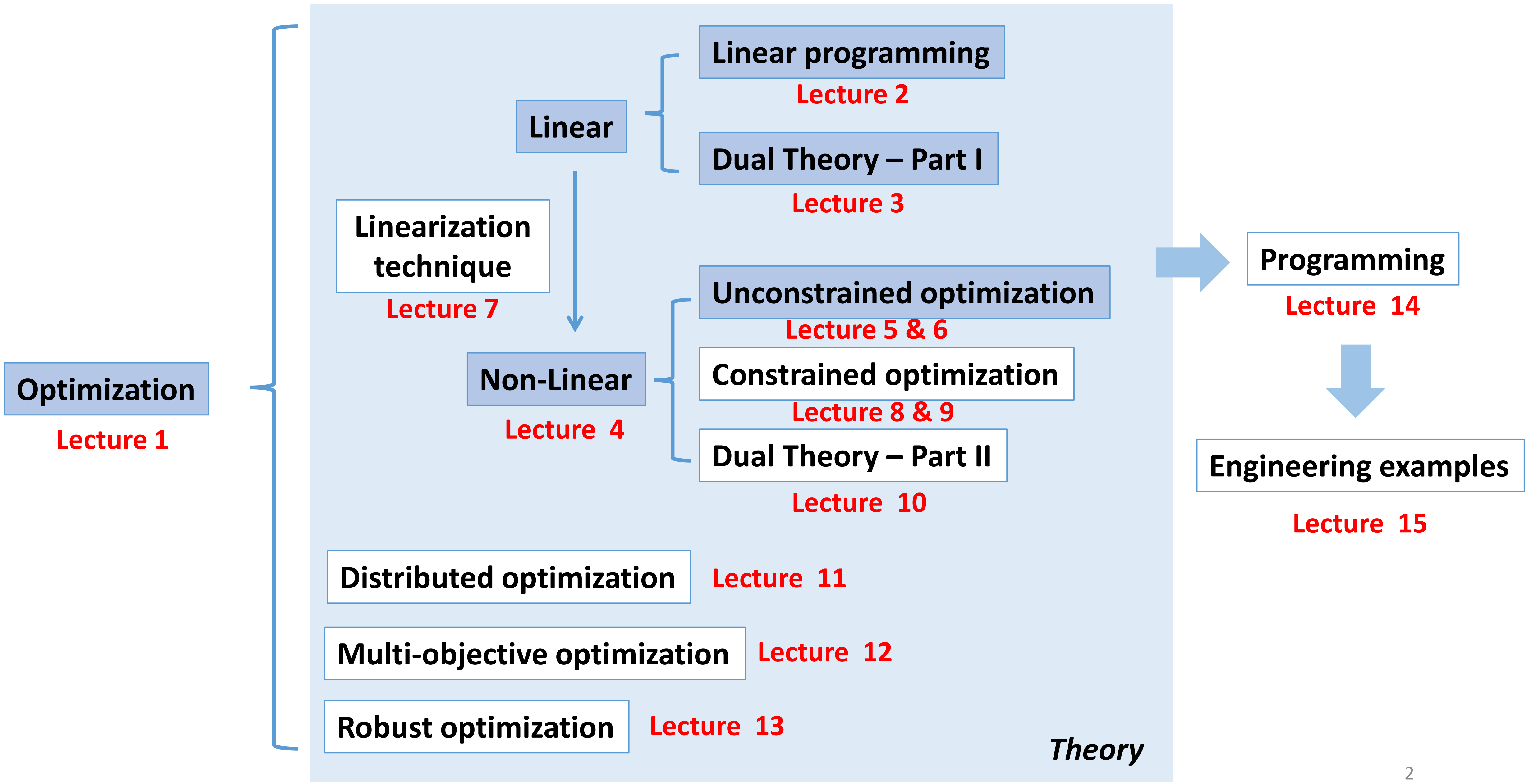
MAEG4070 Engineering Optimization

Lecture 6 Unconstrained Optimization
Gradient Based Methods

Yue Chen
MAE, CUHK
email: yuechen@mae.cuhk.edu.hk
Oct 3, 2022

Content of this course (tentative)



Overview

We consider the unconstrained optimization problem:

$$x^* \in \operatorname{argmin}_x f(x)$$

In the last lecture, we provide necessary (sufficient) conditions for the optimal solution x^* based on gradient and Hessian. However, for high-dimension optimization, to check those conditions can be time-consuming and even impossible.

In practice, we usually use **iterative** algorithms

- Compute sequence of iterates $\{x_k\}$ that converge to x^* at a fast rate
- x_{k+1} is a function of f and previous iterate x_k

or ∇f, $H(x)$, …

What will be learned?

