Unconstrained Optimization

For a continuous function which first and second derivatives exist.

① If function is strictly convex, $\boxed{\frac{\partial f'(x)}{\partial x} = 0}$ is a necessary & sufficient condition for global min
↳ e.g. error function for multiple linear regression / SVM

② Else if function is not convex,
- ▷ $\boxed{\frac{\partial f'(x)}{\partial x} = 0}$ is a necessary but insufficient condition.
  ∵ stationary point may be max / min / inflexion point
- ▷ $\frac{\partial f''(x)}{\partial x} = 0$ is a necessary but insufficient condition.
  ∵ local min, may not be global min
- ▷ $f(x)$ is lowest among all points where $\frac{\partial f'(x)}{\partial x} = 0$ (and $\frac{\partial f''(x)}{\partial x} = 0$) is a necessary and sufficient condition

→ Bisection method / Newton method / Secant Method and Quasi-newton method

To solve for x where $\boxed{\frac{\partial f'(x)}{\partial x} = 0}$ (iteratively): ⇒ solves for stationary point

Step 1 : Initialisation

Step 2 : While termination condition is not met, update the initial value to a more optimised one

Section 1 : Finding global min with one variable
Section 2 : Finding global min with multiple variables

Bisection Method

Initialisation Select some a and b such that $f(a)f(b) \leq 0$ and f is continuous in $[a, b]$.
$\Rightarrow f(x) = 0$ must lie in $[a, b]$.

Update Find $m = \frac{a+b}{2}$. If $f(a)f(m) > 0$, $b = m$, else $a = m$.

Terminating condition E.g. $b - a \leq \varepsilon$ / $|f[\frac{1}{2}(a+b)]| \leq \varepsilon$

- If $f(m) = 0$, $x^* = m$
- Else, $x^* \approx m$. If $f(m) < 0$, $m < x^* < b$. Else, $a < x^* < m$

E.g. Find x at min $g(x) = 2x^6 + 3x^4 - 12x$.
$f(x) = g'(x) = 12x^5 + 12x^3 - 12$. ⇒ Find x where $f(x) = 0$.
$g''(x) = 60x^4 + 36x^2 \geq 0$ for all $x \in \mathbb{R}$ ∴ function is convex
One solution where $g''(x) = 0$ ∴ stationary point is global min

iteration #	a	b	$m = \frac{a+b}{2}$	f(m)	g(m)
0	0	2	$\frac{0+2}{2} = 1$	12	-7
1	0	1	0.5	-10.125	-5.7812
2	0.5	1	0.75	-4.0898	-7.6948
3	0.75	1	0.875	2.1940	-7.8439
4	0.75	0.875	0.8125	-1.3144	-7.8672
5	0.8125	0.875	0.84375	0.3397	-7.8829
6	0.8125	0.84375	0.828125	-0.5113	-7.8815
7	0.828125	0.84375	0.8359375	-0.0919	-7.8839

∵ $f(0) = -12$ $f(2) = 468$ ⇒ $f(0)f(2) \leq 0$

Note: converges to 0

$0.8359375 < x^* < 0.84375$, $x^* \approx 0.835935$

Newton Method

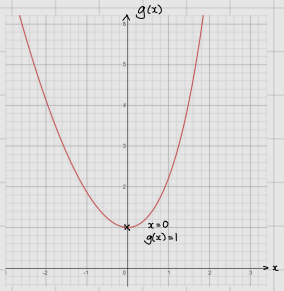
↳ Requires fewer iterations than bisection method, but each iteration is more expensive

Initialisation Select some x_0 → Can will provide a range. In real life, narrow down to a range w knowledge of the f^n } select a random value from the range

Update $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

Termination condition decrement in current iteration $< \varepsilon$

E.g. find x at min $g(x) = e^x + \frac{1}{2}x^2 - x$



iteration # (i)	x_i : $x_{n+1} = x_n - \frac{e^{x_n} + x_n - 1}{e^{x_n} + 1}$	$f(x_i)$
0	1.00000	2.71828
1	0.26894	0.57752
2	0.01878	0.03774
3	0.00009	0.00018
4	0.00000	0.00000

$\therefore x^* = 0$

Extension — problem formulation : Given R, find $f(R)$

E.g. Given R, find $\sqrt{R}$.

