Andrew Ng ML

Coursera

Week 2

Linear regression with Multiple features

Imagine we have many features:

size x_1	# of bedrooms x_2	# of floors x_3	Age of home x_4	price y
2104	5	1	45	460
1416	3	2	40	232

n = number of features

m = data points or # of examples

$x^{(i)}$ = input (features) of ith training example

$x_j^{(i)}$ = value of feature j in ith training example

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix} \in \mathbb{R}^4$$

index into training set

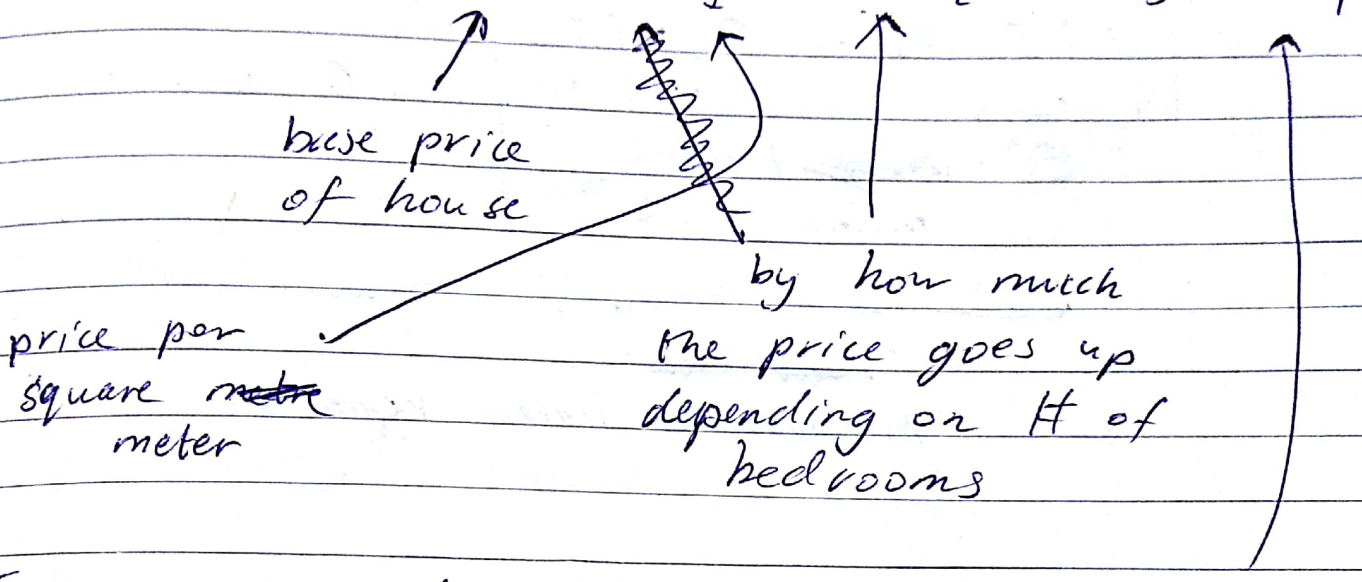
$$x_3^{(2)} = 2 \in \mathbb{R}$$

Note: all values, indexed are 1 indexed $i \& j$

Now, hypothesis :

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

Eg: $h_\theta(x) = 80 + 0.1 x_1 + 0.01 x_2 + 3 x_3 + 2 x_4$



For convenience of notation, define $x_0 = 1$

Then :

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

The hypothesis can now be expressed as such:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n =$$

$$= \underbrace{[\theta_0 \ \theta_1 \ \dots \ \theta_n]}_{\theta^T:\ (n+1) \times (n+1) \text{ matrix}} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

~~Multi-linear~~

Multivariate linear regression

Cost function with new notation:

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

θ: all parameter vector

Gradient descent:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

(simult. update for every $j = 0, \dots, n$)