Today's course may be a little bit complicated, BUT the thing you have to remember is

Algorithm: Choose initial point $x_0 \in \mathbb{R}^n$, repeat:

Gradient Descent: $x_k = x_{k-1} - \alpha \nabla f(x_{k-1})$

Or Newton: $x_k = x_{k-1} - \left[\nabla^2 f(x_{k-1})\right]^{-1} \nabla f(x_{k-1})$

Stop until convergence, e.g. $\|x_k - x_{k-1}\| \leq \varepsilon$

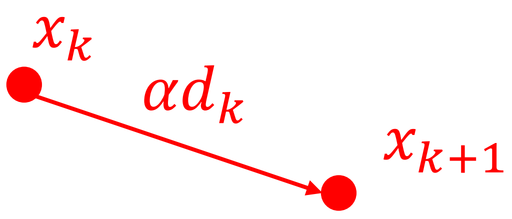
Two kinds of iterative algorithms

Descent & Line search algorithm

- Iteratively find directions d_k and (approximately) solve for

$$\min_{\alpha>0} f(\overbrace{x_k + \alpha d_k}^{x_{k+1}})$$

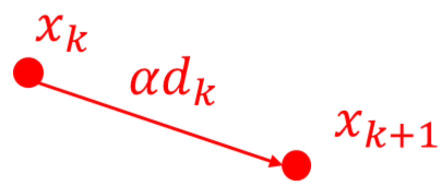
- Some well-known algorithms based on different d_k
  - ✓ Gradient descent
  - ✓ Conjugate descent
  - ✓ Newton
  - ✓ Quasi-Newton
  - ✓ …



Trust Region Algorithm Surrogate model that is easier to solve

- Iteratively solve $\min_{d} s_k(x_k + d)$ where $x_k + d$ lies in some "trust region"; $s_k(.)$ is an approximation of $f(x)$ that is accurate in trust region.

Descent & Line Search Algorithm



Question 1: How to determine the direction d_k?

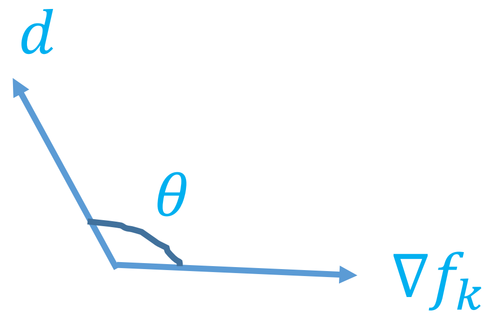
We want to find d_k such that $f(x_k + \alpha d_k) < f(x_k)$; as "steep" as possible

By Taylor's Theorem:

$$f(x_k + \alpha d) = f(x_k) + \alpha d^T \nabla f_k + \frac{1}{2}\alpha^2 d^T \nabla^2 f(x_k + td)d, \text{ for some } t \in (0, \alpha)$$

Rate of change of f along direction d is

$$\lim_{\alpha \to 0} \frac{f(x_k + \alpha d) - f(x_k)}{\alpha} = d^T \nabla f_k$$



Without loss of generality, we assume d is a unit direction

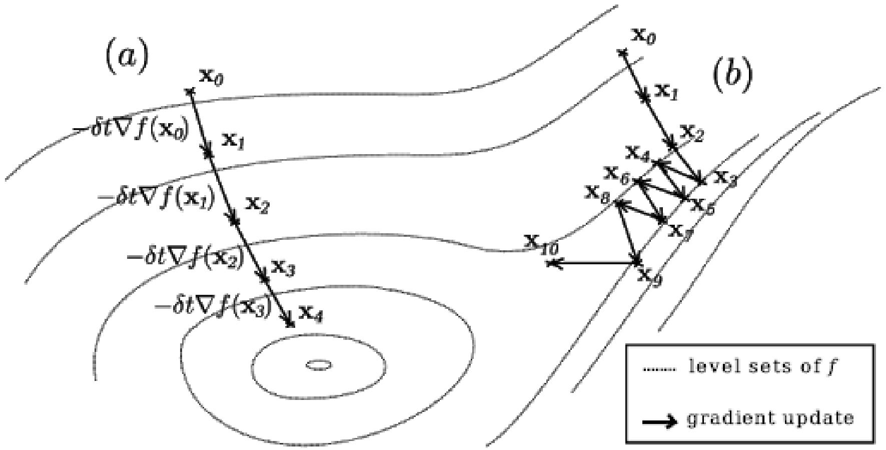
$$\min_d d^T \nabla f_k, \text{ s.t. } ||d|| = 1$$

Therefore

$$d = -\nabla f_k / ||\nabla f_k||$$

Gradient Descent

When $d_k = -\nabla f_k / \|\nabla f_k\|$, it is called the "gradient descent" method.



Algorithm: choose initial point $x_0 \in \mathbb{R}^n$, repeat:

$$x_k = x_{k-1} - \alpha_k \nabla f(x_{k-1}), k = 1, 2, 3, \ldots$$

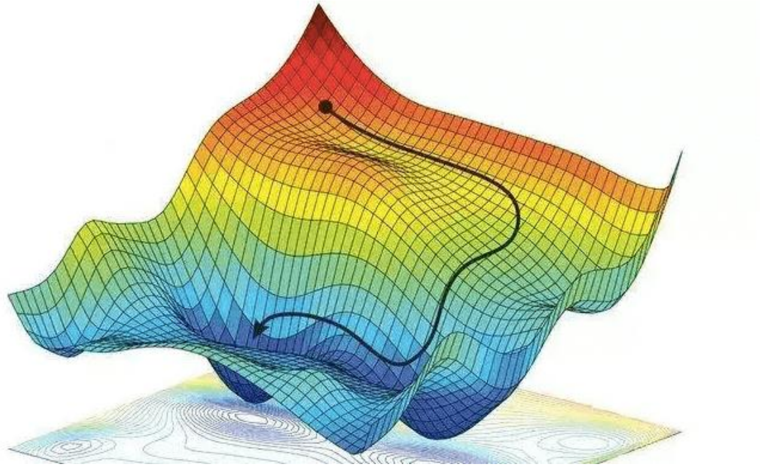
Stop until convergence, e.g. $||x_k - x_{k-1}|| \leq \epsilon$.

