Session 8. Advanced Optimization Models

* Algorithms for Optimization

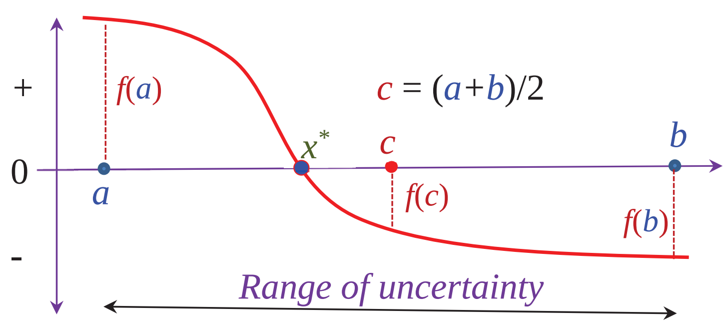
- *Direct methods*: Direct methods compute the solution to a problem in a *finite* number of iterations. These methods would give the precise answer if they were performed in infinite precision arithmetic.
- *Iterative methods*: Iterative methods approach the solution *gradually*, rather than in one large computational step. Therefore, when solving a problem with an iterative method, you can observe the error estimate in the solution *decreases* with the number of iterations.

* Bisection Method

- Let x^* be the root of $f(x)$, i.e., $f(x^*) = 0$. If x^* is known to be located in an initial bracket $[a, b]$, then *bisect* this interval into two intervals $[a, c]$ and $[c, b]$ where c is the *midpoint*.

  If $f(a)\, f(c) < 0$ and $f(c)\, f(b) > 0$, then x^* is located in $[a, c]$

  If $f(a)\, f(c) > 0$ and $f(c)\, f(b) < 0$, then x^* is located in $[c, b]$



- This process may now be *iterated* such that the size of the bracket (as well as the actual error of the estimate) is being divided by 2 every iteration.

A. Gradient Vector*

* Gradient of a Function

- Let $\mathbf{x} = [x_1, x_2, \ldots, x_k]'$ be a vector and $f(\mathbf{x})$ be a *differentiable* function.

- The *gradient* of the function $f(\mathbf{x})$, evaluated at $\mathbf{x}$, is defined as the k x 1 column vector, where ∇ (the *nabla* symbol) denotes the vector differential operator.

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(\boldsymbol{x}) \\ \frac{\partial}{\partial x_2} f(\boldsymbol{x}) \\ \cdots \\ \frac{\partial}{\partial x_k} f(\boldsymbol{x}) \end{bmatrix}$$

Ex 1] Consider the bivariate function, $f(x_1, x_2) = 2x_1^2 - 3x_1x_2 + x_2^2$.

(a) Compute the *partial derivatives* of $f(x_1, x_2)$.

$\frac{\partial}{\partial x_1} f(x_1, x_2) =$

$\frac{\partial}{\partial x_2} f(x_1, x_2) =$

(b) The gradient vector is $\nabla f(x_1, x_2) =$

Ex 2] Compute the *partial derivatives* of the function,

$$f(b_0, b_1) = \sum_{i=1}^{n} [y_i - b_0 - b_1 x_i]^2.$$

(Note that b_0 and b_1 are variables, while x_i and y_i are constants.)

- $\frac{\partial}{\partial b_0} f(b_0, b_1) =$

- $\frac{\partial}{\partial b_1} f(b_0, b_1) =$

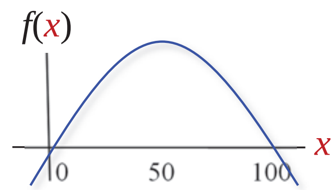
* Interpretation of the Gradient

(a) ***Single* variable**:

- In one dimension, the gradient of a function $f(x)$ is just the *derivative*, which is the *slope* of the *tangent line* to the graph of f at the point x.