}

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

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

$$= \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \frac{\partial}{\partial \theta_j} \left(\sum_{k=0}^{n} \theta_k x_k^{(i)} - y^{(i)} \right) =$$

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

Then gradient descent:

Repeat {

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

(simult update for every $j = 0, \ldots, n$)

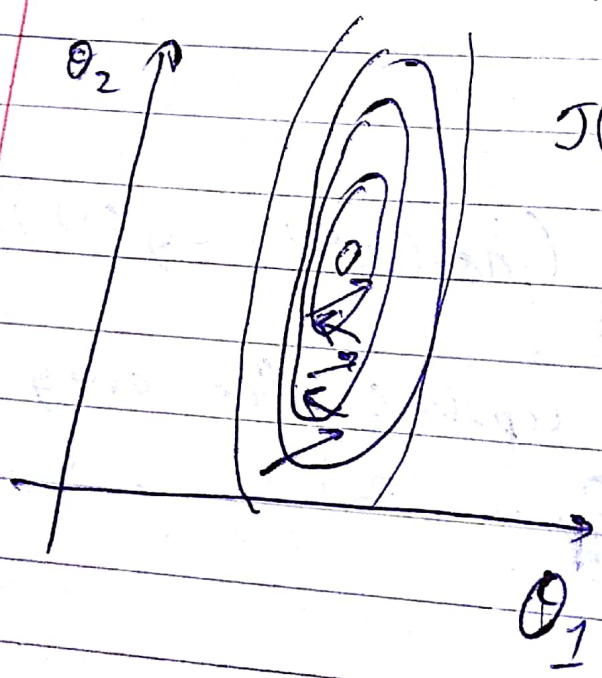
}

Gradient descent feature scaling

Idea: Make sure features are on a similar scale (take on similar range of values) then gradient descent converges quickly

e.g. x_1 = size (0 - 2000 $feet^2$)
x_2 = # of bedrooms (1-5)

suppose that we ignore θ_0 or $\theta_0 = 0$
then the cost ~~fo~~ function of x_1 & x_2 will be ~~et~~ plotted as such:

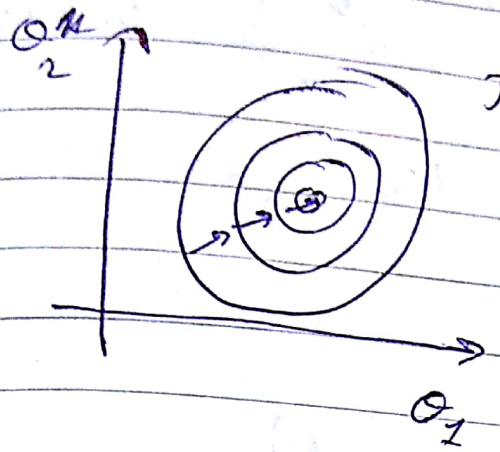


The steps tend to oscilate a lot hence it takes time to reach the global minimum. The taller & skinier the contours, the more steps will be needed

The axes must be inverted however (θ_1 & θ_2 must swap) as θ_2 will have a smaller range as compared to θ_1

Solution:

$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$, $x_2 = \frac{\text{\# of bedrooms}}{5}$



Once the values are on more similar scale the contour becomes less eliptical & less steps are needed (a more direct path is made)

Scale of x_1, x_2 now:

$0 \leq x_1 \leq 1$, $0 \leq x_2 \leq 1$

More generally people try to get every feature into approximately a $-1 \leq x_i \leq 1$ range but if the bound values vary a bit then it's not much cause for concern. General rule of thumb for bounds:

-3 to 3 or $-\frac{1}{3}$ to $\frac{1}{3}$

Anything more or less is usually considered poor scaling:

less than -3 & more than 3 OR

~~More tha~~ $-0.0001 \leq x_4 \leq 0.0001$ (wrong)

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately 0 mean
Thus values of x_i will be shifted as such:

