AI505

Optimization

Derivatives and Gradients

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Derivatives
2. Symbolic Differentiation
3. Numerical Differentiation
4. Automatic Differentiation

Definitions

- $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$ closed interval
  $(a, b) = \{x \in \mathbb{R} \mid a < x < b\}$ open interval
- We denote vectors in bold, eg, $\boldsymbol{x}$ and matrices in uppercase letters, eg, A
- Given a vector $\boldsymbol{x} \in \mathbb{R}^n$, x_i is its i-th component
- We invariably assume that $\boldsymbol{x}$ is a **column vector**,
  which we also write with parentheses as $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ or $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$.
- The transpose $\boldsymbol{x}^T$ is a row vector

$$\boldsymbol{x}^T = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \end{bmatrix}$$

- Scalar product: $\boldsymbol{y}^T \boldsymbol{x} = \sum_{i=1}^{n} y_i x_i$

Definitions

- **linear combination**

$$\boldsymbol{v}_1, \boldsymbol{v}_2 \ldots, \boldsymbol{v}_k \in \mathbb{R}^n \qquad \boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_k]^T \in \mathbb{R}^k$$

$$\boldsymbol{x} = \lambda_1 \boldsymbol{v}_1 + \cdots + \lambda_k \boldsymbol{v}_k = \sum_{i=1}^{k} \lambda_i \boldsymbol{v}_i$$

  moreover:

  $\boldsymbol{\lambda} \geq 0$ **conic combination**

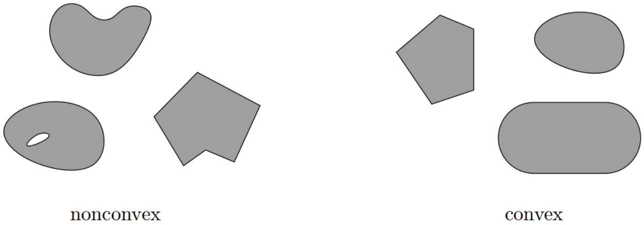
  $\boldsymbol{\lambda}^T 1 = 1$ **affine combination** $\left(\sum_{i=1}^{k} \lambda_i = 1\right)$

  $\boldsymbol{\lambda} \geq 0$ and $\boldsymbol{\lambda}^T 1 = 1$ **convex combination**

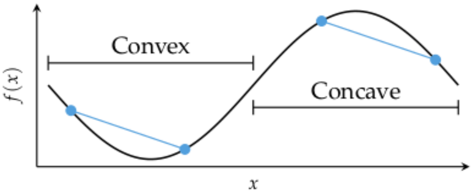
- $A\boldsymbol{x}$ column vector combination of the columns of A;
  $\boldsymbol{u}^T A$ row vector combination of the rows of A

Definitions

- **convex set**: if $\boldsymbol{x}, \boldsymbol{y} \in S$ and $0 \leq \lambda \leq 1$ then $\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in S$



- **convex function**
  - if its epigraph $\{(x, y) \in \mathbb{R}^2 : y \geq f(x)\}$ is a convex set or
  - if $f : \mathbb{R}^n \to \mathbb{R}$ and if $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \alpha \in [0, 1]$ it holds that $f(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{y})$

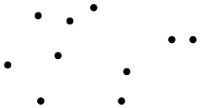


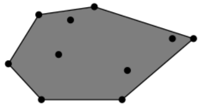
Definitions

- For a set of points $S \subseteq \mathbb{R}^n$

$\mathrm{lin}(S)$	linear hull (span)
$\mathrm{cone}(S)$	conic hull
$\mathrm{aff}(S)$	affine hull
$\mathrm{conv}(S)$	convex hull

X





the convex hull of X

$$\mathrm{conv}(X) = \{\lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2 + \ldots + \lambda_n \boldsymbol{x}_n \;\mid\; \boldsymbol{x}_i \in X,\ \lambda_1, \ldots, \lambda_n \geq 0 \text{ and } \textstyle\sum_i \lambda_i = 1\}$$

Norms

Def. A **norm** is a function that assigns a length to a vector.