Ex 1] Find the *gradient* of the function, $f(x) = -x^2 + 100x$.

$$\nabla f(x) = \frac{d}{dx} f(x) =$$



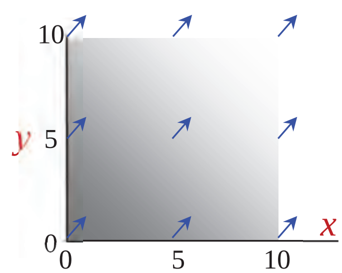
(b) ***Two* variables**:

- Consider a surface whose *height* level at a point x and y is $f(x, y)$. The gradient of the function f at a point is a vector pointing in the direction of the *steepest* slope or grade at that point.

- The *steepness* of the slope at that point is given by the *magnitude* of the gradient vector $\|\nabla f(x, y)\|$.

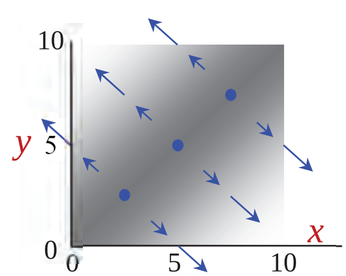
Ex 2] Consider $f(x, y) = x+y$.

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix} =$$



Ex 3] Consider $f(x, y)=(x-y)^2$.

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix} =$$



B. Gradient Ascent Method*

* Gradient Method

- The gradient method is a first-order optimization algorithm, which is also known as *steepest ascent* or *steepest decent*.
- The *gradient* of a function gives the direction of *steepest increase*. Thus, a natural maximization algorithm is to take steps proportional to the gradient of the function at the current point.

* *Maximization* Problem

- **Input**:

  Differentiable function $f(\mathbf{x})$

  Initial solution $\mathbf{x}^{(0)}$

  Learning rate $\delta > 0$

  Tolerance limit $\varepsilon > 0$

- **Output**

  Maximum point $\mathbf{x}^*$

- **Procedure**

  Step 1. $t = 1$

  Step 2. $\mathbf{x}^{(t)} = \mathbf{x}^{(t\text{-}1)} + \delta\ \nabla f(\mathbf{x}^{(t\text{-}1)})$

  Step 3. If $f(\mathbf{x}^{(t)}) - f(\mathbf{x}^{(t\text{-}1)}) > \varepsilon$, then

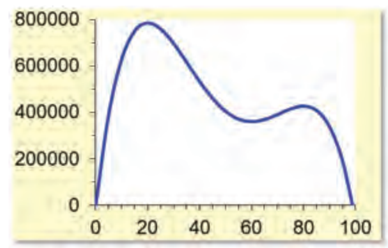
  $t = t+1$ and go to Step 2.

  else $\mathbf{x}^* = \mathbf{x}^{(t)}$ and stop.

It eventually results in the local *maximum* point because

$$f(\mathbf{x}^{(t+1)}) = f[\mathbf{x}^{(t)} + \delta\ \nabla f(\mathbf{x}^{(t)})\ ] > f(\mathbf{x}^{(t)}) > f(\mathbf{x}^{(t\text{-}1)}).$$

Ex 1] Local maximum: Consider the maximization problem. As shown in the graph, the global maximum is at x^*=20, while the local maximum is at x^*=80.



- *Max* $f(x) = -x^4/4 + 160x^3/3 - 3800x^2 + 96000x$
- Gradient: $\nabla f(x) = \frac{d}{dx} f(x) =$
- Iteration: $x^{(t)} = x^{(t\text{-}1)} + \delta\ \nabla f(x^{(t\text{-}1)})$

$$= x^{(t\text{-}1)} + \delta\{-[x^{(t\text{-}1)}]^3 + 160\ [x^{(t\text{-}1)}]^2 - 7600\ [x^{(t\text{-}1)}] + 96000\}$$

Let's consider four different cases with the same initial value $x^{(0)}$=0, but with a different learning rate δ.

- **Case 1**. If δ=0.0005, it finds the global maximum x^*=20.

Iteration, t	0	1	2	3	4	…
$x^{(t)}$	0	48	42.62	35.28	26.83	…

- **Case 2**. If δ=0.0009, it converges to the local maximum x^*=80.