For example: $f(x) = \frac{1}{2}(x_1^2 + \gamma x_2^2)$ with $\gamma > 1$

$$\nabla f = \begin{bmatrix} x_1 \\ \gamma x_2 \end{bmatrix}$$

$$x_{k,1} = x_{(k-1),1} - \alpha_k x_{(k-1),1}$$

$$x_{k,2} = x_{(k-1),2} - \alpha_k \gamma x_{(k-1),2}$$



Pictures from Google image

Gradient Descent

Interpretation:

If we approximate the Hessian $\nabla^2 f$ by $\frac{1}{\alpha} I$, then

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha} \|y - x\|_2^2$$

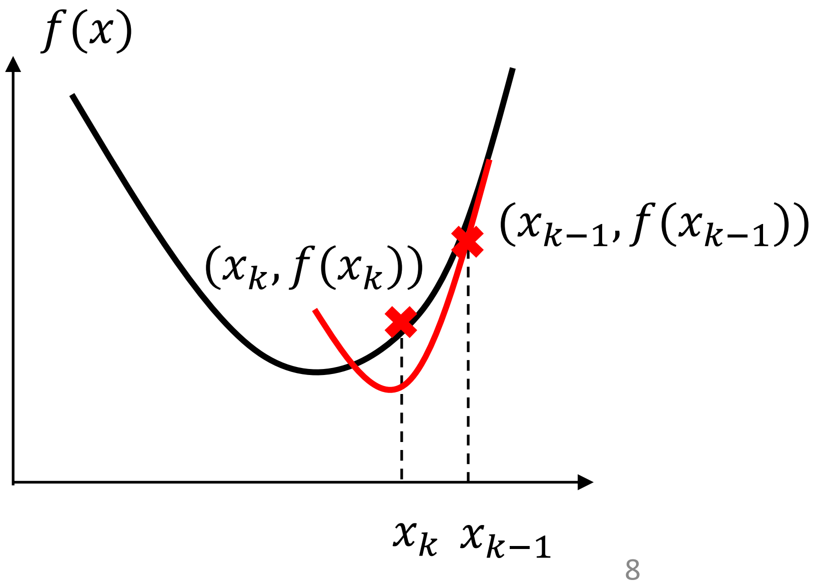
Let $x = x_{k-1}$, we want to choose $x_k = y$ that minimizes $f(y)$

$$\min_y \frac{1}{2\alpha} \|y - x\|_2^2 + \nabla f(x)^T (y - x)$$

$$\frac{1}{\alpha}(y - x) + \nabla f(x_{k-1}) = 0$$

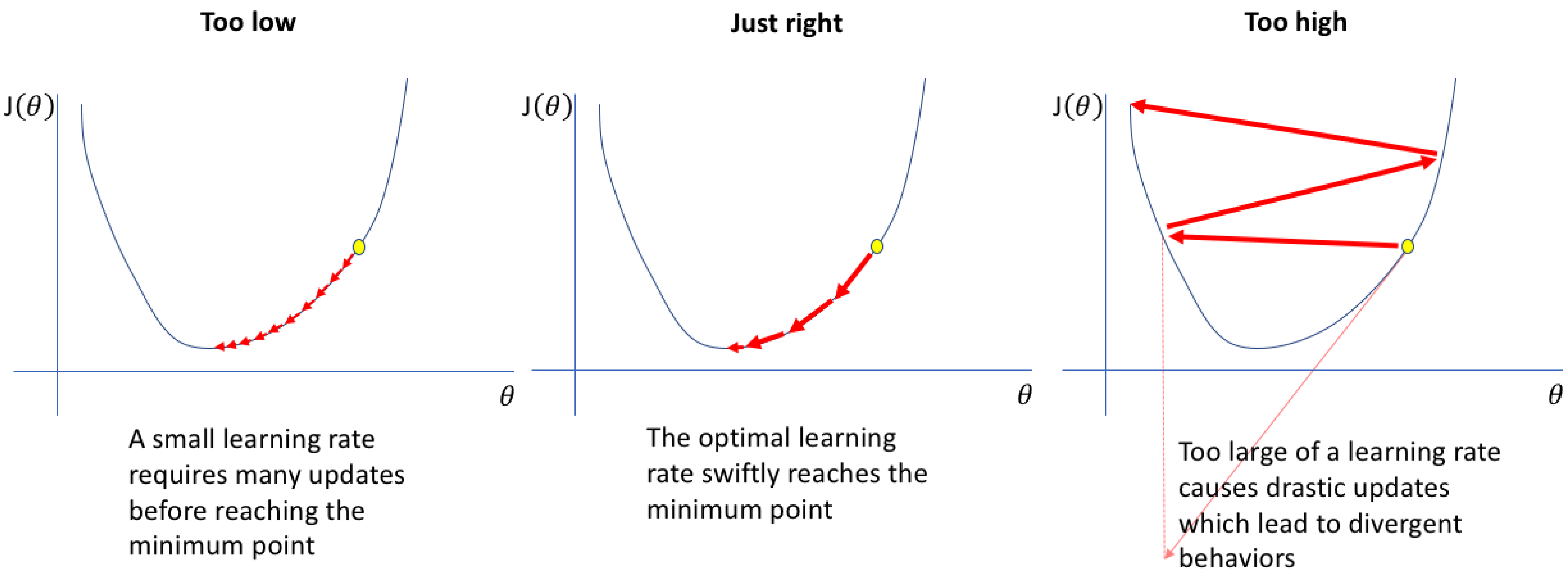
Therefore

$$x_k = x_{k-1} - \alpha \nabla f(x_{k-1})$$



Gradient Descent

Question 2: **How to choose the step size α**:
Gradient descent is also widely used in machine learning, where *step size* is called *learning rate.*



