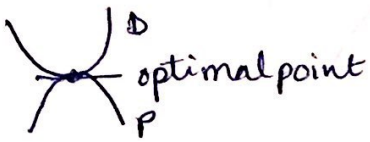
Primal $\equiv \max\{c^T x \mid Ax \leq b, x \geq 0\}$

Dual $\equiv \min\{b^T y \mid A^T y \geq c, y \geq 0\}$

→ Strong duality: $c^T x^* = b^T y^*$



→ Weak duality: $c^T x \leq b^T y$

$\forall$ feasible x & y.

$P = \max\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n\}$

$D = \min\{b^T y \mid A^T y = c, y \geq 0\}$

→ Let x^* be the primal optima

Let I be the set of constraints that are tight.

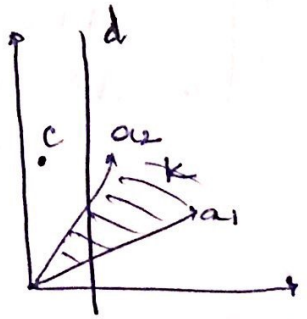
$a_i^T x^* = b, \quad i \in I$

$a_i^T x^* < b, \quad i \notin I.$

Claim: c is in $K = \{x \mid x = \sum_{i \in I} \lambda_i a_i, \lambda_i \geq 0\}$

Proof by contradiction ⇒

[if c is not in K, then there exists "d"]



d separates c from $\{a_i\}$

$a_i^T d \leq 0, \quad i \in I$

$c^T d \geq 0.$

$z = x^* + \varepsilon d$

$a_i^T z \leq b_i, \quad i \notin I$

$$a_i^T z = a_i^T (x^* + \varepsilon d)$$

$$= a_i^T x^* + \varepsilon \underbrace{a_i^T d}_{-ve}$$

↑ feasible point.

$z \rightarrow$ Not going to violate for all constraints.

Objective =

$$c^T z = c^T [x^* + \varepsilon d] = c^T x^* + \varepsilon \underbrace{c^T d}_{+ve}$$

z is superior to x^*

(z) doesn't violate constraints ↓

↓

has a better obj ; it contradicts our statement, x^* is optima.

→ Hence, there exists no such d and

our claim is true.

claim: c is in $K = \{x \mid x = \sum_{i \in I} \lambda_i a_i , \lambda_i \geq 0\}$

$\lambda \geq 0$; $\lambda_i = 0$, when $i \notin I$

$$A^T \lambda = \sum_i \lambda_i a_i = c$$

$$b^T \lambda = \sum_i \lambda_i b_i = \sum a_i^T x^* \lambda_i = c^T x^*$$

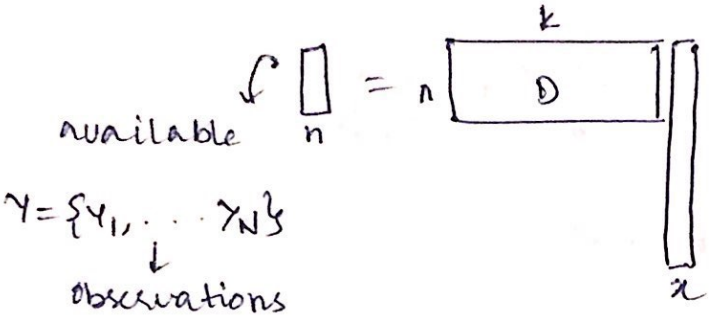
tight.

Sparse coding & Dictionary Learning →

Compact representation → complete basis (fourier..).

↓

Instead of using a standard basis, can we learn it?



N >> K

K >> n

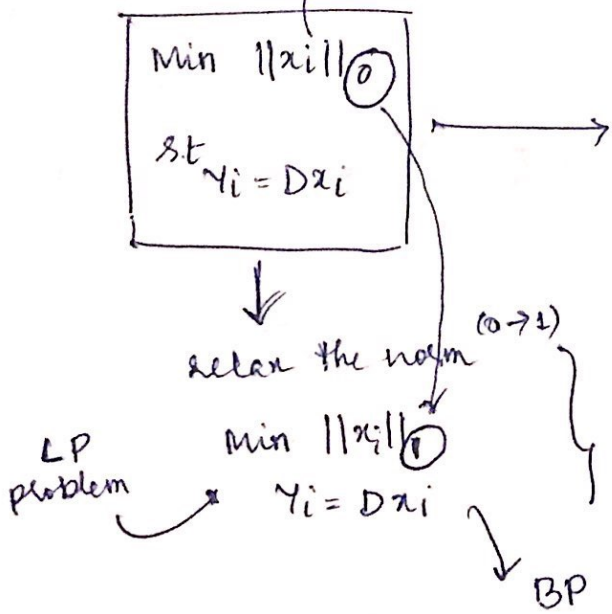
$$Y_i = D x_i$$

Problem :- Find x_i (for each Y_i)?

Sparse coding :- Find the sparsest x, s.t $Y_i = D x_i$

Dictionary Learning :- learn D (and x_i) from Y ⇒ given set of samples.

Y_i and D are given, **Sparse coding** ⇒ and corresponding coeff



OMP ⇒ Min $||Ax-b||$ ↘ minimise the error [residual]

s.t $||x||_0 \leq T$ (atmost T elements)

(1.) Select most correlated column, add to bases.
start with $\underline{Y}$

(2.) Find X(i) corresponding to the new basis.
And calculate the residual.

* if T is larger, error goes lower and lower.

K-SVD & K-Means :-

KMeans → dictionary element ⇒ such that mean square error is small.