Iteration, t	0	1	2	3	4	…
$x^{(t)}$	0	86.4	76.30	79.36	80.02	…

- **Case 3**. If δ=0.0010, x^* oscillates between 14.94 and 29.78.

Iteration, t	0	1	2	3	4	…
$x^{(t)}$	0	96	52.22	45.26	32.33	…

- **Case 4**. If δ=0.0012, it *diverges*.

Iteration, t	0	1	2	3	4	…
$x^{(t)}$	0	115.2	-106.8	4632	$-\infty$	…

Ex 2] Multivariate Optimization

A monopolist produces a single product for two types of customer. To maximize profit, how much should the monopolist sell to each customer?

- Objective function

$$Max\ f(x_1, x_2) = x_1\,(70\text{-}4x_1)+x_2\,(150\text{-}15x_2) - [100+15(x_1+x_2)]$$

- Gradient vector

$$\nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x_1, x_2) \\ \frac{\partial}{\partial x_2} f(x_1, x_2) \end{bmatrix} =$$

(1) Analytical solution:

Set the gradient vector equal to 0 and solve the equations!

Then, $x_1^* =$ and $x_2^* =$

(2) Steepest ascent algorithm: $\mathbf{x}^{(t)} = \mathbf{x}^{(t\text{-}1)} + \delta\ \nabla f(\mathbf{x}^{(t\text{-}1)})$ or

$$\begin{bmatrix} x_1^{(t)} \\ x_2^{(t)} \end{bmatrix} = \begin{bmatrix} x_1^{(t-1)} \\ x_2^{(t-1)} \end{bmatrix} + \delta \begin{bmatrix} 55 - \ \ 8x_1^{(t-1)} \\ 135 - 30x_2^{(t-1)} \end{bmatrix}.$$

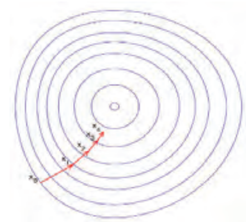
Table. 20 iterations with the learning rate $\delta = 0.05$ and the initial value $\mathbf{x}^{(0)} = [0, 0]$.

t	0	1	2	3	4	..	10	..	20
$x_1^{(t)}$	0	2.75		5.390	5.984	..	6.833	..	6.875
$x_2^{(t)}$	0	6.75		5.063	4.219	..	4.509	..	4.500

C. Gradient Descent Method*

* *Minimization* Problem

- If you take steps proportional to the *negative* of the gradient, you approach a local *minimum* of the function; the procedure is then known as *gradient descent*.



- **Input**:

Differentiable function	$f(\mathbf{x})$
Initial solution	$\mathbf{x}^{(0)}$
Learning rate	$\delta > 0$
Tolerance limit	$\varepsilon > 0$

- **Output**

Minimum point	$\mathbf{x}^*$

- **Procedure**

Step 1. $t = 1$

Step 2. $\mathbf{x}^{(t)} = \mathbf{x}^{(t-1)} - \delta\, \nabla f(\mathbf{x}^{(t-1)})$

Step 3. If $f(\mathbf{x}^{(t-1)}) - f(\mathbf{x}^{(t)}) > \varepsilon$, then
$t = t+1$ and go to Step 2.
else $\mathbf{x}^* = \mathbf{x}^{(t)}$ and stop.

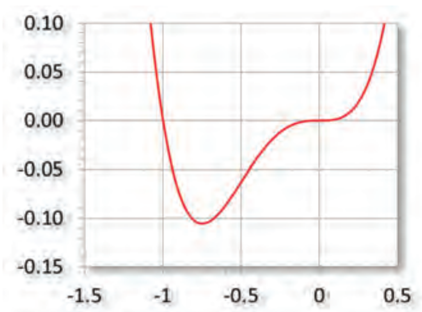
It eventually results in the local *minimum* point because

$$f(\mathbf{x}^{(t+1)}) = f[\mathbf{x}^{(t)} - \delta\, \nabla f(\mathbf{x}^{(t)})\,] < f(\mathbf{x}^{(t)}) < f(\mathbf{x}^{(t-1)}).$$

How to determine the learning rate δ and the initial values $\mathbf{x}^{(0)}$?