① $f(x) = \square - R$

$\sqrt{R} \xrightarrow{x \to x^2} R$ (derive function that converts $\sqrt{R}$ to R)

$x \xrightarrow{x \to x^2} x^2$ (apply the same function to x)

$\therefore f(x) = x^2 - R$

② $f'(x) = 2x$

③ Newton's iteration : $x_{k+1} = x_k - \frac{x_k^2 - R}{2x_k} = \frac{1}{2}x_k + \frac{R}{2x_k}$

Secant Method

↳ Approximates $f'(x_n)$ using $\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$

∴ Chosen over Newton method when $f'(x)$ is unavailable or hard to compute

Initialisation Select some x_{-1} and x_0 (order does not matter)

Update $x_{n+1} = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})}$

Termination condition

Gradient Descent

Initialisation Select some x_0

Update $x_{n+1} = x_n - \lambda f'(x_n)$

Termination condition

Note : When working with non-convex functions, for gradient descent, if $\lambda f'(x_n) < 0$, do not update x_n. Instead, $\downarrow \lambda$ and recalculate $\Rightarrow$ resultant stationary point will not be local maximum, but either local minimum or inflexion point. For all other methods, resultant stationary point may be local maximum, local minimum or inflexion point.

Analytical soln w gradient descent (if Jacobian is friendly — eg linear eqns) :

$\nabla f(x) = \begin{pmatrix} 2x_1 - 2x_2 \\ 4x_2 - 2x_1 - 2 \end{pmatrix}$ $\Rightarrow$ will eventually converge when $\nabla f(x) = 0$

$2x_1 - 2x_2 = 0$ & $4x_2 - 2x_1 - 2 = 0$ $\Rightarrow x_1 = 1$, $x_2 = 1$

Section 1 : Finding global min with one variable
Section 2 : Finding global min with multiple variables

→ derived from Taylor series

Gradient Descent

(a) Gradient descent

↳ batch update : $x_{k+1} = x_k + \lambda \sum_{n=1}^{N} \nabla L_n(x_k)$, where L is loss
∵ Expensive when training set is large

unconstrained optimisation with n variables is converted to unconstrained optimisation with 1 variable

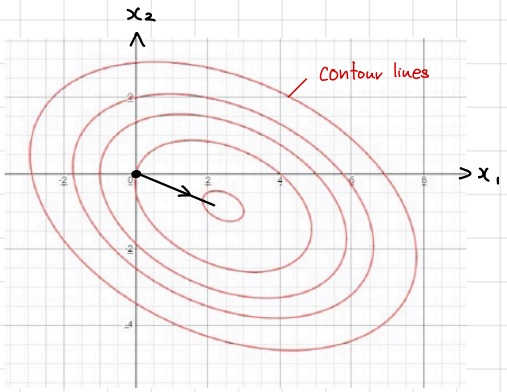
Initialisation Select some x_0.

Update If λ is not given :

Step 0. Find optimal step size (λ) using exact line search :
(1) Find direction $d = -\nabla f(x_k)$
(2) Find λ for $\min_{\lambda} f(x_k + \lambda)$ for some $\lambda > 0 \implies g(\lambda) = \frac{\partial}{\partial \lambda} f(x_k + \lambda d_k) = 0$
↳ Intuition for when λ is sub-optimal :
- If function value ↑, step is too large. Undo the step & ↓ step size.
- If function value ↓ too slowly, step is too small. ↑ step size.

Step 1 : $x_{k+1} = x_k - \lambda_k \nabla f(x_k)$

Use case 1 : min $f(x)$ for "non linear-combination"



Starting from initial point $x_0 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, compute x_1 using the steepest descent approach with optimal stepsize (exact line approach) to find the minimum of the function
$f(x_1, x_2) = x_1^3 + x_2^3 - 2x_1^2 + 3x_2^2 - 8$

① Find gradient: $\nabla f(x_1, x_2) = \begin{pmatrix} 3x_1^2 - 4x_1 \\ 3x_2^2 + 6x_2 \end{pmatrix}$
$\nabla f(1, -1) = \begin{pmatrix} -1 \\ -3 \end{pmatrix}$

② Since stepsize (λ) is not given, find λ : $d = -\nabla f(1, -1) = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$x + \lambda d = \begin{pmatrix} 1+\lambda \\ -1+3\lambda \end{pmatrix}$$

$$\nabla f(x + \lambda d) = \begin{pmatrix} 3(1+\lambda)^2 - 4(1+\lambda) \\ 3(-1+3\lambda)^2 + 6(-1+3\lambda) \end{pmatrix}$$

$$f'(\lambda) = \nabla f(x + \lambda d)^T d = 3(1+\lambda)^2 - 4(1+\lambda) + 9(-1+3\lambda)^2 + 18(-1+3\lambda)$$
$$= -10 + 2\lambda + 84\lambda^2 = 0$$

$\therefore \lambda = \frac{1}{3}$ or $\lambda = -\frac{5}{14}$ (rej. as $\lambda > 0$)