Pictures from Google image

Descent based algorithms with better guarantees

Algorithms with improved convergence
- Newton methods
- Quasi-Newton methods
- Conjugate gradient method
- Accelerated gradient method

Algorithms for nondifferentiable or constrained problems
- Subgradient method
- Proximal gradient method
- Smoothing methods
- Cutting-plane methods

Newton Method

Newton method was originally developed to find a root of an equation $f(x) = 0$.

First, we give the linear approximation of function $f(x)$

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x$$

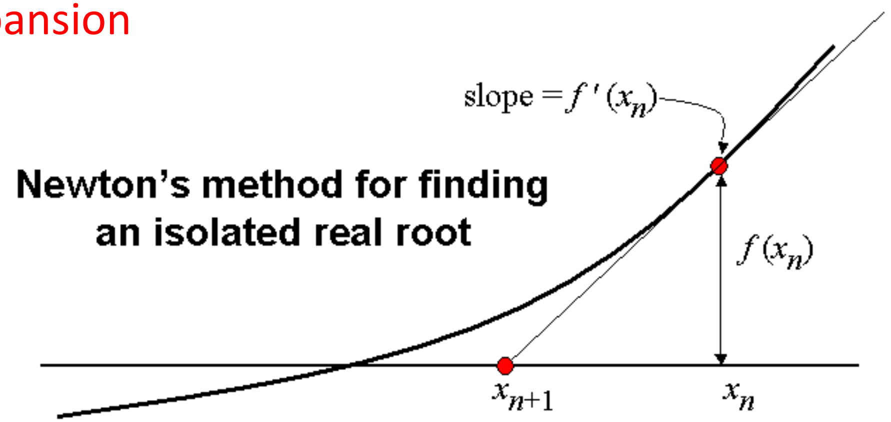
Taylor expansion

Let $f(x + \Delta x) = 0$, we have

$$\Delta x = -\frac{f(x)}{f'(x)}$$

Then

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



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

Pictures from Google image

Newton Method

It can be easily extended to multi-variate case as

$$x_k = x_{k-1} - [\nabla f(x_{k-1})]^{-1} f(x_{k-1})$$

$$\mathbb{R}^n \quad \mathbb{R}^n \quad \mathbb{R}^{n\times m} \quad \mathbb{R}^m$$

Newton method for the unconstrained optimization problem

$$\min_x f(x)$$

Is the same as Newton method for finding a root of

$$\nabla f(x) = 0$$

History: Newton (1685) and Raphson (1690) originally focused on finding the roots of polynomials. Simpson (1740) applied this idea to general nonlinear equations and minimizations.

Newton Method

Consider the unconstrained optimization problem

$$\min_x f(x)$$

Where $f\colon\ \mathbb{R}^n \to \mathbb{R}$ is twice differentiable

$$\underbrace{\boxed{\nabla f(x^*)}}_{}$$

We want to find the optimal point that satisfies $\overset{g(x)}{\boxed{\nabla f(x^*)}} = 0_n$

Note that $\Delta f\colon \mathbb{R}^n \to \mathbb{R}^n$

$g(x + \Delta x) \approx g(x) + g'(x)\Delta x$

Similarly, through linear approximation

$$\nabla f(x + \Delta x) \approx \nabla f(x) + \nabla^2 f(x)\Delta x = 0_n$$

Newton step: $x_k = x_{k-1} - \boxed{\left[\nabla^2 f(x_{k-1})\right]^{-1} \nabla f(x_{k-1})}$

Iterate until convergence, or exceed a maximum number of iterates

Newton Method

Interpretation:

Consider the second-order Taylor approximation

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$$

Assume $\nabla^2 f(x)$ is positive definite, so that $f(x)$ has a strict global optimum. Let $x = x_{k-1}$, we want to choose $x_k = y$ that minimizes $f(y)$

