Matrix calculus

Useful definitions and notations

We will treat all vectors as column vectors by default.

Matrix and vector multiplication

Let A be $m \times n$, and B be $n \times p$, and let the product AB be

$$C = AB$$

then C is a $m \times p$ matrix, with element (i, j) given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Let A be $m \times n$, and x be $n \times 1$, then the typical element of the product

$$z = Ax$$

is given by

$$z_i = \sum_{k=1}^{n} a_{ik} x_k$$

Finally, just to remind:

- $C = AB \quad C^\top = B^\top A^\top$
- $AB \neq BA$
- $e^{A} = \sum\limits_{k=0}^{\infty} \frac{1}{k!} A^k$
- $e^{A+B} \neq e^{A} e^{B}$
- $\langle x, Ay \rangle = \langle A^\top x, y \rangle$

Gradient

Gradient Let $f(x) : \mathbb{R}^n \to \mathbb{R}$, then vector, which contains all first order partial derivatives:

$$\nabla f(x) = \dfrac{df}{dx} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Hessian

Let $f(x) : \mathbb{R}^n \to \mathbb{R}$, then matrix, containing all the second order partial derivatives:

$$
f''(x) = \frac{\partial^2 f}{\partial x_i \partial x_j} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}
$$

But actually, Hessian could be a tensor in such a way: $(f(x) : \mathbb{R}^n \to \mathbb{R}^m)$ is just 3d tensor, every slice is just hessian of corresponding scalar function $(H(f_1(x)), H(f_2(x)), \dots, H(f_m(x)))$

Jacobian

The extension of the gradient of multidimensional $f(x) : \mathbb{R}^n \to \mathbb{R}^m$:

$$
f'(x) = \frac{df}{dx^T} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}
$$

Summary

$$
f(x) : X \to Y; \qquad \frac{\partial f(x)}{\partial x} \in G
$$

X	Y	G	Name
$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}$	$f'(x)$ (derivative)
$\mathbb{R}^n$	$\mathbb{R}$	$\mathbb{R}^{\mathrm{n}}$	$\frac{\partial f}{\partial x_i}$ (gradient)
$\mathbb{R}^n$	$\mathbb{R}^m$	$\mathbb{R}^{m \times n}$	$\frac{\partial f_i}{\partial x_j}$ (jacobian)
$\mathbb{R}^{m \times n}$	$\mathbb{R}$	$\mathbb{R}^{m \times n}$	$\frac{\partial f}{\partial x_{ij}}$

named gradient of $f(x)$. This vector indicates the direction of steepest ascent. Thus, vector $-\nabla f(x)$ means the direction of the steepest descent of the function in the point. Moreover, the gradient vector is always orthogonal to the contour line in the point.

General concept

Naive approach

The basic idea of naive approach is to reduce matrix\vector derivatives to the well-known scalar derivatives.

Matrix notation of a function

$$f(x) = c^\top x$$

Scalar notation of a function

$$f(x) = \sum_{i=1}^{n} c_i x_i$$

Matrix notation of a gradient

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

$$\frac{\partial f(x)}{\partial x_k} = c_k$$

Simple derivative

$$\frac{\partial f(x)}{\partial x_k} = \frac{\partial \left(\sum_{i=1}^{n} c_i x_i\right)}{\partial x_k}$$

One of the most important practical trick here is to separate indices of sum (i) and partial derivatives (k). Ignoring this simple rule tends to produce mistakes.

Guru approach

The guru approach implies formulating a set of simple rules, which allows you to calculate derivatives just like in a scalar case. It might be convenient to use the differential notation here.

Differentials

After obtaining the differential notation of df we can retrieve the gradient using following formula:

$$df(x) = \langle \nabla f(x), dx \rangle$$

Then, if we have differential of the above form and we need to calculate the second derivative of the matrix\vector function, we treat "old" dx as the constant dx_1, then calculate $d(df)$

$$d^2 f(x) = \langle \nabla^2 f(x) dx_1, dx_2 \rangle = \langle H_f(x) dx_1, dx_2 \rangle$$

Properties

Let A and B be the constant matrices, while X and Y are the variables (or matrix functions).

