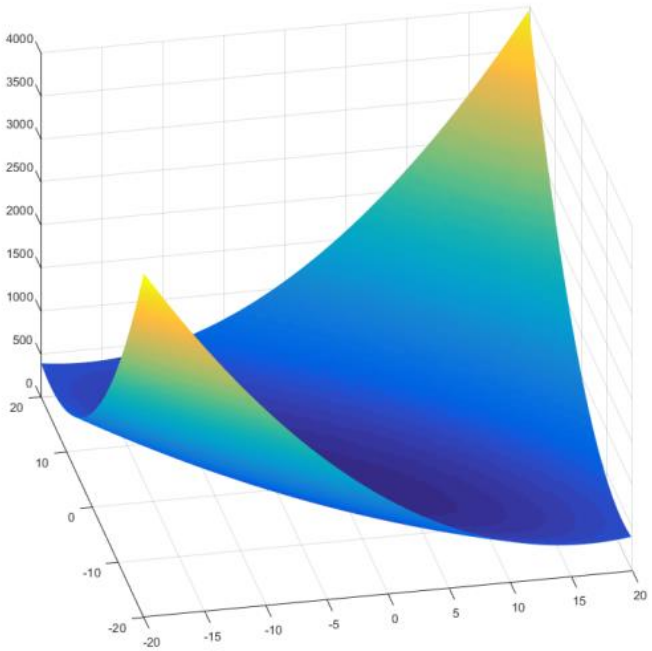
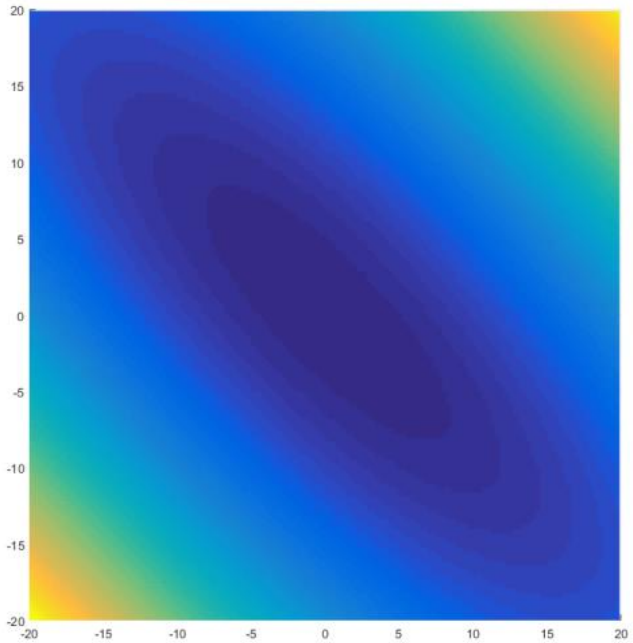
Lecture 13
CM50264: Machine Learning 1
Optimization Basics 2

Kwang In Kim

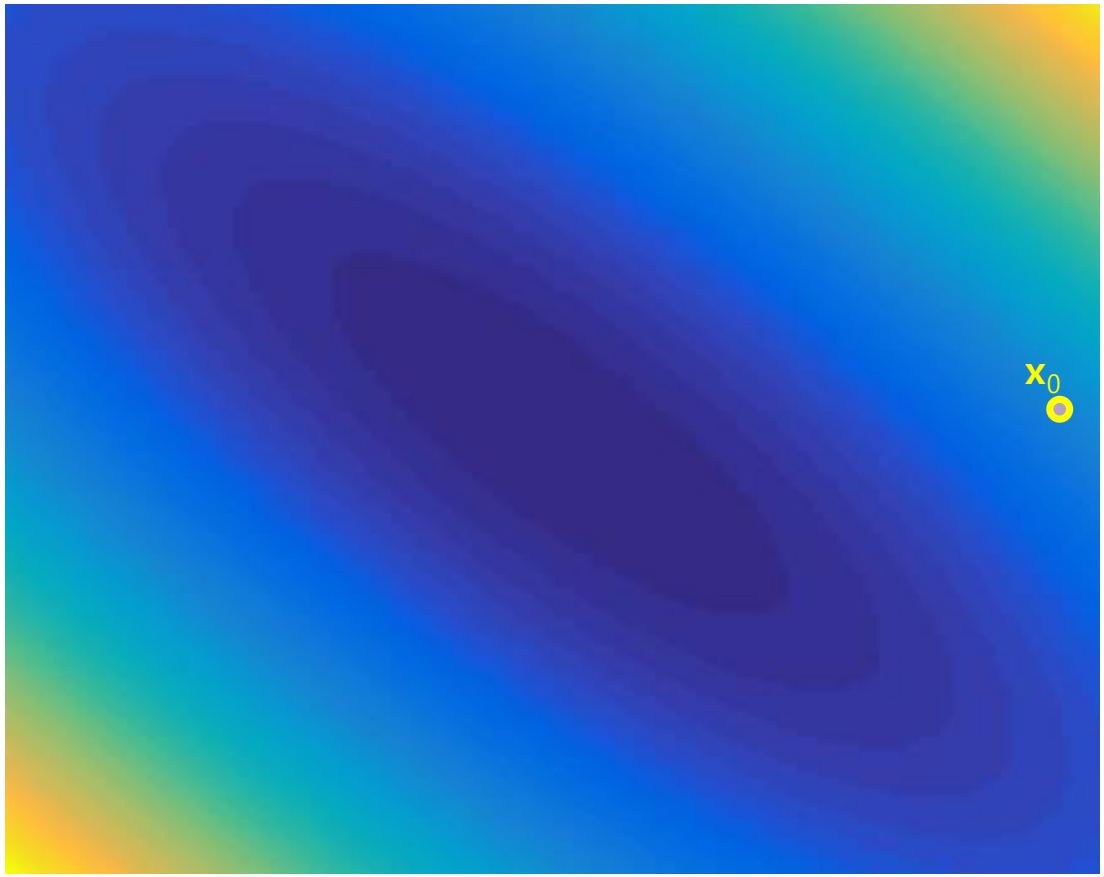
Previously in optimization basics ...
A simple 2D optimization problem





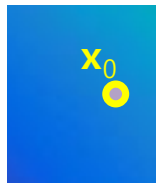
Our objective function f is two-dimensional: $f : \mathbb{R}^2 \to \mathbb{R}$.

A simple 2D optimization problem



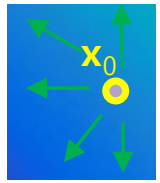
We start the optimization with an initial (zero-th) solution $\mathbf{x}_0$.

A simple 2D optimization problem



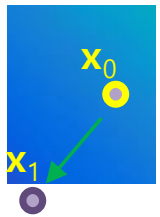
Again, we can observe f only in a small neighborhood of $\mathbf{x}_0$.

A simple 2D optimization problem



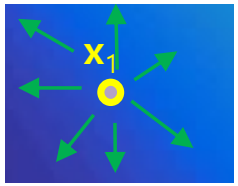
By inspecting f around $\mathbf{x}_0$ ...

A simple 2D optimization problem



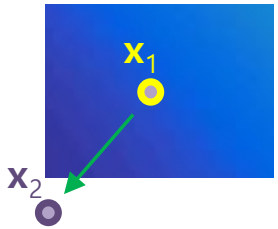
We decide a direction to move (to decrease the f value).

A simple 2D optimization problem



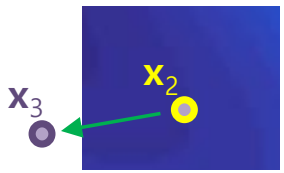
Now we are at the first solution $\mathbf{x}_1$, and observing f around ...

A simple 2D optimization problem



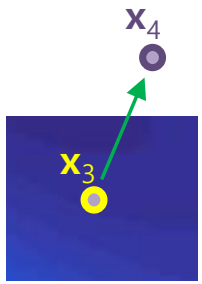
and decide a new direction ...

A simple 2D optimization problem



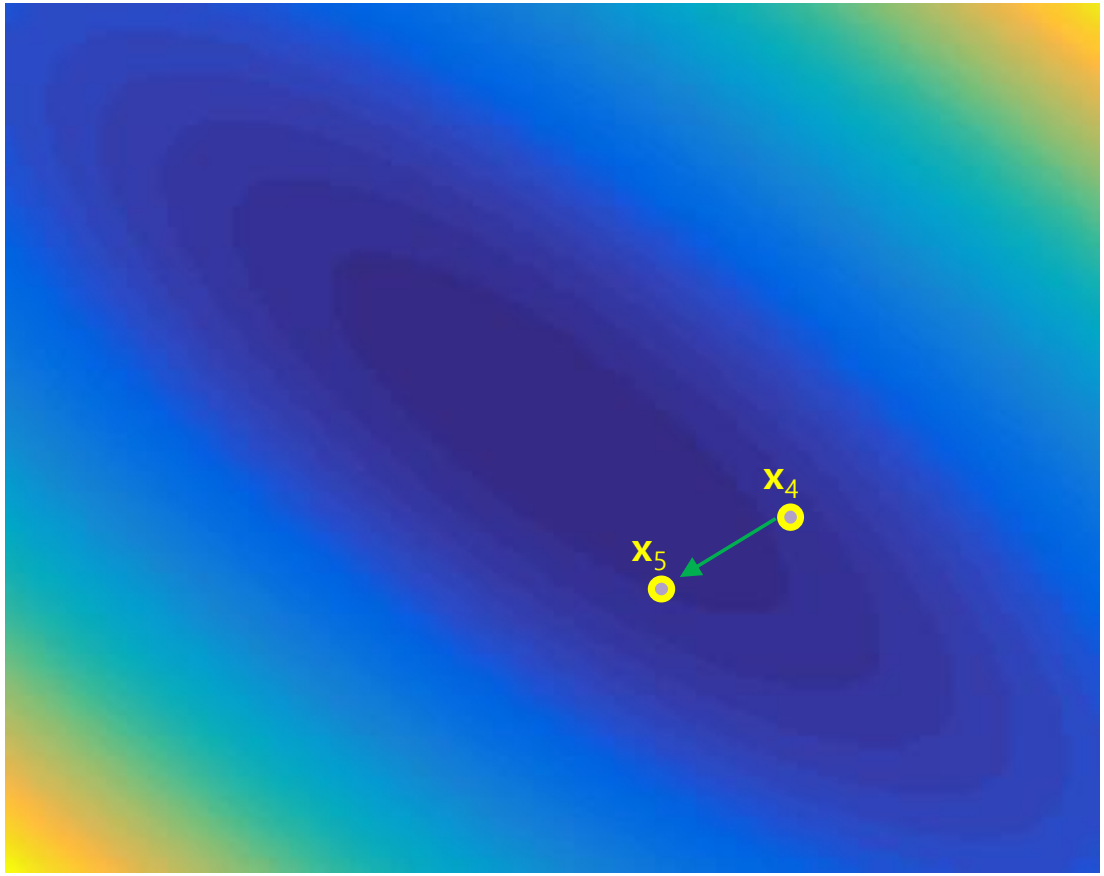
From the second solution to the third ...