Ex 1] Apply the steepest descent algorithm to find a local minimum of the univariate function,

$$f(x) = x^4 + x^3.$$



- *Min* $f(x) = x^4 + x^3$
- Gradient: $\nabla f(x) = \dfrac{d}{dx} f(x) =$

  Thus, the stationary points are $x^* =$
- Iteration: $x^{(t)} = x^{(t\text{-}1)} - \delta\ \nabla f(x^{(t\text{-}1)})$

  $= x^{(t\text{-}1)} - \delta\ \{\ 4[x^{(t\text{-}1)}]^3 + 3\ [x^{(t\text{-}1)}]^2\ \}$
- Initial value is $x^{(0)} = -1.1$ and the learning rate is δ.

- **Case 1**. If δ=0.9, it finds the global minimum x^* = -0.75.

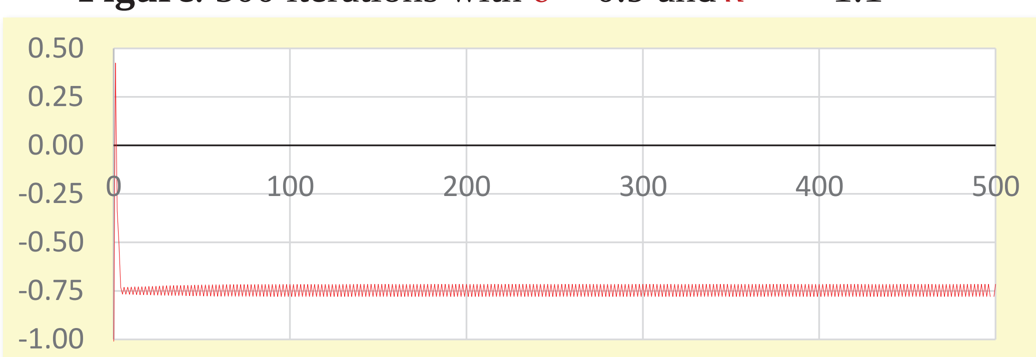
Iteration, t	0	1	2	3	4	...
$x^{(t)}$	-1.1	0.425	-0.338	-0.507	-0.732	...

- **Case 2**. If δ=0.8, it reaches at the inflection point x^* = 0.

5

Iteration, t	0	1	2	3	4	...
$x^{(t)}$	-1.1	0.255	0.046	0.040	0.036	...

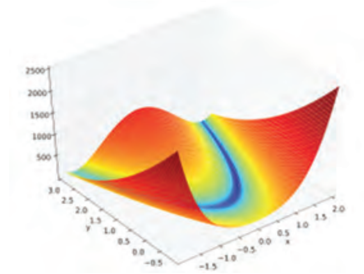
Figure. 500 iterations with δ = 0.9 and $x^{(0)}$ = -1.1



Ex 2] Rosenbrock Function

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

which is a non-convex function used as a performance test problem for optimization algorithms.