- $dA = 0$
- $d(\alpha X) = \alpha(dX)$
- $d(AXB) = A(dX)B$
- $d(X + Y) = dX + dY$
- $d(X^\top) = (dX)^\top$
- $d(XY) = (dX)Y + X(dY)$
- $d\langle X, Y \rangle = \langle dX, Y \rangle + \langle X, dY \rangle$
- $d\left(\dfrac{X}{\phi}\right) = \dfrac{\phi dX - (d\phi)X}{\phi^2}$
- $d\left(\det X\right) = \det X \langle X^{-\top}, dX \rangle$
- $d\text{tr } X = \langle I, dX \rangle$

- $df(g(x)) = \dfrac{df}{dg} \cdot dg(x)$
- $H = (J(\nabla f))^T$
- $d(X^{-1}) = -X^{-1}(dX)X^{-1}$

References

- Good introduction
- The Matrix Cookbook
- MSU seminars (Rus.)
- Online tool for analytic expression of a derivative.
- Determinant derivative

Example 1

Find $\nabla f(x)$, if $f(x) = \dfrac{1}{2}x^TAx + b^Tx + c$.

Example 2

Find $\nabla f(x)$, $f''(x)$, if $f(x) = -e^{-x^Tx}$.

Example 3

Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \dfrac{1}{2}\|Ax - b\|_2^2$.

Example 4

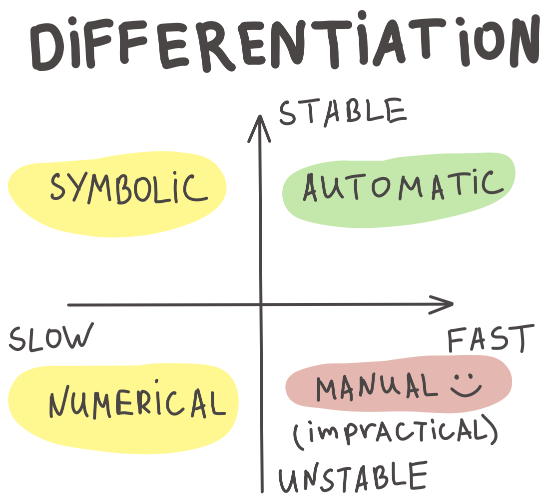
Calculate: $\dfrac{\partial}{\partial X} \sum \text{eig}(X), \quad \dfrac{\partial}{\partial X} \prod \text{eig}(X), \quad \dfrac{\partial}{\partial X}\text{tr}(X), \quad \dfrac{\partial}{\partial X}\text{det}(X)$

Example 5

Find $\nabla f(X)$, if $f(X) = \langle S, X\rangle - \log\det X$

Automatic differentiation

Idea



Automatic differentiation is a scheme, that allow you to compute a value of gradient of function with a cost of computing function itself only twice.

Chain rule

We will illustrate some important matrix calculus facts for specific cases

Univariate chain rule

Suppose, we have the following functions $R: \mathbb{R} \to \mathbb{R}, L: \mathbb{R} \to \mathbb{R}$ and $W \in \mathbb{R}$. Then

$$
\frac{\partial R}{\partial W} = \frac{\partial R}{\partial L} \frac{\partial L}{\partial W}
$$

Multivariate chain rule

The simplest example:

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

Now, we'll consider $f : \mathbb{R}^n \to \mathbb{R}$:

$$\frac{\partial}{\partial t} f(x_1(t), \ldots, x_n(t)) = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \ldots + \frac{\partial f}{\partial x_n}\frac{\partial x_n}{\partial t}$$

But what if we will add another dimension $f : \mathbb{R}^n \to \mathbb{R}^m$, than the j-th output of f will be:

$$\frac{\partial}{\partial t} f_j(x_1(t), \ldots, x_n(t)) = \sum_{i=1}^{n} \frac{\partial f_j}{\partial x_i}\frac{\partial x_i}{\partial t} = \sum_{i=1}^{n} J_{ji}\frac{\partial x_i}{\partial t},$$