A simple 2D optimization problem



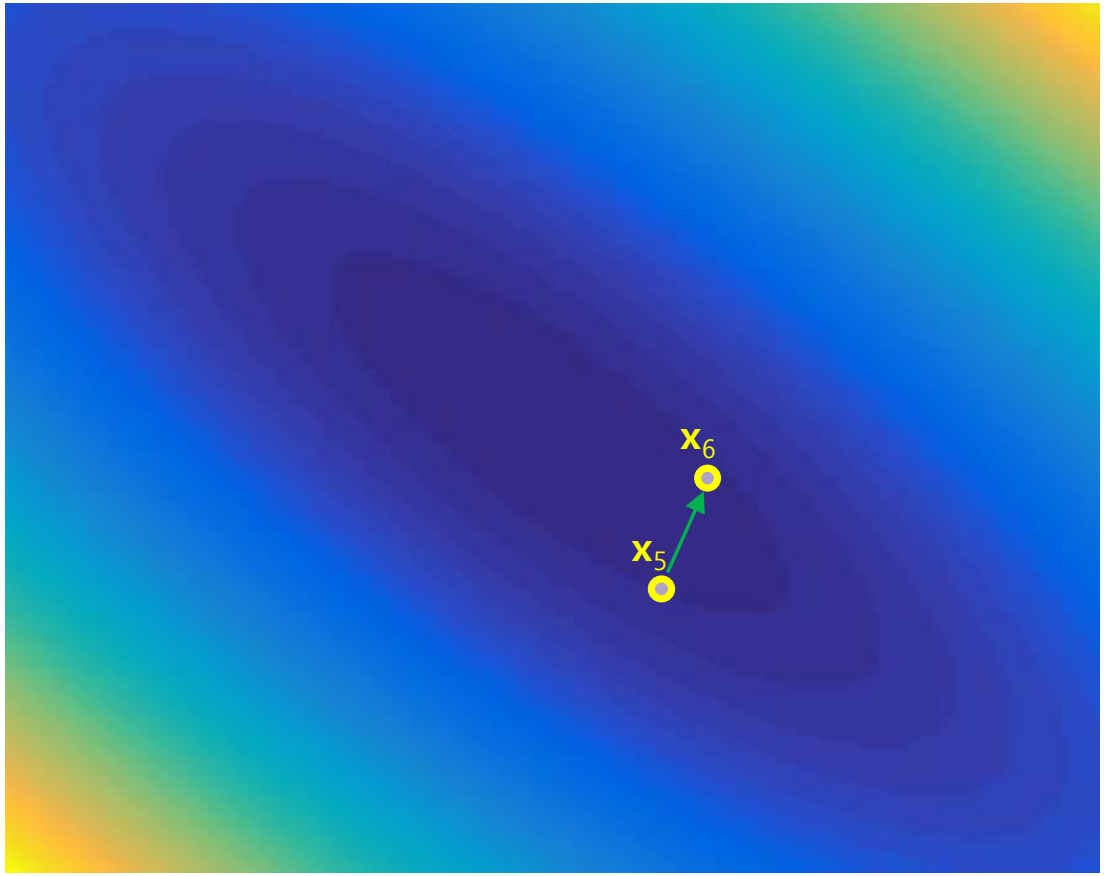
from the third solution to the fourth ...

A simple 2D optimization problem



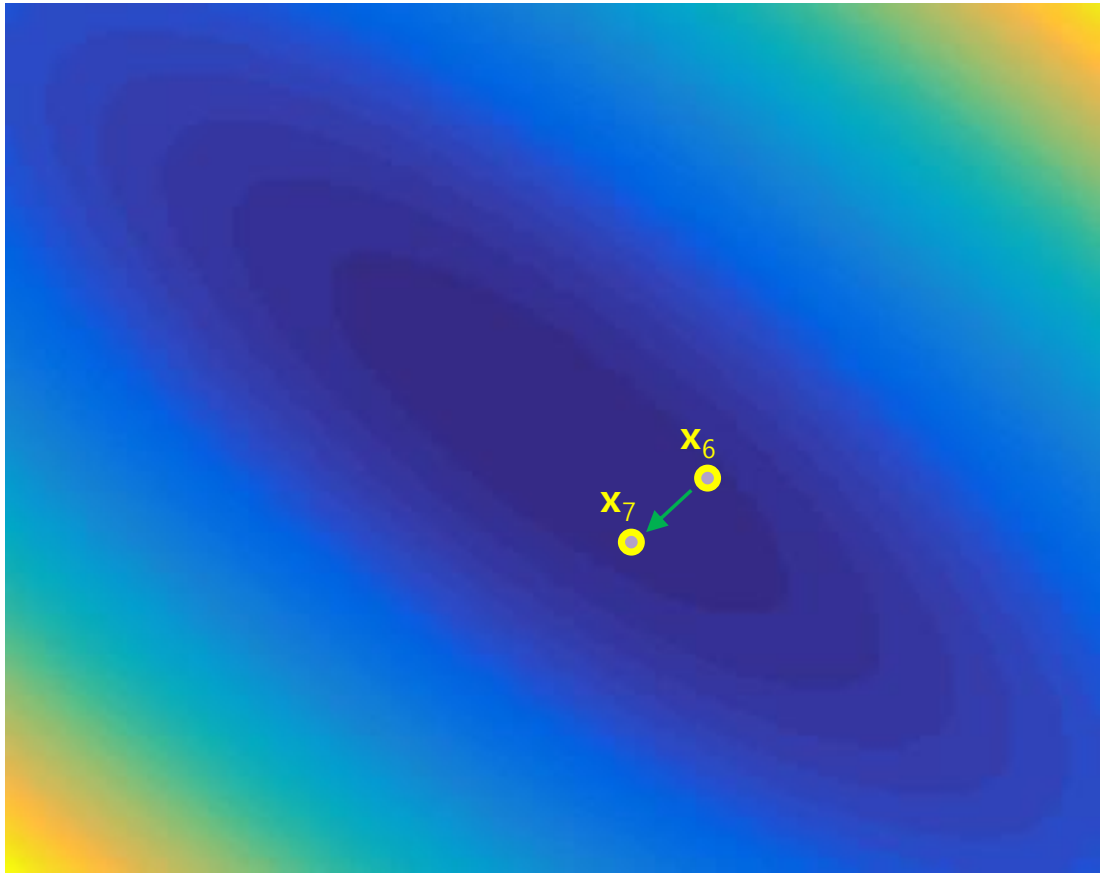
from $\mathbf{x}_4$ to $\mathbf{x}_5$ ...

A simple 2D optimization problem



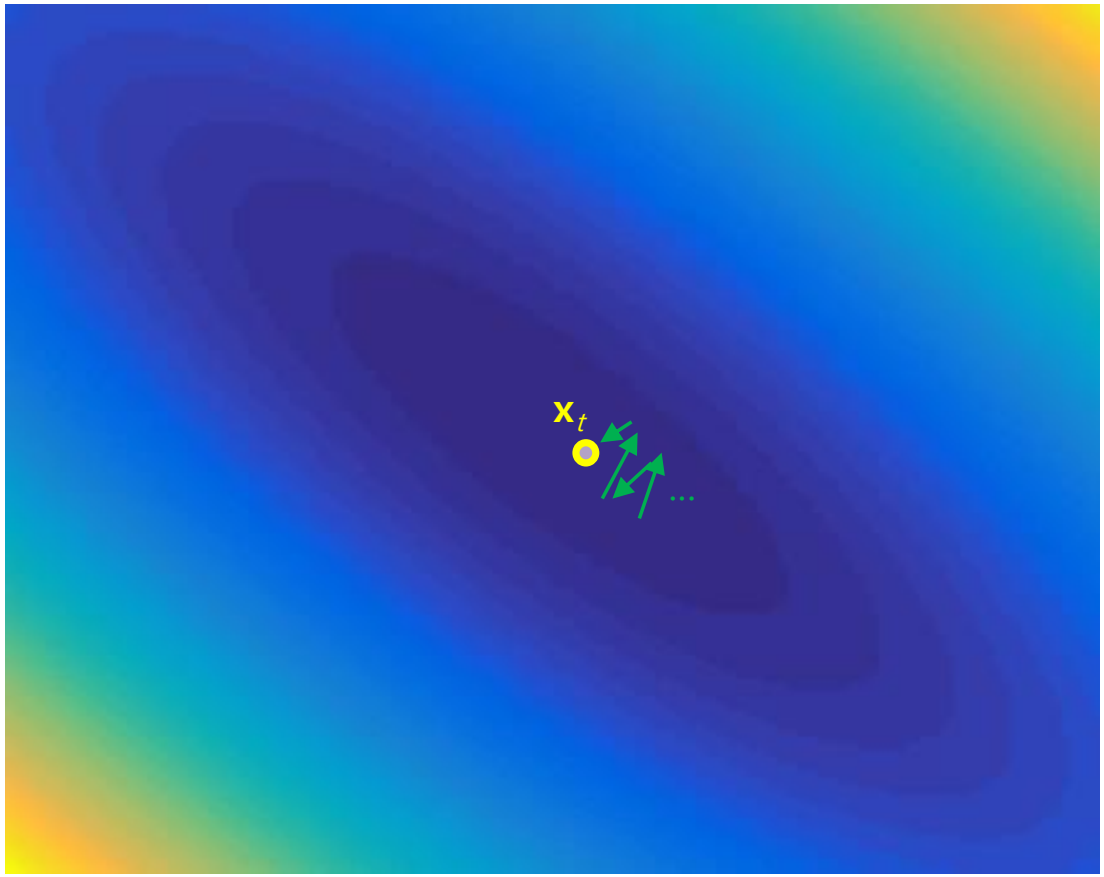
from $\mathbf{x}_5$ to $\mathbf{x}_6$ ...