A function f is a norm if:

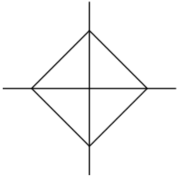
1. $f(\boldsymbol{x}) = 0$ if and only if $\boldsymbol{x}$ is the zero vector
2. $f(a\boldsymbol{x}) = |a| f(\boldsymbol{x})$, such that lengths scale
3. $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$, also known as triangle inequality

L_p norms are commonly used set of norms parameterized by a scalar $p \geq 1$:

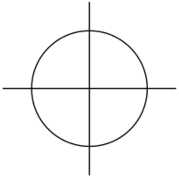
$$\|\boldsymbol{x}\|_p = \lim_{\rho \to p} \left(|x_1|^\rho + |x_2|^\rho + \ldots + |x_n|^\rho\right)^{\frac{1}{\rho}}$$

L_∞ is also called the max norm, Chebyshev distance or chessboard distance.

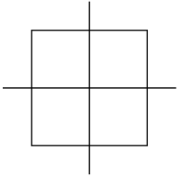
$$L_1\colon \|\mathbf{x}\|_1 = |x_1| + |x_2| + \cdots + |x_n|$$



$$L_2\colon \|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$



$$L_\infty\colon \|\mathbf{x}\|_\infty = \max(|x_1|, |x_2|, \cdots, |x_n|)$$



Matrix Norms

$$\|A\| \stackrel{\text{def}}{=} \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}. \tag{A.7}$$

The matrix norms defined in this way are said to be *consistent* with the vector norms (A.2). Explicit formulae for these norms are as follows:

$$\|A\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^{m} |A_{ij}|, \tag{A.8a}$$

$$\|A\|_2 = \text{largest eigenvalue of } (A^T A)^{1/2}, \tag{A.8b}$$

$$\|A\|_\infty = \max_{i=1,\ldots,m} \sum_{j=1}^{n} |A_{ij}|. \tag{A.8c}$$

The Frobenius norm $\|A\|_F$ of the matrix A is defined by

$$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2 \right)^{1/2}. \tag{A.9}$$

Condition Number

For a nonsingular matrix A, the **condition number** $\kappa(A)$ is defined as

$$\kappa(A) = \|A\| \left\|A^{-1}\right\|$$

where $\|\cdot\|$ is any matrix norm. The condition number of a matrix is a measure of how close the matrix is to being singular. A matrix with a large condition number is said to be ill-conditioned, and small changes in the input can lead to large changes in the output when solving linear systems or performing matrix operations involving that matrix.

Outline

1. Derivatives
2. Symbolic Differentiation
3. Numerical Differentiation
4. Automatic Differentiation

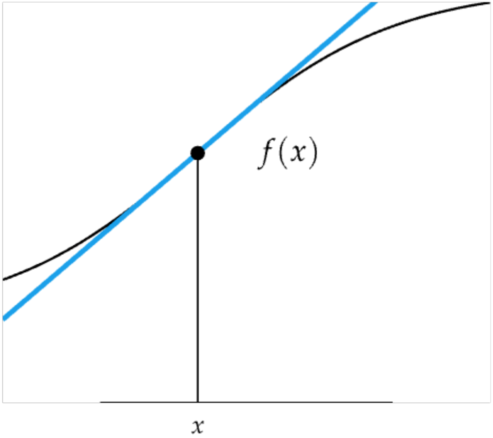
Derivatives

- Derivatives tell us which direction to search for a solution

- Slope of Tangent Line

$$f'(x) := \frac{\mathrm{d}f(x)}{\mathrm{d}x}$$

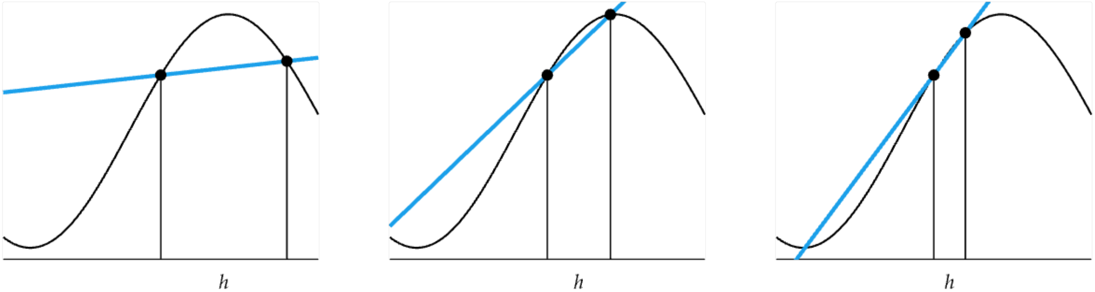
(Leibniz notation)