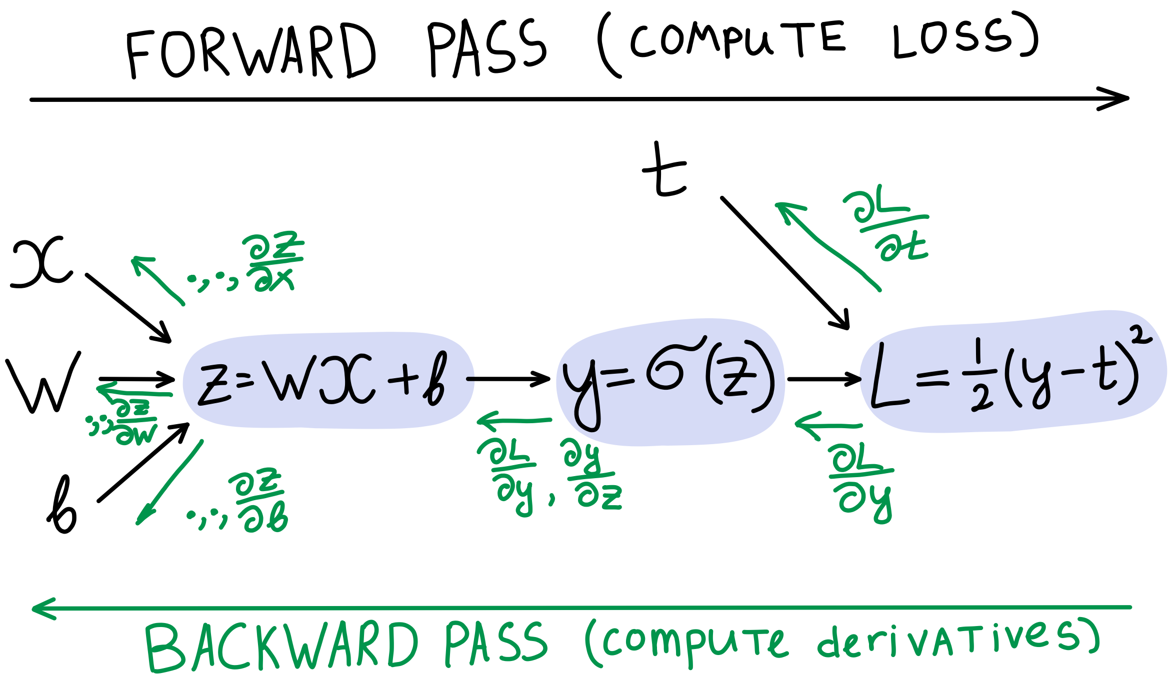
where matrix $J \in \mathbb{R}^{m \times n}$ is the jacobian of the f. Hence, we could write it in a vector way:

$$\frac{\partial f}{\partial t} = J^\top \frac{\partial x}{\partial t} \quad \Longleftrightarrow \quad \left(\frac{\partial f}{\partial t}\right)^\top = \left(\frac{\partial x}{\partial t}\right)^\top J$$

Backpropagation

The whole idea came from the applying chain rule to the computation graph of primitive operations

$$L = L\left(y\left(z(w, x, b)\right), t\right)$$



$$\begin{aligned}
z &= wx + b & \frac{\partial z}{\partial w} &= x, \frac{\partial z}{\partial x} = w, \frac{\partial z}{\partial b} = 0 \\
y &= \sigma(z) & \frac{\partial y}{\partial z} &= \sigma'(z) \\
L &= \frac{1}{2}(y - t)^2 & \frac{\partial L}{\partial y} &= y - t, \frac{\partial L}{\partial t} = t - y
\end{aligned}$$

All frameworks for automatic differentiation construct (implicitly or explicitly) computation graph. In deep learning we typically want to compute the derivatives of the loss function L w.r.t. each intermediate parameters in order to tune them via gradient descent. For this purpose it is convenient to use the following notation:

$$\overline{v_i} = \frac{\partial L}{\partial v_i}$$

Let $v_1, \ldots, v_N$ be a topological ordering of the computation graph (i.e. parents come before children). v_N denotes the variable we're trying to compute derivatives of (e.g. loss).

Forward pass:

- For $i = 1, \ldots, N$:
  - Compute v_i as a function of its parents.

Backward pass:

- $\overline{v_N} = 1$
- For $i = N-1, \ldots, 1$:
  - Compute derivatives $\overline{v_i} = \sum_{j \in \text{Children}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$

Note, that $\overline{v_j}$ term is coming from the children of $\overline{v_i}$, while $\frac{\partial v_j}{\partial v_i}$ is already precomputed effectively.