③ Find x_1 : $x_1 = x_0 + \lambda_1 d$
$= \begin{pmatrix} 1 \\ -1 \end{pmatrix} + \frac{1}{3} \begin{pmatrix} 1 \\ 3 \end{pmatrix}$
$= \begin{pmatrix} \frac{4}{3} \\ 0 \end{pmatrix}$

Use case 2: min sum of squared error

mxn-matrix, m-vector

$$E(x) = \frac{1}{2}\|Ax-b\|^2 = \frac{1}{2}(Ax-b)^T(Ax-b)$$
$$\nabla E(x) = A^T(Ax-b)$$

Note: If A^TA is invertible, x can be computed directly

At min $E(x)$, $\nabla E(x) = A^T(Ax-b) = 0$

$A^T(Ax-b) = A^TAx - A^Tb = 0 \Rightarrow x = (A^TA)^{-1}A^Tb$

symmetrical positive-definite nxn-matrix, n-vector

Use Case 3: min $f(x) = \frac{1}{2}x^TAx - b^Tx$

With this coefficient, $\nabla f(x) = Ax-b$

① $r_k = -\nabla f(x_k) = b - Ax_k$

② $\lambda_k = \frac{r_k^T r_k}{r_k^T A r_k}$

③ $x_{k+1} = x_k + \lambda_k r_k$

Given $f(x_1, x_2) = \frac{1}{2}\begin{pmatrix}x_1 & x_2\end{pmatrix}\begin{pmatrix}2 & 1\\ 1 & 20\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} - \begin{pmatrix}5 & 3\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}$, $\lambda = 0.085$, $x_0 = \begin{pmatrix}-3\\ -1\end{pmatrix}$

$\Rightarrow \nabla f(x_1, x_2) = \begin{pmatrix}2 & 1\\ 1 & 20\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} - \begin{pmatrix}5\\ 3\end{pmatrix}$

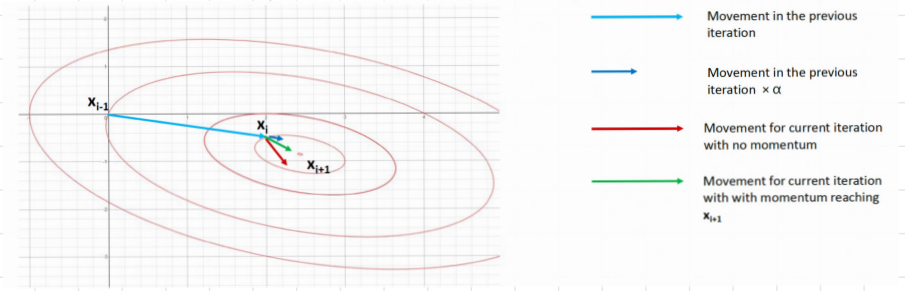
- At x_0, gradient $= \nabla f(-3,-1) = \begin{pmatrix}-12\\ -26\end{pmatrix}$
- $x_1 = x_0 - \lambda\nabla f(x_0) = \begin{pmatrix}-3\\ -1\end{pmatrix} - 0.085\begin{pmatrix}-12\\ -26\end{pmatrix} = \begin{pmatrix}-1.98\\ -1.21\end{pmatrix}$

Use case 4: Neural network ⇒ Refer to notes on "Neural Networks"

Termination condition E.g. $\|\nabla f(x_k)\| \leq \varepsilon$ / max # iterations

(b) Gradient descent with momentum

Modify step 2.2 to $x_{k+1} = x_k - \lambda_k\nabla f(x_k) + \alpha(x_k - x_{k-1})$, where $\alpha > 0$ is the momentum parameter



(c) Stochastic gradient descent

Approximate d_k using mini-batch gradient update or, in extreme case, randomly select one sample and update model weights with it.

Newton Method

↳ expensive ∵ $H(x_k)$ & $H^{-1}(x_k)$ must be computed in every iteration

Initialisation Select some starting point x_0.

Update $x_{k+1} = x_k - (H_k)^{-1} \nabla f(x_k)$

∵ If H_k is invertible, $\min f_k(x)$ is obtained when $\nabla f(x_k) + H_k(x - x_k) = 0$

Termination condition $\| \nabla f(x_k) \| \leq \varepsilon$

Quasi-newton Method

↳ Approximates $H(x_k)^{-1}$ using a positive definite matrix D_k

Initialisation Select some starting point x_0.

