutation of integers 1..N:

randperm(N);

- Now suppose you have two files in the working directory, featuresX.dat

and pricesY.dat

- In order to load those data in the workspace use *load*

```
>> load featuresX.dat
>> load priceY.dat
```

- Keywords *who* and *whos* give informations about the variables loaded in the current workspace

```
>> who
Variables in the current scope:

A          a          c          priceY     w          y
Z          ans        featuresX  v          x

>> whos
Variables in the current scope:

   Attr Name           Size                     Bytes  Class
   ==== ====           ====                     =====  =====
        A              3x2                         48  double
        Z              2x3                         48  double
        a              1x1                          8  double
        ans            1x2                         16  double
        c              1x1                          1  logical
        featuresX      8x2                        128  double
        priceY         8x1                         64  double
        v              1x5                         24  double
        w              1x10000                  80000  double
        x              1x4                         32  double
        y              1x10000                  80000  double

Total is 20049 elements using 160369 bytes
```

- To delete a variable use *clear <variable>*. *clear* without any arguments deletes all variables
- To save a variable in a file use *save*

```
>> v = priceY(1:4)
v =

   3000
   2000
   3500
   2900

>> save price-subset.m v;
```

- To save in a readable format use this

```
>> save price-subset.txt v -ascii;
```

- Other operations on matrices

```
>> A = [1,2;3,4;5,6]
A =

   1   2
   3   4
   5   6

>> A(:,2)
ans =

   2
   4
   6

>> A(2,:)
ans =

   3   4

>> A(:)
ans =

   1
   3
   5
   2
   4
   6
```

- The semicolon means “go to the next line”
- The colon means “every element in that dimension”
- The dot in front of a operator means “perform the operation elementwise”

```
A =

   1   2
   3   4
   5   6

>> B
B =

   11   12
   13   14
   15   16

>> A.*B
ans =

   11   24
   39   56
   75   96
```

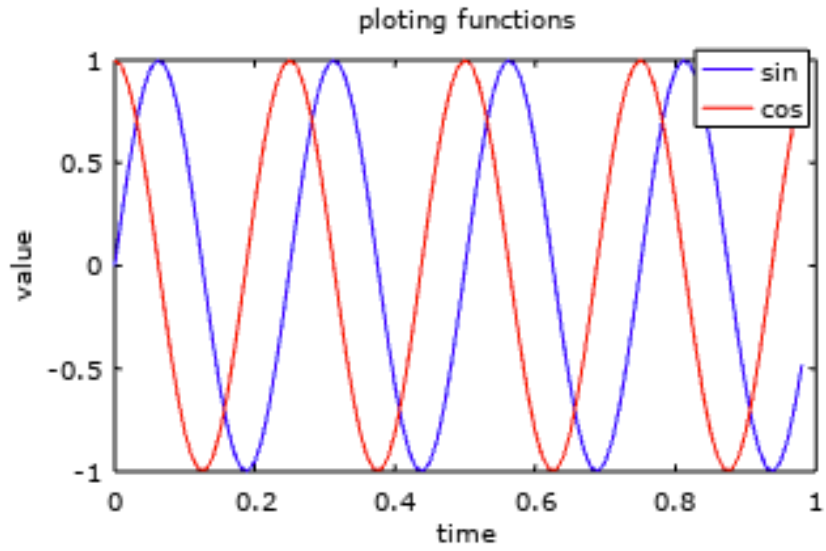
```
>> v = [1; 2; 3]
v =

   1
   2
   3

>> 1./v
ans =

   1.00000
   0.50000
   0.33333
```

- some more functions

```
>> v = [1 .3 2 12]
v =

    1.00000    0.30000    2.00000   12.00000

>> sum(v)
ans =  15.300
>> prod(v)
ans =  7.2000
>> floor(v)
ans =

    1    0    2   12

>> ceil(v)
ans =

    1    1    2   12

>> max(v)
ans =  12
>> [val,ind] = max(v)
val =  12
ind =  4
```

- flipud(A) : gives A with columns flipped up side down
- We have pinv(A) and inv(A). The first one gives you a value of inverse of A even if A is not invertible
- Plotting functions

```
>> t = 0:0.01:0.98;
>> y1 = sin(2*pi*4*t);
>> plot(t,y1);
>> hold on;
>> y2 = cos(2*pi*4*t);
>> plot(t,y2,'r');
>> xlabel('time');
>> ylabel('value');
>> legend('sin','cos');
>> title('ploting functions');
```