↓ sparsest code [extreme]

↓ representation of the elements.

find the mean & encoding, given samples

↳ Ily, given Y, find D and X

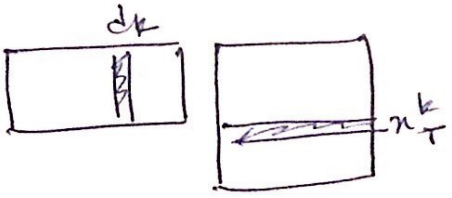
↑ the right dictionary.

$$\underset{Y}{[y_1 \dots y_N]} = \underset{\text{Dict } D}{[d_1 \dots d_K]} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & \vdots \end{bmatrix} \leftarrow e_i$$

code X

$Y = DX$

Min $||Y - Dx||_2$, $\quad Y - DX = Y - \sum_{i=1}^{k} d_k x_T^k$



$$Y - DX = Y - \sum_{i=1}^{k} d_i x_T^i$$

↳ T representation : row vector.

$$= \left[Y - \sum_{\substack{j=1 \\ i \neq j}}^{k} d_i x_i^T \right] - d_j x_T^j$$

$$= E - d_j x_T^j$$

E ↳ assume. except j, all are fixed

Compute SVD;

on computing SVD, x_T^j → not going to be sparse.

modify by retaining sparsity; Y_R

$$E_R - d_j x_T^j$$ ↳ only non-zero elements are considered

1. Initialize D (like means for K-means)
2. Do sparsecoding.
3. Find D with the same sparsity (find new set of means)
4. Repeat 2 & 3.

$$Y = DX$$

Sparse coding

Given Y_i & D, find sparsest X

→ greedy → OMP

→ L1 → BP (LP)

Dictionary Learning.

given Y;

Find D and X

Non linear Optimisation →

Convexity →

Convex set, $\equiv$ C

$x, y \in C$ & $\theta \in \mathbb{R}$

$\theta x + (1-\theta) y \in C$ [Convex combination]

↓

should also lie in the convex set

Convex function $\equiv$> $D(f) \rightarrow$ domain of f [convex]

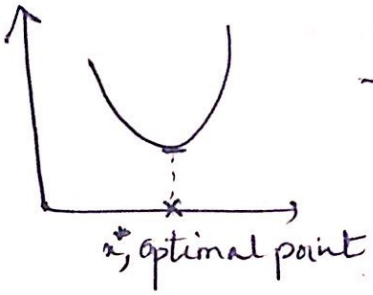
If $x, y \in D(f)$

$$f(\theta x + (1-\theta) y) \leq \theta f(x) + (1-\theta) f(y)$$

line should lie above the function.

Unconstrained –

$\min_x f(x)$



→ **Necessary condition**

$\nabla_x f(x) = 0$

↓

generally, not alone enough.

so, we need →

$\nabla_x^2 f(x) \geq 0$.

Constrained –

$\min_x f(x)$

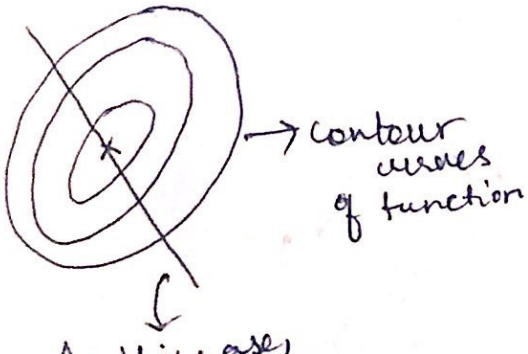
s.t $h(x) = 0$.

} **Necessary condition**

$h(x) = 0$.

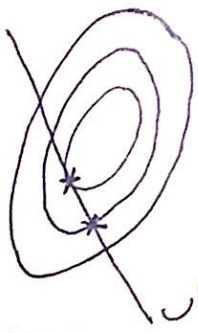
$\nabla_x f(x) = 0$.

* Contour curve – along the curve, value doesn't change



In this case, constraint doesn't matter.

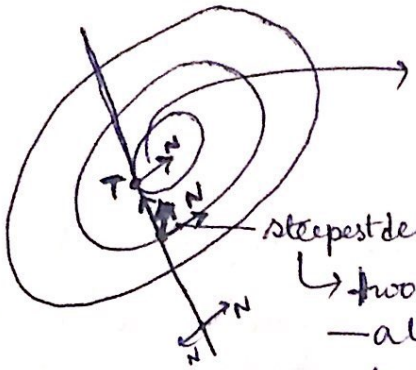
in this case →



2 conditions

① h(x) tangent to contours

② h(x) intersecting contours



→ orthogonal to both the contour curve and constraint

"optimal"

steepest descent

↳ two components

— along the constraint, we can reduce the function value, hence our point is not optimal

Tangent point $\Longrightarrow h(x) = 0$

$\hookrightarrow \quad \nabla_x f(x) + \lambda \nabla_x h(x) = 0$

$$\min_x f(x) \quad \text{s.t } h(x) = 0 \longrightarrow \text{unconstrained problem} \longrightarrow \min_x f(x) + \lambda h(x)$$

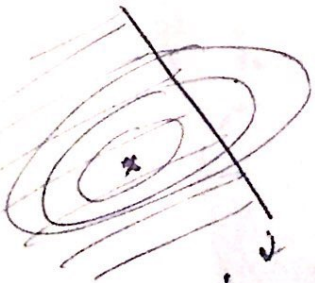
multiple constraints

$$\text{s.t } h_i(x) = 0, \quad i = 1, 2, \ldots m \longrightarrow \min_x f(x) + \sum_i \lambda_i h_i(x)$$