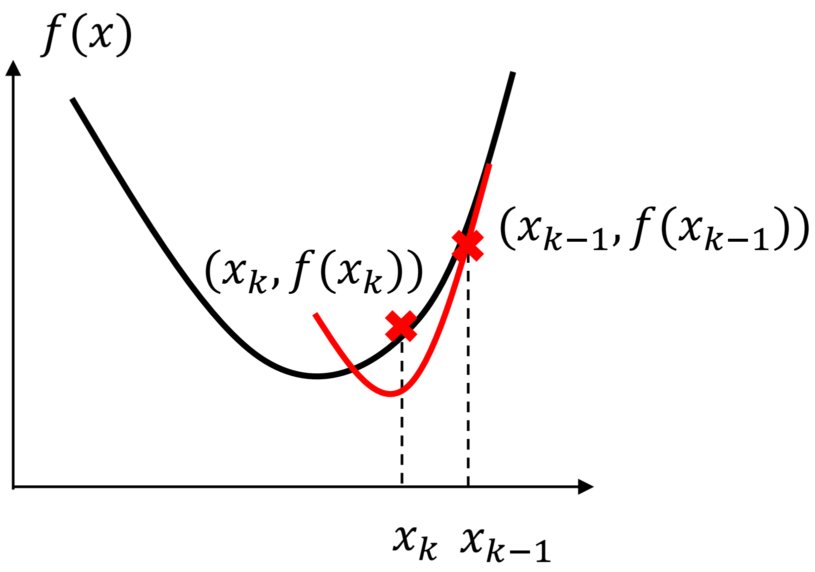
$$\min_y \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x) + \nabla f(x)^T (y - x)$$

$\nabla^2 f(x)(y - x) + \nabla f(x) = 0$

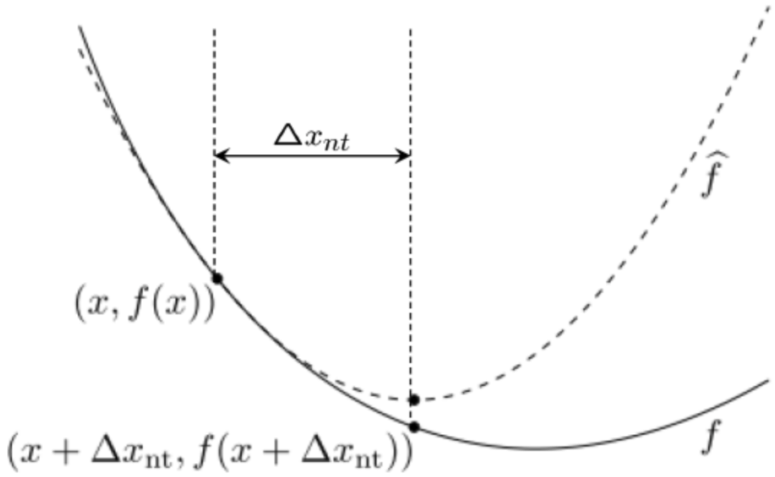
Therefore

$$x_k = x_{k-1} - \left[\nabla^2 f(x_{k-1})\right]^{-1} \nabla f(x_{k-1})$$

Comparison of Gradient Descent & Newton Method



Gradient Descent



Newton Method

Newton method is obtained by minimizing over quadratic approximation:

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$$

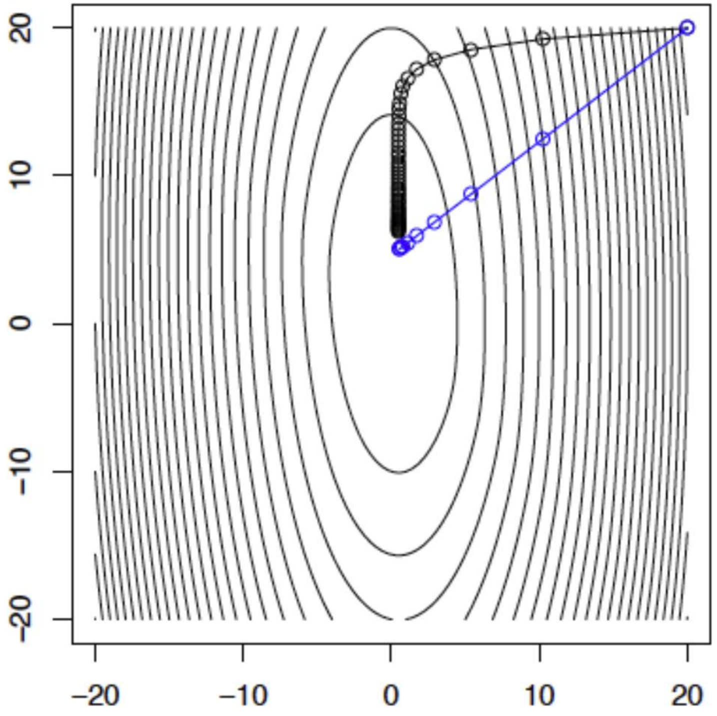
Gradient descent uses another quadratic approximation:

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha}\|y - x\|_2^2$$

Pictures from Google image

Comparison of Gradient Descent & Newton Method

Example: for $f(x) = \frac{10x_1^2+x_2^2}{2} + 5\log(1+e^{-x_1-x_2})$, the result of gradient descent (black) and Newton method (blue) is compared. (step size are similar)



Newton method is faster

Pictures from Google image

Example

Solve the optimization $\min_{x_1,x_2} f(x) = 2x_1^2 + 4x_2^2$ for one step, using gradient descent and Newton method, respectively. Choose $\alpha = 0.1$.