- Gradient vector

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

(1) Analytical solution? $\quad x_1^* =$ $\qquad$ and $x_2^* =$

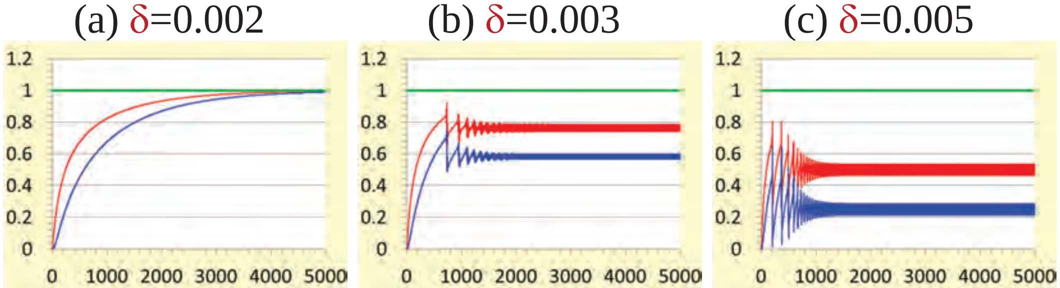
(2) Steepest descent algorithm: $\quad \mathbf{x}^{(t)} = \mathbf{x}^{(t\text{-}1)} - \delta\, \nabla f(\mathbf{x}^{(t\text{-}1)})$

which is $\begin{bmatrix} x_1{}^{(t)} \\ x_2{}^{(t)} \end{bmatrix} = \begin{bmatrix} x_1{}^{(t-1)} \\ x_2{}^{(t-1)} \end{bmatrix}$

$$-\delta \begin{bmatrix} -400x_1{}^{(t-1)}\{x_2{}^{(t-1)} - [x_1{}^{(t-1)}]^2\} - 2\{1 - x_1{}^{(t-1)}\} \\ 200\,\{x_2{}^{(t-1)} - [x_1{}^{(t-1)}]^2\} \end{bmatrix}.$$

Figure. 5,000 iterations with δ and $\mathbf{x}^{(0)} = [0, 0]$.

(a) δ=0.002 $\qquad$ (b) δ=0.003 $\qquad$ (c) δ=0.005



*D. Gradient Methods for Prediction and Classification**

I. Gradient Descent for Regression Models

- *Minimization* of the *sum of squared errors* (*SSE*)

  - Objective function: $f(\mathbf{b})=\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n}[y_i - \hat{y}_i(\boldsymbol{b})]^2$,

    where $\mathbf{b}=[b_0, b_1, \ldots, b_k]$ is a parameter vector.

  - Gradient vector of the objective function $f(\mathbf{b})$ for the kth parameter:

$$\nabla f(b_k) = \left[\frac{\partial}{\partial b_k} f(\boldsymbol{b})\right] = \left[\frac{\partial}{\partial b_k}\sum_{i=1}^{n}[y_i - \hat{y}_i(\boldsymbol{b})]^2\right]$$

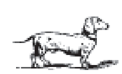
$$= -2\left[\sum_{i=1}^{n}[y_i - \hat{y}_i(\boldsymbol{b})]\ \frac{\partial}{\partial b_k}\hat{y}_i(\boldsymbol{b})\right]$$

$$= -2\left[\sum_{i=1}^{n} e_i \frac{\partial}{\partial b_k}\hat{y}_i(\boldsymbol{b})\right].$$

  - Steepest *descent* method for the minimization problem:

$$b_k^{(t)} = b_k^{(t-1)} - \delta\ \nabla f(\mathbf{b}_k^{(t-1)}) \text{ or}$$

$$b_k^{(t)} = b_k^{(t-1)} + 2\delta\ \sum_{i=1}^{n} e_i^{(t-1)}\ \frac{\partial}{\partial b_k}\hat{y}_i(\boldsymbol{b}).$$



Model, $\hat{y}_i(\boldsymbol{b})$	Gradient vector
▪ Polynomial regression $\hat{y}_i(\mathbf{b})=b_0+b_1x_i+b_2x_i^2$	$\frac{\partial}{\partial b_k}\hat{y}_i(\boldsymbol{b}) = \begin{cases} 1 & \text{for } k=0 \\ x_i & \text{for } k=1 \\ x_i^2 & \text{for } k=2 \end{cases}$
▪ Linear regression $\hat{y}_i(\mathbf{b})=b_0+b_1x_i+\ldots+b_kx_k$	$\frac{\partial}{\partial b_k}\hat{y}_i(\boldsymbol{b}) = \begin{cases} 1 & \text{for } k=0 \\ x_k & \text{for } k\geq 1 \end{cases}$

Ex 1] Simple Linear Regression Model

y_i	x_i
3	2
4	4
6	3
5	5

Find the least square estimates $\mathbf{b}=(b_0, b_1)$ that minimize the sum of squared errors (SSE).

