Lecture 16

We saw that the steepest descent method give exact soln. in few iteration.

We saw that one uses the stationary condition

$$\frac{df(\alpha_n)}{d\alpha_n} = 0$$

One can say as we use this stationary condition for $f(\alpha_0)$, one should directly use same method to get min. point of $f(x)$?

① Even for complicated multiple variables $f(x_1, \ldots x_p)$ (let say $p = 1000$); then $f(\alpha_0)$ at any step n is still a univariate function & the optimization of such $f(\alpha_n)$ is much simpler compared with original multivariate problem

② Steepest descent is typically slow once the local minimization is near.

↳ $\because$ near local min. the gradient is nearly zero $\therefore$ rate of descent is also slow.

If high accuracy is needed, other local search methods should be used

There are many variations of the steepest descent method : Simulated annealing, Momentum Based Method, Adaptive learning rate method L16

If max. is needed, then this method become hill-climbing method

* Standard steepest descent method works well for convex function and near a local peak of most smooth multi modal function, though this local peak is not necessarily the global peak

For some functions, it may not be a good method.

Eg. Let us minimize the so-called banana function (introduced by Rosenbrock)

$$f(x_1, x_2) = (1-x_1)^2 + 100(x_2 - x_1^2)^2$$

This function has global min. $f_{min} = 0$ at $(1,1)$

$$\frac{df}{\partial x_1} = \cancel{2}(-2(1-x_1) - 400(x_2 - x_1^2) = 0$$

$$\frac{df}{dx_2} = 200(x_2 - x_1)^2 = 0$$

whose unique soln. are

$$x_1 = 1 \qquad x_2 = 1$$

2 a | Using steepest descent method

initial guess $x^{(0)} = (5,5)$

$\nabla f = \left(-2(1-x_1) - 400x_1(x_2-x_1^2),\ 200(x_2-x_1^2)\right)^T$

$\nabla f(x^{(0)}) = (40008, -4000)^T$

1st

$$x^{(1)} = x^{(0)} - \alpha_0 \begin{pmatrix} 40008 \\ -4000 \end{pmatrix}$$

$$f(\alpha_0) = \left[1-(5-40008\alpha_0)\right]^2 + 100\left((5+4000\alpha_0)-(5-40008\alpha_0)^2\right]^2$$

should be minimized.

The stationary condition become

$$\frac{df}{d\alpha_0} = 1.0248\times10^{21}\alpha_0^3 - 3.8807\times10^{17}\alpha_0^2 + 0.4546\times10^{14}\alpha_0 - 1.6166\times10^9 = 0$$

3 soln.

$\alpha_0 \approx 0.00006761,\ 0.0001212,\ 0.0001840$

Use any of them.

new iteration is always greater that $x_2^{(0)} = 5$.

which moves away from the best solution (1,1)

The difficulty in this example arises because of the scaling difference where factor 100 is associated with the second term.

Further, Rosenbrock's banana function is very tough test function for optimization algorithms

Q-1a

Momentum Based Method

It is an enhancement of the gradient descent method
It carry an additional term called momentum to smooth the trajectory, allowing optimization to accelerate relevant direction

Ball is rolling down a hill that gathers enough momentum to overcome a plateau region & make to global minima instead of stuck the local minima.

steepest descent :

$$x^{n+1} = x^{n} - \alpha \nabla f(x^{n})$$

momentum based

is

$$v^{n+1} = \beta v^{n} + \alpha \nabla f(x^{n})$$

$\beta \rightarrow$ momentum factor (typically b/w 0.9 & 0.99)

$\alpha \rightarrow$ learning rate $\rightarrow$ Some put it $(1-\beta)$

$v \rightarrow$ velocity (momentum)

$$x^{n+1} = x^{n} - v^{n+1}$$

initialize $v^{0} = 0$ & $\beta \rightarrow$ momentum factor to 0.9

$(x_2 - x$

Gradient Based

Steepest Descent moves steadily in smaller steps, making gradual progress toward minimum

→ Sensitive to learning rate

Momentum based

Momentum based Method takes larger steps early on & accelerates the updates, leading to faster convergence

→ less sensitive to learning rate

Momentum method tend to converge faster compared to gradient descent in practice

Adaptive learning rate

Refers to optimization algorithms which adjust the learning rate dynamically based on the progress of the training ~~method~~ process, rather than using fixed learning rate as is traditional gradient descent method.