From ~~0~~ $50 \leq x_i \leq 100$ to

$-25 \leq x_i \leq 25$

(Does not apply to $x_0 = 1$)

E.g. $x_1 = \frac{\text{size} - 1000}{2000}$ thus

if ~~$0 \leq$~~ $0 \leq \text{size} \leq 2000$ then

$-0.5 \leq \text{size} \leq 0.5$

General formula:

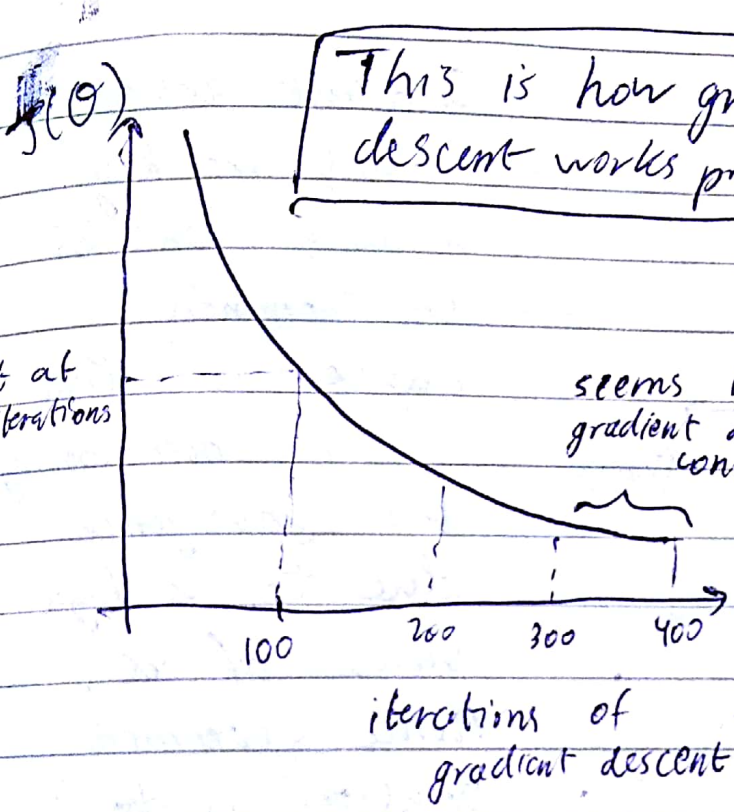
$$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$$

μ_1 ← avg value of x in training set

s_1 ↗ range or std deviation

Questions:

- How to make sure gradient descent is working correctly? "debugging"
- How to choose learning rate α?



This is how gradient descent works properly

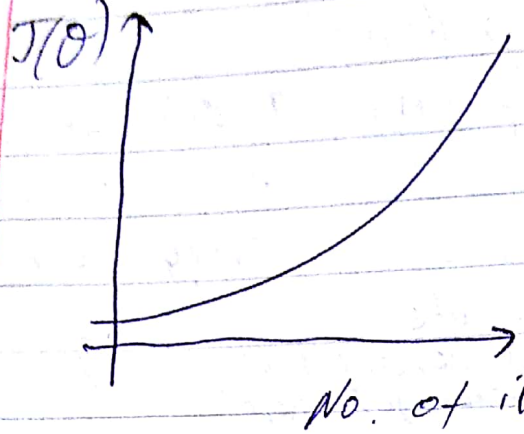
$J(\theta)$ should decrease after every iteration