Derivatives

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

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



Symbolic Differentiation

$$f'(x) \equiv \underbrace{\lim_{h\to 0} \frac{f(x+h)-f(x)}{h}}_{\text{forward difference}} = \underbrace{\lim_{h\to 0} \frac{f(x+h/2)-f(x-h/2)}{h}}_{\text{central difference}} = \underbrace{\lim_{h\to 0} \frac{f(x)-f(x-h)}{h}}_{\text{backward difference}}$$

Symbolic Differentiation

```
import sympy as sp

# Define the variable
x = sp.symbols('x')

# Define the function
f = x**2 + x/2 - sp.sin(x)/x

# Compute the derivative
df_dx = sp.diff(f, x)

# Display the result
print("The symbolic derivative of f is:")
print(df_dx)
```

derivative.py

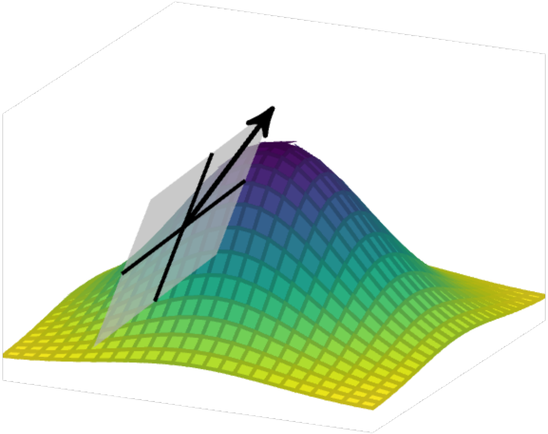
Derivatives in Multiple Dimensions

- Gradient Vector

$$\nabla f(\mathbf{x})^T = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

- Hessian Matrix

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \, \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \, \partial x_n} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \, \partial x_2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \, \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \, \partial x_n} \\ \vdots & \ddots & & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \, \partial x_n} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \, \partial x_n} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \, \partial x_n} \end{bmatrix}$$



Directional derivative

The **directional derivative** $\nabla_{\boldsymbol{s}} f(\boldsymbol{x})$ of a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$ is the instantaneous rate of change of $f(\boldsymbol{x})$ as $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$ is moved with velocity $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]$.

$$\nabla_{\mathbf{s}} f(\mathbf{x}) \equiv \underbrace{\lim_{h\to 0} \frac{f(\mathbf{x}+h\mathbf{s}) - f(\mathbf{x})}{h}}_{\text{forward difference}} = \underbrace{\lim_{h\to 0} \frac{f(\mathbf{x}+h\mathbf{s}/2) - f(\mathbf{x}-h\mathbf{s}/2)}{h}}_{\text{central difference}} = \underbrace{\lim_{h\to 0} \frac{f(\mathbf{x}) - f(\mathbf{x}-h\mathbf{s})}{h}}_{\text{backward difference}}$$

To compute $\nabla_{\boldsymbol{s}} f(\boldsymbol{x})$:

- compute $\boxed{\nabla_{\boldsymbol{s}} f(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial x_1} s_1 + \frac{\partial f(\boldsymbol{x})}{\partial x_2} s_2 + \ldots + \frac{\partial f(\boldsymbol{x})}{\partial x_n} s_n = \nabla f(\boldsymbol{x})^T \boldsymbol{s} = \nabla f(\boldsymbol{x}) \cdot \boldsymbol{s}}$
- $g(\alpha) := f(\boldsymbol{x} + \alpha \boldsymbol{s})$ and then compute $g'(0)$

We wish to compute the directional derivative of $f(\mathbf{x}) = x_1 x_2$ at $\mathbf{x} = [1, 0]$ in the direction $\mathbf{s} = [-1, -1]$:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}, & \frac{\partial f}{\partial x_2} \end{bmatrix} = [x_2, x_1]$$

$$\nabla_{\mathbf{s}} f(\mathbf{x}) = \nabla f(\mathbf{x})^\top \mathbf{s} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = -1$$