Jacobian vector product

The reason why it works so fast in practice is that the Jacobian of the operations are already developed in effective manner in automatic differentiation frameworks. Typically, we even do not construct or store the full Jacobian, doing matvec directly instead.

Example: element-wise exponent

$$y = \exp(z) \qquad J = \text{diag}(\exp(z)) \qquad \overline{z} = \overline{y} J$$

See the examples of Vector-Jacobian Products from autodidact library:

```
defvjp(anp.add,         lambda g, ans, x, y : unbroadcast(x, g),
                        lambda g, ans, x, y : unbroadcast(y, g))
defvjp(anp.multiply,    lambda g, ans, x, y : unbroadcast(x, y * g),
                        lambda g, ans, x, y : unbroadcast(y, x * g))
defvjp(anp.subtract,    lambda g, ans, x, y : unbroadcast(x, g),
                        lambda g, ans, x, y : unbroadcast(y, -g))
defvjp(anp.divide,      lambda g, ans, x, y : unbroadcast(x,   g / y),
                        lambda g, ans, x, y : unbroadcast(y, - g * x / y**2))
defvjp(anp.true_divide, lambda g, ans, x, y : unbroadcast(x,   g / y),
                        lambda g, ans, x, y : unbroadcast(y, - g * x / y**2))
```

Hessian vector product

Interesting, that the similar idea could be used to compute Hessian-vector products, which is essential for second order optimization or conjugate gradient methods. For a scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$ with continuous second derivatives (so that the Hessian matrix is symmetric), the Hessian at a point $x \in \mathbb{R}^n$ is written as $\partial^2 f(x)$. A Hessian-vector product function is then able to evaluate

$$v \mapsto \partial^2 f(x) \cdot v$$

for any vector $v \in \mathbb{R}^n$.

The trick is not to instantiate the full Hessian matrix: if n is large, perhaps in the millions or billions in the context of neural networks, then that might be impossible to store. Luckily, `grad` (in the jax/autograd/pytorch/tensorflow) already gives us a way to write an efficient Hessian-vector product function. We just have to use the identity

$$\partial^2 f(x) v = \partial [x \mapsto \partial f(x) \cdot v] = \partial g(x),$$

where $g(x) = \partial f(x) \cdot v$ is a new scalar-valued function that dots the gradient of f at x with the vector v. Notice that we're only ever differentiating scalar-valued functions of vector-valued arguments, which is exactly where we know `grad` is efficient.

```
import jax.numpy as jnp

def hvp(f, x, v):
    return grad(lambda x: jnp.vdot(grad(f)(x), v))(x)
```

Code

Materials

- Autodidact - a pedagogical implementation of Autograd
- CSC321 Lecture 6
- CSC321 Lecture 10
- Why you should understand backpropagation :)
- JAX autodiff cookbook

Convex set

In this chapter variety of convex sets and related definitions are described.