Solution: Let $x^{(0)} = (1,1)^T$, then

$$\nabla f\left(x^{(0)}\right) = \begin{pmatrix} 4x_1 \\ 8x_2 \end{pmatrix}\Bigg|_{x^{(0)}} = \begin{pmatrix} 4 \\ 8 \end{pmatrix}$$

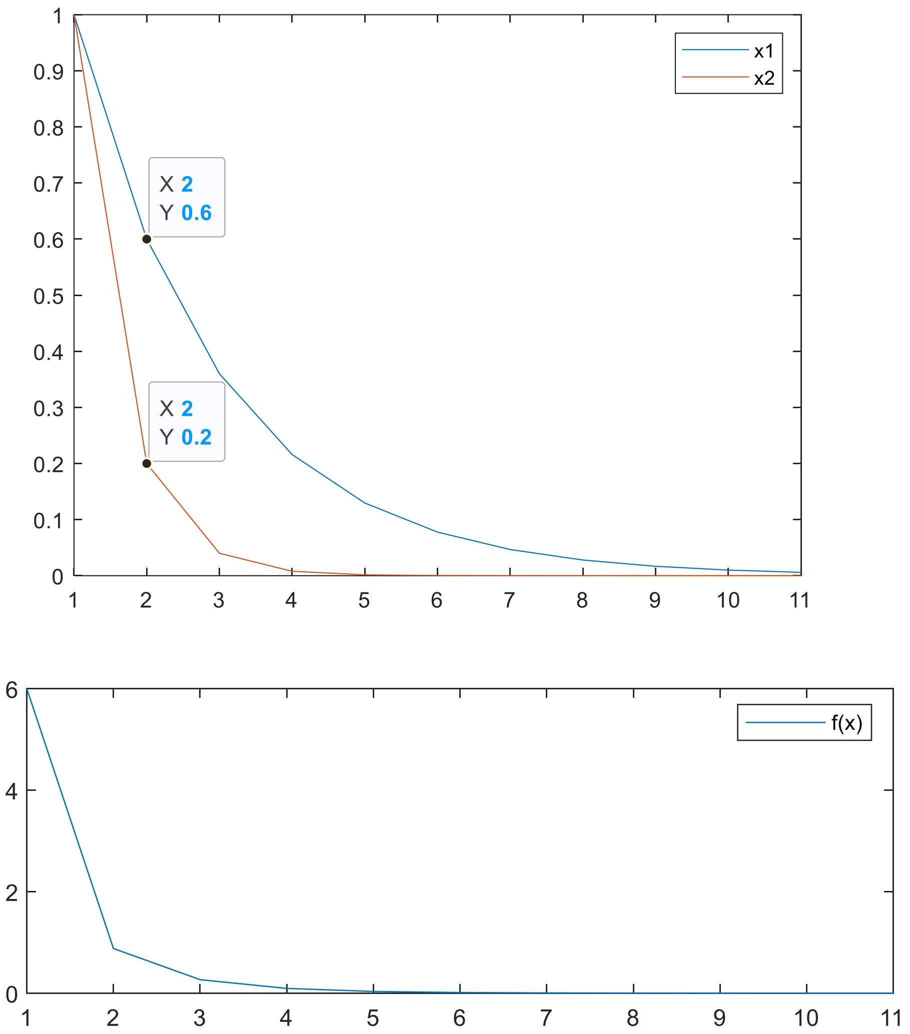
$$\nabla^2 f\left(x^{(0)}\right) = \begin{pmatrix} 4 & 0 \\ 0 & 8 \end{pmatrix}, \nabla^2 f\left(x^{(0)}\right)^{-1} = \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{8} \end{pmatrix}$$

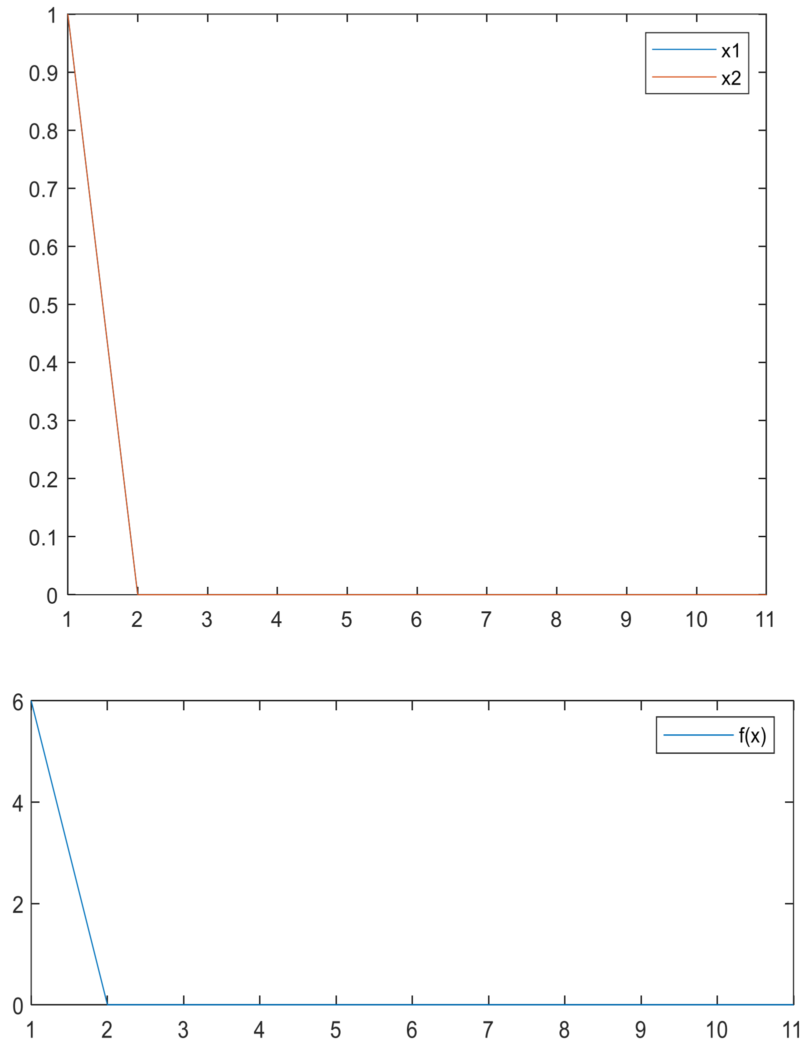
Gradient descent: $x^{(1)} = x^{(0)} - \alpha \nabla f\left(x^{(0)}\right) = \begin{pmatrix} 0.6 \\ 0.2 \end{pmatrix}$