Types

AdaGrad (Adaptive Gradient Algorithm)

adapts learning rate for each parameter individually ~~individually~~ reducing the learning rate for parameters with large gradients & ~~~~ ↑sing it for small gradient

$$x^{n+1} = x^n - \frac{\alpha}{\sqrt{G_n + \epsilon}} \nabla f(x^n)$$

$G_n \rightarrow$ sum of previous gradients squared.

ϵ is small constant to prevent division by zero $\sim 10^{-8}$

In some place

$$x^{n+1} = x^n - \frac{\alpha}{\sqrt{G_n} + \epsilon} \nabla f(x^n)$$

Works great on sparse data.

learning rate may slow over time, resulting slower convergence

One can use them in momentum also

$$v^{n+1} = \beta v^n + \frac{\alpha}{\sqrt{G_n + \epsilon}} \nabla f(x^n)$$

$$x^{n+1} = x^n - v^{n+1}$$

② <u>RMSProp</u> (Root Mean Square Propagation)

Instead of summing all past squared gradients it uses exponentially weighted moving averages of squared gradients to update parameter

$$x^{n+1} = x^n - \frac{\alpha}{\sqrt{E[\nabla f(x)^2]_n + \epsilon}} \nabla f(x^n)$$

$$E[\nabla f(x)^2]_n = \gamma E[\nabla f(x)^2]_{n-1} + (1-\gamma)(\nabla f(x)_n)^2$$

$$v^{n+1} = \beta v^n + \alpha \frac{\nabla f(x^n)}{\sqrt{E[\nabla f(x^n)^2] + \epsilon}}$$

$$\boxed{x^{n+1} = x^n - v^{n+1}}$$

Widely used, works well with non-stationary ~~and~~ noisy objects

Adam (Adaptive Moment Estimation)

Combines the first moment (mean of gradient) and second moment (variance of gradient) to adapt learning rate for each parameter

$$x^{n+1} = x^n - \frac{\alpha}{\sqrt{\hat{v}_n + \epsilon}} \hat{S}_n$$

$\hat{S}_n \to$ bias-corrected first moment (mean)

$\hat{v}_n \to$ bias corrected second moment (variance)

$$\cancel{\hat{S}_n = \beta S_{n-1} + (1-\beta) \nabla f(x_n)}$$

$$S_n = \beta_1 S_{n-1} + (1-\beta_1) \nabla f(x_n)$$ β typically 0.9

$$v_n = \beta_2 v_{n-1} + (1-\beta_2)(\nabla f(x_n))^2$$ $\beta_2 \to 0.999$

$$\hat{S}_n = \frac{S_n}{1-\beta_1^n}, \quad \hat{v}_n = \frac{v_n}{1-\beta_2^n}$$

combination of RMSProp + momentum that already includes momentum-like behavior

I outperform in terms of convergence speed + performance

Adadelta

Extension of AdaGrand optimization algorithm that aims to improve

→ No learning rate.

Use Accumulated gradient like RMS prop

$$E[\nabla f^2]_n = \gamma E[\nabla f^2]_{n-1} + (1-\gamma)(\nabla f(x)_n)^2$$

γ → ~~decent~~ decay constant, ~0.9 (typically)

~~RMS~~

$$\Delta x_n = \frac{RMS[\Delta x_{n-1}]}{\sqrt{E[\nabla f^2]_n + \epsilon}} \nabla f(x_n)$$

Δx_n → update applied to parameter at step n

$RMS[\Delta x_{n-1})$ is root mean square of previous update

$E(\nabla f^2]_n$ → moving average of sq...

RMS

$$E[\Delta x^2]_n = \gamma E(\Delta x^2)_{n-1} + (1-\gamma)(\Delta x_n)^2$$

$$x^{n+1} = x^n + \Delta x^n$$

No learning rate require

Simulated Annealing

It is a random search technique for global optimization problems, and mimics the annealing process in material processing when a metal cools + freezes into a crystalline state with minimum energy + larger crystal size.