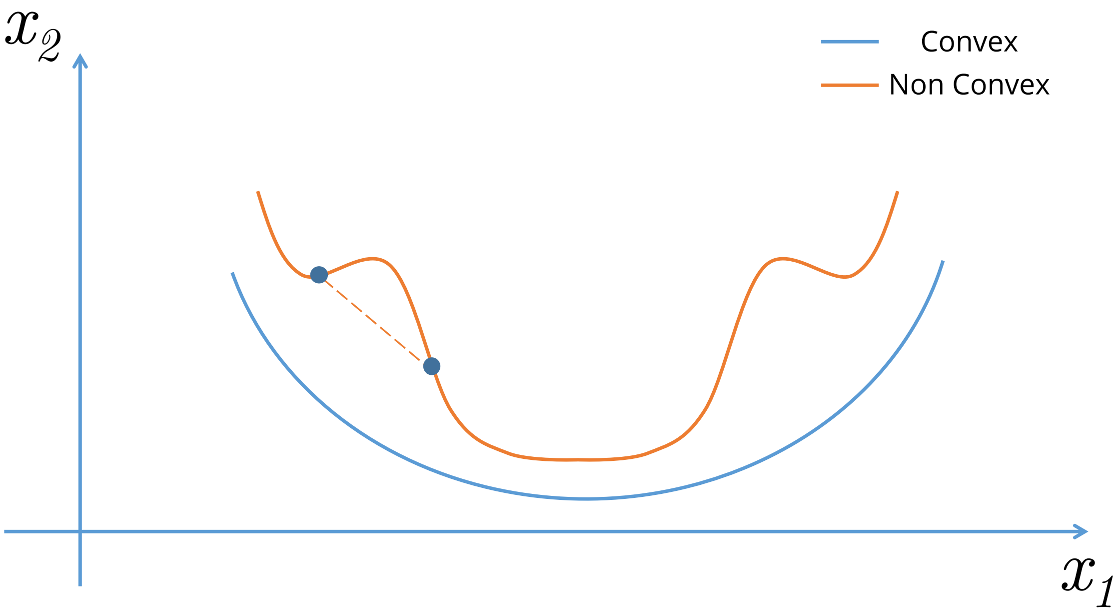
Convex function

Convex function

The function $f(x)$, **which is defined on the convex set** $S \subseteq \mathbb{R}^n$, is called **convex** S, if:

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda) f(x_2)$$

for any $x_1, x_2 \in S$ and $0 \leq \lambda \leq 1$.
If above inequality holds as strict inequality $x_1 \neq x_2$ and $0 < \lambda < 1$, then function is called strictly convex S



Examples

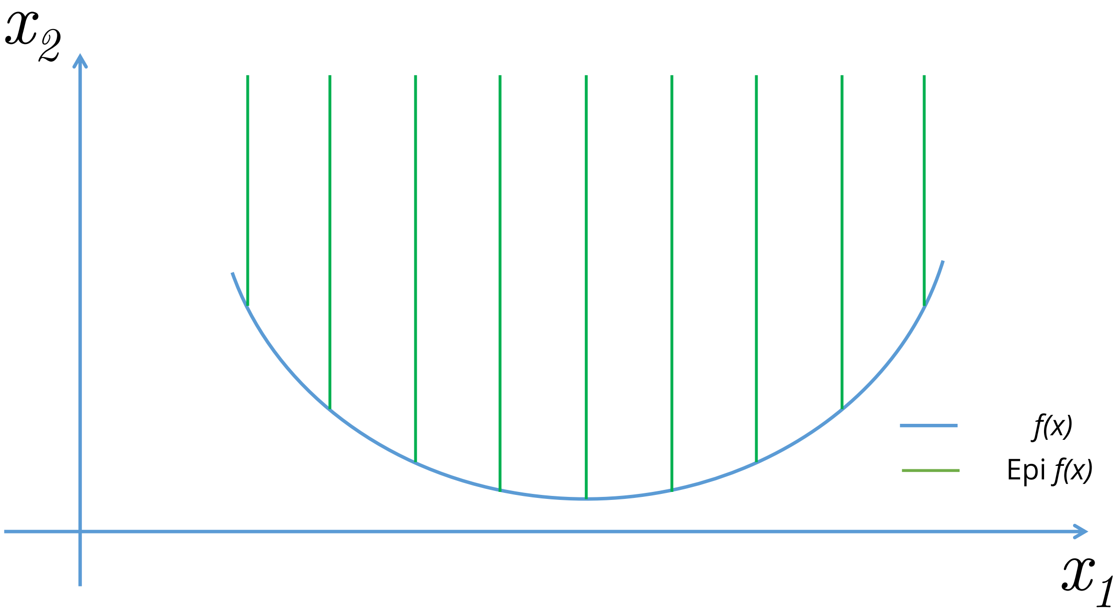
- $f(x) = x^p, p > 1, \quad S = \mathbb{R}_+$
- $f(x) = \|x\|^p, \quad p > 1, S = \mathbb{R}$
- $f(x) = e^{cx}, \quad c \in \mathbb{R}, S = \mathbb{R}$
- $f(x) = -\ln x, \quad S = \mathbb{R}_{++}$
- $f(x) = x \ln x, \quad S = \mathbb{R}_{++}$
- The sum of the largest k coordinates $f(x) = x_{(1)} + \ldots + x_{(k)}, \quad S = \mathbb{R}^n$
- $f(X) = \lambda_{max}(X), \quad X = X^T$
- $f(X) = -\log \det X, \quad S = S^n_{++}$

