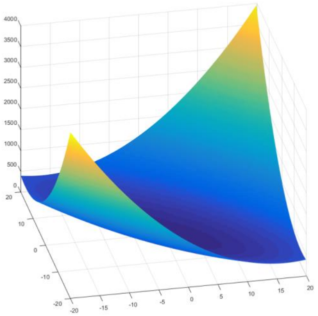
Lecture 14

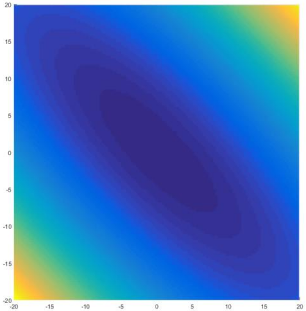
CM50264: Machine Learning 1

Optimization Basics 2

Kwang In Kim

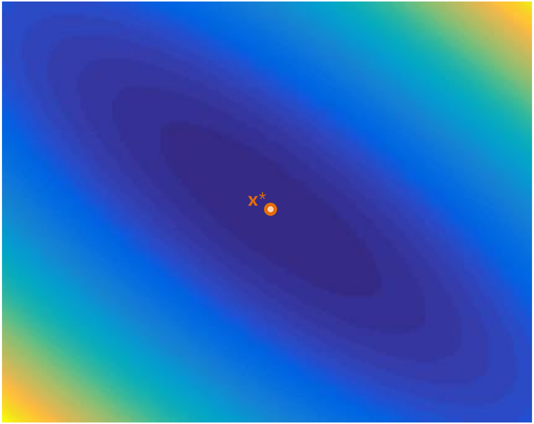
Previously in optimization basics ...
A simple 2D optimization problem





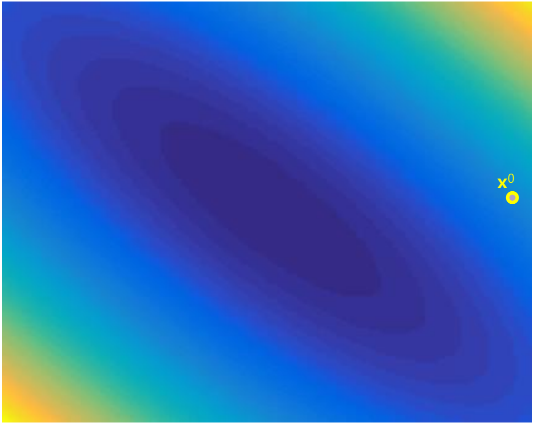
Our objective function f is two-dimensional: $f : \mathbb{R}^2 \rightarrow \mathbb{R}$.

A simple 2D optimization problem



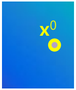
Our objective function f is two-dimensional: $f : \mathbb{R}^2 \to \mathbb{R}$.

A simple 2D optimization problem



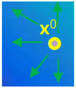
We start the optimization with an initial (zero-th) solution $\mathbf{x}^0$.

A simple 2D optimization problem



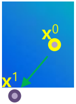
Again, we can observe f only in a small neighborhood of $\mathbf{x}^0$.

A simple 2D optimization problem



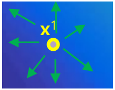
By inspecting f around $\mathbf{x}^0$ ...

A simple 2D optimization problem



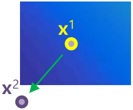
We decide a direction to move (to decrease the f value).

A simple 2D optimization problem



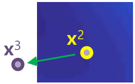
Now we are at the first solution $\mathbf{x}^1$, and observing f around ...

A simple 2D optimization problem



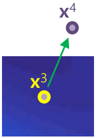
and decide a new direction ...

A simple 2D optimization problem



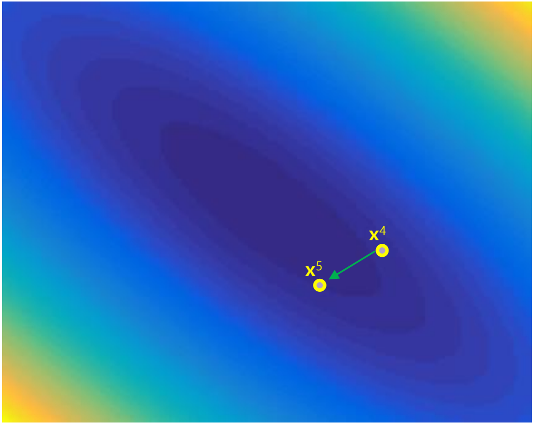
From the second solution to the third ...

A simple 2D optimization problem



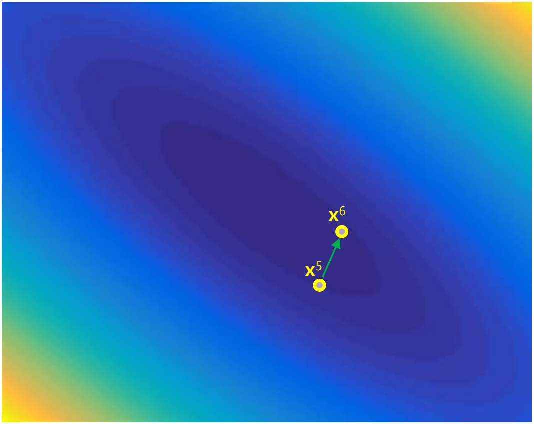
from the third solution to the fourth ...