- Use *close* to remove the figure, *clf* to delete the plot leaving the figure
- changing directors and save the currently displayed plot as a PNG image

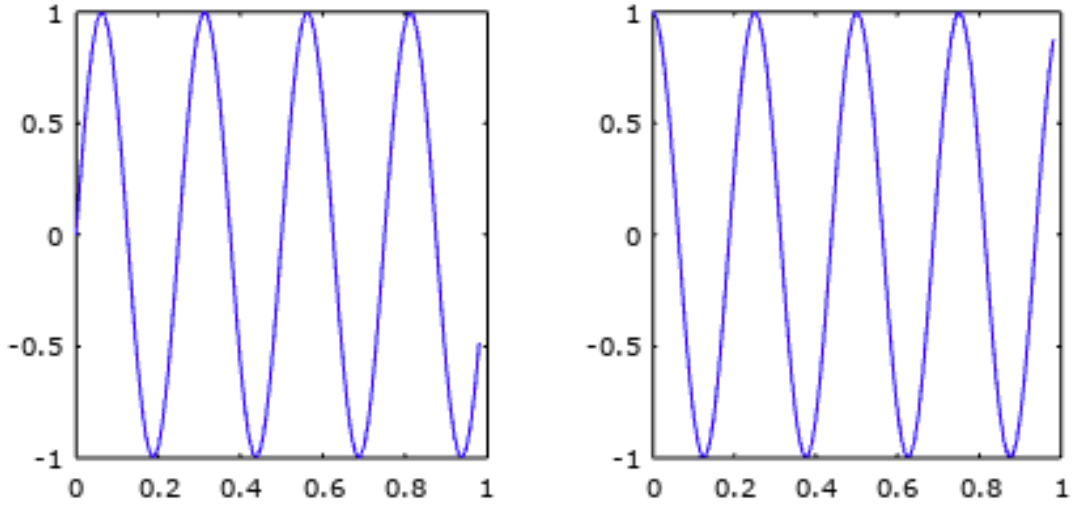
```
>> cd OctaveProjects/ ; print -dpng myPlot.png
```

- Plot two functions in two different windows

```
>> figure(1); plot(t,y1);
>> figure(2); plot(t,y2);
```

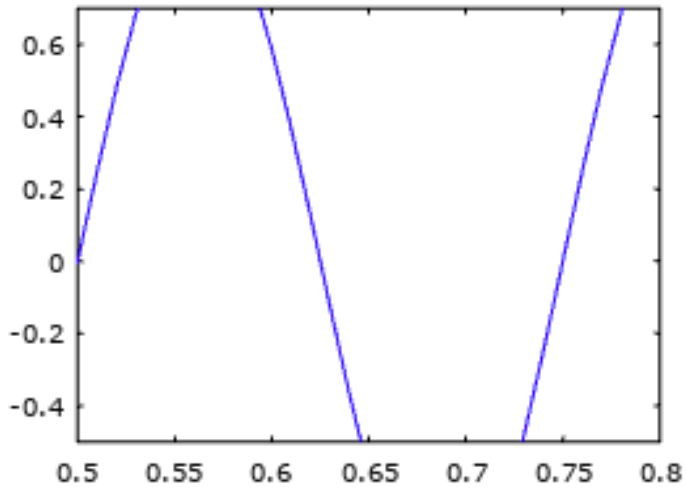
- Subplotting

```
>> subplot(1,2,1);
>> plot(t,y1);
>> subplot(1,2,2);
>> plot(t,y2);
```



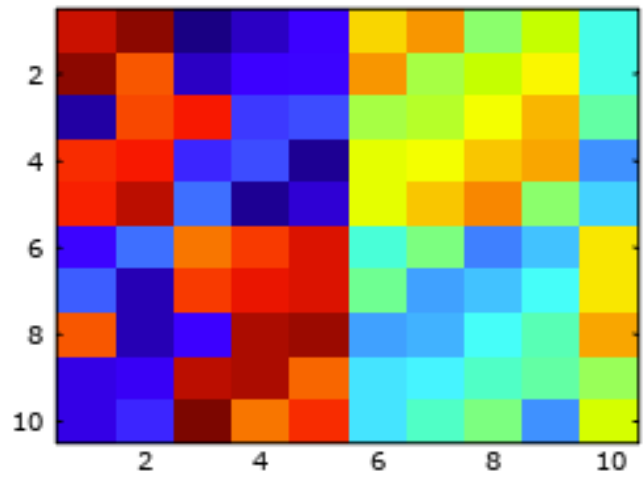
- Set the range for axis

```
>> plot(t,y)
>> axis([.5 .8 -.5 0.7])
```