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 $\mathbf{x}_t = \mathbf{x}_*$.

Iterative descent algorithm

Summarizing our optimization approach...

(I) $t = 0$; Make an initial guess $\mathbf{x}_t$;

(II) Iterate until the termination condition is met.

(i) Find a direction $\mathbf{p}_t$ to move;

(ii) Decide how much (α_t) to move along $\mathbf{p}_t$ direction;

(iii) $\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t \mathbf{p}_t$;

(iv) $t = t + 1$;

How do we decide

- direction to move $\mathbf{p}_t$,
- step size α_t,
- when to stop (termination condition)?

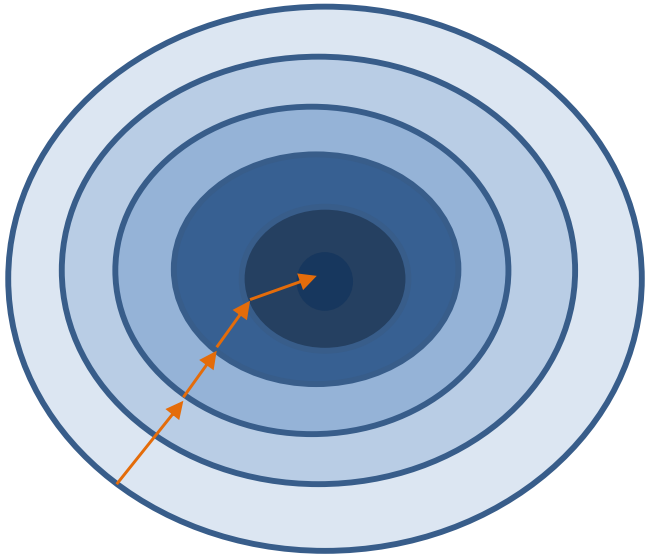
Steepest descent

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t \mathbf{p}_t;$$

- $\mathbf{p}_t = -\nabla f(\mathbf{x}_t)$.
- How do we decide the step size $\alpha_t > 0$?
  $\rightarrow$ A simple solution (among others): fix it to a constant value.
- 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 $T > 0$.

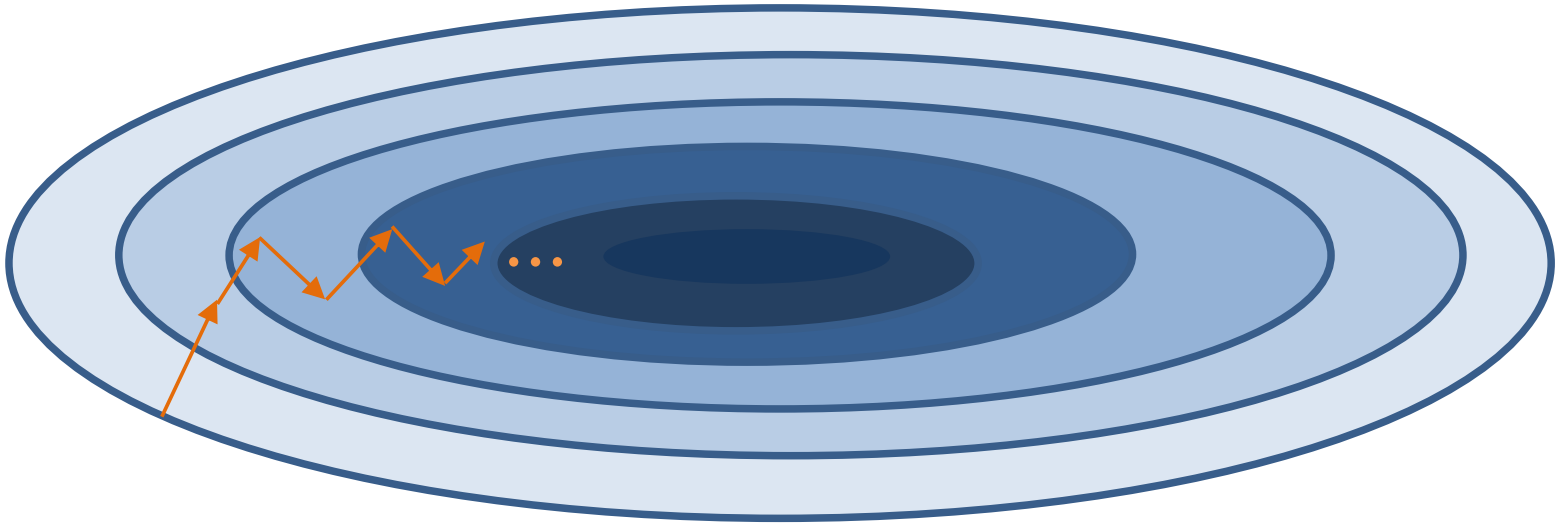
Steepest descent algorithm

Good for isotropic functions.



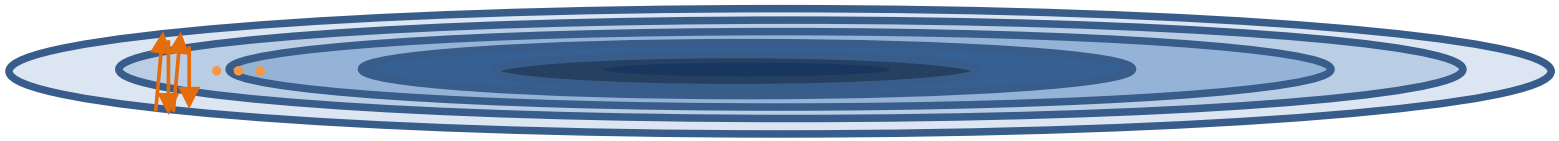
Steepest descent algorithm

Not good for anisotropic functions.



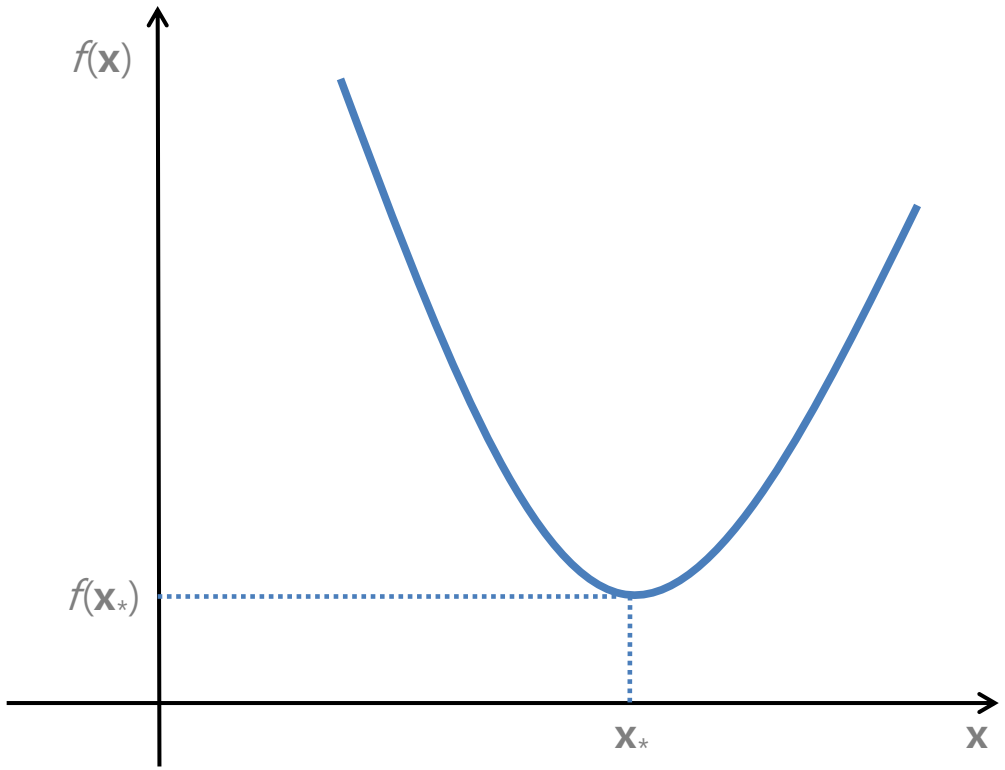
Steepest descent algorithm

Not good for anisotropic functions.