A simple 2D optimization problem



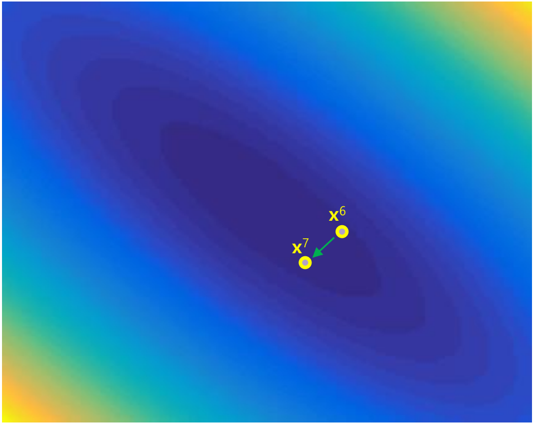
from $\mathbf{x}^4$ to $\mathbf{x}^5$ ...

A simple 2D optimization problem



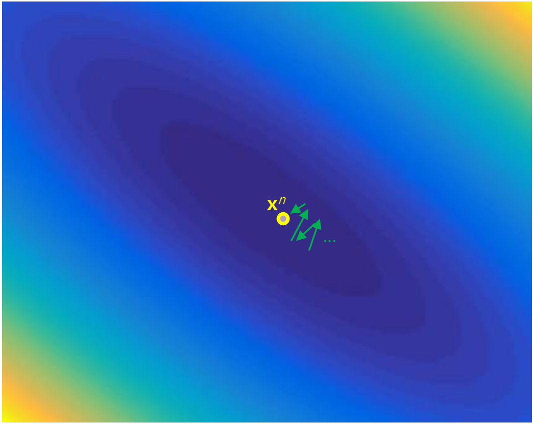
from $\mathbf{x}^5$ to $\mathbf{x}^6$ ...

A simple 2D optimization problem



from $\mathbf{x}^6$ to $\mathbf{x}^7$ ...

A simple 2D optimization problem



After a few iterations, we arrive at the optimum $\mathbf{x}^n = \mathbf{x}^*$.

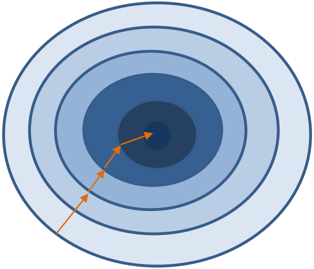
Steepest descent

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \alpha^t \mathbf{p}^t;$$

- $\mathbf{p}^t = -\nabla f(\mathbf{x}^t)$.
- How do we decide the step size $\alpha^t > 0$?
  $\rightarrow$ A simple solution (among others): fix it to a constant value α.
- When to terminate the optimization process?
  $\rightarrow$ A simple solution (among others):
  terminate when $\frac{\|f(\mathbf{x}^t)\|}{\|f(\mathbf{x}^0)\|} < T$ for a constant $T > 0$.

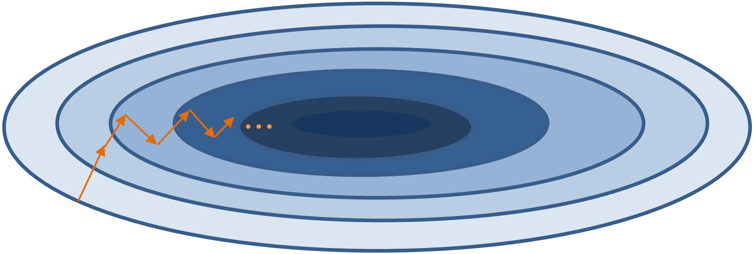
Steepest descent

Good for isotropic functions.



Steepest descent

Not good for anisotropic functions.

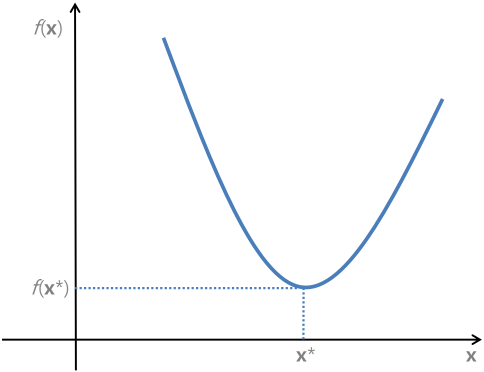


Steepest descent

Not good for anisotropic functions.