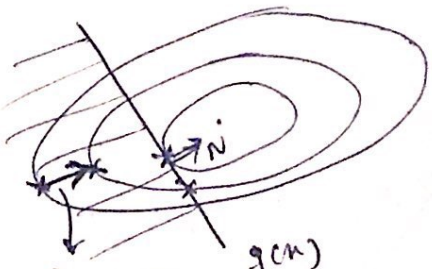
$$\downarrow$$

$$\nabla_x f(x) + \sum_i \lambda_i h_i(x) = 0.$$

Inequality constraint :— $\min f(x)$ s.t $g(x) \leq 0$



becomes an inactive constraint



Can move to the next min contour plot

$\nabla_x f(x) = 0$

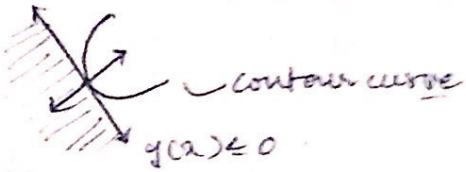
$\nabla_x f(x) + \mu \nabla_x g(x) = 0$

$\mu = 0$ or $g(x) = 0 \Rightarrow \boxed{\mu g(x) = 0}$

$g(x) \leq 0$

$\nabla_x f(x) + \mu \nabla_x g(x) = 0$

$\boxed{\mu \geq 0}$ → gradients can only be antiparallel [not parallel]



Multiple constraints :

$g_j(x) \leq 0 \ \forall j \ ; \ h_i(x) = 0 \ \forall i$

$g_j(x)$ → inequality cond^{n}s

$h_i(x)$ → equality cond^{n}s

$\nabla_x f(x) + \sum_j \mu_j \nabla g_j(x) + \sum_i \lambda_i \nabla_x h_i(x)$

$\mu_j \geq 0 \quad \forall j$

$\mu_j g_j(x) = 0 \quad \forall j$ → KKT Conditions

↓

not always necessary for the optimal sol^{n}s

Opt problems have to satisfy

↓

Sufficiency conditions ← for KKT to be sufficient

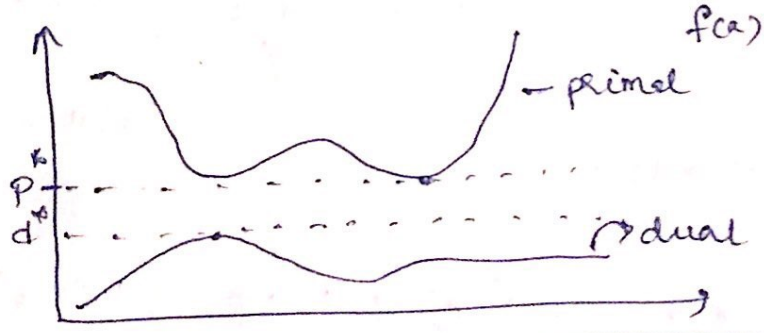
→ Convex optimisation ⇒ Slater conditions

Duality :- $P \equiv \min_x f(x)$

$s.t \quad g_j(x) \leq 0 \ , \ j = 1, \ldots n$

$h_i(x) = 0 \ , \ j = 1, \ldots m.$

} Primal



both the terms boil down to zero

opt : $[\mu = 0]$

$P \equiv \min_x \max_{\mu \geq 0, \lambda} f(x) + \sum_j \mu_j g_j(x) + \sum_i \lambda_i h_i(x)$ → $[h_i(x) = 0 \ \forall i]$

$L(x, \mu \geq 0, \lambda) \Longrightarrow \min_x \begin{cases} f(x) , & x \text{ is feasible} \\ \infty , & \text{otherwise} \end{cases}$

$$P \equiv \min_{x} \max_{\mu \geq 0, \lambda} f(x) + \sum \mu_j g_j(x) + \sum_i \lambda_i h_i(x)$$

$$\downarrow$$

$$D \equiv \max_{\mu \geq 0, \lambda} \min_{x} f(x) + \sum_j \mu_j g_j(x) + \sum_i \lambda_i h_i(x)$$

→ Lagrangian dual problem.

$\beta = [\mu, \lambda]$ (dual variable)

whereas, x = primal variable.

Dual lower bound of primal :-

$$\max_{\substack{\mu \geq 0 \\ \lambda}} \min_{x} L(\cdot) \leq \min_{x} \max_{\substack{\mu \geq 0 \\ \lambda}} L(\cdot)$$

Lagrangian dual problem ≡ always concave in q $f(x)$

lower bound to primal

$$\max_{\substack{\mu \geq 0 \\ \lambda}} \min_{x} \left[f(x) + \sum_j \mu_i g_i(x) + \sum_i \lambda_i h_i(x) \right]$$

$$\downarrow$$

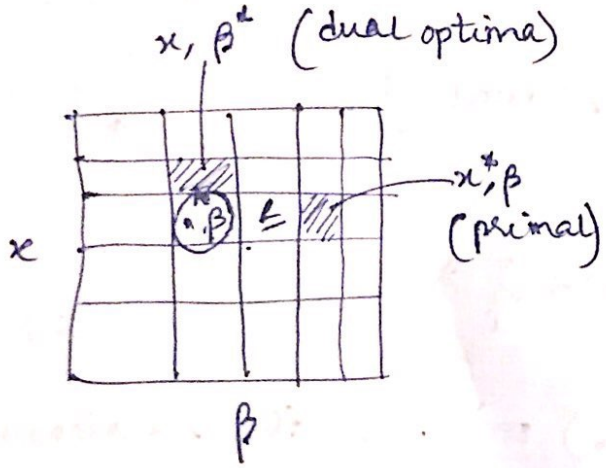
$$\max_{\substack{\mu \geq 0 \\ \lambda}} \underbrace{\left[k_1 + \mu k_2 + \lambda k_3 \right]}_{\text{linear function } q(\mu, \lambda)} \rightarrow \text{maximising over all the linear functions.}$$

Lagrangian function :- $L(x, \beta)$

P: $\min_{x} \max_{\beta} L(x, \beta)$ — first rows then columns

D: $\max_{\beta} \min_{x} L(x, \beta)$ — first cols then rows

first x and then β



$$L(x^*, \beta) \geq L(x, \beta) \geq L(x, \beta^*)$$

Primal obj ≥ Dual obj

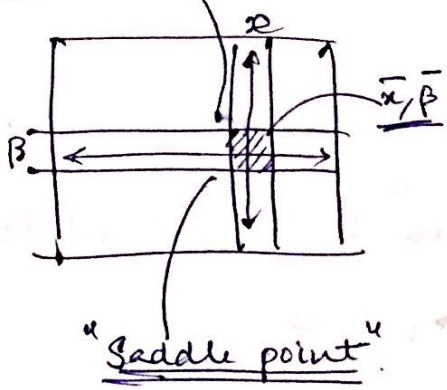
primal optima (p^*) ≥ dual optima (d^*)

$p^* \geq d^*$

$p^* \geq d^*$ (weak duality)

$p^* = d^*$ (strong duality)

red and green in the same box → saddle point



If a box, minimises L → along the ~~column~~ row

maximises L → along the ~~column~~ col.