Newton method: $x^{(1)} = x^{(0)} - \nabla^2 f\left(x^{(0)}\right)^{-1} \nabla f\left(x^{(0)}\right)$

$$= \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{8} \end{pmatrix} \begin{pmatrix} 4 \\ 8 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Example





Gradient Descent

Newton Method

Acceleration Techniques

Recall Gradient Descent $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$

- It can be slow since it relies too much on local information $\nabla f(x_k)$ to decide the iterate direction
- To accelerate this process, an "momentum" item is included

Heavy Ball Algorithm

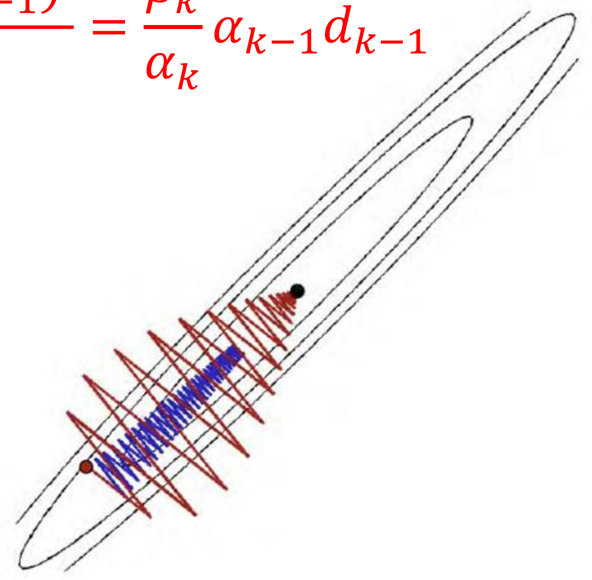
$$\frac{\beta_k(x_k - x_{k-1})}{\alpha_k} = \frac{\beta_k}{\alpha_k}\alpha_{k-1}d_{k-1}$$

- $x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k(x_k - x_{k-1})$
- It can be rewritten as

$$d_k = -\nabla f(x_k) + \frac{\alpha_{k-1}}{\alpha_k}\beta_k d_{k-1}$$

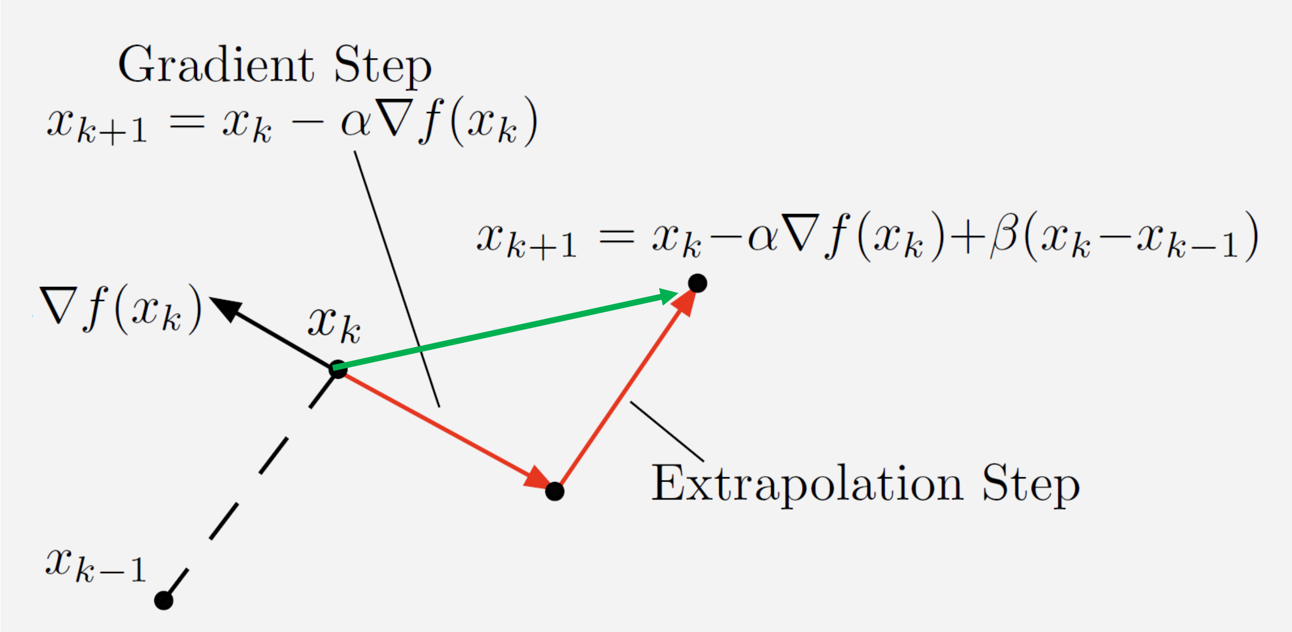
$$x_{k+1} = x_k + \alpha_k d_k$$

- When f is quadratic, this is the Chebyshev Iterative Method
- Momentum prevents oscillation due to local-driven direction