Annealing process involve careful control of temperature + cooling rate.

In 1983 done by Kirkpatrick, Gelatt + Vecchi

Unlike gradient based method and other deterministic search methods which has disadvantage of becoming trapped in local minima.

Main advantage of S.A. is that the ability to avoid being trapped in local minima.

Basic idea of S.A.

↳ use random search which not only accept changes that improve the objective function, but also keep some changes that are not ideal.

In minimization problem.

any better moves or changes that decrease the cost (or value) of objective function f will be accepted, however some changes that increase f will also be accepted with a probability p.

This probability p, also called the transition probability is determined by

$$p = \exp\left[-\frac{\delta E}{k_B T}\right]$$

k_B is Boltzmann's const & T is temp. for controlling the annealing process.

δE is change in energy level.

This transition probability is based on Boltzmann distribution in physics.

Simplest way to link δE with change of objective function δf is to use

$$\delta E = \gamma \, \delta f \qquad \gamma \to \text{real const.}$$

For simplicity without losing generality we can use

$k_B = 1$ & $\gamma = 1$. Thus

$$p(\delta f, T) = e^{-\frac{\delta f}{T}}$$

Whether or not to accept a change, we ~~usually~~ usually use a random no. r (drawn from uniform $[0,1]$) as threshold. Then, if

$$p = e^{-\frac{\delta f}{T}} > r \quad \text{it is accepted}$$

The choice of right temperature is crucial

If $T \to \infty$, then $p \to 1$ almost all changes will be accepted

If $T \to 0$, then any $\delta f > 0$ (worse soln.) rarely anything will be accepted

Special case $T \to 0$ corresponds to gradient-based method

Another issue is to how to control the cooling process so that the system cools down gradually from higher temperature to ultimately freeze to a global minimum state

Two commonly used cooling schedules are

Linear cooling process

~~$T = T_0 - \beta t$~~ $T \to T - \delta T$

$T = T_0 - \beta t$ with temp. increment δT.

T_0 is initial temp. & t is pseudo time for iteration

β is cooling rate &

it should be chosen in such a way that

$T \to 0$ when $t \to t_f$ (max. no. of iterations)

which gives $\beta = T_0 / t_f$.

Geometric cooling essentially decrease temp. by cooling factor $0 < \alpha < 1$ so that T is replaced by αT or

$$T(t) = T_0 \alpha^t \quad , t = 1, 2, \ldots, t_f$$

$T \to 0$ when $t \to \infty$.

In practice $\alpha = 0.7 \sim 0.99$

To find suitable starting temp. T_0, we can use any information about objective function.

$$T_0 \approx \frac{\max(\delta f)}{\ln p_0}$$

If we know max. change $\max(\delta f)$ of objective fn, one can use it to estimate an initial temp T_0 for a given probability p_0.

T_f is set at $10^{-10} \sim 10^{-5}$.

$T_0 \to 1$

$f(x)$; $x = (x_1, \ldots x_d)^T$

Initialize ~~temp~~ initial temp T_0 + initial guess $x^{(0)}$

Set final temp. T_f + max no. of iteration N.

Define cooling schedule $T \mapsto \alpha T$ $(0 < \alpha < 1)$

while ($T > T_f$ + $n < N$)

Move randomly to new locations

$x_{n+1} = x_n + \text{random}$

Calculate $\delta f = f_{n+1}(x_{n+1}) - f_n(x_n)$

Accept new soln. if better

if not improved.

Generate random no. r

Accept if $p = \exp[-\delta f / T] > r$

update best x^* + f^*

2 a)

Exponential weighted moving average

3 square gradient are 9, 4, 1

$E[g^2]_1 = 0.9 \times 0 + 0.1 \times 9 = 0.9$

$E(g^2)_2 = 0.9 \times 0.9 + 0.1 \times 4 = 1.21$

$E(g^2)_3 = 0.9 \times 1.21 + 0.1 \times 1 = 1.189$

Sum of Previous Squared Gradient

$G_1 = 9$

$G_2 = 9 + 4 = 13$

$G_3 = 13 + 1 = 14$

Linear Programming

Basic idea in L.P. is to find the max. or min. of a linear objective under linear constraints.