seems like gradient descent has converged here

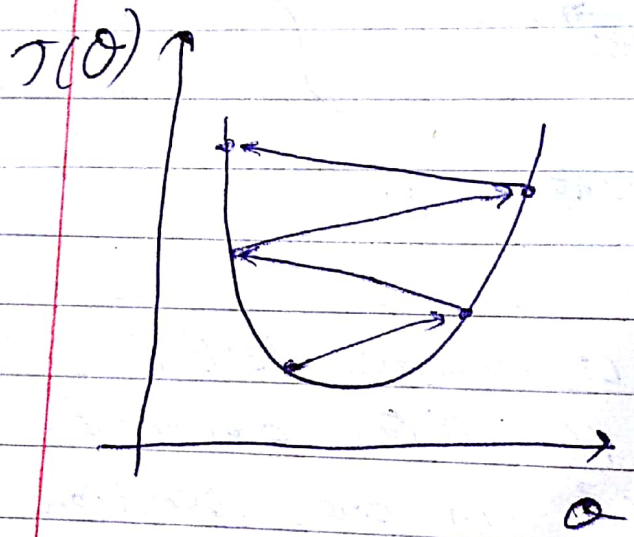
Example automatic convergence test:
Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration

↳ difficult to choose what this value is hence it is better to use this plot instead

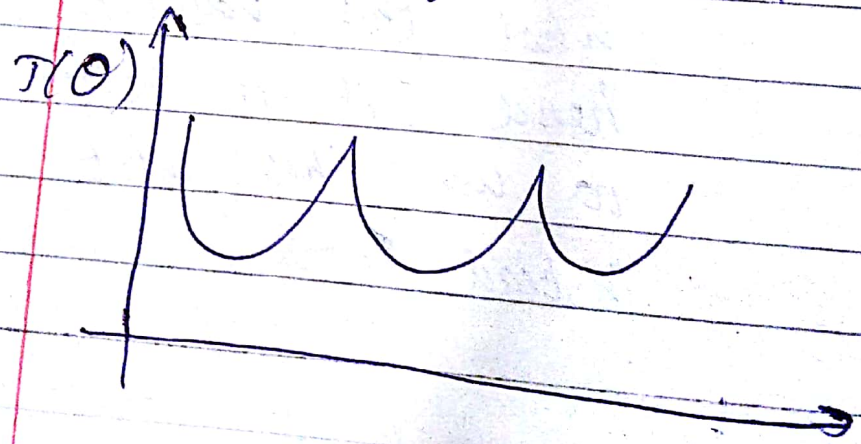
This is how gradient descent does not work properly



Gradient descent is not working properly here & the common cause is that it is overshooting the convergence due to a high value of α. Hence common solution is to decrease the value of α



Alternatively this can happen:



Use this solution for such a case

- For sufficiently small α, $J(\theta)$ should decrease on every iteration
- But if α is too small, gradient descent can be slow to converge

To choose α, try:

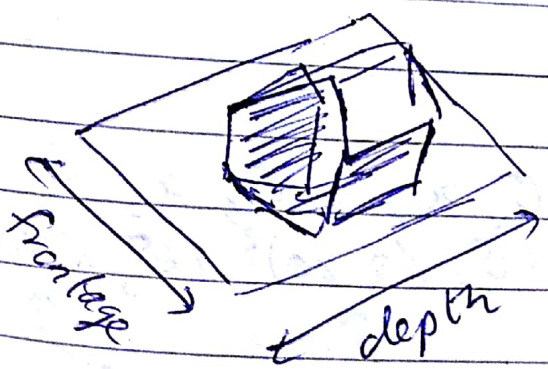
0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1

Features & Polynomial Regression

Housing price prediction

Suppose we have the following hypothesis:

$h_\theta(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$



Suppose you realise that area is a better predictor of price than frontage & depth hence you make a new feature – area