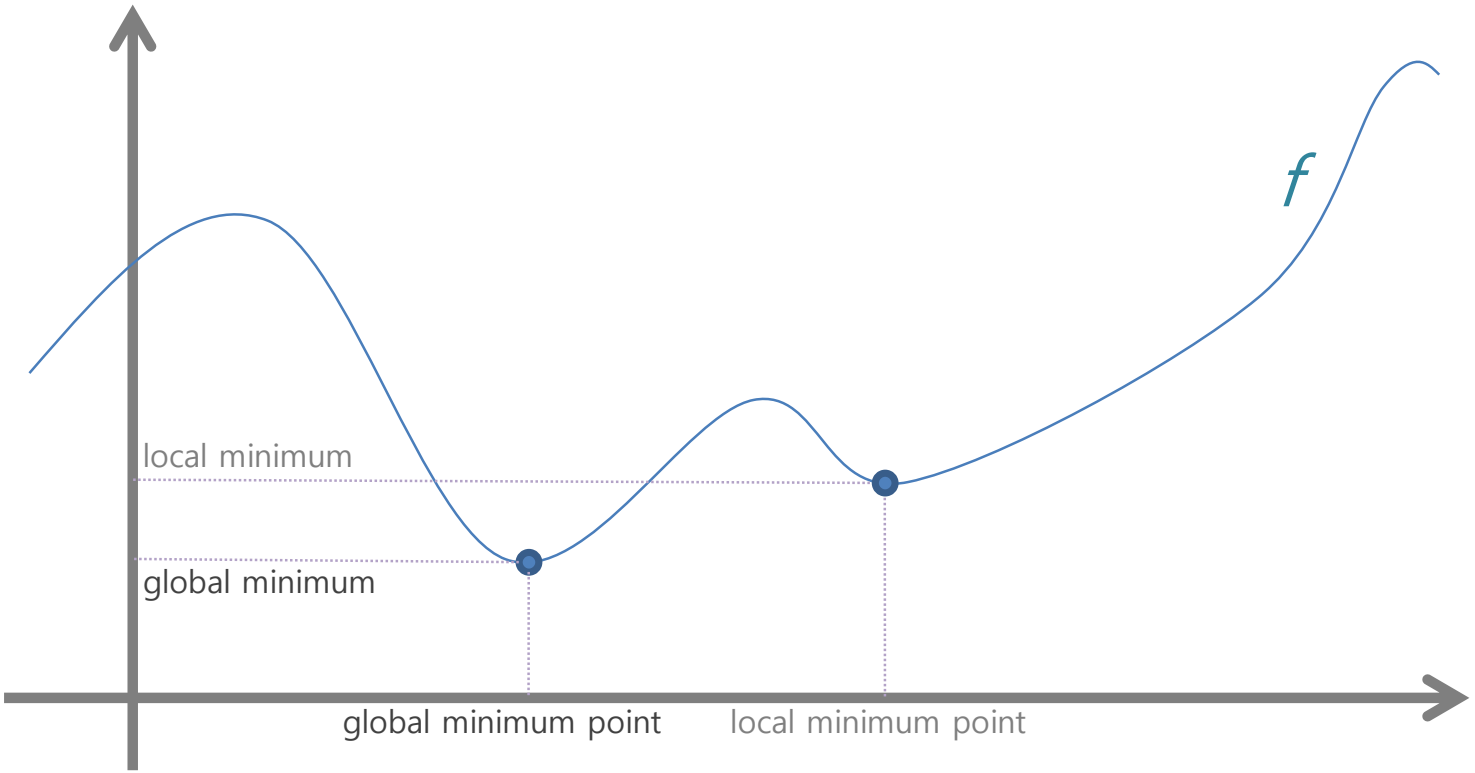
Steepest descent algorithm

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



Steepest descent algorithm

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



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

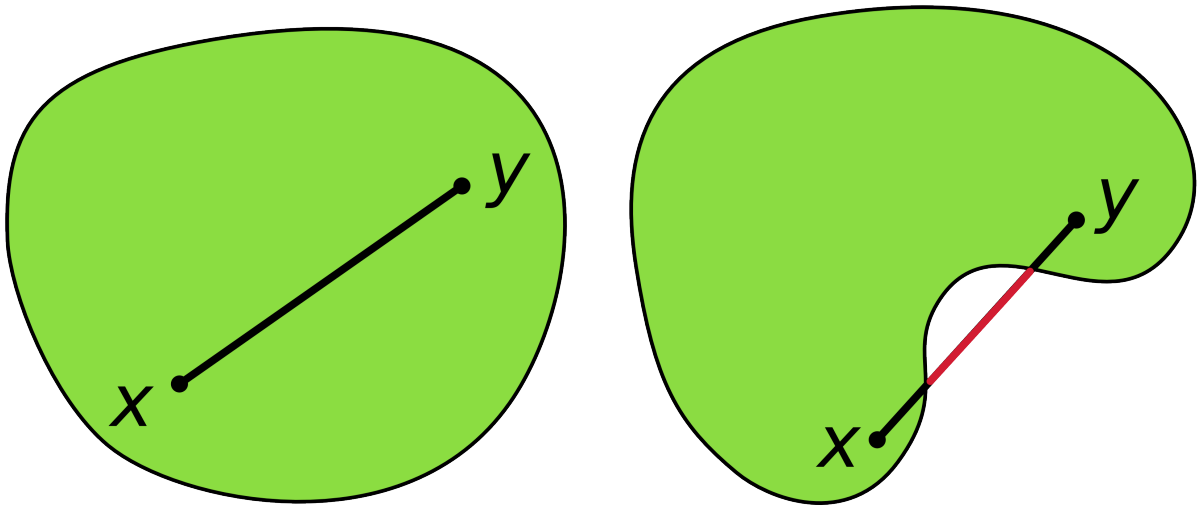
- Global minimum (point) $\mathbf{x}_*$: $f(\mathbf{x}_*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$.
- Local minimum (point): a point which becomes a global minimum point when we restrict the domain to a (arbitrary small) neighborhood of itself.

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

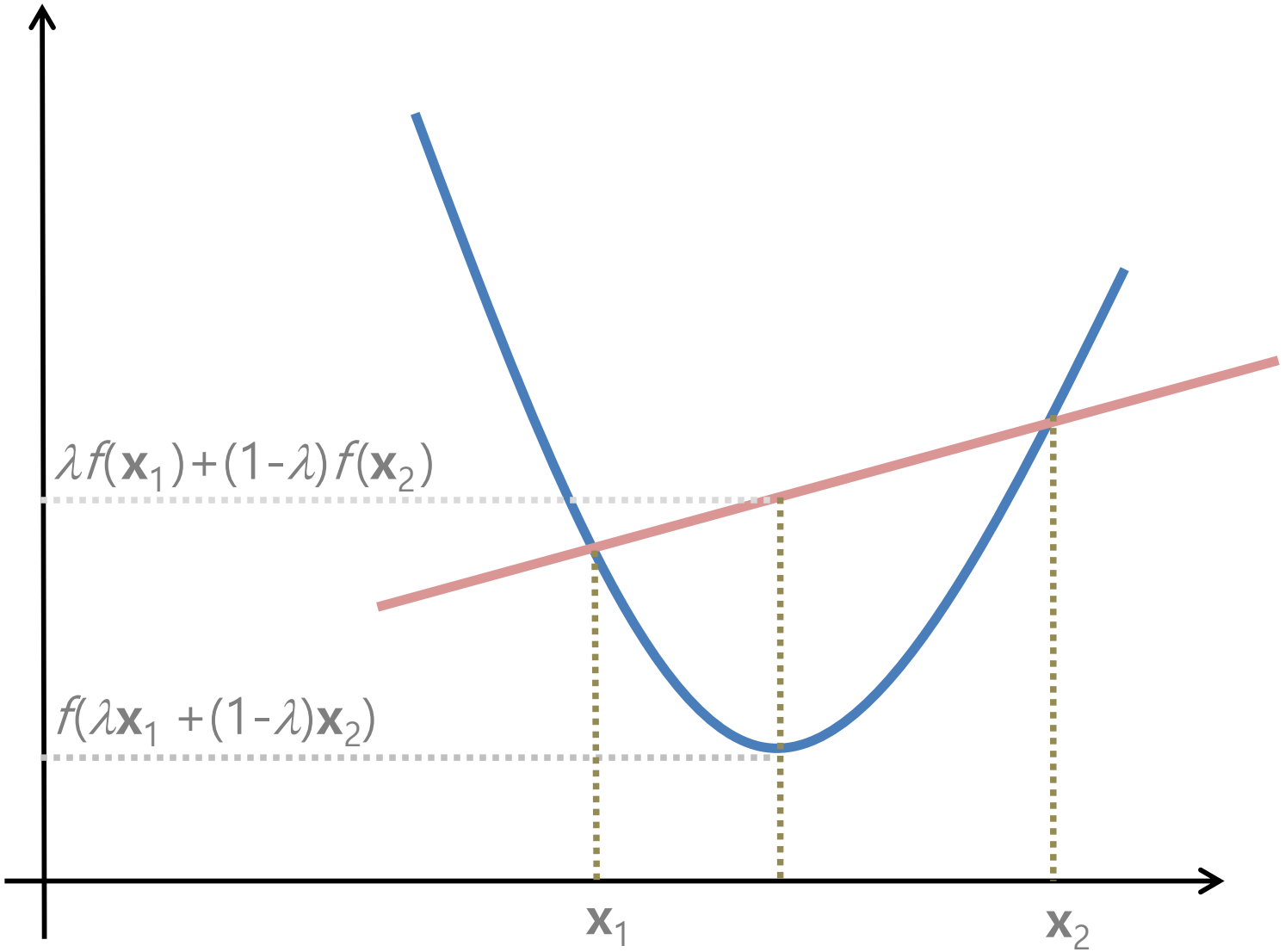
$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2),$$

- strictly convex if

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) < \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

$\forall \mathbf{x}_1, \mathbf{x}_2 \in C$ and $\lambda \in [0, 1]$.