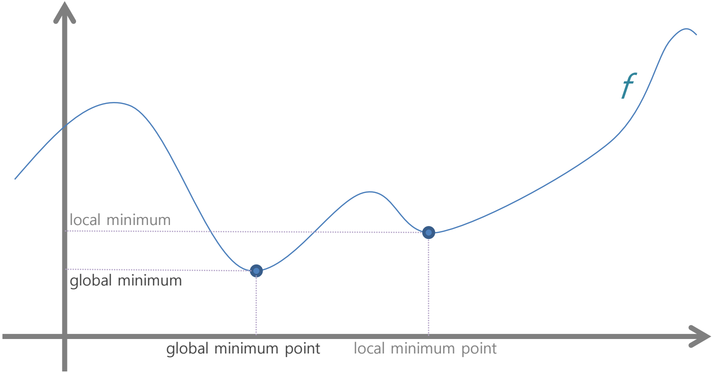
Steepest descent

finds the global minimum $\mathbf{x}^*$ for



Steepest descent

may not find the global minimum $\mathbf{x}^*$ when f looks like



Finding the global optimum is difficult even in 1D case.

- Global minimum (point) $\mathbf{x}^*$: $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$.
- Local minimum (point): a point which becomes a global minimum point when we restrict the domain to a (arbitrary small) neighborhood of itself.

Optimization (minimization) problem

Given a function $f : \mathcal{X} \subset \mathbb{R}^n \mapsto \mathbb{R}$, find an element $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$.

- Maximization can be converted to a minimization by multiplying f by -1.
- Optimization problem can be accompanied by constraints (constrained optimization problem):

$$
\begin{aligned}
\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathcal{X}} & f(\mathbf{x}), \\
\text{s.t. } & g_i(\mathbf{x}) \leq 0, i = \{1, \ldots, k\}, \\
& h_j(\mathbf{x}) = 0, j = \{1, \ldots, l\}.
\end{aligned}
$$

$\{g_i\}$ and $\{h_j\}$ are (inequality & equality) *constraint functions*.

Optimization problem types

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}),$$
$$\text{s.t. } g_i(\mathbf{x}) \leq 0, i = \{1, \ldots, k\},$$
$$h_j(\mathbf{x}) = 0, j = \{1, \ldots, l\}.$$

- Constrained vs. unconstrained optimization.
- Discrete vs. continuous optimization.
- Deterministic vs. stochastic optimization.
- Convex vs. non-convex optimization.

Linear programming

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{c}^\top \mathbf{x},$$
$$\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b},$$
$$\mathbf{x} \geq 0.$$

Example: a company manufactures two types of boxes, A and B.

- The boxes undergo two major processes: cutting and pinning operations.
- The profits per unit are 6 for A and 4 for B.
- A requires 2 minutes for cutting and 3 minutes for pinning.
- B requires 2 minutes for cutting and 1 minutes for pinning.
- Available operating time is 120 minutes and 60 minutes for cutting and pinning machines.
- We have to decide the optimum A and B quantities to maximize the profits.

Linear programming

- The boxes undergo two major processes: cutting and pinning operations.
- The profits per unit are 6 for A and 4 for B.
- A requires 2 minutes for cutting and 3 minutes for pinning.
- B requires 2 minutes for cutting and 1 minutes for pinning.
- Available operating time is 120 minutes and 60 minutes for cutting and pinning machines.
- We have to decide the optimum A and B quantities to maximize the profits.

$$
\begin{aligned}
\mathbf{x} &= [x_A, x_B]^\top \\
\mathbf{x}^* &= \arg\max_{\mathbf{x}\in\mathbb{R}^2} f(\mathbf{x}) = 6x_A + 4x_B \text{ (objective function)} \\
&\qquad \text{s.t. } 2x_A + 2x_B \le 120 \text{ (cutting constraint)} \\
&\qquad\qquad 3x_A + x_B \le 60 \text{ (pinning constraint)} \\
&\qquad\qquad x_A, x_B \ge 0 \text{ (box quantities cannot be negative)}\ .
\end{aligned}
$$

Integer programming

$$
\begin{aligned}
\mathbf{x} &= [x_A, x_B]^\top \\
\mathbf{x}^* &= \arg\max_{\mathbf{x}\in\mathbb{R}^2} f(\mathbf{x}) = 6x_A + 4x_B \text{ (objective function)} \\
&\quad \text{s.t. } 2x_A + 2x_B \leq 120 \text{ (cutting constraint)} \\
&\qquad 3x_A + x_B \leq 60 \text{ (pinning constraint)} \\
&\qquad x_A, x_B \geq 0 \text{ (box quantities cannot be negative)} .
\end{aligned}
$$

If box quantities need to be integers ...

$$
\begin{aligned}
\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathbb{Z}^2} \; & 6x_A + 4x_B, \\
\text{s.t. } & 2x_A + 2x_B \leq 120, \\
& 3x_A + x_B \leq 60 \\
& x_A, x_B \geq 0.
\end{aligned}
$$

- Quadratic programming:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{c}^\top \mathbf{x},$$
$$\text{s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{b},$$