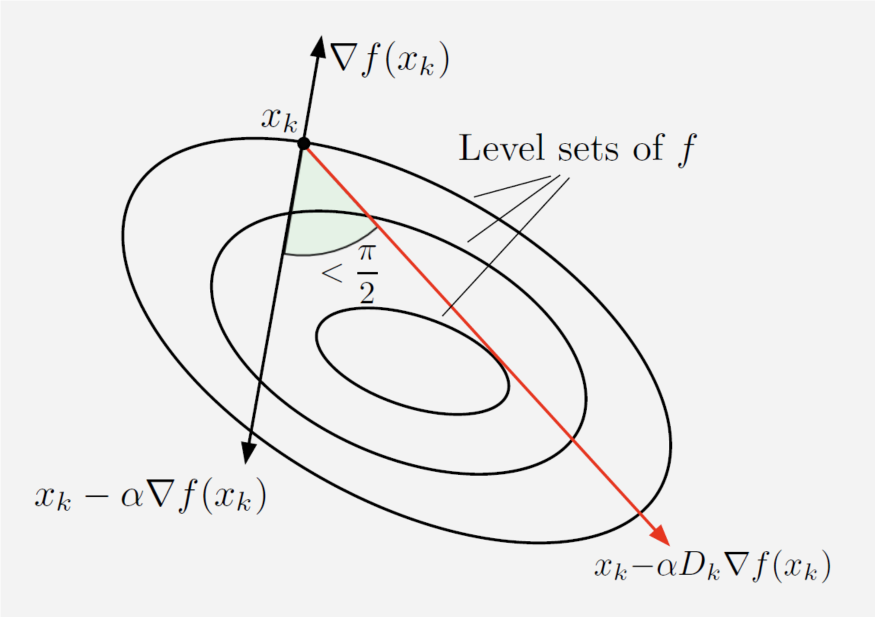


Pictures from Google image

Acceleration Techniques



Interpretation



Need not be a descent direction

Pictures from Google image

Acceleration Techniques

Methods using Momentum:

- **Heavy ball Method**

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1})$$

- **Conjugate Gradient**

$$d_k = -\nabla f(x_k) + \beta_k d_{k-1}$$
$$x_{k+1} = x_k + \alpha_k d_k$$

Let u, v be vectors in $\mathbb{R}^n$ and let A be a positive definite $n \times n$ matrix. u and v are said to be mutually A-conjugate if and only if $u^T A v$=0.

Similar to Heavy ball, but β_k is specially chosen to ensure d_k is conjugate to $\{d_1, \ldots, d_{k-1}\}$

- **Nesterov's Optimal Method**

$$d_k = -\nabla f\big(x_k + \beta_k (x_k - x_{k-1})\big) + \beta_k d_{k-1}$$
$$x_{k+1} = x_k + \alpha_k d_k$$

Summary

Algorithm: Choose initial point $x_0 \in \mathbb{R}^n$, repeat:

Gradient Descent: $x_k = x_{k-1} - \alpha \nabla f(x_{k-1})$

Or Newton: $x_k = x_{k-1} - \left[\nabla^2 f(x_{k-1})\right]^{-1} \nabla f(x_{k-1})$

Stop until convergence, e.g. $\|x_k - x_{k-1}\| \le \varepsilon$

Other issues (outside the scope of this lecture)

- How to prove the convergence of the algorithms? (will reach optimum or not)
- What's the convergence rate of the algorithms? (How fast)

Gradient based methods

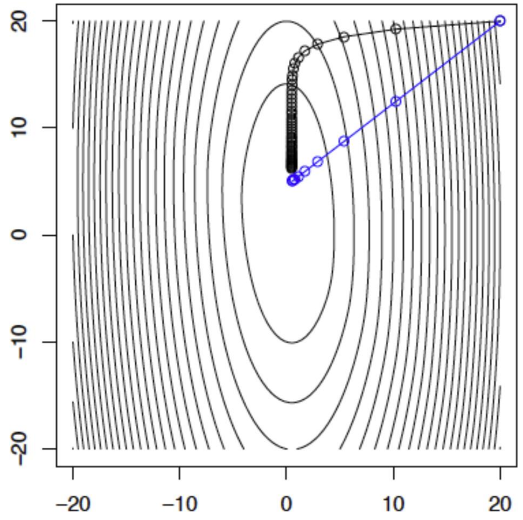
How to apply gradient descent in machine learning

https://www.youtube.com/watch?v=sDv4f4s2SB8

Why not Netwon method widely used in machine learning?

- Though it takes fewer steps to converge
- But it has significant computational burden
  - full Hessian is costly
  - inverted Hessian is costly and unstable
  - Hard to implement online
  - etc.
- More discussion:

https://stats.stackexchange.com/questions/253632/why-is-newtons-method-not-widely-used-in-machine-learning



Thanks！