E.g.

BSNL provide 2 different services x_1 and x_2

Profit of first service $\rightarrow \alpha x_1$

Profit of 2nd ——— $\rightarrow \beta x_2$

$$\beta > \alpha > 0.$$

Total profit

$$P(x) = \alpha x_1 + \beta x_2 \qquad \beta/\alpha > 1$$

↑se profit as much possible

However the services are limited by the total bandwidth

At most n_1 of first & n_2 of second can be provided per unit of time

$$x_1 \leq n_1 \qquad x_2 \leq n_2$$

Let say that staff employed can maintain only at max n services. so.

$$x_1 + x_2 \leq n$$

Addition constraint avoid unphysical values

$$n, n_1, n_2 > 0$$

$$0 \leq x_1 \leq n_1, \quad 0 \leq x_2 \leq n_2$$

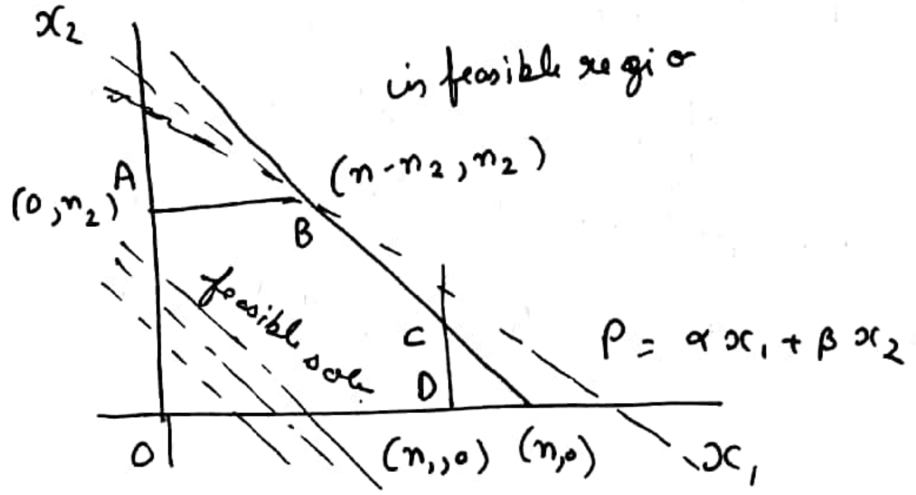
Problem now is to find the best x_1 & x_2 so that profit P is max.

Mathematically.

maximize $P(x_1, x_2) = \alpha x_1 + \beta x_2$
$(x_1, x_2) \in N^2$

subject to $x_1 + x_2 \leq n$

$0 \leq x_1 \leq n_1$; $0 \leq x_2 \leq n_2$



Feasible soln. can graphically be represented as inside region of polygon OABCD.

Aim is to maximize P.
thus optimal solution is at extreme point B with
$(n - n_2, n_2)$ & $P = \alpha(n - n_2) + \beta n_2$

If $\alpha = 2$, $\beta = 3$, $n_1 = 16$, $n_2 = 10$, and $n = 20$, then optimal soln. occurs at $x_1 = n - n_2 = 10$ & $x_2 = n_2 = 10$.
with total Profit $P = 2 \times (20 - 10) + 3 \times 10 = 50$.

The no. of feasible soln is infinite.

To find the best soln, one first plot out all the constraints as straight lines and all feasible solutions satisfying all the constraints as straight lines. All feasible solns satisfying vertices of the polygon form the set of the extreme points

Then one plot objective fn. P as family of parallel lines. The highest value of P corresponds to case when objective lines goes through extreme point B.

$\therefore x_1 = n - n_2$ & $x_2 = n_2$ at point B are the best soln.

This is bit simple

For complicated problems ;→
need a formal approach.

One of the most widely used method is the Simplex method.