(a) Matrix approach

$$\underset{4\times1}{\mathbf{y}} = \begin{bmatrix}3\\4\\6\\5\end{bmatrix} \quad \text{and} \quad \underset{4\times2}{\mathbf{X}} = \begin{bmatrix}1 & 2\\1 & 4\\1 & 3\\1 & 5\end{bmatrix}.$$

$$\text{Thus, } \mathbf{b} = (\underset{2\times4}{\mathbf{X}'}\underset{4\times2}{\mathbf{X}})^{-1}(\underset{2\times4}{\mathbf{X}'}\underset{4\times1}{\mathbf{y}}) = \begin{bmatrix}3.1\\0.4\end{bmatrix}.$$

(b) Microsoft Excel – Data Analysis

Regression Statistics	
Multiple R	0.4
R Square	0.16
Adjusted R Square	-0.26
Standard Error	1.4491
Observations	4

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	1	0.8	0.8	0.3810	0.6
Residual	2	4.2	2.1		
Total	3	5.0			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*
Intercept	3.1	2.381	1.302	0.323	-7.145	13.345
X	0.4	0.648	0.617	0.6	-2.388	3.188

(c) Gradient descent method with the learning rate $\delta=0.01$ and the initial solutions, $b_0^{(0)}=3$ and $b_1^{(0)}=0.3$.

y_i	x_i	$\hat{y}_i$	e_i	$e_i x_i$
3	2	3.6	-0.6	-1.2
4	4			
6	3	3.9	2.1	6.3
5	5	4.5	0.5	2.5
$b_0^{(0)}=3$	$b_1^{(0)}=0.3$	Total =		

▪ **Step 1**. The solutions at t=1 are $b_0^{(1)}$ =3.036 and $b_1^{(1)}$=0.436,

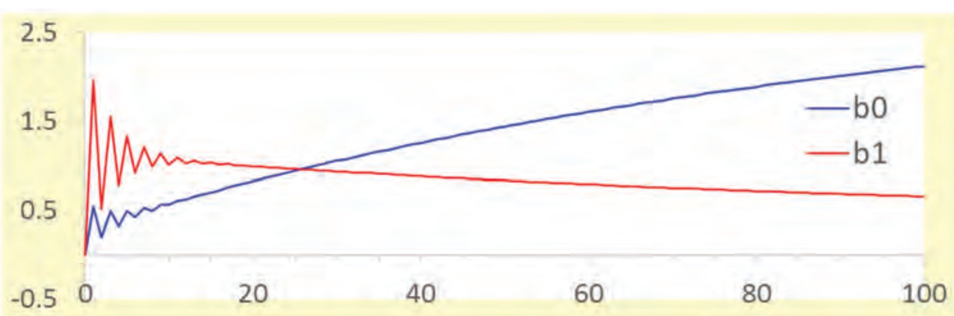
because $\begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \end{bmatrix} = \begin{bmatrix} 3.0 \\ 0.3 \end{bmatrix} + 2 \times 0.01 \begin{bmatrix} 1.8 \\ 6.8 \end{bmatrix}$ = .

y_i	x_i	$\hat{y}_i$	e_i	$e_i x_i$
3	2	3.908	-0.908	-1.816
4	4			
6	3	4.344	1.656	4.968
5	5	5.216	-0.216	-1.080
$b_0^{(1)}=3.036$	$b_1^{(1)}=0.436$	Total =		

▪ **Step 2**. The solutions at t=2 are $b_0^{(2)}$= and $b_1^{(2)}$=

because $\begin{bmatrix} b_0^{(2)} \\ b_1^{(2)} \end{bmatrix} = \begin{bmatrix} 3.036 \\ 0.436 \end{bmatrix} + 2 \times 0.01 \begin{bmatrix} -0.248 \\ -1.048 \end{bmatrix}$=

Figure. 100 iterations with δ =0.015 and $\mathbf{b}^{(0)}$ = [0, 0].