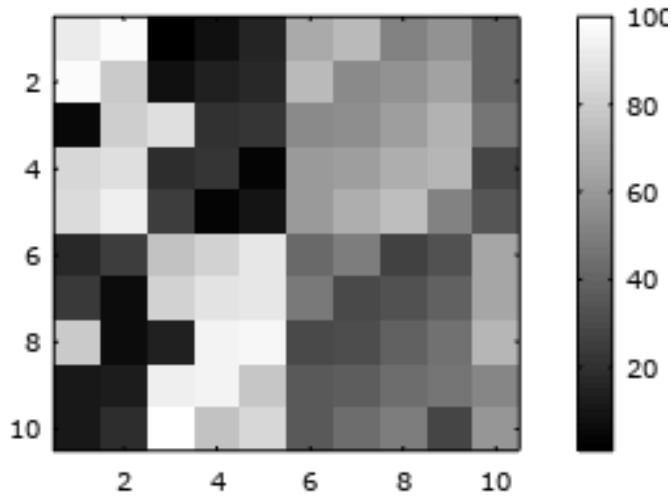
- Visualize a matrix with color code

```
>> A = magic(10)
A =

    92    99     1     8    15    67    74    51    58    40
    98    80     7    14    16    73    55    57    64    41
     4    81    88    20    22    54    56    63    70    47
    85    87    19    21     3    60    62    69    71    28
    86    93    25     2     9    61    68    75    52    34
    17    24    76    83    90    42    49    26    33    65
    23     5    82    89    91    48    30    32    39    66
    79     6    13    95    97    29    31    38    45    72
    10    12    94    96    78    35    37    44    46    53
    11    18   100    77    84    36    43    50    27    59

>> imagesc(A)
```



- You can also use a grey scale

```
>> imagesc(A), colorbar, colormap gray;
```



- Unrolling and reshaping matrices

```
>> A = magic(3)
A =

   8   1   6
   3   5   7
   4   9   2

>> unrolledA = A(:)
unrolledA =

   8
   3
   4
   1
   5
   9
   6
   7
   2

>> reshapedA = reshape(unrolledA(1:9),3,3)
reshapedA =

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

- For cycle

```
>> v = ones(1,10)
v =

   1   1   1   1   1   1   1   1   1   1

>> for i=1:10,
v(i) = 2^i;
end;
>> v
v =

      2       4       8      16      32      64     128     256     512    1024
```

- while loop and break statement

```
v =

   1   1   1   1   1   1   1   1   1   1

>> i=1;
>> while true,
   v(i)=999;
   i += 1;
   if i > 5,
      break;
   end;
end;
>> v
v =

   999   999   999   999   999     1     1     1     1     1
```

To define a function edit a <filename>.m file in a <dirname> directory, for example *squareThisNumber.m*, like this

squareThisNumber.m

```
1  function y = squareThisNumber(x)
2  y = x^2;
```

then call the function from the Command Window

```
>> s = 3;
>> squareThisNumber(s)
ans =  9
```

Note: to make <dirname> in the search path of octave, use
>> addpath('<dirpath>');
Example

```
>> addpath('/Users/gabrielefilosofi/OctaveProjects/');
```

- Let's define the cost function for a linear regression model for the training set (1,-1),(2,3),(3,8)

costFunction.m

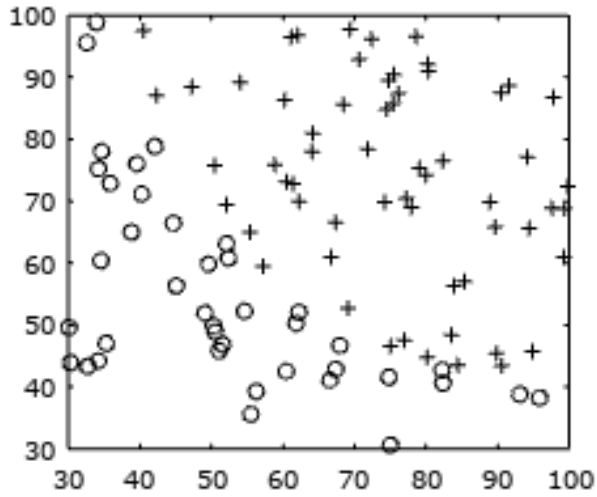
```
1  function J = costFunction(X, y, theta)
2  m = size(X,1); % number of training examples
3  deltas = (X*theta - y).^2;
4  J = 1/(2*m) * sum(deltas);
```