Update
- Set $g_k = \nabla f(x_k)$ and $s_k = -D_k g_k$
- $x_{k+1} = f(x_k) + \lambda_k s_k$
- Update D_k to D_{k+1} :

let $\delta_k = \lambda_k s_k = x_{k+1} - x_k$ and $\gamma_k = g_{k+1} - g_k$

(a) Symmetric rank-one (SR1) update: $D_{k+1} = D_k + \frac{(\delta_k - D_k \gamma_k)(\delta_k - D_k \gamma_k)^T}{(\delta_k - D_k \gamma_k)^T \gamma_k}$

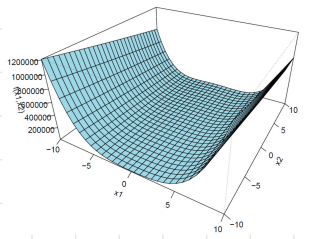
(b) David-Fletcher-Powell (DFP) update: $D_{k+1} = D_k + \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} - \frac{D_k \gamma_k \gamma_k^T D_k}{\gamma_k^T D_k \gamma_k}$

(c) Brayden-Fletcher-Goldfarb Shanno (BFGS) update: $D_{k+1} = D_k + \left(1 + \frac{\delta_k D_k \delta_k^T}{\delta_k^T \gamma_k}\right) \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} - \frac{\delta_k \gamma_k^T D_k + D_k \gamma_k \delta_k^T}{\delta_k^T \gamma_k}$

Use case 1 : min f(x) for "non linear-combination"

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

$g = \nabla f(x_1, x_2) = \begin{pmatrix} -400 x_1 (x_2 - x_1^2) - (1 - x_1) \\ -200(x_2 - x_1^2) \end{pmatrix}$



```
> fr <- function(x) { ## Rosenbrock Banana function
    x1 <- x[1]
    x2 <- x[2]
    100 * (x2 -x1 * x1)^2 + (1 -x1)^2
  }

> grr <- function(x) { ## Gradient of 'fr'
    x1 <-x[1]
    x2 <-x[2]
    c(-400 * x1 * (x2 -x1 * x1) -2 * (1 -x1), 200 * (x2 -x1 * x1))
}

> optim(c(-1.2,1), fr, grr, method = "BFGS")
```

```
$par
[1] 1 1

$value
[1] 9.594956e-18

$counts
function gradient
110 43

$convergence
[1] 0
```

Use case 2 : min sum of squared error

let $e_i = y_{actual} - y_{predicted}$

$\min S(\beta) = \frac{1}{2}(e_1^2 + e_2^2 + \dots + e_n^2) = \frac{1}{2}(y - X\beta)^T(y - X\beta)$

$\nabla_\beta S(\beta) = -X^T(y - X\beta)$

Data

X =

Size	Floor	Broadband rate
500	4	8
550	7	50
620	9	7
630	5	24
665	8	100
700	4	8
770	10	7
880	12	50
920	14	8
1000	9	24

Y =

Rental price
320
380
400
390
385
410
480
600
570
620

Input

```
errorFunction <- function(beta) {

    x <- rbind( c(1,500,4,8),
                c(1,550,7,50),
                c(1,620,9,7),
                c(1,630,5,24),
                c(1,665,8,100),
                c(1,700,4,8),
                c(1,770,10,7),
                c(1,880,12,50),
                c(1,920,14,8),
                c(1,1000,9,24))

    yactual <-rbind(320,
                    380,
                    400,
                    390,
                    385,
                    410,
                    480,
                    600,
                    570,
                    620)

    ypred <- X %*% beta

    error <- yactual - ypred

    (t(error) %*% error)/2

}

gradientFunction <- function(beta){

    X <- rbind( c(1,500,4,8),
                c(1,550,7,50),
                c(1,620,9,7),
                c(1,630,5,24),
                c(1,665,8,100),
                c(1,700,4,8),
                c(1,770,10,7),
                c(1,880,12,50),
                c(1,920,14,8),
                c(1,1000,9,24))
    XT <- t(X)

    ypred <- X %*% beta

    yactual <- rbind( 320,
                      380,
                      400,
                      390,
                      385,
                      410,
                      480,
                      600,
                      570,
                      620)

    error <- yactual -ypred
    deriv <- -XT %*% error
    c(deriv[1], deriv[2], deriv[3], deriv[4])}

res <- optim(c(0,0,0,0),
             fn = ErrorF,
             gr = GradF,
             method = "BFGS")
```

Output

$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = (X^TX)^{-1}X^Ty = \begin{pmatrix} 19.5615 \\ 0.5487 \\ 4.9635 \\ -0.0621 \end{pmatrix}$

```
$par
[1] 19.56155889 0.54873984 4.96354729 -0.06209514

$value
[1] 2242.282
```

→ optimal SSE = $\frac{1}{2}$ · (sum of sq errors) = 2242.282

```
$counts
functiongradient
48 8

$convergence
[1] 0

$message
NULL

> GradF(res$par)
[1] 1.587328e-06 1.469361e-03 4.441470e-05
1.688157e-04
```