Epigraph

For the function $f(x)$, defined on $S \subseteq \mathbb{R}^n$, the following set:

$$\text{epi } f = \{[x, \mu] \in S \times \mathbb{R} : f(x) \leq \mu\}$$

is called **epigraph** of the function $f(x)$

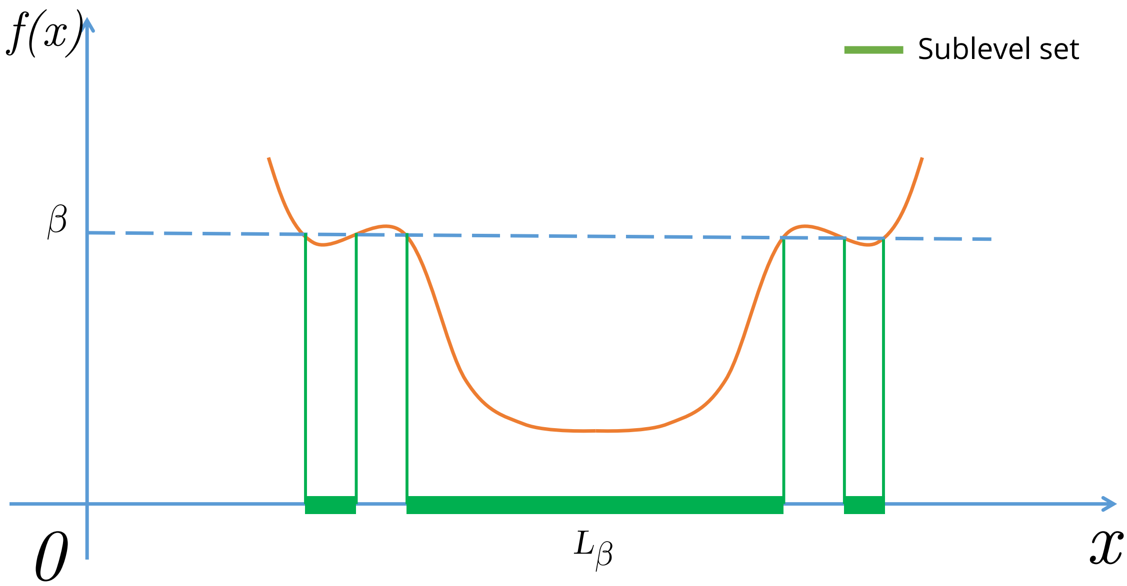


Sublevel set

For the function $f(x)$, defined on $S \subseteq \mathbb{R}^n$, the following set:

$$\mathcal{L}_\beta = \{x \in S : f(x) \leq \beta\}$$

is called **sublevel set** or Lebesgue set of the function $f(x)$



Criteria of convexity

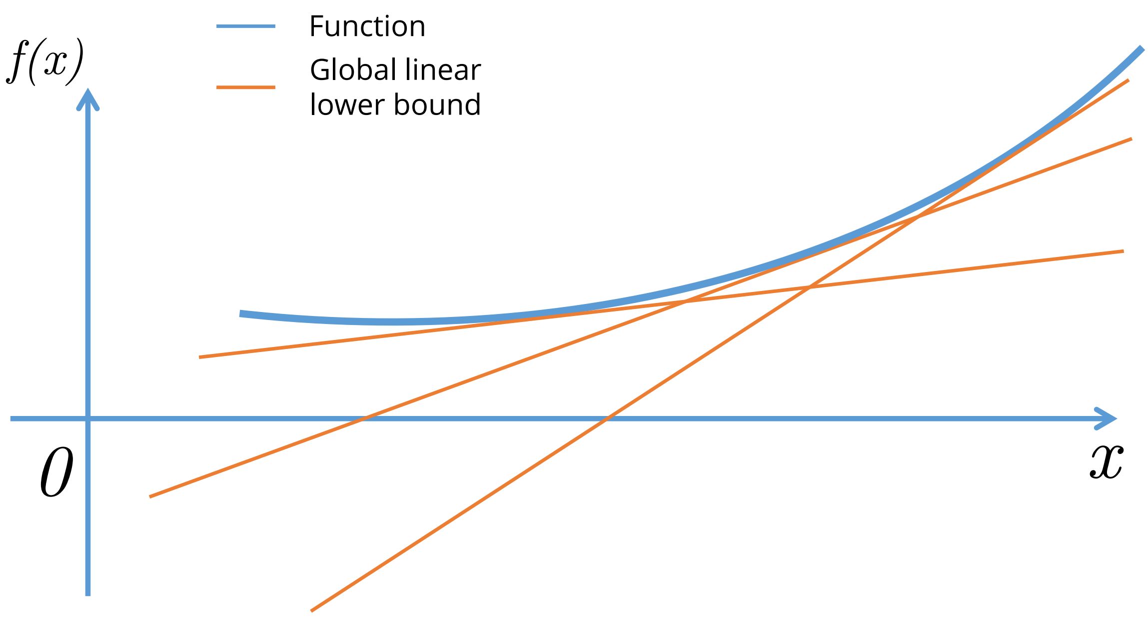
First order differential criterion of convexity

