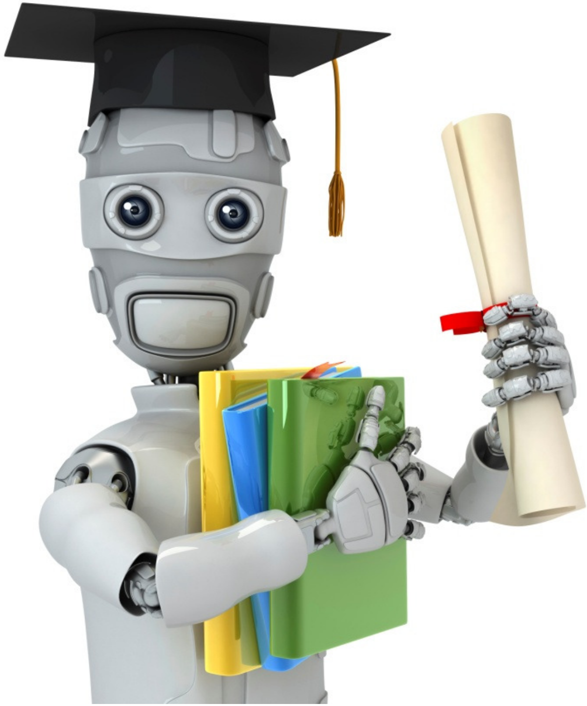


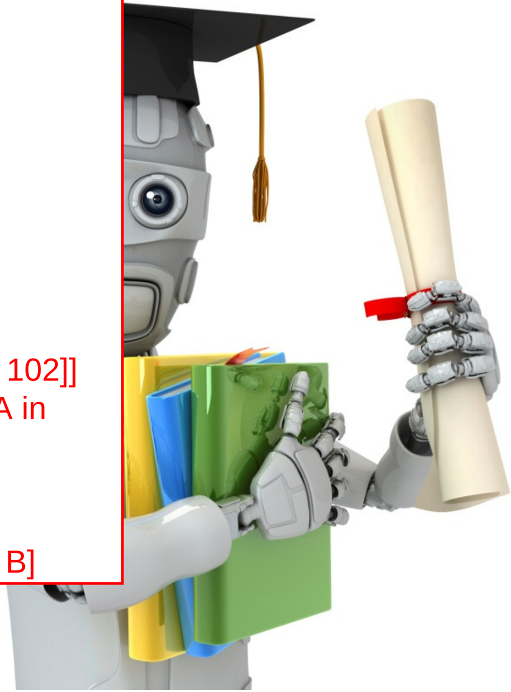
Machine Learning

# Octave Tutorial

# Basic operations

help

size;
length;
pwd;
cd;
ls;
load;
who;
whos;
clear;
save;
save -ascii
A(3,2)
A([1 3], :)
A = [A, [100; 101; 102]]
A(:) put all ele of A in single vector
C = [A B]
C = [A; B]
[A, B] same as [A B]



Machine Learning

# Octave Tutorial

## Moving data around

# Octave Tutorial

## Computing on data

A*B matrix multi
A .* B element multi
A .^ 2
log
exp
abs
v + ones(length(v),1) same as v+1
A' is transpose
max()
[val, ind] = max(a) gives value and index
a < 3 is element wise comparison
find (a < 3)
magic(3) to create a square matrix
[r,c] = find(A >= 7) give row and column of element
sum
prod gives product
floor
ceil
max(A,[],2)
max(A,[],1)
max(max(A)) or max(A(:))
flipup(eye(9))
pinv(A)

# Octave Tutorial

## Plotting data

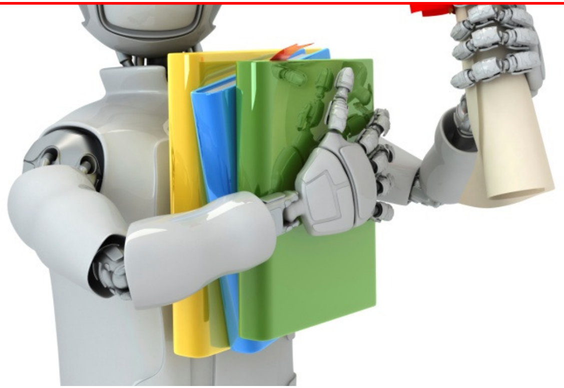
```
plot(t,y1,'r')
hold on;
xlabel('time')
legend('sin')
title('my plot')
print -dpng 'myplot.png'
close to close the figure
figure(1); plot(t,y1)
figure(2); plot(t,y2)
subplot(1,2,1);
axis([0.5 1 -1 1])
clf to clear the figure
imagesc(A) to visualize the matrix
colorbar
colormap gray
use ; to put multiple command together
```

Machine Learning

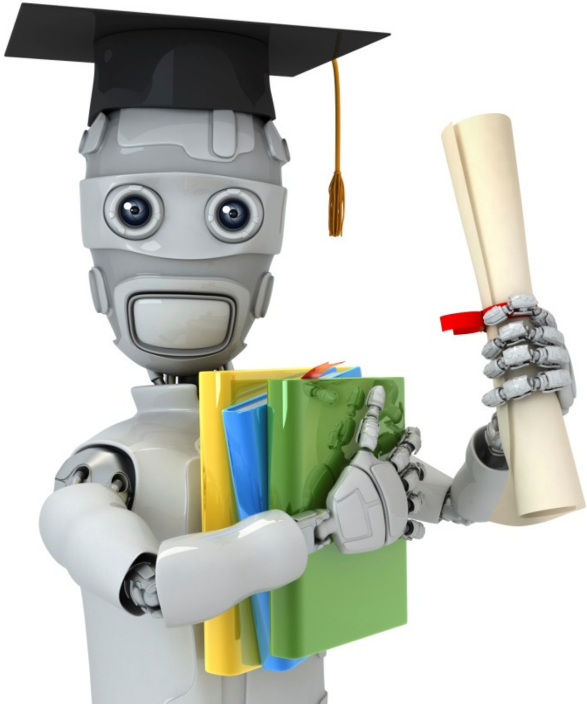
for
break
continue
while
if, elseif, else, end
quit
exit
create a file named function
addpath('c:\user/ang\desktop')
function [y1,y2] = dquareandcubethisnumber(x)

# Octave Tutorial

## Control statements: for, while, if statements



Machine Learning



Machine Learning

# Octave Tutorial

## Vectorial implementation

## Vectorization example.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \qquad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

$$h_\theta(x) = \sum_{j=\theta}^{n} \theta_j x_j$$

$$= \theta^T x$$

<u>Unvectorized implementation</u>

```
prediction = 0.0;
for j = 1:n+1,
  prediction = prediction +
               theta(j) * x(y)
end;
```

<u>Vectorized implementation</u>

```
prediction = theta' * x;
```

C++

# Vectorization example.

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

$$= \theta^T x$$

Unvectorized implementation

```
double prediction = 0.0;
for (int j = 0; j < n; j++)
  prediction += theta[j] * x[y];
```

Vectorized implementation

```
double prediction
   = theta.transpose() * x;
```

# Gradient descent

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for all $j$)

---

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$(n = 2)$

$u(j) = 2v(j) + 5w(j)$ (for all $j$)

$u = 2v + 5w$