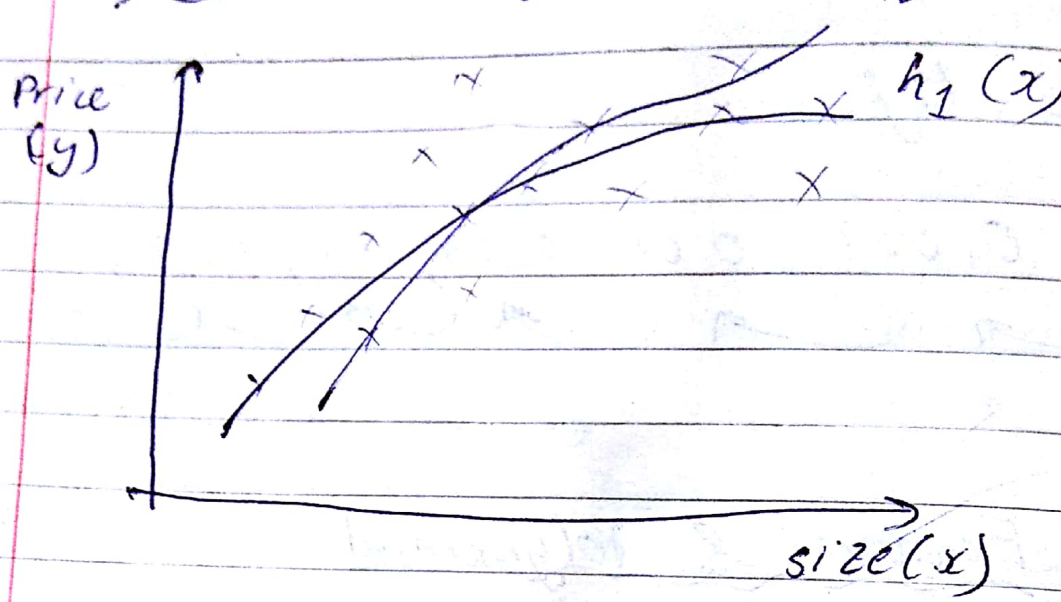
$\text{area} = \text{frontage} * \text{depth}$

Now change hypothesis:

$h_\theta(x) = \theta_0 + \theta_1 \times \text{area}$

New features might at times contribute to a better model

Polynomial regression



Depending on the data set you may choose to fit a quadratic or cubic model

$$h_1(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$
$$h_2(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 =$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (siz$$

just individual features

$x_1 = size$
$x_2 = size^2$
$x_3 = size^3$

When using polynomial regression the ranges can vary drammatically & hence feature scaling becomes much more important

size: 1 – 1000
$size^2$: 1 – 10^6
$size^3$: 1 – 10^9

You have an almost unlimited choice of ~~features~~ the type of features you can possibly use:

$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{size}$$

& etc

Algorithms exist that can tell you which features you can use

Normal Equation

~~Solve for~~ Method to solve for θ analytically

Intuition: If 1D ($\theta \in \mathbb{R}$)

$$J(\theta) = a\theta^2 + b\theta + c$$

$$\frac{d}{d\theta} J(\theta) = \dots \overset{\text{set}}{=} 0$$

↑ differential of $J(\theta)$ with respect to θ

Now solve for θ

$\theta \in \mathbb{R}^{n+1}$ ⇛

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

$$\frac{\partial}{\partial \theta_j} J(\theta) = \dots \overset{\text{set}}{=} 0 \quad \text{(for every } j\text{)}$$

Solve for $\theta_0, \theta_1, \dots, \theta_n$

Suppose the following dataset ($m = 4$):

x_0	size x_1	# bedrooms x_2	# floor x_3	age of home x_4	price y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	175

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

$m \times (n+1)$ matrix

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 175 \end{bmatrix}$$

m-dimensional vector

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

↳ gives value of θ that minimises cost function

General example :

m examples : $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \quad m \times (n+1) \text{ matrix}$$

design matrix

Eg if $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \\ 1 & x_1^{(m)} \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$\theta = (X^T X)^{-1} X^T y$ ← normal equation method

in octave this would be calculated as such:

pinv (X' * X) * X' * y

(X' = X^T; $(X^T X)^{-1}$; X' = X^T)

Feature scaling is not necessary for normal equation method as it is in gradient descent (no step by step iteration)