The differentiable function $f(x)$ defined on the convex set $S \subseteq \mathbb{R}^n$ is convex if and only if $\forall x, y \in S$:

$$f(y) \geq f(x) + \nabla f^T(x)(y - x)$$

Let $y = x + \Delta x$, then the criterion will become more tractable:

$$f(x + \Delta x) \geq f(x) + \nabla f^T(x) \Delta x$$



Second order differential criterion of convexity

Twice differentiable function $f(x)$ defined on the convex set $S \subseteq \mathbb{R}^n$ is convex if and only if $\forall x \in \mathbf{int}(S) \neq \emptyset$:

$$\nabla^2 f(x) \succeq 0$$

In other words, $\forall y \in \mathbb{R}^n$:

$$\langle y, \nabla^2 f(x) y \rangle \geq 0$$

Connection with epigraph

The function is convex if and only if its epigraph is convex set.

Connection with sublevel set

If $f(x)$ - is a convex function defined on the convex set $S \subseteq \mathbb{R}^n$, then for any β sublevel set $\mathcal{L}_\beta$ is convex.

The function $f(x)$ defined on the convex set $S \subseteq \mathbb{R}^n$ is closed if and only if for any β sublevel set $\mathcal{L}_\beta$ is closed.

Reduction to a line

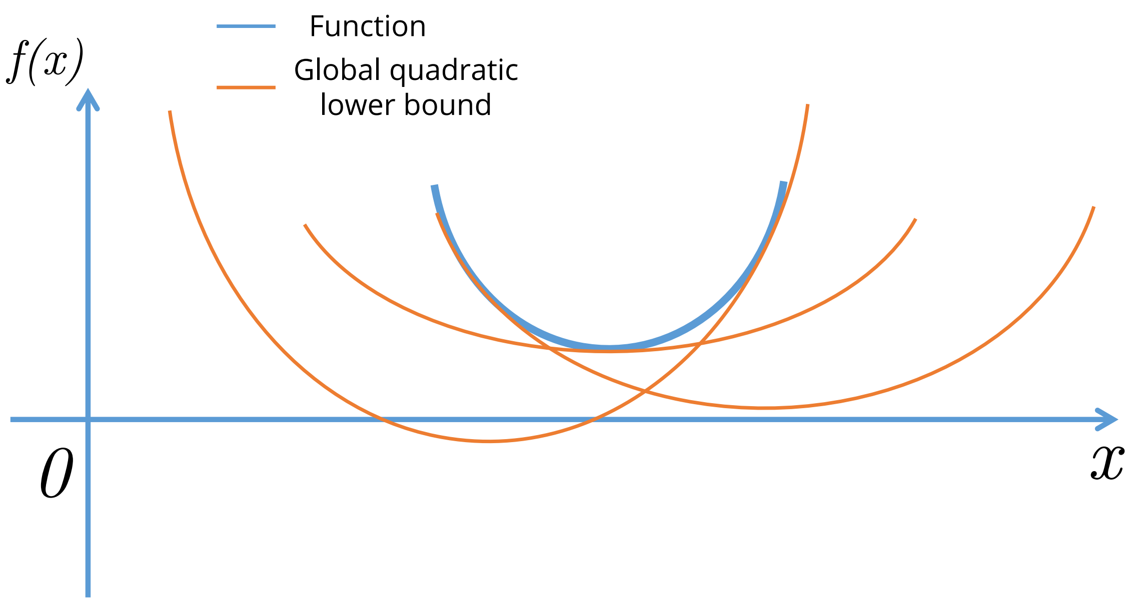
$f : S \to \mathbb{R}$ is convex if and only if S is convex set and the function $g(t) = f(x + tv)$ defined on $\{t \mid x + tv \in S\}$ is convex for any $x \in S, v \in \mathbb{R}^n$, which allows to check convexity of the scalar function in order to establish covexity of the vector function.