Find the max. value of

$$Z = 2x + 3y$$

subject to the constraints

$$x + y \leq 30 \quad , y \geq 3$$

$$0 \leq y \leq 12 \quad , x - y \geq 0$$

$$0 \leq x \leq 20$$

As $x \geq 0$ & $y \geq 0$ soln. lies in first quadrant

$x + y \leq 30$

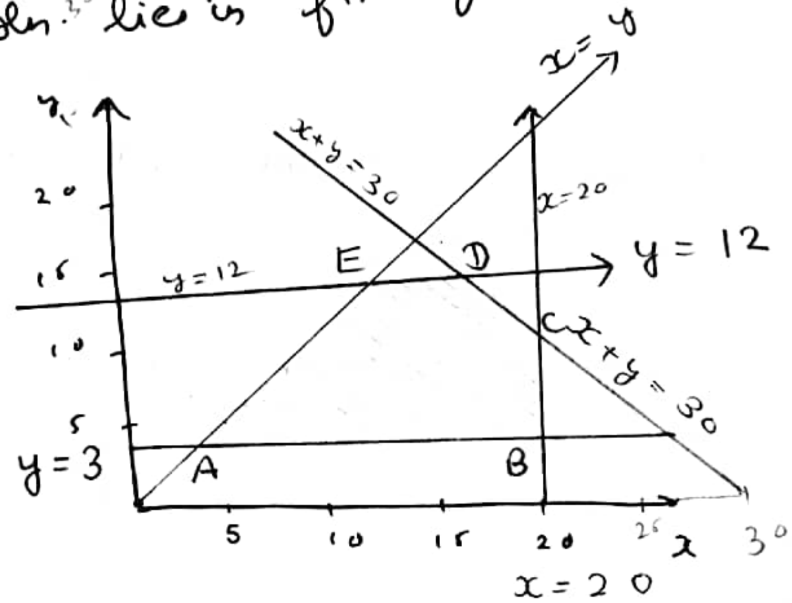
$y \geq 3$

$y \leq 12$

$x \geq y$

$x \leq 20$

ABCDE is
Convex region



A (3,3)

B (20,3)

C (20,10)

D (18,12)

E (12,12)

Value of z at 5 vertices are

$Z(A) = 15$

$Z(B) = 49$

$Z(C) = 70$

$Z(D) = 72$

$Z(E) = 60$.

Since the max. value of Z is 72 which occurs at vertex D; the solution of L.P.P. is

$$x = 18, \ y = 12 \ \& \ \max. Z = 72$$

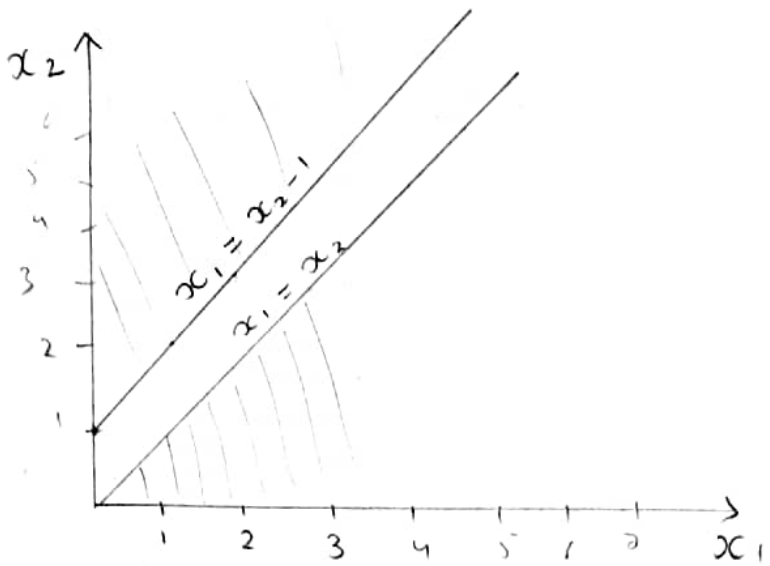
Solve

max. $Z = 4x_1 + 3x_2$

s.t. $x_1 - x_2 \le -1$

$-x_1 + x_2 \le 0$

$x_1, x_2 \ge 0$



$x_1, x_2 \to$ first quadrant.

No soln. as the constraints are incompatible.

