Multiple Features:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_n x_n$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad x_0 = 1$$

$\rightarrow n+1$ dimention

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

$$h_\theta(x) = \theta^T x = \begin{bmatrix} \theta_0 & \theta_1 & \dots \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$= \theta_0 + \theta_1 x_1 + \dots \theta_n x_n$$

Now, $\theta \rightarrow n+1$ dimentional vector

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

New Algo for gradient descent:

repeat untill convergence { $\rightarrow \theta - \frac{\alpha}{m}(x'(x\theta - y))$

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

} simultaneously update θ_j for $j = 0 \dots n$

Feature Scaling: It is needed to make the gradient descent work well.

We will try to make every feature into approximately $-1 \leq x_i \leq 1$ range.

$0 \leq x \leq 3$ ✓

$-2 \leq x \leq 0.5$ ✓

$-100 \leq x \leq 200$ ✗

$0.0001 \leq x \leq 0.00001$ ✗

Mean normalization:

$$x_i = \frac{x_i - \mu_i}{s_i}$$

$\mu_i \rightarrow$ mean/average

$s_i \rightarrow$ range (max - min)

$s_i \rightarrow$ standard deviation

Debugging: Making sure gradient descent is working correctly.

Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration

If α is too

small: slow convergence

large: may not decrease on every iteration and thus may not coverge

Polynomial Regression:

We can change the behaviour of curve of our hypothesis by making it quadric, cube or square or any form.

$h_\theta(x) = \theta_0 + \theta_1 x$

$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ → $x_1 = 1 \to 10$, $x_2 = 1 \to 100$, $x_3 = 1 \to 10000$

$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x}$ → $x_1 = 1 \to 100$, $x_2 = \sqrt{x} = 1 \to 10$

Normal Equation:

Without taking derivative for many times, we can use the formula,

$$\theta = (X^T X)^{-1} X^T y$$

X = design matrix $= \begin{bmatrix} 1 & 1 & 1 & & 1 \\ x_0 & x_1 & x_2 & \cdots & x_n \\ 1 & 1 & 1 & & 1 \end{bmatrix}$

→ Here, feature scaling is not important.

→ Need to compute $(X^T X)^{-1} \rightarrow O(n^3)$

slow if n is very large

If n is large, gradient descent is better

$O(kn^2)$ ↙

If $(X^T X)$ is singular/degenerate NE → X

$(X^T X)$ is non invertible

- Some features are redundant → linearly dependent
- Too much features

Soln: delete some features

In octave 'pinve' function will give inverse, though (X^TX) is non-invertiable.
So we will use pinv to find NE.

Octave Tutorial:

== equal
~= not equal

xor(1,0) → 1

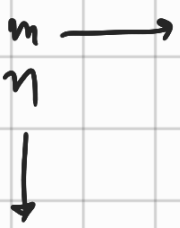
semicolon suppresses output

disp(_ _ _);
↳ output

disp(sprintf('2 decimals: %0.2f', pi));
↳ 2 decimals: 3.14

format long
a = pi;
a → 3._ _ _ ... 14 digit

ones (n x m)



zeros

rand

randn

sqrt

eye

help

size

length

load

who

whos

clear

save hello.txt ✓ -ascii %save as txt file

A(3,2) → an element

A(2,:) → second row

A(:,2) → second column

A([1 3],:) → everything from 1st & 3rd row

A(:,2) = [10; 11; 12] → assignment, them to 2nd column

```
A = [A, [100; 111; 122]]; % append new column vector to right
V = A(:) % put all element of A into a single vector
C = [A  B] % concatenate A & B matrix
C = [A; B] % concatenate vertically

A * C

A. * B  → elementwise multiplication
A.^2    →   "          square
A./V    →   "          division
log(v)  →   "          logarithm
exp(v)  →   "          e^
abs(v)  →   "          absolute value
 -V     → negative v
V + ones(length(v), 1) == v+1
A'      → Tranpose
max(A)
```

[val, ind] = max (A)

value ↵ ↳ index

a < 3 → elementwise comparison

find (a<3) → indexs of true values

magic (3)

[r, c] = find (A >= 7)

sum(a)

prod(a)

floor(a)

ceil (a)

max (rand (3), rand(3))

max(A, [], 1) → columnwise max

sum (a, 1) → columnwise summation

sum (a, 2) → rowise summation

sum(sum(A .* eye(9)))

flipud (eye (9)) → Anti diagonal Identity matrix of 9x9

pinv

sin

```
plot (x, y);
hold on;
plot (x, y, 'r');
                ↳ red color
xlabel ('time')
ylabel ('value')
legend ('sin', 'cos')
title ('my plot')
print -dpng 'myplot.png'
close
figure (1); plot (x, y1);
figure (2); plot (x, y2);
subplot (1,2,1) % divide plot a 1X2 grid.
                  access first element
plot ___
subplot (1, 2, 2) % access second element
plot ___
axis ([0.5 1  -1 1])
```

```
clf;
imagesec(A)
imagesec(A), colorbar, colormap gray;

for i=1:4,
    v(i) = 2 ^ i;
end;

v  →  2
      4
      8
     16

indices = 1:10;
for i = indices,
    disp(i);
end;

i=1
while true,
    v(i) = 999;
    if i == 6,
        break;
    end;
end;
```

```
if v(1)==1,
   disp(- - -);
elseif v(1)==2,
   disp(- - -);
else
   disp(- - -);
end;
```

Function:

return　　name　　parameter

```
function y = squareThisNumber(x)
y = x^2;   ← body
```

addpath('C:\...')

function [y1, y2] = ___ .

↳ return two numbers

vectorization → use LA.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad 3 \times 2 \qquad B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad 2 \times 3$$

$$A' = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \quad 2 \times 3$$