```
>> X = [1 1; 1 2; 1 3];
>> y = [-1; 3; 8];
>> theta = [0;1];
>> costFunction(X,y,theta)
ans =  5
```

- Let's define a function that plots the training set of some binary target function. The points are passed as rows of a nx2 matrix X, and the target function is passed as a vector y

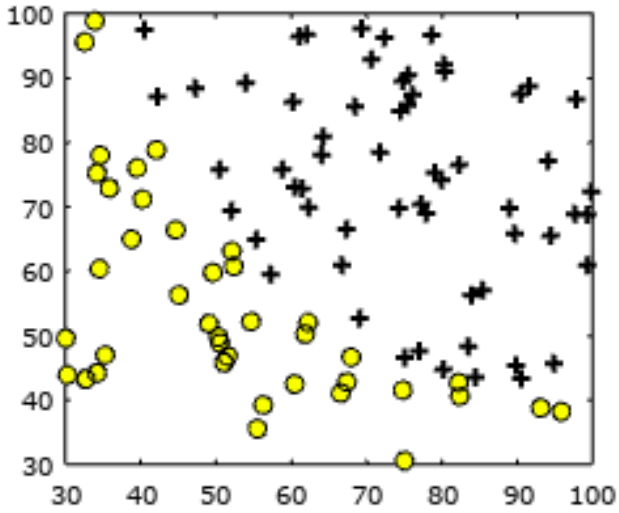
```
function plotData(X, y)
%PLOTDATA Plots the data points X and y into a new figure
%   PLOTDATA(x,y) plots the data points with + for the positive examples
%   and o for the negative examples. X is assumed to be a Mx2 matrix.

% Create New Figure
figure; hold on;

for i=1:size(X,1),
  if y(i) == 0,
    plot(X(i,1),X(i,2),'ko');
  else
    plot(X(i,1),X(i,2),'k+');
  end
end

hold off;
end
```

```
>> load ex2data1.txt
>> X = [ ex2data1(:,1) ex2data1(:,2) ];
>> y = ex2data1(:,3);
>> plotData(X,y)
```



The same problem could even be solved exploiting the find function, which returns a vector of indices of nonzero elements of a matrix, as a row if X is a row vector or as a column otherwise

```
>> pos = find(y == 1);
>> neg = find(y == 0);
>> hold  on;
>> plot(X(pos,1),X(pos,2), 'k+', 'LineWidth', 2, 'MarkerSize', 7);
>> plot(X(neg,1),X(neg,2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7);
```



- To put a pause in a script

```
fprintf("\nProgram paused. Press enter to continue.\n");
pause;
```

MATLAB MAT-files

You can save and load parts of variables directly in MAT-files without loading them into memory using the matfile function. The primary advantage of using the matfile function over the load or save functions is that you can process parts of very large data sets that are otherwise too large to fit in memory. When working with these large variables, read and write as much data into memory as possible at a time.

Example: let's create a version 7.3 MAT-file filled with some data

```
>> A = rand(5);
>> B = magic(10);
>> save example.mat A B -v7.3;
>> clear A B
```

Now create a matlab.io.MatFile object from the MAT-file, then load the first column of B, multiply by 2 and save.

```
>> exampleObject = matfile('example.mat');
>> firstRowB = exampleObject.B(1,:);
>> firstRowB = 2 * firstRowB;
>> exampleObject.B(1,:) = firstRowB;
Cannot change 'B' because Properties.Writable is false.  To modify 'B', set
Properties.Writable to true.
```

The save instruction failed because the MAT-file was read-only.
Set to true the Writable property of the file and repeat the operation.

```
>> exampleObject.Properties.Writable = true;
>> exampleObject.B(1,:) = firstRowB;
```