General linear programing problem

Any L.P. problem involving more than 2 variables may be expressed as :

Find the values of variables $x_1, x_2, \ldots, x_n$ which maximize (or minimize) the objective function

$$Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

s.t. constraints

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \le b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \le b_2$$

$$\cdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \le b_m$$

and meet the non-negativity restrictions

$$x_1, x_2, \ldots, x_n \ge 0.$$

Def 1 – Set of values $x_1, x_2, \ldots, x_n$ which satisfies the constraints of L.P.P. is called its soln.

Any soln. to a L.P.P. which satisfyes the non-negativity restriction of the problem is called feasible soln.

Any feasible soln. which maximize (or minimize) objective fn is called its optimal soln.

Inequality constraints are changed to equalities by adding (or subtracting) non-negative variables to the L.H.S. of such constraint.

→ If constraints of a general L.P.P. be

$$\sum_{j=1}^{n} a_{ij} x_i \leq b_i \quad (i=1,2,\ldots k)$$

then non-negative variables S_i is added

$$\sum_{j=1}^{n} a_{ij} x_i + S_i = b_i \quad (i=1,2,\ldots,k)$$

slack variables.

→ If constraints of a general L.P.P. be

$$\sum_{j=1}^{n} a_{ij} x_i \geq b_i \quad (i=k,k+1,\ldots)$$

then $$\sum_{j=1}^{n} a_{ij} x_i - S_i = b_i \quad (i=k,k+1,\ldots)$$

are calls surplus variables

Scanned with OKEN Scanner

Canonical and Standard forms of L·P·P.

After the formation of L·P·P., obvious next step is to obtain its soln.

But before that, problem must be presented in a suitable form.

1) Canonical form.

General L·P·P. can always be expressed in following form:

Maximize $Z = c_1x_1 + c_2x_2 + \dots + c_nx_n$

s.t. $a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n \leq b_i \; ; i = 1, 2, \dots m$

$x_1, x_2, \dots, x_n \geq 0$.

by doing some elementary transformation

This form of L·P·P. is called its canonical form.

& has following characteristic

① $f(x)$ is of maximization type.

② All constraints are of $(\leq)$ type.

③ all Variables x_i are non-negative

→ Finds its use in Duality theory.

2) Standard form

The general L.P.P. can also be put in the following form.

$$\text{Max. } Z = c_1x_1 + c_2x_2 + \dots + c_nx_n$$

S.t.

$$\alpha_{i1}x_1 + \alpha_{i2}x_2 + \dots + \alpha_{in}x_n = b_i \; ; \; i = 1, 2, \dots, m$$

$$x_1, x_2, \dots x_n \geq 0$$

Standard form has following characteristic

① $f(x)$ is maximization type.
② All constraints are expressed as =ns.
③ R.H.S. of each constraint is non-negative
④ All variable are non-negative

Minimize $Z = c_1 x_1 + c_2 x_2 + \dots + c_n x_n$

can be made equivalent to $Z' = -Z = -c_1x_1 - c_2x_2 + \dots - c_nx_n$

inequality constraint ~~can~~ are converted to equalities by adding (or subtracting) the slack (or surplus) variables to the left hand side of such constraints.

L1

If a variable is $x_1, x_2, \ldots, x_n$ is -ve then it can always be expressed as difference of two non-negative variables

$$x_i < 0 \rightarrow \quad x_i = x_i' - x_i''$$

$$x_i' \geq 0 \; ; \; x_i'' \geq 0$$

Ex Express following into standard form.

Maximize $Z = 3x_1 + 5x_2 + 7x_3$

s.t. $6x_1 - 4x_2 \leq 5$

$3x_1 + 2x_2 + 5x_3 \geq 11$

$4x_1 + 3x_3 \leq 2$

$x_1, x_2 \geq 0$

As x_3 is unrestricted Let $x_3 = x_3' - x_3''$ where $x_3', x_3'' \geq 0$.

Now. $Z = 3x_1 + 5x_2 + 7x_3' - 7x_3''$

s.t. $6x_1 - 4x_2 + S_1 = 5$

$3x_1 + 2x_2 + 5x_3' - 5x_3'' - S_2 = 11$

$4x_1 + 3x_3' - 3x_3'' + S_3 = 2$

$x_1, x_2, x_3', x_3'', S_1, S_2, S_3 \geq 0$