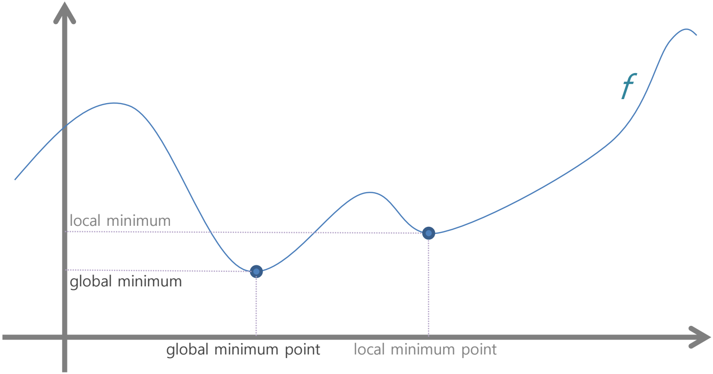
  where $\mathbf{Q}$ is a real symmetric matrix.

- Nonlinear programming:

$$\begin{aligned} \mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} & f(\mathbf{x}), \\ \text{s.t. } g(\mathbf{x}) &\leq 0, \\ h(\mathbf{x}) &= 0, \end{aligned}$$

  where f, g, h are all nonlinear.

Optimization



Finding the global optimum is difficult even in 1D case.

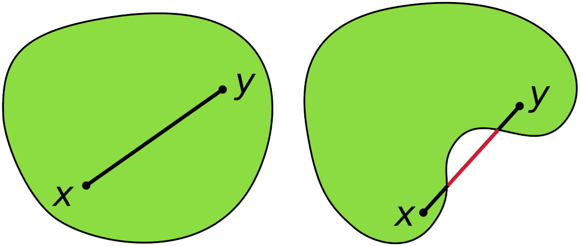
- Global minimum (point) $\mathbf{x}^*$: $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$.
- Local minimum (point): a point which becomes a global minimum point when we restrict the domain to a (arbitrary small) neighborhood of itself.

Convex set

A subset C of a vector space is called convex if $\forall \mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$

$$\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in C.$$

examples:

- 

- $\mathbb{R}^n$
- a cone in $\mathbb{R}^n$

[Wikipedia]

Convex function

A function on a convex set C is

- convex if

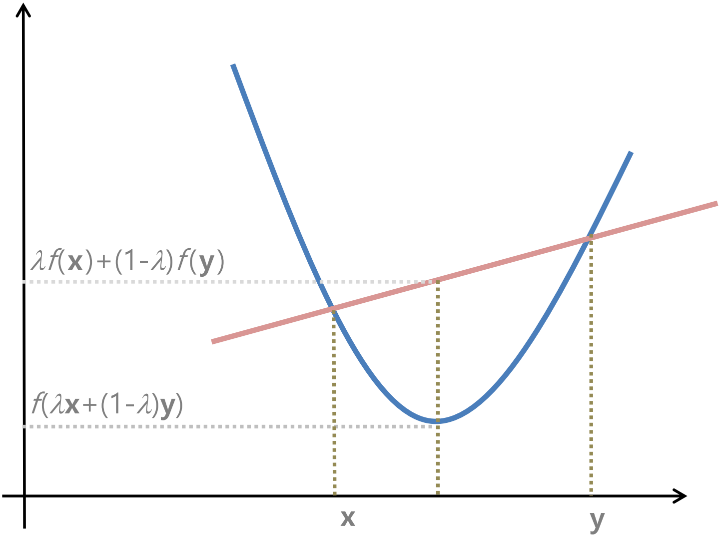
$$f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}),$$

- strictly convex if

$$f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) < \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y})$$

$\forall \mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$.

Convex function



Convex function

If f is twice continuously differentiable,

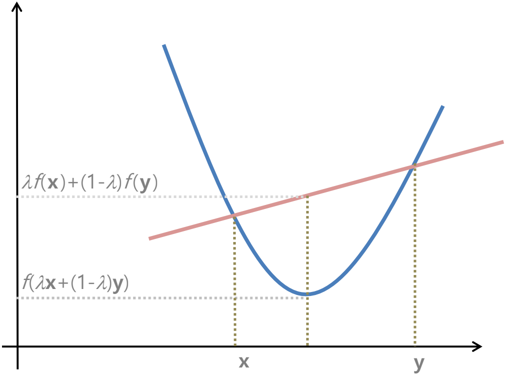
- $f : \mathbb{R} \mapsto \mathbb{R}$ is convex if $\frac{d^2 f}{d\mathbf{x}^2}$ is positive everywhere.
- $f : \mathbb{R}^n \mapsto \mathbb{R}$ is convex if the Hessian matrix is positive definite everywhere.

Example: second-order polynomial

$$p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c$$

is convex if $\mathbf{A}$ is (symmetric) positive definite:
The Hessian matrix $\frac{d^2 p}{d\mathbf{x}^2}[\mathbf{x}]$ of p is $\mathbf{A}$ for all $\mathbf{x}$.

Convex optimization



$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}),$$
$$\text{s.t. } g(\mathbf{x}) \leq 0,$$

where $f, g : \mathbb{R}^n \to \mathbb{R}$ are convex.