Convex function



(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:

$$\begin{aligned}
\mathbf{x}^\top \mathbf{A}\mathbf{x} &= \mathbf{x}^\top (\mathbf{E}\Lambda\mathbf{E}^{-1})\mathbf{x} \\
&= \mathbf{x}^\top (\mathbf{E}\Lambda\mathbf{E}^\top)\mathbf{x} \\
&= (\mathbf{E}^\top \mathbf{x})^\top \Lambda (\mathbf{E}^\top \mathbf{x}) \\
&= \mathbf{y}^\top \Lambda \mathbf{y} \text{ with } (\mathbf{y} = \mathbf{E}^\top \mathbf{x}). \\
&\Rightarrow \mathbf{x}^\top \mathbf{A}\mathbf{x} > 0 \text{ if elements of } \Lambda \text{ are positive.}
\end{aligned}$$

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: a second-order polynomial

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

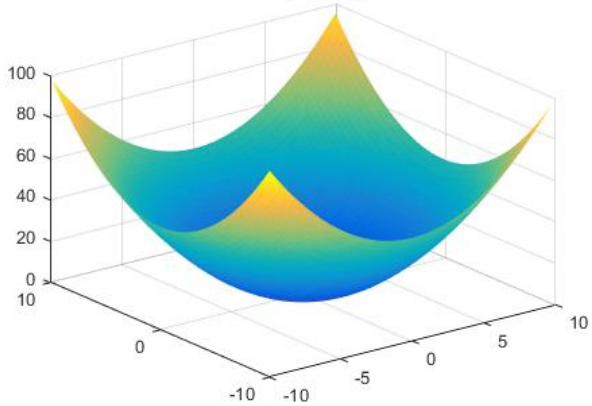
is convex if $\mathbf{A}$ is (symmetric) positive definite.

The Hessian matrix $\frac{d^2 p}{d\mathbf{x}^2}[\mathbf{x}]$ of p is A for all $\mathbf{x}$.

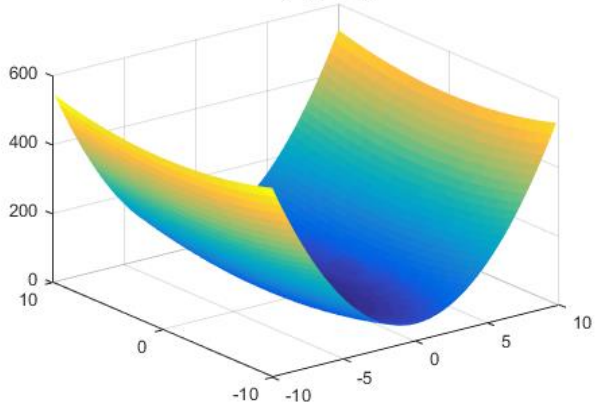
2D examples

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

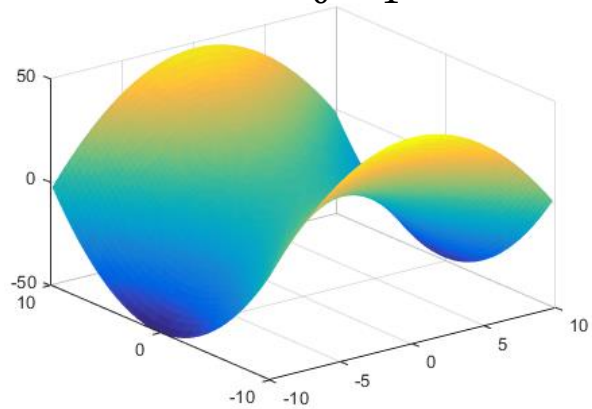
$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$



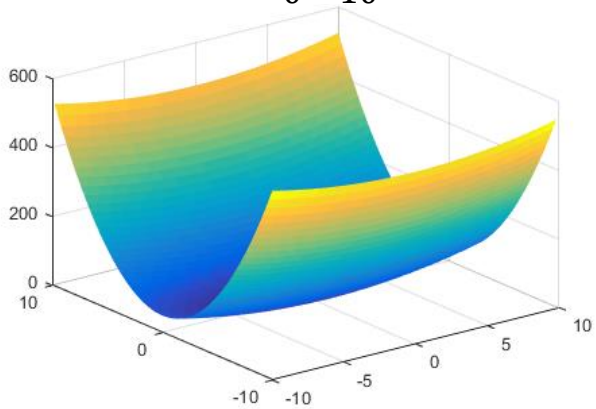
$\mathbf{A} = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}$



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



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



2D example 1

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

$$\begin{aligned}
\mathbf{A} &= \mathbf{E}\Lambda\mathbf{E}^{-1} = \mathbf{E}\Lambda\mathbf{E}^{\top} \\
&= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
&= (\mathbf{e}_1, \mathbf{e}_2) \begin{pmatrix} \lambda^1 & 0 \\ 0 & \lambda^2 \end{pmatrix} (\mathbf{e}_1, \mathbf{e}_2)^{\top} \\
&= \lambda^1 \mathbf{e}_1 \mathbf{e}_1{}^{\top} + \lambda^2 \mathbf{e}_2 \mathbf{e}_2{}^{\top},
\end{aligned}$$

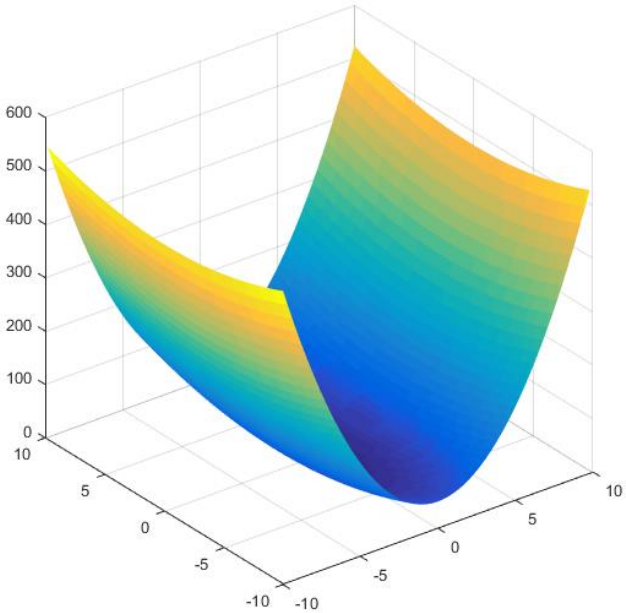
where

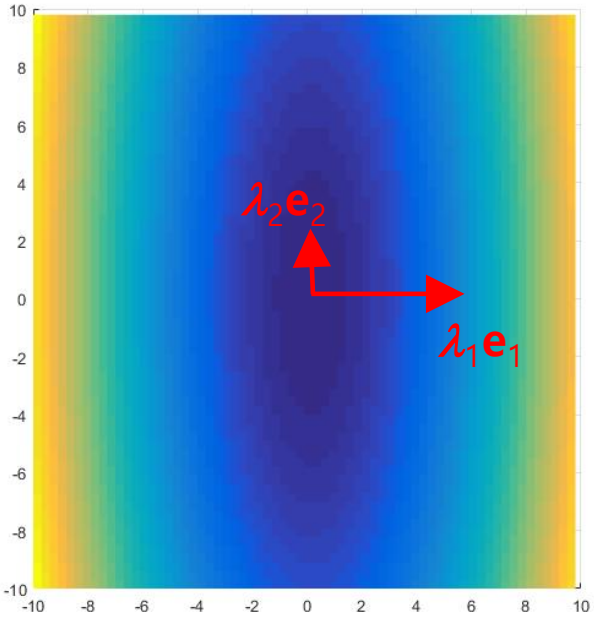
$$\mathbf{e}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \mathbf{e}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \lambda^1 = 10, \lambda^2 = 1.$$

Note: Eigenvectors of $\mathbf{A}$ form a basis.