They approach the optimal solutions, $\mathbf{b}^*$=[3.1, 0.4].

II. Gradient Ascent for Logistic Regression Model

▪ *Maximization* of the *log-likelihood* function:

- Objective function

$$f(\mathbf{b}) = \sum_{i=1}^{n} y_i \, ln \, \pi_i + \sum_{i=1}^{n} (1 - y_i) \, ln(1 - \pi_i),$$

where $\pi_i = \frac{1}{1+exp(-\boldsymbol{b}\boldsymbol{x}_i)}$.

- Gradient vector of the objective function $f(\mathbf{b})$ for the kth parameter:

$$\nabla f(b_k)$$

$$= \frac{\partial}{\partial b_k} \sum_{i=1}^{n} y_i \, ln \, \pi_i + \frac{\partial}{\partial b_k} \sum_{i=1}^{n} (1 - y_i) \, ln(1 - \pi_i)$$

$$= \sum_{i=1}^{n} y_i \frac{1}{\pi_i} \frac{\partial}{\partial b_k} \pi_i + \sum_{i=1}^{n} (1 - y_i) \frac{1}{1-\pi_i} \frac{\partial}{\partial b_k} (1 - \pi_i)$$

$$= \left(\sum_{i=1}^{n} \left[\frac{y_i}{\pi_i} - \frac{(1-y_i)}{1-\pi_i} \right] \right) \frac{\partial}{\partial b_k} \pi_i$$

$$= \left(\sum_{i=1}^{n} \left[\frac{y_i - \pi_i}{\pi_i(1-\pi_i)} \right] \right) \frac{\partial}{\partial b_k} \pi_i,$$

which can be shown to be

$$\nabla f(b_k) = \sum_{i=1}^{n} (y_i - \pi_i) x_{i,k}.$$

- Steepest *ascent* method:

$b_k^{(t)} = b_k^{(t-1)} + \delta \; \nabla f(\mathbf{b}_k^{(t-1)})$ or

$$b_k^{(t)} = b_k^{(t-1)} + \delta \; \sum_{i=1}^{n} \left(y_i - \pi_i^{(t)} \right) x_{i,k}.$$

Ex 2] Logistic Regression Model

Consider the simple logistic regression model for a classification problem with 8 sample observations.

	A	B	C	D	E	F
1	y_i	x_i	$b_0+b_1x_i$	π_i	$y_i-\pi_i$	$(y_i-\pi_i)x_i$
2	1	4	1.0	0.731	0.269	1.076
3	1	6	2.0	0.881	0.119	0.715
4	1	8	3.0	0.953	0.047	0.379
5	1	9	3.5	0.971	0.029	0.264
6	0	1	-0.5	0.378	-0.378	-0.378
7	0	3	0.5	0.622	-0.622	-1.867
8	0	5	1.5	0.818	-0.818	-4.088
9	0	7	2.5	0.924	-0.924	-6.469
10	b_0	b_1		Sum=	-2.277	-10.368
11	-1.0	0.5				

Suppose that the solution at t-1 is $\mathbf{b}^{(t-1)} = [-1, 0.5]$. Find the solution at t with the learning rate $\delta = 0.1$.

$$\begin{bmatrix} b_0^{(t)} \\ b_1^{(t)} \end{bmatrix} = \begin{bmatrix} b_0^{(t-1)} \\ b_1^{(t-1)} \end{bmatrix} + \delta \begin{bmatrix} \sum_{i=1}^{n} \left[y_i - \pi_i^{(t-1)}\right] \\ \sum_{i=1}^{n} \left[y_i - \pi_i^{(t-1)}\right] x_i \end{bmatrix}$$

$$= \begin{bmatrix} -1 \\ 0.5 \end{bmatrix} + 0.1 \begin{bmatrix} -2.277 \\ -10.368 \end{bmatrix} =$$