Lagrangian [with saddle point]

$\Rightarrow L(\bar{x}, \beta) \leq L(\bar{x}, \bar{\beta}) \leq L(x, \bar{\beta})$

$$L(\bar{x}, \bar{\beta}) \leq \max_{\beta} L(x, \beta)$$

At the saddle point, $L(\bar{x}, \bar{\beta}) = \min_{x} \max_{\beta} (L(x, \beta)) = p^*$ (primal)

↳ primal optima

$$L(\bar{x}, \bar{\beta}) \geq \min_{x} L(x, \beta)$$

At the saddle point, $L(\bar{x}, \bar{\beta}) = \max_{\beta} \min_{x} L(x, \beta) = d^*$ (Dual)

↳ dual optima.

$L(\bar{x}, \bar{\beta}) \Rightarrow$

$p^* = d^*$

$\bar{x} = x^*, \bar{\beta} = \beta^*$

[at the saddle point]

Nonlinear Equations and optimisation:-

$\min \|f(x)\|$

$\Downarrow$

$f(x) = 0$

$f_1(x_1, x_2, \ldots x_n) = 0.$

↓ multivariable

$f_2(x_1, x_2, \ldots x_n) = 0.$

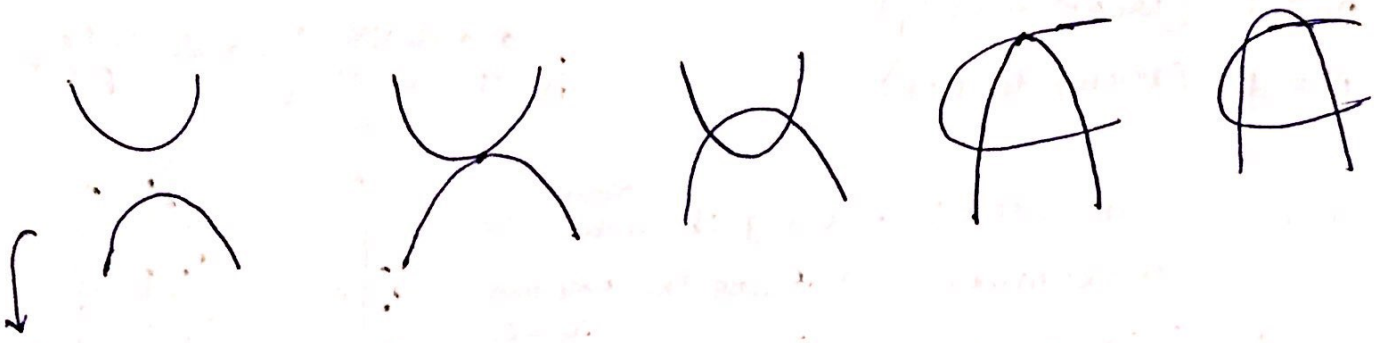
$\vdots$

$f(\bar{x}) = 0 \equiv \mathbb{R}^n \rightarrow \mathbb{R}^m.$

$x^0 \rightarrow x^1 \rightarrow x^2 \rightarrow \ldots x^k \rightarrow x^{k+1}$

Iterative :-

* Initialize
* Update
* Iterate/Repeat until convergence.



possible cases of solutions.

→ $f(x) = 0$.

→ How many times function is evaluated?

→ How fast does it converge?

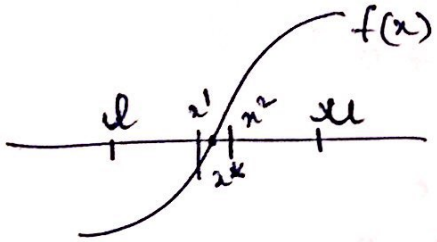
→ Local or global?

Situations :-

① We know the form of $f(x)$

② We know only to evaluate $f(x)$ [$f(x)$ – black box] → $f'(x)$

* Bisecting →



[sign of $f(x)$ differs at u and l
↳ this implies that there is a solution in between u and l.
(exists a root)

Initialize → u, l

$x^k = \frac{u+l}{2}$.

if $f(x^1)\, f(u) < 0$, $l = x^1$

if $f(x^1)\, f(l) < 0$, $u = x^1$

$k = k+1$ → slowly converges to the true value.

$f(x) = x^2 - x - 2$

$x^* = -1, 2$

Initialize; $u = 5, l = 1$

$x^1 = 3$

$x^2 = 2$

$x^3 = 2.5$

What's good about this scheme?

→ We don't need to Know the form of the function.

→ We dont need to differentiate

→ there is a bound on the number of steps

Initialising u and l → might be a problem.

Local optimum soln

~~How many~~ Iterations → interval is becoming half.

$|x^k - x^{k-1}| < \varepsilon \longrightarrow k \geq$ [log of 2 in terms of the initial interval $[u, l]$ and ε]

$x^2 - x - 2 = 0$

$x^2 = x + 2 \Rightarrow \boxed{x = 1 + \frac{2}{x}}$

* **Fixed point :-**

p is a fixed point.