2D example 1

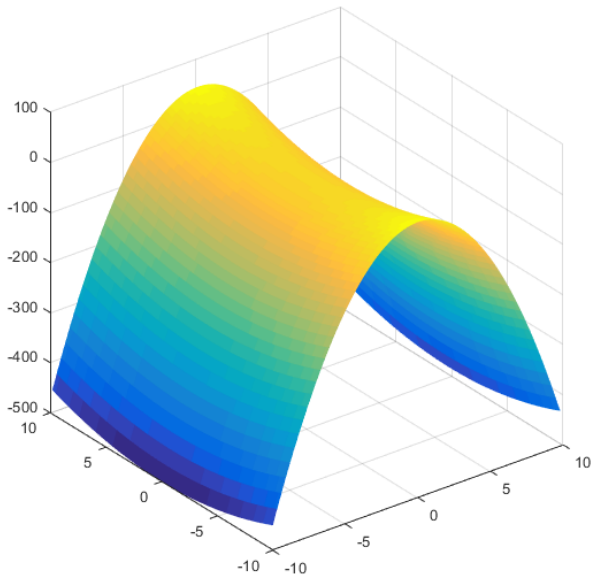
$$p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}, \mathbf{A} = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} = \lambda^1 \mathbf{e}_1 \mathbf{e}_1{}^\top + \lambda^2 \mathbf{e}_2 \mathbf{e}_2{}^\top.$$

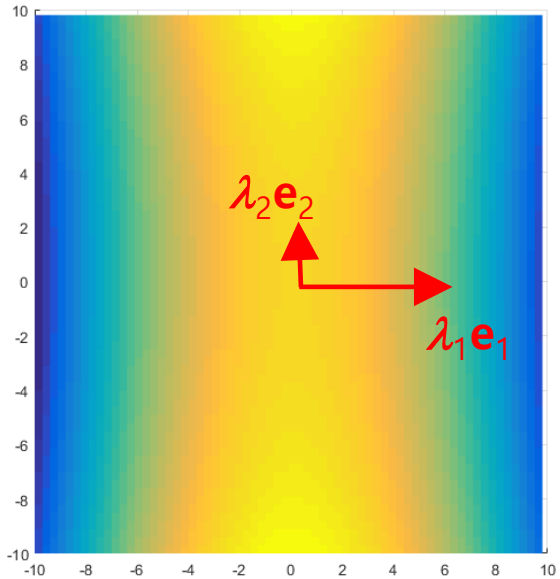




2D example 2

$$p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}, \mathbf{A} = \begin{pmatrix} -10 & 0 \\ 0 & 1 \end{pmatrix} = \lambda^1 \mathbf{e}_1 \mathbf{e}_1{}^\top + \lambda^2 \mathbf{e}_2 \mathbf{e}_2{}^\top.$$





2D example 3

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

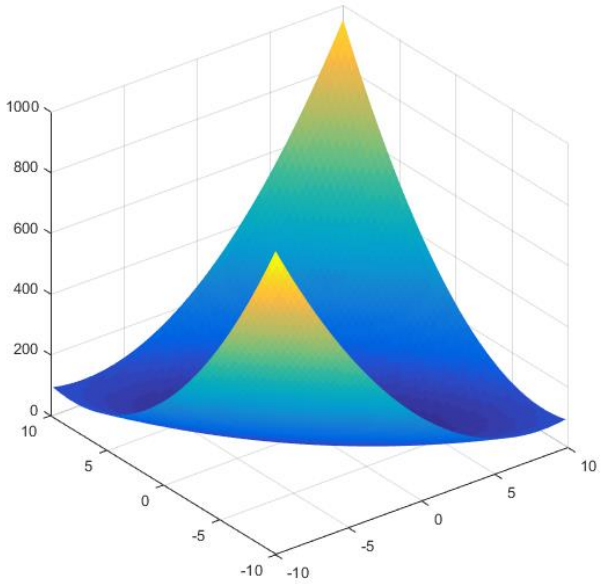
$$\begin{aligned}
\mathbf{A} &= \mathbf{E}\Lambda\mathbf{E}^{-1} = \mathbf{E}\Lambda\mathbf{E}^{\top} \\
&= \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \\
&= (\mathbf{e}_1, \mathbf{e}_2) \begin{pmatrix} \lambda^1 & 0 \\ 0 & \lambda^2 \end{pmatrix} (\mathbf{e}_1, \mathbf{e}_2)^{\top} \\
&= \lambda^1 \mathbf{e}_1 {\mathbf{e}_1}^{\top} + \lambda^2 \mathbf{e}_2 {\mathbf{e}_2}^{\top},
\end{aligned}$$

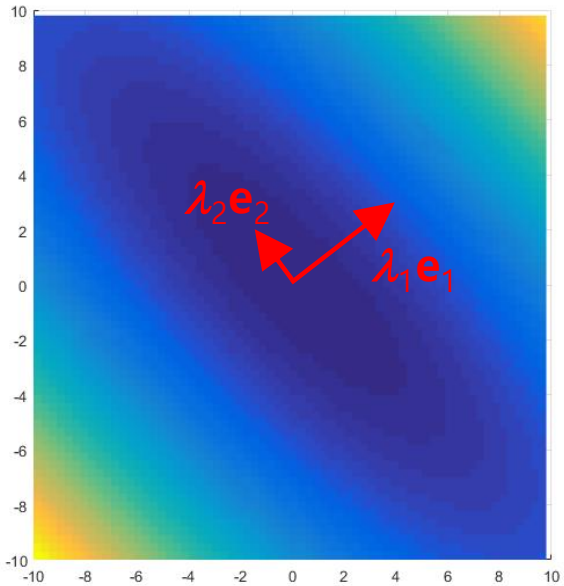
where

$$\mathbf{e}_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}, \mathbf{e}_2 = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}, \lambda^1 = 10, \lambda^2 = 1.$$

2D example 3

$$p(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}, \mathbf{A} = \begin{pmatrix} 5.5 & 4.5 \\ 4.5 & 5.5 \end{pmatrix} = \lambda^1 \mathbf{e}_1 \mathbf{e}_1{}^\top + \lambda^2 \mathbf{e}_2 \mathbf{e}_2{}^\top.$$





Taylor series

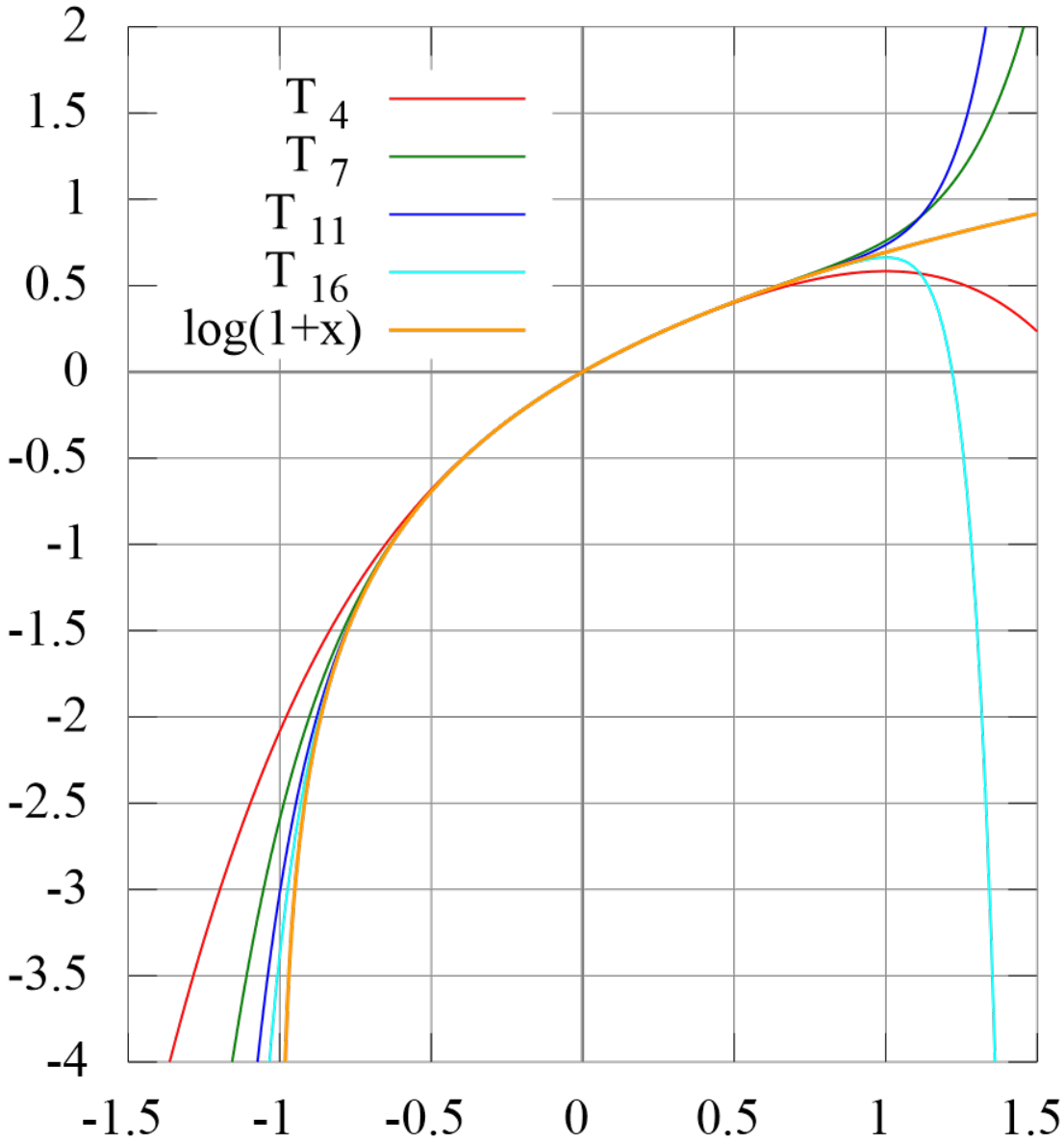
The Taylor series of an infinitely differentiable function $f : \mathbb{R} \mapsto \mathbb{R}$ expanded at $\mathbf{a}$ is

$$f(\mathbf{a}) + \frac{1}{1!} f'(\mathbf{a})(\mathbf{x} - \mathbf{a}) + \frac{1}{2!} f''(\mathbf{a})(\mathbf{x} - \mathbf{a})^2 + \frac{1}{3!} f^{(3)}(\mathbf{a})(\mathbf{x} - \mathbf{a})^3 + \dots$$

- The series converges to $f(\mathbf{x})$.
- Multivariate case ($f : \mathbb{R}^n \mapsto \mathbb{R}$):

$$f(\mathbf{x}) = f(\mathbf{a}) + (\mathbf{x} - \mathbf{a})^\top \nabla f(\mathbf{a}) + \frac{1}{2} (\mathbf{x} - \mathbf{a})^\top H[f](\mathbf{a})(\mathbf{x} - \mathbf{a}) + \dots$$

Taylor series approximation example



[Wikipedia]

Taylor series approximation

If $f : \mathbb{R}^n \mapsto \mathbb{R}$ is twice continuously differentiable and $\mathbf{p} \in \mathbb{R}^n$, then, $\exists t \in (0, 1)$

$$f(\mathbf{x} + \mathbf{p}) \approx \bar{f}(\mathbf{x} + \mathbf{p}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top H[f](\mathbf{x} + t\mathbf{p})\mathbf{p}.$$

- Locally, a smooth function f can be well-approximated based on the second-order Taylor polynomial $\bar{f}(\mathbf{x})$.
- Many optimization techniques are derived from this approximation.