Convex function

- If f is twice continuously differentiable, $f : \mathbb{R} \mapsto \mathbb{R}$ is convex if $\frac{d^2 f}{d\mathbf{x}^2}$ is positive everywhere.
- If f is twice continuously differentiable, $f : \mathbb{R}^n \mapsto \mathbb{R}$ is convex if the Hessian matrix $Hf|_{\mathbf{x}}$ is positive definite everywhere.
- The second order polynomial

$$p(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c$$

  is convex if $\mathbf{A}$ is positive definite.
- An increasing function of a convex function is convex, e.g. if $f(x)$ is convex, $\log(f(x))$ and $\exp(f(x))$ are convex.
- If f and g are convex, $f + g$ is convex.
- If a local optimum exists for a convex function f, it is a global minimum.
- A point $\mathbf{x}^*$ is a local minimum if $\nabla_f(\mathbf{x}^*) = 0$.

(symmetric) Positive definite matrix

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} > 0, \forall \mathbf{x} \neq 0$$

- $\mathbf{A}$ is invertible and $\mathbf{A}^{-1}$ is positive definite.
- Eigenvalues of $\mathbf{A}$ are positive:

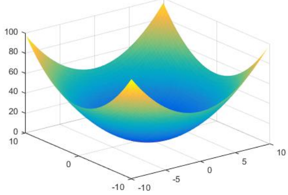
$$\mathbf{x}^\top \mathbf{A}\mathbf{x} = \mathbf{x}^\top (\mathbf{E}\Lambda\mathbf{E}^{-1})\mathbf{x} = \mathbf{x}^\top (\mathbf{E}\Lambda\mathbf{E}^\top)\mathbf{x} = (\mathbf{E}^\top\mathbf{x})^\top \Lambda (\mathbf{E}^\top \mathbf{x}).$$

$$\Rightarrow \mathbf{x}^\top \mathbf{A}\mathbf{x} > 0 \text{ if elements of } \Lambda \text{ are positive.}$$

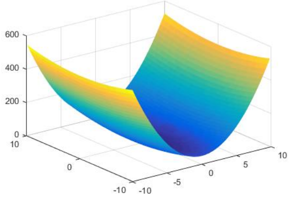
2D examples

$$p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}.$$

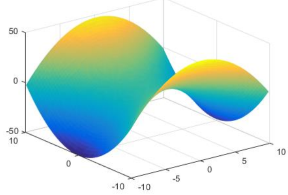
$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$



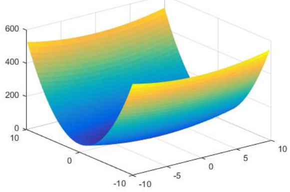
$\mathbf{A} = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}$



$\mathbf{A} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$



$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}$



2D example 1

Eigen-decomposition of the diagonal matrix $\mathbf{A} = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}$:

$$\begin{aligned} \mathbf{A} &= \mathbf{E}\Lambda\mathbf{E}^{-1} = \mathbf{E}\Lambda\mathbf{E}^{\top} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ &= (\mathbf{e}^1, \mathbf{e}^2) \begin{pmatrix} \lambda^1 & 0 \\ 0 & \lambda^2 \end{pmatrix} (\mathbf{e}^1, \mathbf{e}^2)^{\top} \\ &= \lambda^1 \mathbf{e}^1 {\mathbf{e}^1}^{\top} + \lambda^2 \mathbf{e}^2 {\mathbf{e}^2}^{\top}, \end{aligned}$$

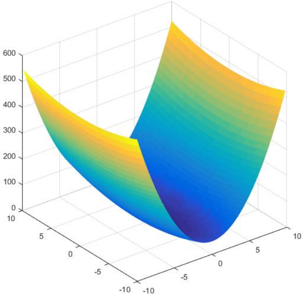
where

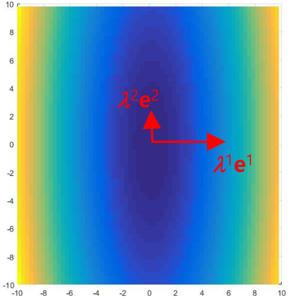
$$\mathbf{e}^1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \mathbf{e}^2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \lambda^1 = 10, \lambda^2 = 1.$$

Note: Eigenvectors of **A** form a basis.

2D example 1

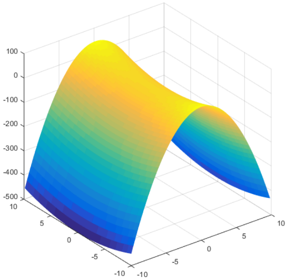
$$p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}, \mathbf{A} = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} = \lambda^1 \mathbf{e}^1 \mathbf{e}^{1\top} + \lambda^2 \mathbf{e}^2 \mathbf{e}^{2\top}.$$