Strong convexity

$f(x)$, **defined on the convex set** $S \subseteq \mathbb{R}^n$, is called μ-strongly convex (strogly convex) on S, if:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2) - \mu \lambda (1-\lambda) \|x_1 - x_2\|^2$$

for any $x_1, x_2 \in S$ and $0 \leq \lambda \leq 1$ for some $\mu > 0$.



Criteria of strong convexity

First order differential criterion of strong convexity

Differentiable $f(x)$ defined on the convex set $S \subseteq \mathbb{R}^n$ μ-strongly convex if and only if $\forall x, y \in S$:

$$f(y) \geq f(x) + \nabla f^T(x)(y - x) + \frac{\mu}{2}\|y - x\|^2$$

Let $y = x + \Delta x$, then the criterion will become more tractable:

$$f(x + \Delta x) \geq f(x) + \nabla f^T(x)\Delta x + \frac{\mu}{2}\|\Delta x\|^2$$

Second order differential criterion of strong convexity

Twice differentiable function $f(x)$ defined on the convex set $S \subseteq \mathbb{R}^n$ is called μ-strongly convex if and only if $\forall x \in \mathbf{int}(S) \neq \emptyset$:

$$\nabla^2 f(x) \succeq \mu I$$

In other words:

$$\langle y, \nabla^2 f(x) y\rangle \geq \mu\|y\|^2$$

Facts

- $f(x)$ is called (strictly) concave, if the function $-f(x)$ - (strictly) convex.
- Jensen's inequality for the convex functions:

  $$f\left(\sum_{i=1}^n \alpha_i x_i\right) \leq \sum_{i=1}^n \alpha_i f(x_i)$$

  for $\alpha_i \geq 0; \quad \sum\limits_{i=1}^n \alpha_i = 1$ (probability simplex)

  For the infinite dimension case:

  $$f\left(\int\limits_S x p(x)dx\right) \leq \int\limits_S f(x)p(x)dx$$

  If the integrals exist and $p(x) \geq 0, \quad \int\limits_S p(x)dx = 1$

- If the function $f(x)$ and the set S are convex, then any local minimum $x^* = \arg\min\limits_{x \in S} f(x)$ will be the global one. Strong convexity guarantees the uniqueness of the solution.

Operations that preserve convexity

- Non-negative sum of the convex functions: $\alpha f(x) + \beta g(x), (\alpha \geq 0, \beta \geq 0)$
- Composition with affine function $f(Ax+b)$ is convex, if $f(x)$ is convex
- Pointwise maximum (supremum): If $f_1(x), \ldots, f_m(x)$ are convex, then $f(x) = \max\{f_1(x), \ldots, f_m(x)\}$ is convex
- If $f(x,y)$ is convex on x for any $y \in Y$: $g(x) = \sup\limits_{y \in Y} f(x,y)$ is convex
- If $f(x)$ is convex on S, then $g(x,t) = tf(x/t)$ - is convex with $x/t \in S, t > 0$
- Let $f_1 : S_1 \to \mathbb{R}$ and $f_2 : S_2 \to \mathbb{R}$, where $\mathrm{range}(f_1) \subseteq S_2$. If f_1 and f_2 are convex, and f_2 is increasing, then $f_2 \circ f_1$ is convex on S_1

Other forms of convexity

- Log-convex: $\log f$ is convex; Log convexity implies convexity.
- Log-concavity: $\log f$ concave; **not** closed under addition!
- Exponentially convex: $[f(x_i + x_j)] \succeq 0$, for $x_1, \ldots, x_n$
- Operator convex: $f(\lambda X + (1-\lambda)Y) \preceq \lambda f(X) + (1-\lambda)f(Y)$
- Quasiconvex: $f(\lambda x + (1-\lambda)y) \leq \max\{f(x), f(y)\}$
- Pseudoconvex: $\langle \nabla f(y), x - y \rangle \geq 0 \longrightarrow f(x) \geq f(y)$
- Discrete convexity: $f : \mathbb{Z}^n \to \mathbb{Z}$; “convexity + matroid theory.”

References

- Steven Boyd lectures
- Suvrit Sra lectures
- Martin Jaggi lectures