Disadvantages & Advantages

Gradient descent	Normal equation
• Need to choose a • Needs many iterations • Works well when n is large	• No need to choose a • No need to iterate • Need to compute $(X^T X)^{-1}$ $[(n+1) \times m$ by $m \times (n+1)]$ hence $(X^T X)^{-1}$ is approx $n \times n$ matrix • Slow if n is very large, Computing inverse is $O(n^3)$ of $(X^T X)$

inverting when n = 100 is no issue for current PCs, at n = 10000 gradient descent may become more viable

Summary - as long as the value of n is not too large, the normal equation serves as a great alternative to gradient descent

The normal equation does not work for all algorithms - while gradient descent works for all learning algo apart from regression

Normal equation & noninvertibility

What if X^TX is non-invertible in $(X^TX)^{-1}X^Ty$? (singular)

Octave has 2 functions for inverting matrices :

1) pinv (pseudo-inverse) - will compute inverse even when the matrix is singular
2) inv (inverse) - will not compute inverse when the matrix is singular

Hence calculate θ as such:

θ = pinv (X' * X) * X' * y

Causes of non-invertibility:

- Redundant features (linearly dependent)

  E.g. x_1 = size in $feet^2$

  x_2 = size in m^2

  $1m = 3.28$ feet

  Hence, $x_1 = (3.28)^2 x_2$

- Too many features (e.g. $m \leq n$)

  ↳ to remove this issue either delete some features or use regularization

  ↳ might not cause non-invertibility at times

  ↳ fit many features with few data points

If you find that $X^T X$ is singular then go through the possible causes & see if there is any linear dependence (delete one in this case). If not redundant then if $m \leq n$ delete features or apply regularization

Octave

- Comments start with '%'
- 1 not equal to 2 is:
  1 ~= 2 % true
  1 != 2 % true
- To not print out the result of a statement end it with a semicolon:
  a = 6;
- String assignment:
  b = 'hi';
- true is the same as 1
- false is the same as 0
- >> sprintf ('2 decimals: %0.2f', a)
  → 2 decimals: 0.31
- >> format long
  ↳ display all decimals & numbers in a bigger format
  >> format short
  ↳ display all decimals & numbers in a smaller format
- A = [1 2; 3 4; 5 6]
  ↳ $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$ matrix
- v = 1:0.1:2 → creates a row vector of 11 elements: [1.0 1.1 1.2 ... 1.9 2.0

>> v = 1:6

↳ v = [1 2 3 4 5 6]

>> ones (2, 3)

↳ ones = $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

>> 2 * ones (2, 3)

↳ $\begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$

>> zeros (1, 3)

↳ [0 0 0]

>> rand (2, 3)

↳ $\begin{bmatrix} 0.9498417 & 0.8200758 & 0.6006023 \\ 0.0008707 & 0.7265427 & 0.3778833 \end{bmatrix}$

Random numbers drawn from uniform distribution between 0 & 1

>> randn (1, 3)

↳ 1 by 3 matrix of values from gaussian distribution with mean 0 & ~~variance~~ std 1

>> hist (w) ↳ plot histogram of matrix w

>> hist (w, 50)

↳ plot histogram with 50 buckets

- >> eye(4)
  ↳ $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- help("eye") or help eye
  ↳ shows all the help needed for the eye command
- >> a = [1 2; 3 4; 5 6]
  >> size(a)
  ↳ returns [3 2]
  (3 = # of rows, 2 = # of columns)
- >> size(a, 1) → give back 1st. dimension which is #. of rows
- >> size(a, 2) → give back 2nd dimension which is # of columns
- >> length(v) → gives the size of the longest dimension
  ↳ usually applied to vectors
- >> pwd → current path of where octave is running
- use cd & ls to move around & print contents of directories