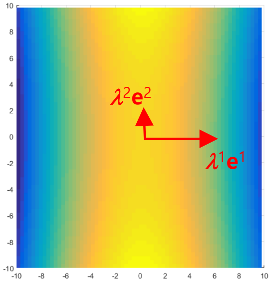




2D example 2

$$p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}, \mathbf{A} = \begin{pmatrix} -10 & 0 \\ 0 & 1 \end{pmatrix} = \lambda^1 \mathbf{e}^1 \mathbf{e}^{1\top} + \lambda^2 \mathbf{e}^2 \mathbf{e}^{2\top}.$$





Optimization problem types

Convex functions

Steepest descent

Newton method

2D example 3

Eigen-decomposition of symmetric matrix $\mathbf{A} = \begin{pmatrix} 5.5 & 4.5 \\ 4.5 & 5.5 \end{pmatrix}$:

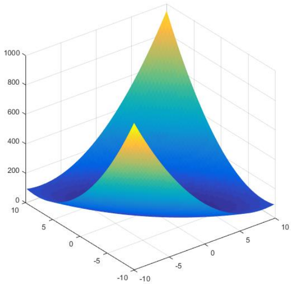
$$\begin{aligned}
\mathbf{A} &= \mathbf{E}\Lambda\mathbf{E}^{-1} = \mathbf{E}\Lambda\mathbf{E}^\top \\
&= \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \\
&= (\mathbf{e}^1, \mathbf{e}^2) \begin{pmatrix} \lambda^1 & 0 \\ 0 & \lambda^2 \end{pmatrix} (\mathbf{e}^1, \mathbf{e}^2)^\top \\
&= \lambda^1 \mathbf{e}^1 {\mathbf{e}^1}^\top + \lambda^2 \mathbf{e}^2 {\mathbf{e}^2}^\top,
\end{aligned}$$

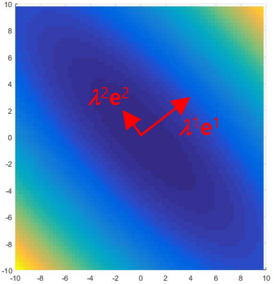
where

$$\mathbf{e}^1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}, \mathbf{e}^2 = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}, \lambda^1 = 10, \lambda^2 = 1.$$

2D example 3

$$p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}, \mathbf{A} = \begin{pmatrix} 5.5 & 4.5 \\ 4.5 & 5.5 \end{pmatrix} = \lambda^1 \mathbf{e}^1 {\mathbf{e}^1}^\top + \lambda^2 \mathbf{e}^2 {\mathbf{e}^2}^\top.$$





Gradient descent

What do we have to consider?

- Which direction ($\mathbf{q}_k$) to move?
- How far (α_k) should it move?
- When to stop the iteration?

When to stop gradient descent?

In practice the sufficient conditions

- $\nabla_f(\mathbf{x}^*) = 0$
- $H_f(\mathbf{x}^*)$ is positive definite

may be hard to satisfy exactly.

Alternative stopping criteria:

- $\|\nabla_f(\mathbf{x}_k)\| < T_1$,
- $\left\|\frac{\nabla_f(\mathbf{x}_k)}{\nabla_f(\mathbf{x}_0)}\right\| < T_2$,

where $T_1, T_2 \geq 0$ are predetermined (hyper-)parameters.

How far should it move?
Line search step (choosing α_k)

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{q}_k$$

Various methods for choosing α_k:

- Fix at a small constant: often, the case for training neural networks.
- Decay as k increases: e.g., for training neural networks.
- Solve a one-dimensional optimization problem.
- Choose such that a sufficient decrease of f is assured: Strong Wolfe conditions

$$\begin{aligned} f(\mathbf{x}_k + \alpha_k \mathbf{q}_k) &\leq f(\mathbf{x}_k) + c_1 \alpha_k \nabla_f(\mathbf{x}_k)^\top \mathbf{q}_k \\ |\nabla_f(\mathbf{x}_k + \alpha_k \mathbf{q}_k)^\top \mathbf{q}_k| &\leq c_2 |\nabla_f(\mathbf{x}_k)^\top \mathbf{q}_k|, \end{aligned}$$

with $0 < c_1 < c_2 < 1$ guarantee that the line search-type algorithms discussed hereafter converge.

Ch3 of [NOS] for details

Backtracking line search algorithm

1. Choose α_0, $\tau \in (0,1)$, and $c_1 \in (0,1)$;
2. $j = 0$;
3. Iterate until $f(\mathbf{x}_k + \alpha_j \mathbf{q}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_j \nabla_f(\mathbf{x}_k)^\top \mathbf{q}_k$,
    1. $\alpha_j = \tau \alpha_{j-1}$;
    2. $j = j + 1$;

Backtracking starts with a large step size α_0 and reduce α_0 by a factor of τ.
For sufficiently large α_0, the corresponding gradient descent algorithms converge.

Which direction ($\mathbf{q}_k$) to move?

Steepest descent

moves along the direction of steepest descent of f (opposite to the direction of the gradient)

$$\begin{aligned} \mathbf{q}_k &= -\nabla_f(\mathbf{x}_k) \\ \Leftrightarrow \mathbf{x}_{k+1} &= \mathbf{x}_k - \alpha_k \nabla_f(\mathbf{x}_k). \end{aligned}$$

- Intuitive.
- Easy to implement.
- Memory efficient.