→ if $p = g(p)$

For Ex : $g(x) = 1 + \frac{2}{x}$

$x^0 = 1$

Iterations

$x^1 = g(x^0) = 3$

$x^2 = g(x^1) = 1.6$

$x^3 = g(x^2) = 2.9$

$x^4 = g(x^3) = 1.9.$

converged to 2

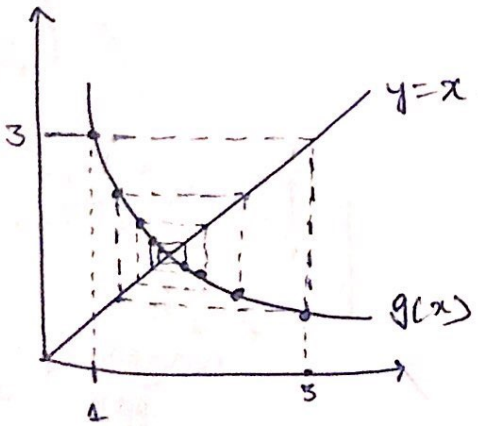
→ Initialize

Iterate

$x^{k+1} \leftarrow g(x^k)$

until convergence.

} fixed point iteration scheme.



or $x^2 - x - 2 = 0$

$x = x^2 - 2$

$g(x) = x^2 - 2$

if $x^0 = 2.1$, $x^1 = 2.4$, $x^2 = 2.5$ ↓ diverging (bad scheme)

$x^0 = 1$, $x^1 = -1$, $x^2 = -1$ ↓ converges to -1 (good scheme)

Fixed point iteration scheme →

$\Rightarrow x^2 = a \quad ; \quad x = a/x$

$x^2 - a = 0$

$(x + \sqrt{a})(x - \sqrt{a}) = 0$

$g(x) = a/x$

$x^0 = 1$

$x^1 = g(x^0) = a$

$x^2 = g(x^1) = 1$

$x^3 = g(x^2) = a$

↓ doesn't converge.

↳ continuously oscillating

→ Not all $g(x)$'s work / converge.

$g(x) = \frac{1}{2}\left(\frac{a}{x} + 1\right)$

$x^0 = 2$

$x^1 = g(x^0) = \frac{a+2}{4}$

$x^2 = g(x^1) = \frac{5a+2}{2(a+2)}$

⋮

if $a = 9$

$x^0 = 2$

$x^1 = 3.25$

$x^2 = 3.009$

↓

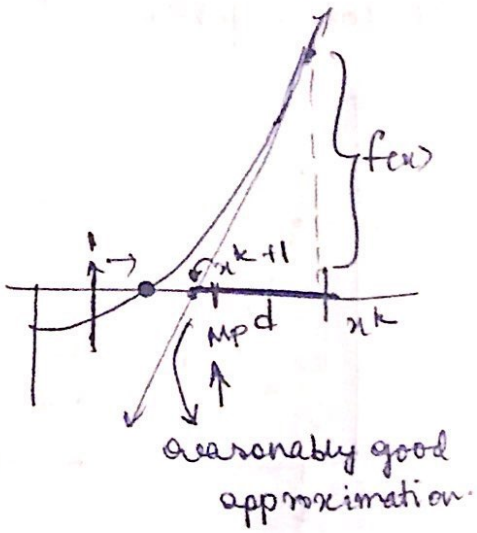
converging.

* If $|g'(x^*)| < 1$, the scheme is locally convergent

→ Many $g(x)$'s can be created from the original problem.

Newton's Method →

$$x^{k+1} \leftarrow x^k - \frac{f(x)}{f'(x)}$$



reasonably good approximation.

$\tan\theta = f'(x) = \frac{f(x)}{d}$

$d = \frac{f(x)}{f'(x)}$

$x^{k+1} \leftarrow x^k - d \Rightarrow x^{k+1} \leftarrow x^k - f(x)/f'(x)$

Limitations of Newton's Method

↳ We need to find the differentiation of $f(x)$

Iterative process.

$$x^1 \leftarrow x^0 - \frac{f(x)}{f'(x)}$$

$$\downarrow$$

$$x^{k+1} \leftarrow x^k - \frac{f(x^k)}{f'(x^k)}$$

$f(x) = x^2 - x - 2$.

start with $x^0 = 1$;

$$x^1 \leftarrow 1 - \frac{[-2]}{1} = 3$$

$$x^2 \leftarrow 3 - \frac{4}{5} = 2.2.$$

$$\downarrow$$

slowly converges to $x^i = 2$.

Secant Method ⟹

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$ ⟶ instead of differentiating;

$$f'(x^k) = \frac{f(x^k) - f(x^{k-1})}{x^k - x^{k-1}}$$

$$= x^k - \frac{f(x^k) \cdot (x^k - x^{k-1})}{f(x^k) - f(x^{k-1})}$$

↓

Advantage over Newton's method.

↳ No need to compute the derivative, just evaluating will be enough

Disadvantage ⟹ approximate of derivative is computed
↳ might take more time to converge.

Computation – becomes easier
↓
small amount of memory is needed.
(recevaluation is not needed).

$p = g(p)$

$x^{k+1} \leftarrow g(x^k)$

→ Locally converging

Min $|g(x)|$

↓

Solve: $g(x) = 0$.

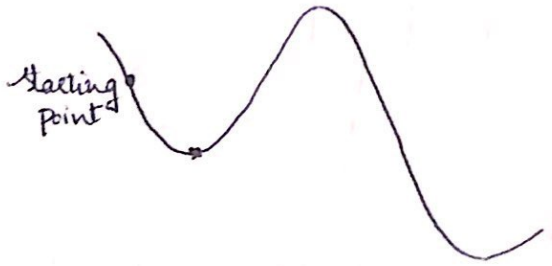
↓

m var, n equations.