- >> load featuresX.dat
  OR
  >> load("featuresX.dat")
  will load ~~these files~~ this file from the current directory
- >> who
  ↳ shows the variables that are currently loaded in memory
- >> whos
  ↳ shows more detail on the variables loaded in memory
- >> clear a
  ↳ removes variable 'a' from loaded variables
- >> v = priceY(1:10)
  - v: choose the first 10 elements of vector priceY
  - priceY: 47×1 vector
- >> b = priceY(2)
  ↑ stores the second element of priceY in b
- >> save hello.dat v;
  ↳ will save the contents of v into a file called hello.dat in the current directory
  ↳ saves in a compressed format
- >> clear
  ↳ simply typing clear will remove all variables from memory

- >> save hello.dat v -ascii
  ↳ save the variable v in a human readable format in the file hello.dat

- >> a = [1 2; 3 4; 5 6]
  >> a(2,2)
  ↑ 2nd row, ↖ 2nd column
  a(2,2) = 4

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

- >> a(2, :)
  ↳ fetch everything in the 2nd ~~row~~
  a(2, :) ⇒ [3 4]

- >> a(:, 2)
  ↳ fetch ~~everything~~ in 2nd column

- >> a([1 3], :)
  ↑ get everything from the first & ~~second~~ third row (all columns)

- >> a(:, 2) = [10; 11; 12]
  ↳ assign the vector $\begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix}$ to the second column of a

>> a = [a, [100; 110; 101]]

if $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$ then it becomes $\begin{bmatrix} 1 & 2 & 100 \\ 3 & 4 & 110 \\ 5 & 6 & 101 \end{bmatrix}$

>> A(:) → put all elements into a single column vector. Suppose $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ then A(:) is $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$.

Suppose that a, b, c are matrices where A is 3×2, B is 3×2, C is 2×2

>> A * C → matrix ~~mult~~ mult of A & C

>> A .* B → element wise multiplication of A & B

~~period~~

period is used to denote element wise operations in octave

>> A .^ 2 → element wise ~~exponentiation~~ power

>> log (A) → element-wise logarithm

>> exp (A) → element-wise exponentiation

>> abs (A) → element wise absolute value

- >> a + 1
  - ↳ adds 1 to every element
- >> a'
  - ↳ transpose of matrix a
- >> max(v)
  - ↳ if $v = \begin{bmatrix} 1 \\ 4 \\ 5 \end{bmatrix}$ then will return 5
- >> [val, ind] = max(v)
  - val ↙ will store max value
  - ind ↘ will store index of max value
- >> max(A)
  - ↳ if $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$ then max(A) returns the column-wise maximum : [5 6] (max in each column)
- >> v < 3
  - ↳ does element-wise comparison
  - if $v = \begin{bmatrix} 1 \\ 4 \\ 5 \end{bmatrix}$ then v < 3 is $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

>> find([1 0 0 1 0 1])

↳ [1 4 6] - indicies where the 1s are present

>> magic(3)

↳ creates such a 3×3 matrix where the columns & rows sum up to ~~or~~ the same number (diagonals as well)

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

will hold the row & column value

A >= 7 creates a matrix where the values are 1 or 0 & the find ~~finds~~ returns the column & row index of the 1s

- >> sum(a) → sum of all elements in a
- >> prod(a) → product of all elements
- >> floor(a)
  ↳ rounds down the elements
- >> ceil(a)
  ↳ rounds up to the nearest integer

- `>> max(A, [], 1)`
  - ↳ takes the max of the first dimension (rows) ↳ default
- `>> max(A, [], 2)`
  - ↳ takes the max of the second dimension (columns)
- `>> sum(A, 1)`
  - ↳ sums all the row elements (columnwise)
- `>> sum(A, 2)`
  - ↳ sums all the column elements (rowwise)
- `>> pinv(A)`
  - ↳ pseudo inverse of A

```
>> t = [0.0:0.01:0.98]
>> y1 = sin(2*pi*4*t)
>> plot(t, y1)
```

t → x, y1 → y

```
>> y2 = cos(2*pi*4*t)
>> hold on;
```

keeps the previous plot plotted & any new plotting will be on the same graph

```
>> plot(t, y2, 'r')
```