Commonly used in training neural networks with fixed or decaying step-size α_k.

Convergence behavior of steepest descent

Suppose that f is a quadratic polynomial:

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x},$$

with **A** positive definite.

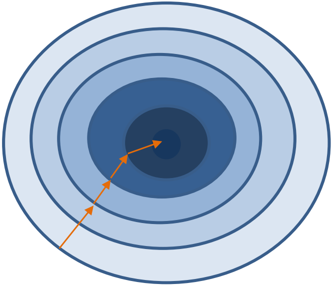
When exact line search step is used

$$\begin{aligned} f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*) &\leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 (f(\mathbf{x}_k) - f(\mathbf{x}^*)) \\ \|\mathbf{x}_{k+1} - \mathbf{x}^*\|_{\mathbf{A}}^2 &\leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 \|\mathbf{x}_k - \mathbf{x}^*\|_{\mathbf{A}}^2 \end{aligned}$$

where $\{\lambda_i\}$ is the set of eigenvalues of **A** and $\|\mathbf{x}\|_{\mathbf{A}} = \mathbf{x}^\top \mathbf{A}\mathbf{x}$.
$\Rightarrow$ Linear rate of convergence.

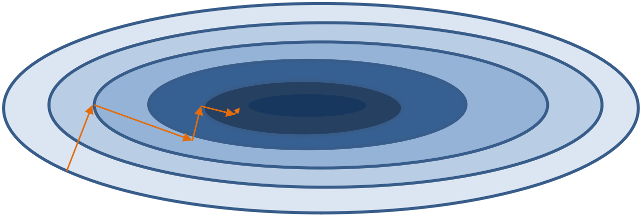
Steepest descent on quadratic functions

Good for isotropically-shaped functions.



Steepest descent on quadratic functions

Bad for anisotropically-shaped functions: typically zig-zagging.



- Elongation depends on ratios of eigenvalues of $\mathbf{A}$
  $\Rightarrow$ severely elongated if $\mathbf{A}$ is badly conditioned.
- $\mathbf{A}$ is the Hessian of f.
- For a general function f, the speed of convergence depends on the condition numbers of local Hessians $\{H_f(\mathbf{x}_k)\}$.

(pure) Newton method

At each iteration k, (pure) Newton method approximates f:

$$f(\mathbf{x}) \approx m_k(\mathbf{x}) = f(\mathbf{x}_k) + (\mathbf{x} - \mathbf{x}_k)^\top \nabla_f(\mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\top H_f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)$$

and solve the corresponding analytical optimization: For a positive definite H_f, m_k is a convex function:
Setting the derivative of m_k to zero

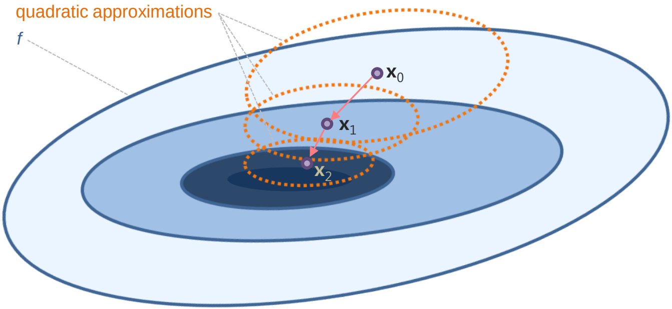
$$\nabla_f(\mathbf{x}_k) + H_f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = 0,$$

we obtain

$$\mathbf{x}_{k+1} \Leftarrow \mathbf{x}^* = \mathbf{x}_k - H_f(\mathbf{x}_k)^{-1} \nabla_f(\mathbf{x}_k).$$

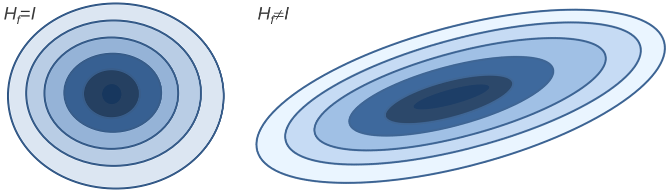
The (pure) Newton method does not use α_k:

(pure) Newton method

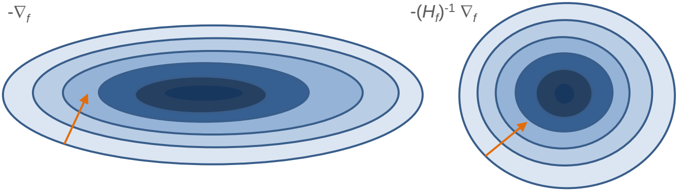


(pure) Newton method

Hessian (H_f) is responsible for the elongation:



Multiplying H_f^{-1} to ∇_f has the effect of squeezing the function along the elongated direction:



Multiplying H_f^{-1}?