* Fixed point method –

$h(x)$ → locally converges if →

$$|h'(x^*)| < 1$$



$$\underbrace{x^{k+1} - x^*}_{e^{k+1}} = h(x^k) - h(x^*)$$

— From Meanvalue theorem

$$e^{k+1} = h'(\,)\,[x^k - x^*]$$

$$e^{k+1} = \underbrace{h'(\,)}_{(h'(x)<1)} \cdot e^k$$

→ on increasing iterations, error decreases.

$e^k \rightarrow e^{k+1}$ ↓

* Newton's method –

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$

$$h(x) = x - f(x)/f'(x) \implies h'(x) = 1 - \frac{f' \cdot f' - f \cdot f''}{f' \cdot f'}$$

$$= \frac{f \cdot f''}{f' \cdot f'}$$

Ex: $x^3 - 2x - 5 = f(x)$

Newton's iteration — $x^0 = -1$

$$x^{k+1} \leftarrow x^k - \left[\frac{x^3 - 2x - 5}{3x^2 - 2}\right]$$

$x^1 \leftarrow 3$

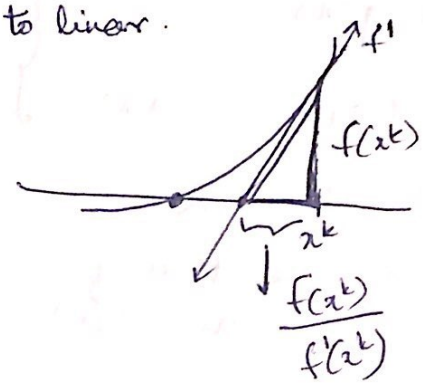
$x^2 \leftarrow 59/25$

⋮

approximates to ~ 2.1

$$x^{k+1} - x^k = -\frac{f(x^k)}{f'(x^k)}$$

limiting to linear.



Taylor series —

$$f(y) = f(x) + \frac{f'(x)}{1!}(y-x) + \frac{f''(x)}{2!}(y-x)^2 + \dots$$

we have to find, x^{k+1} such that $f(x^{k+1}) = 0$.

$y = x^{k+1}$

$$f(x^{k+1}) = f(x^k) + \frac{f'(x^k)}{1!}(x^{k+1} - x^k) + \underbrace{\frac{f''(x^k)}{2!}(x^{k+1} - x^k)^2 + \dots}$$

($f(x^{k+1}) \to 0$)

approximated as linear, so higher order terms $= 0$

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$

Quadratic approximation →

$$f(x^{k+1}) = f(x^k) + \frac{f'(x^k)}{1!}(x^{k+1} - x^k) + \frac{f''(x^k)}{2!}(x^{k+1} - x^k)^2$$

($f(x^{k+1}) \to 0$)

$$0 = f(x^k) + \frac{f'(x^k)}{1!}(x^{k+1} - x^k) + \frac{f''(x^k)}{2!}(x^{k+1} - x^k)^2$$

→ Solve for x^{k+1}.

(but Hessian matrix (memory intensive))

⇒ gives a better solution than linear approximation and even more higher order terms give better and better solutions → so why are we not doing it?

— computationally intensive

In higher order approximations
↳ no. of iterations – will be slow
but each iteration is computationally complex to solve.

Min $g(x)$
↓
$g(x_1, x_2, \ldots x_n)$
$g : \mathbb{R}^N \to \mathbb{R}$
(scalar function of n variables)

$\to \nabla g = 0$.

$$\begin{bmatrix} \partial g/\partial x_1 \\ \partial g/\partial x_2 \\ \vdots \\ \partial g/\partial x_n \end{bmatrix} = \bar{0}$$ } system of equations.

n equations.

$\Rightarrow \bar{f}(\bar{x}) = \bar{0}$ ⟵ optimising $g(x)$

where $\bar{f} : \mathbb{R}^n \to \mathbb{R}^m$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots \\ \vdots & \\ \frac{\partial f_1}{\partial x_n} & \cdots \end{bmatrix}$$ Jacobian

$\Downarrow$ equivalent

$$H = \begin{bmatrix} \frac{\partial g}{\partial x_1 \partial x_1} & \frac{\partial g}{\partial x_1 \partial x_2} \cdots \\ \vdots & \end{bmatrix}$$ Hessian (second derivative matrix)

system of equations $\Rightarrow \bar{f}(\bar{x}) = 0$.

$$\bar{x}^{k+1} \leftarrow \bar{x}^k + s.$$

$$\bar{f}(\bar{x} + s) = \bar{f}(\bar{x}) + J.S$$

↓ assume = 0, just like in Taylor's series

$$J.s = -\bar{f}$$

J → Jacobian
s → change or Δx

$J.s = -\bar{f}$ (Solve $\bar{F}(x) = 0$)

→ in terms of g; Min $g(x)$

$$\Rightarrow \boxed{\nabla^2 g \cdot s = -\nabla g}$$

$$\Rightarrow \boxed{x^{new} \leftarrow x^{old} - (\nabla^2 g)^{-1} \nabla g}$$

* $g(x) = (x-2)^2 - 5$ → Min = 2 ⟹ $\eta^* = 1/2$ [optimal learning rate]

$= x^2 - 4x - 1$

$g'(x) = 2x - 4$

$g''(x) = 2.$

$\Rightarrow x^{new} \leftarrow x^{old} - \frac{1}{2}[2x - 4]$

Suppose; $x^0 = 5$

$x^1 = 5 - \frac{1}{2}[10-4] = 2 = x^*$

$x^0 = 6$

$x^1 = 6 - \frac{1}{2}[12-4] = 2 = x^*$

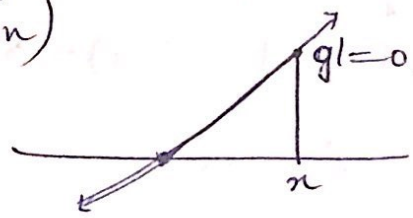
Why? x gets cancelled out and we end up with opt, why?