↳ will plot cosine func on top of sine function in red color

```
>> xlabel('time')
>> ylabel('value')
```

} sets the labels of the axes

```
>> legend('sin', 'cos')
```

↳ labels the 2 lines, first one will be labeled 'sin' & the other 'cos'

```
>> title('my plot')
```

← title of the plot is set

```
>> print -dpng 'myplot.png'
```

↳ ~~file of~~ p image of plot saved as png by name of 'myplot'

>> close ← removes the plot window

- >> figure(1); plot(t, y1); } plots on
  >> figure(2); plot(t, y2); } 2 windows side by side

- >> subplot(1, 2, 1);

  divides window into 1×2 matrix

  select the first element in matrix & plot there

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

  ← select the 2nd element in matrix & plot there

>> plot(t, y2)
>> axis([0.5 1 -1 1])

  horizontal axes to 0.5 to 1 range

  vertical to -1 to 1 range

>> clf → clears a figure
  ↳ "clear figure"

>> imagesc (A) → plots a matrix A with the help of colours

>> imagesc(A), colorbar, colormap grey;

colorbar → add a bar that shows which colour corresponds to which value

colormap grey → colourscheme for connecting colours & values

- >> a = 1, b = 2, c = 3
  ↳ comma chaining where the results are printed out for each statement
- >> a = 1; b = 2; c = 3
  ↳ results not printed out but both method just simply run multiple statements one after the other

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

```
>> i = 1;
>> while i <= 5,
   v(i) = 100;
   i = i+1;
   end;
```

'continue' & 'break' statements are available as well

each block starts with a comma & ends with an 'end' statement

control constructs available:
- if,
- else
- while
- for
- ~~if else~~ elseif

```
>> if v(1) == 1,
     disp ('The value is one');
   elseif v(1) == 2,
     disp ('The value is 2');
   else
     disp ('The value is not 1 or 2');
   end;
```

↑ end only at the end of all of the if statements

Creating functions in octave:
Create a file with extension .m that will hold ~~name~~ code of function

function code in "SquareThisNumber.m":
function y = SquareThisNumber(x)
y = x^2;

y: the value returned
x: parameter passed

In Octave:
>> squareThisNumber(5)
ans = 25

add another search path

>> addpath('C:\Users\ang\Desktop')
↳ Octave will search this directory as well as the current directory when searching for files & etc (.m files from other locations can be used like this)

- Octave allows you to define functions that can return multiple values (not as such in other languages)

"SquareAndCubeThisNumber.m":
function [y1, y2] = squareAndCubeThisNumber(x)
y1 = x^2;
y2 = x^3;

Vectorization

If you use vectorization implementation (linear algebra libraries) you usually end up with highly optimised & simpler code.

Vectorization example:

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \quad \left.\right\} \text{non-vectorised}$$

$$= \theta^T x \quad \left.\right\} \text{vectorised}$$

Unvectorised implementation:

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

{ accounts for 1-indexed octave

Matlab & octave ~~are~~ have 1 indexed matricies hence θ_0 must be mapped to θ_1, θ_1 must be mapped to θ_2 & etc

Vectorised implementation:

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

Vectorize gradient descent:

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

for $j = 0, 1, 2 \quad (n = 2)$

Implementation:

$$\theta := \theta - \alpha \delta$$

θ: $\mathbb{R}^{n+1}$ vector; α: $\mathbb{R}$; δ: $\mathbb{R}^{n+1}$

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

$$\delta = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \end{bmatrix}$$

$$\delta_0 = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)} \quad (\mathbb{R}^{n+1})$$

and etc

$$\left[h_\theta(x^{(1)}) - y^{(1)} \right] \cdot x^{(1)} + \left[h_\theta(x^{(2)}) - y^{(2)} \right] \cdot x^{(2)} + \dots$$

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$$

$\mathbb{R}^{n+1}$