Eigen-decomposition of the Hessian matrix H_f and its inverse H_f^{-1}:

$$H_f = (\mathbf{e}^1, \mathbf{e}^2) \begin{pmatrix} \lambda^1 & 0 \\ 0 & \lambda^2 \end{pmatrix} (\mathbf{e}^1, \mathbf{e}^2)^\top = \lambda^1 \mathbf{e}^1 {\mathbf{e}^1}^\top + \lambda^2 \mathbf{e}^2 {\mathbf{e}^2}^\top$$

$$H_f^{-1} = (\mathbf{e}^1, \mathbf{e}^2) \begin{pmatrix} \frac{1}{\lambda^1} & 0 \\ 0 & \frac{1}{\lambda^2} \end{pmatrix} (\mathbf{e}^1, \mathbf{e}^2)^\top = \frac{1}{\lambda^1} \mathbf{e}^1 {\mathbf{e}^1}^\top + \frac{1}{\lambda^2} \mathbf{e}^2 {\mathbf{e}^2}^\top.$$

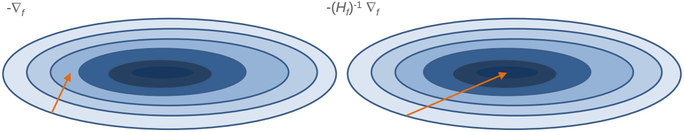
Multiplying the gradient vector ∇^f by H_f^{-1}:

$$\begin{aligned} H_f^{-1} \nabla^f &= \left[\frac{1}{\lambda^1} \mathbf{e}^1 {\mathbf{e}^1}^\top + \frac{1}{\lambda^2} \mathbf{e}^2 {\mathbf{e}^2}^\top \right] \nabla^f \\ &= \frac{1}{\lambda^1} \mathbf{e}^1 \left[{\mathbf{e}^1}^\top \nabla^f \right] + \frac{1}{\lambda^2} \mathbf{e}^2 \left[{\mathbf{e}^2}^\top \nabla^f \right]. \end{aligned}$$

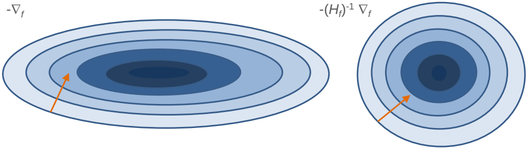
$\Rightarrow$ Projections to basis vectors $\{\mathbf{e}^i\}$ weighted by $\{1/\lambda^i\}$.

Multiplying H_f^{-1}?

Multiplying H_f^{-1} to ∇_f has the effect of stretching the gradient vector ∇_f along the elongated direction:



Multiplying H_f^{-1} to ∇_f has the effect of squeezing the function along the elongated direction:



(pure) Newton method

- Good convergence: quadratic rate.
- May not be easy to implement: requires calculating H_f explicitly.
- May require large memory: to store $H_f(\mathbf{x}_k)$ and solve $H_f(\mathbf{x}_k)^{-1}\nabla_f(\mathbf{x}_k)$.

In practice, $H_f(\mathbf{x}_k)^{-1}$ may not always be positive definite.
If $H_f(\mathbf{x}_k)^{-1}$ is not positive definite, $H_f(\mathbf{x}_k)^{-1}\nabla_f(\mathbf{x}_k)$ is an ascending direction.
To ensure gradient descent, one could instead use

$$\mathbf{B}_k = H_f(\mathbf{x}_k)^{-1} + \mathbf{E}$$

where $\mathbf{E}$ is chosen such that $\mathbf{B}_k$ is positive definite.
E.g., $\mathbf{E} = \lambda I$, $\lambda \geq 0$.

Summary

- Almost all machine learning tasks are formulated as optimization problems.
- Steepest descent: one of the simplest optimization algorithms.
  - requires evaluating gradient.
  - good for isotropic functions.
  - slow for anisotropic functions.
- (Pure) Newton method: one of the fastest iterative optimization algorithms.
  - requires evaluating Hessian.
  - even good for anisotropic functions.

References

Noc J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer (second edition).

Ber D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific (second edition).

Teu Teukolsky, Vetterling, and Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press (any edition). `http://www.nr.com/`

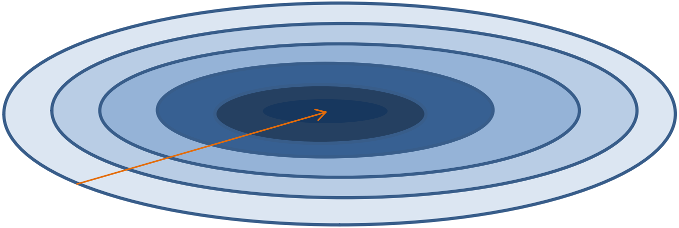
Pet K. B. Petersen and M. S. Pedersen, *The Matrix Cookbook*.

Newton method

Newton method uses the direction obtained from the pure Newton method in the line search optimization:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H_f(\mathbf{x}_k)^{-1} \nabla_f(\mathbf{x}_k)$$

For a quadratic function, optimization is done in a single step:



independently of how f is elongated.