→ we are getting the optima [true solution] independent of the initialisation

Problem = min $g(x)$

But, we are solving ⟹ $\nabla g = 0$ (a linear function)

So in one go, we get the solution.



$\left[\eta = 1/\nabla^2 g\right]$

$\Rightarrow x^{new} \leftarrow x^{old} - \eta \nabla g$ ← learning rate [in gradient descent]

[in the above case, if $\eta^* = 1/2$, it worked perfectly]

Let us take $\eta = 1$;

$x^{new} \leftarrow x^{old} - \nabla g.$

$= x^k - 1.[2x^k - 4]$

$x^0 = 5$

$x^1 \leftarrow 5 - [10-4] = -1$

$x^2 \leftarrow -1 - [-6] = 5$ } oscillates

$\eta = 2$;

$x^{new} \leftarrow x^{old} - 2\nabla g$

$= x^k - 2[2.x^k - 4]$

$x^0 = 5;$

$x^1 = 5 - 2[2.5 - 4] = -7$

$x^2 = -7 - 2[2.-7 - 4] = 29$

↓ Diverging.

Min $g(x)$

$\nabla g = 0$

$$\begin{bmatrix} \frac{\partial g}{\partial x_1} \\ \vdots \\ \frac{\partial g}{\partial x_n} \end{bmatrix} = 0 \quad // f(x) = 0.$$

$$H = \begin{bmatrix} \frac{\partial^2 g}{\partial x_1^2} & \cdots & \frac{\partial^2 g}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 g}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 g}{\partial x_n^2} \end{bmatrix}$$

Solve $\bar{f}(\bar{x}) = \bar{0}$

$$\bar{f}(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{bmatrix}$$

$$J = \begin{bmatrix} \frac{\partial \bar{f}}{\partial x_1} & \frac{\partial \bar{f}}{\partial x_2} & \cdots & \frac{\partial \bar{f}}{\partial x_n} \end{bmatrix}$$

Jacobian

$$\equiv = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

→ H and J are the same matrices because $f = \nabla g$;

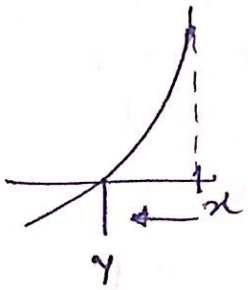
$J_f = \nabla^2 g$ ↪ Hessian of g.

Problem: Solve $f(x) = 0$;

$$f(\bar{y}) = f(\bar{x}) + [\bar{y} - \bar{x}]^T \nabla f(\bar{x})$$

$y - x = s$

Find y st. $f(y) = 0$.



$$\bar{f} + J \cdot s = 0 \qquad , x \text{ is a vector.}$$

$$Js = -\bar{f}$$

$$s = -J^{-1}\bar{f}$$

$$y = x + s$$

$$= x - J^{-1}\bar{f} \longrightarrow x^{k+1} \leftarrow x^k - [\nabla^2 g]^{-1} \nabla g$$

Iterative Algorithm:-

Initialize x^0

For $k = 0 \rightarrow$ Solve $s = -J^{-1}f$

$$x^{k+1} \leftarrow x^k + s = x^k - J^{-1}f$$

$\bar{J}^{-1}$ or $[\nabla^2 g]^{-1}$ computation is expensive

$x^{k+1} \leftarrow x^k - [\nabla^2 g]^{-1} \nabla g$ → right step size [but complex computation]

$\Downarrow$

$x^{k+1} \leftarrow x^k - \eta \nabla g \equiv$ Gradeint Descent.

(putting a scalar in the place of $\nabla^2 g$)

matrix sizes are usually huge in practice [in deeplearning]

→ Though second order approximation is useful, it is not practically feasible [gradeint descent is much easier]

ex: $g = (x-2)^2 - 5 = x^2 - 4x - 1$

$\nabla g = 2x - 4$.

$\nabla^2 g = 2$.

$x^0 = 5$

$x^1 = 5 - \frac{1}{2}(6) = 2$ ✓

using second order derivative.

adhoc learning rate

instead, if we use gradeint descent

$x^{k+1} \leftarrow x^k - \eta \nabla g$

$x^0 = 5$, $\eta = 1$

$x^1 = -5 - 6 = -1$

$x^2 = -1 - (-6) = 5$

— oscillating.

$x^0 = 5$, $\eta = 2$

$x^1 = 5 - 2(6) = -7$

$x^2 = -7 - 2(-18) = 20+$

— diverging.

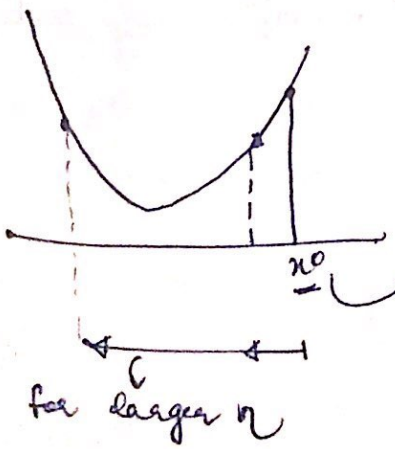
$\eta = 1/2$ optimal solution

$\eta = 1$ oscillating

$\eta = 2$ diverging.

} even when the function is convex.

that is why, we need the appropriate learning rate.