Gradient descent

How do we decide

- direction to move $\mathbf{p}_t$,
- step size α_t,
- when to stop (termination condition)?

Step size ?
Line search step (choosing α_t)

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t \mathbf{p}_t$$

Various methods for choosing α_t:

- Fix at a small constant: often, the case for training neural networks.
- Decay as t 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

$$\begin{aligned} f(\mathbf{x}_t + \alpha_t \mathbf{p}_t) &\leq f(\mathbf{x}_t) + c_1 \alpha_t \nabla f(\mathbf{x}_t)^\top \mathbf{p}_t \\ |\nabla f(\mathbf{x}_t + \alpha_t \mathbf{p}_t)^\top \mathbf{p}_t| &\leq c_2 |\nabla f(\mathbf{x}_t)^\top \mathbf{p}_t| \end{aligned}$$

with $0 < c_1 < c_2 < 1$ guarantees that the steepest descent algorithm and Newton method (discussed later) 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 $f(\mathbf{x}_t + \alpha_j \mathbf{p}_t) \leq f(\mathbf{x}_t) + c_1 \alpha_j \nabla f(\mathbf{x}_t)^\top \mathbf{p}_t$,
   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.

Gradient descent

How do we decide

- direction to move $\mathbf{p}_t$,
- step size α_t,
- when to stop (termination condition)?

Steepest descent

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

$$\begin{aligned} \mathbf{p}_t &= -\nabla f(\mathbf{x}_t) \\ \Leftrightarrow \mathbf{x}_{t+1} &= \mathbf{x}_t - \alpha_t \nabla f(\mathbf{x}_t). \end{aligned}$$

- intuitive.
- easy to implement.
- memory efficient.

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

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 $\mathbf{A}$ positive definite.

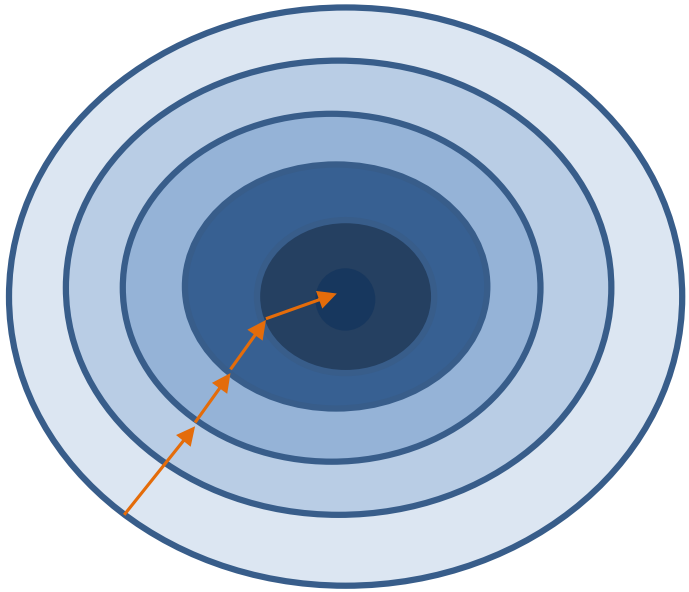
When exact line search step (1D optimization) is used

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

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

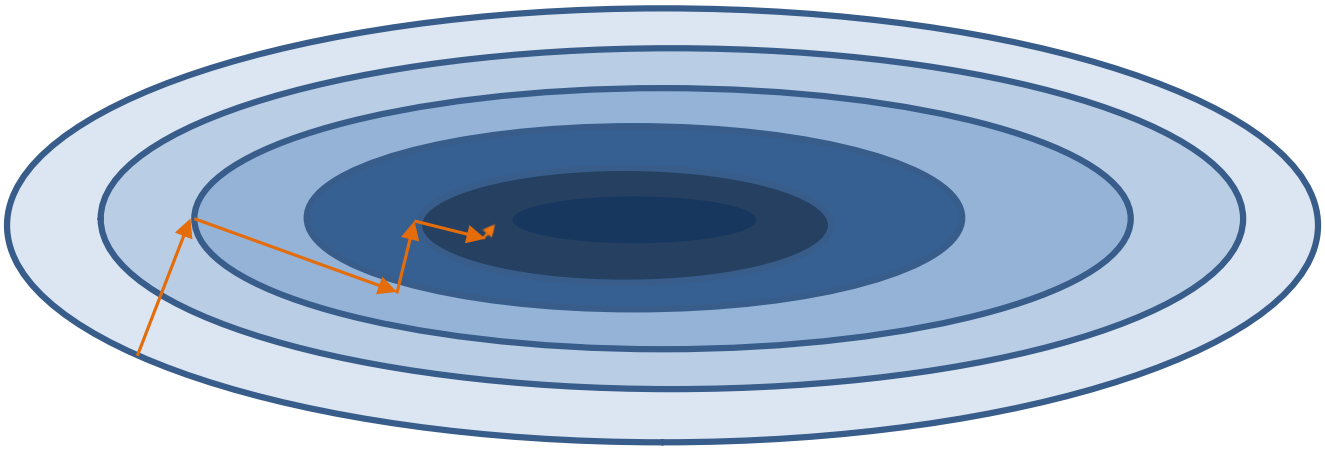
Steepest descent on quadratic functions

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 $\{H[f](\mathbf{x}_t)\}$.

(pure) Newton method

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

$$f(\mathbf{x}) \approx m_t(\mathbf{x}) = f(\mathbf{x}_t) + (\mathbf{x} - \mathbf{x}_t)^\top \nabla f(\mathbf{x}_t) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_t)^\top H[f](\mathbf{x}_t)(\mathbf{x} - \mathbf{x}_t)$$

and solve the corresponding analytical optimization: For a positive definite $H[f]$, m_t is a convex function:
Setting the derivative of m_t to zero

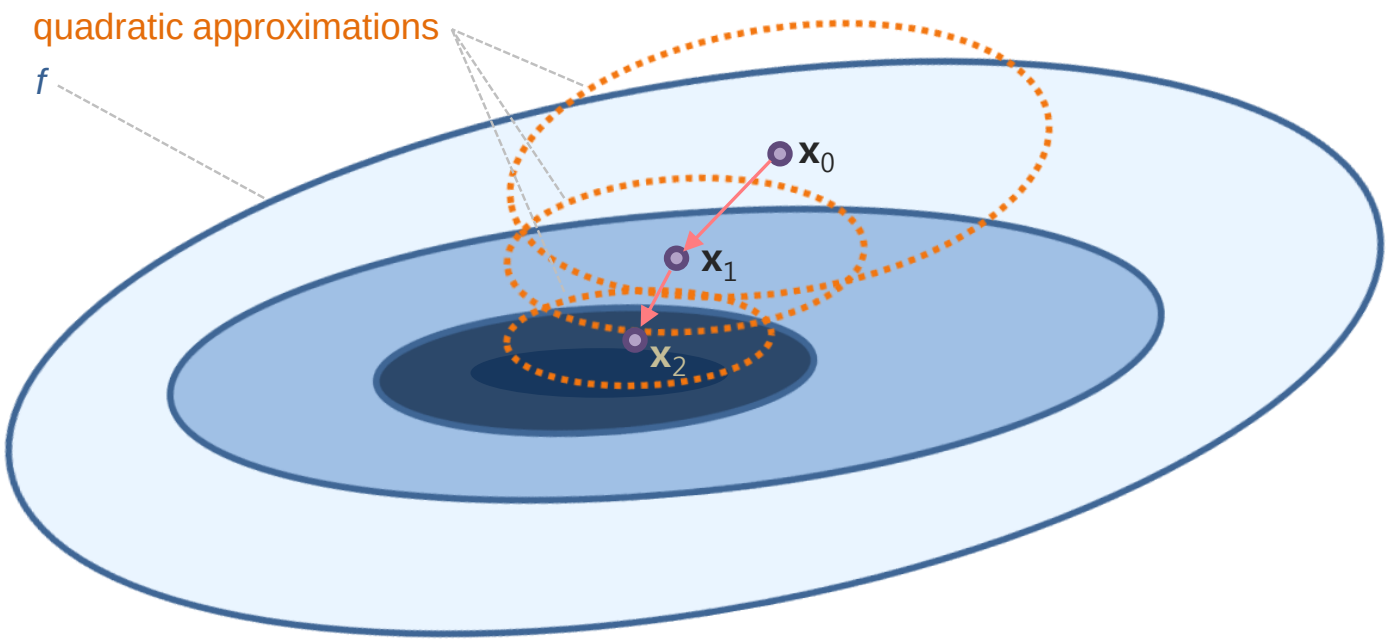
$$\nabla f(\mathbf{x}_t) + H[f](\mathbf{x}_t)(\mathbf{x} - \mathbf{x}_t) = 0,$$

we obtain

$$\mathbf{x}_{k+1} \Leftarrow \mathbf{x}_* = \mathbf{x}_t - H[f](\mathbf{x}_t)^{-1} \nabla f(\mathbf{x}_t).$$

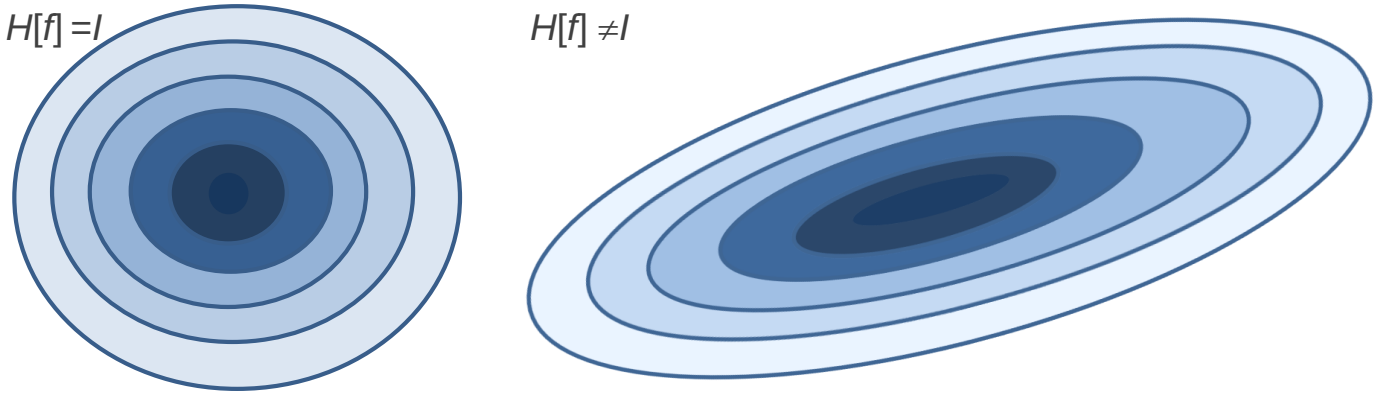
The (pure) Newton method does not use α_t:

(pure) Newton method

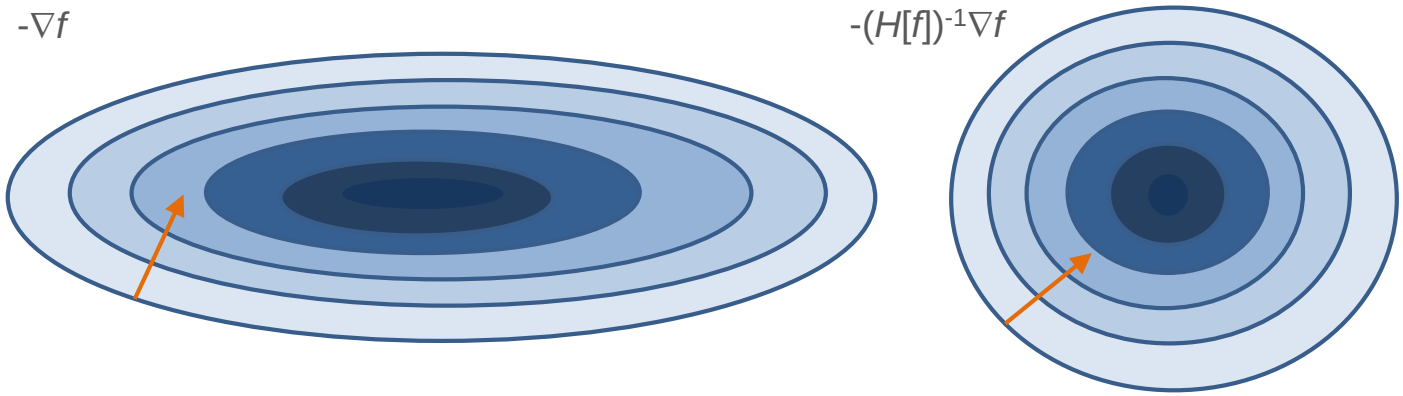


(pure) Newton method

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 $H[f]^{-1}$:

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

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

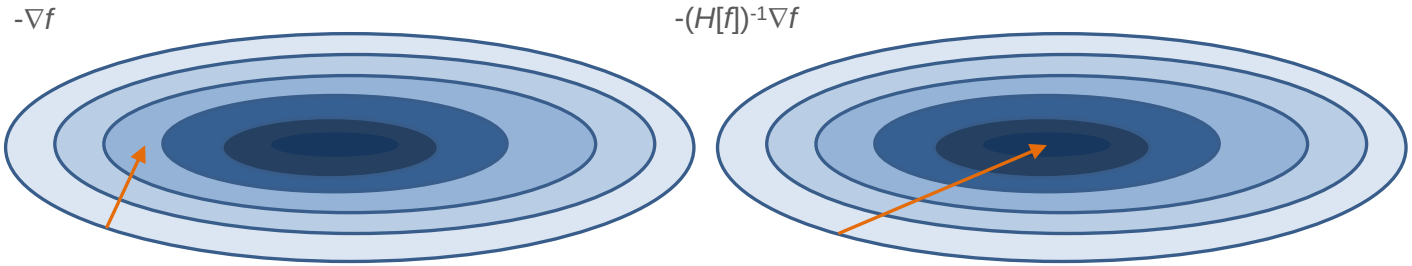
Multiplying the gradient vector ∇f by $H[f]^{-1}$:

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

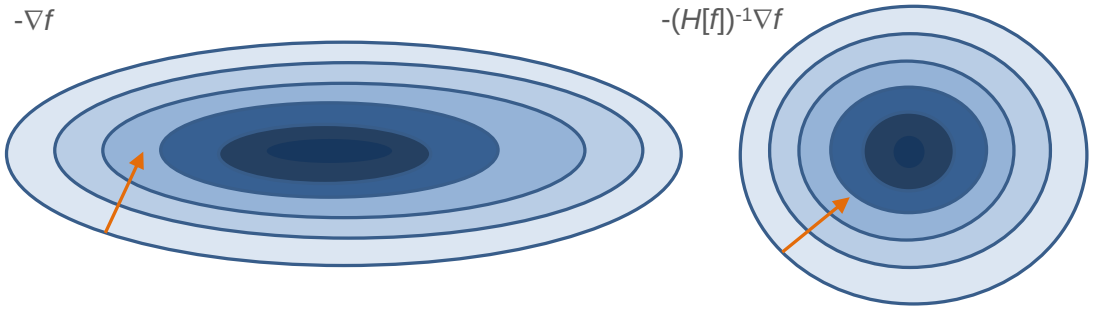
$\Rightarrow$ Projections to basis vectors $\{\mathbf{e}^i\}$ weighted by $\{1/\lambda^i\}$.

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 $H[f](\mathbf{x}_t)$ and solve $H[f](\mathbf{x}_t)^{-1}\nabla f(\mathbf{x}_t)$.

In practice, $H[f](\mathbf{x}_t)^{-1}$ may not always be positive definite.
If $H[f](\mathbf{x}_t)^{-1}$ is not positive definite, $H[f](\mathbf{x}_t)^{-1}\nabla f(\mathbf{x}_t)$ is an ascending direction.
To ensure gradient descent, one could instead use

$$\mathbf{B}_t = H[f](\mathbf{x}_t)^{-1} + \mathbf{E}$$

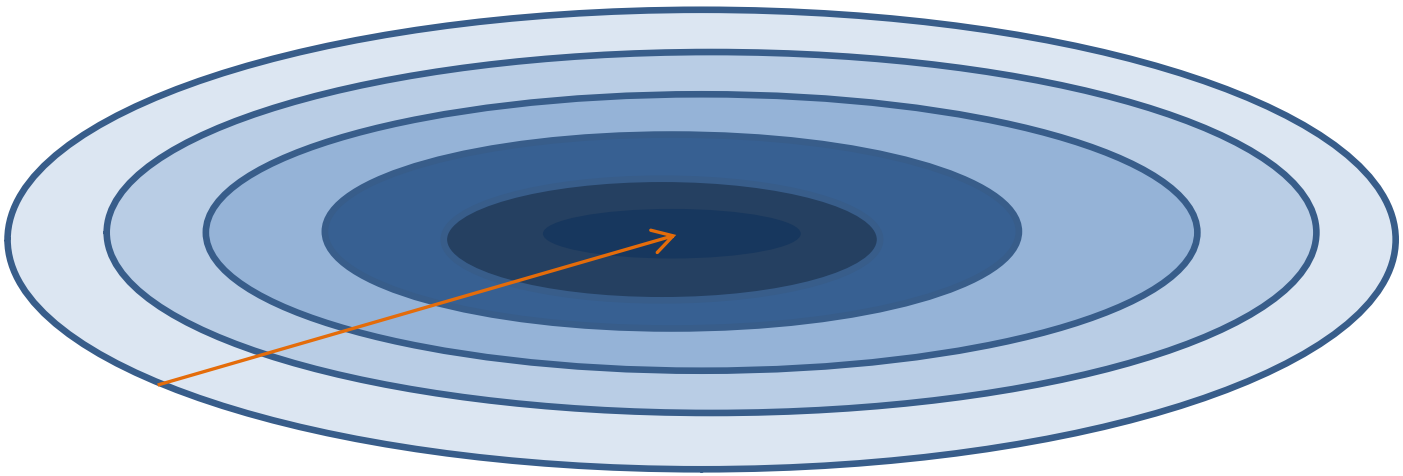
where $\mathbf{E}$ is chosen such that $\mathbf{B}_t$ is positive definite.
E.g., $\mathbf{E} = \lambda I$, $\lambda \geq 0$.

Newton method

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

$$\mathbf{x}_{k+1} = \mathbf{x}_t - \alpha_t H[f](\mathbf{x}_t)^{-1} \nabla f(\mathbf{x}_t)$$

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



independently of how f is elongated.

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.
  - Good even 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*.