Least Square problem ⇒

$$\text{Min } q(\bar{x}) = [Ax-b]^T[Ax-b]$$

$$= x^TA^TAx - 2(Ax)^Tb + b^Tb.$$

$$\nabla q = 2A^TAx - 2A^Tb.$$

$$\nabla^2 q = 2A^TA.$$

Initialise x^0;

$$x^1 \leftarrow x^0 - (2A^TA)^{-1}[2A^TAx^0 - 2A^Tb]$$

$$\leftarrow x^0 - (A^TA)^{-1}.(A^TA)x^0 + (A^TA)^{-1}A^Tb.$$

$$\leftarrow x^0 - x^0 + (A^TA)^{-1}A^Tb.$$

$$\leftarrow (A^TA)^{-1}A^Tb.$$

(~ independent of initialisation)
→ you get the solution in one single step.
[No need of iterative process]

Even in the case of nonlinear least squares problem, we can use this →

$$Ax = b$$

$$a_i^Tx = b_i \quad \text{—[Linear]}$$

Non linear case ⇒

$$f_i(x_i) = b_i$$

Focus:

$$\Rightarrow f_1(x) = a_{10}x_1^2 + a_{11}x_2^2 + a_{12}x_1x_2 = b_1$$

$$f_2(x) = a_{20}x_1^2 + a_{21}x_2^2 \ldots = b_2$$

$$\vdots$$

$$\bar{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \leftarrow$$

Tradeoff between cost per iteration and no. of iterations.

Gradient Descent [GD] vs Stochastic Gradient Descent [SGD] ⇒

$$g(\bar{w}) \rightarrow \sum_{i=1}^{N} \phi(x_i, w)$$

$$w^{k+1} \leftarrow w^k - \eta \nabla g \longrightarrow$$ an estimate of ∇g is computed on a much smaller sample set. [random M samples]

$$\boxed{\tilde{\nabla g} = \sum_{i=1}^{M}} \rightarrow \text{where } \boxed{M << N}$$

⇒ updating w is not a big deal but computation of ∇g is computationally intensive

→ so we are reducing the cost per iteration by estimating gradient from M samples. [where $M << N$]

↓

GD implemented as SGD.

→ if samples are good, then with a smaller set, ~~we can~~ the process converges.

Issues with Gradient Descent –

* GD and SGD
* Optimality
* Convergence [at what rate will it converge?]
* Non-differentiable
* Higher order terms (like Hessian for finding learning rates)
* Practical issues

Newton's method with backtracking =>

for k = 1 . . .

$x^{k+1} \leftarrow x^k + s$

if $g(x^{k+1}) > g(x^k)$ → To avoid divergence or oscillations.

$s = s/2$

GD for ML →

Weights are learned

w

$w^* = \underset{w}{\text{argmin}}(\min f(w))$ — No explicit form of w that we are minimising

↓

f is usually a expectation over a distribution, D

samples 'n' ← $\left\{ \sum_{i=1}^{n} \phi(w_e, x_i, y_i) \right.$ → loss function

(x_i, y_i)

↑ data ↑ label.

$x_i \Rightarrow D$

n samples coming from a distribution

$w^{k+1} \leftarrow w^k - \eta \nabla f$

Finding the gradient ∇f → stochastic gradient

∇f is computed from a random subset of the samples

↓

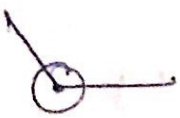
computation — faster

— removes oscillations (because of the randomisation introduced in the samples)

→ Subgradient

- function might not be differentiable at all points.

[Ex: hinge loss]



Instead of one gradient (∇f), we compute a set of gradients $\{\nabla\}$

→ only for selected points

↳ Finding the best out of all potential directions.

Minimising $\rightarrow [Y - Xw]^T[Y - Xw] \rightarrow w^* = (X^TX)^{-1}X^TY$

(X^TX): $d \times d$ (usually high)

$f(w) = \sum_{i=1}^{n} (y_i - x_i^T w)^2$ → If this is done over a subset of samples → SGD.

$w^{k+1} \leftarrow w^k - \eta \nabla f$

PCA: $\max w^T \Sigma w \quad ; \|w\| = 1$

w — eigen vectors of covariance matrix.

update functions [

$\tilde{w} \leftarrow w^k - \eta \nabla f$

$w^{k+1} \leftarrow \frac{\tilde{w}}{\|\tilde{w}\|}$ [Makes sure Norm remains 1]

If n is very large, computed estimate will be closer to the real gradient

If n becomes very low → it becomes noisy.

$f(w+s) = f(w^k) + s^T \nabla f \ldots$ (can be ignored)

$\approx f(w^k) - \eta \nabla f^T \nabla f$

$\Rightarrow f(w^k) - \eta$ → [kept small] — to (not to jump too far)

$f(w+s) \leq f(w^k) \Rightarrow f(w^{k+1}) \leq f(w^k)$

($w^{k+1} = w + s$)

— In every iteration, new value of the function is smaller than the current value

~ converging

Why do oscillations happen?

→ Assumed that taylor series approximation is accurate.

$f(w^{k+1})$ — estimation can be highly inaccurate.

→ Finding s becomes complex (when additional terms are used in the Taylor series)

[for quadratic - computing hessian inverse] $s = H^{-1}\nabla f$.

→ though iterations become simpler

How to find η?

η - has to be small → good for convergence but bad for computation time ↑
- but how small

Larger η → risk of convergence and stability

Trade off between ⇒ computation/itr and #iterations
stability and speed.

vary η over time
- η/t (initially make big jumps and when you come closer to the optima, reduce it slowly)

Instead of using a single scalar for all the dimensions — X
not good as, so, we have to take a d dimensional η vector

Most popular.
Stable gradient descent implementations ⇒ ADAGRAD

ADAGRAD still does $\eta \nabla f$, but ∇f is modified as —

Normalizing ← $\nabla f^* = \frac{\nabla f^i}{\sqrt{g^i + \epsilon}}$ maintain a g vector and divide ∇f by g vector.

→ ∇f is scaled non-uniformly across diff dimensions

this is same as to having diff gradients for diff dimensions.

g → cumulative gradient; $g_i = \sum_{i=1}^{k} (\nabla f^i)^2$

what is the use of ϵ → when $g^i \to 0$, stability.

$s = -\bar{H}^{-1} \nabla f$

$H_{d \times d}$ (Computing inverse — is difficult)

$s = -H^{-1} \nabla f$

if H — diagonal — H^{-1} can be computed easily

– "Hessian free" method – uses conjugate gradient