We can also compute the directional derivative as follows:

$$g(\alpha) = f(\mathbf{x} + \alpha \mathbf{s}) = (1 - \alpha)(-\alpha) = \alpha^2 - \alpha$$

$$g'(\alpha) = 2\alpha - 1$$

$$g'(0) = -1$$

Matrix Calculus

Common gradient:

$$\nabla_{\boldsymbol{x}} \boldsymbol{b}^T \boldsymbol{x} = ?$$

$$\boldsymbol{b}^T \boldsymbol{x} = [b_1 x_1 + b_2 x_2 + \ldots + b_n x_n]$$

$$\frac{\partial \boldsymbol{b}^T \boldsymbol{x}}{\partial x_i} = b_i$$

$$\nabla_{\boldsymbol{x}} \boldsymbol{b}^T \boldsymbol{x} = \nabla_{\boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{b} = \boldsymbol{b}$$

Matrix Calculus

Common gradient:

$$\nabla_{\boldsymbol{x}} \boldsymbol{x}^T A \boldsymbol{x} = ?$$

$$\boldsymbol{x}^T A \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}^T \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}^T \begin{bmatrix} x_1 a_{11} + x_2 a_{12} + \dots + x_n a_{1n} \\ x_1 a_{21} + x_2 a_{22} + \dots + x_n a_{2n} \\ \vdots \\ x_1 a_{n1} + x_2 a_{n2} + \dots + x_n a_{nn} \end{bmatrix}$$

$$= \begin{matrix} x_1^2 a_{11} + x_1 x_2 a_{12} + \dots + x_1 x_n a_{1n} + \\ x_1 x_2 a_{21} + x_2^2 a_{22} + \dots + x_2 x_n a_{2n} + \\ \vdots \\ x_1 x_n a_{n1} + x_2 x_n a_{n2} + \dots + x_n^2 a_{nn} \end{matrix}$$

$$\frac{\partial}{\partial x_i}\boldsymbol{x}^T A\boldsymbol{x} = \sum_{j=1}^{n} x_j\left(a_{ij} + a_{ji}\right)$$

$$\nabla_x \boldsymbol{x}^T A\boldsymbol{x} = \begin{bmatrix} \sum_{j=1}^{n} x_j\left(a_{1j} + a_{j1}\right) \\ \sum_{j=1}^{n} x_j\left(a_{2j} + a_{j2}\right) \\ \vdots \\ \sum_{j=1}^{n} x_j\left(a_{nj} + a_{jn}\right) \end{bmatrix} = \begin{bmatrix} a_{11} + a_{11} & a_{12} + a_{21} & \dots & a_{1n} + a_{n1} \\ a_{21} + a_{12} & a_{22} + a_{22} & \dots & a_{2n} + a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} + a_{1n} & a_{n2} + a_{2n} & \dots & a_{nn} + a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \left(A + A^T\right)\boldsymbol{x}$$

Smoothness

Def. The **smoothness** of a function is a property measured by the number of continuous derivatives (differentiability class) it has over its domain.

A function of **class** C^k is a function of smoothness at least k; that is, a function of class C^k is a function that has a kth derivative that is continuous in its domain.

The term **smooth function** refers to a C^∞-function. However, it may also mean “sufficiently differentiable” for the problem under consideration.

Smoothness

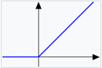
- Let U be an open set on the real line and a function f defined on U with real values. Let k be a non-negative integer.
- The function f is said to be of **differentiability class** C^k if the derivatives $f', f'', \ldots, f^{(k)}$ exist and are continuous on U.
- If f is k-differentiable on U, then it is at least in the class C^{k-1} since $f', f'', \ldots, f^{(k-1)}$ are continuous on U.
- The function f is said to be **infinitely differentiable**, **smooth**, or of **class** C^∞, if it has derivatives of all orders (continuous) on U.
- The function f is said to be of **class** C^ω, or **analytic**, if f is smooth and its Taylor series expansion around any point in its domain converges to the function in some neighborhood of the point.
- There exist functions that are smooth but not analytic; C^ω is thus strictly contained in C^∞. Bump functions are examples of functions with this property.

Example: continuous (C^0) but not differentiable [edit]

The function

$$f(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0 \end{cases}$$

is continuous, but not differentiable at $x = 0$, so it is of class C^0, but not of class C^1.



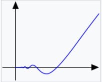
The C^0 function $f(x) = x$ for $x \geq 0$ and 0 otherwise.

Example: finitely-times differentiable (C^k) [edit]

For each even integer k, the function

$$f(x) = |x|^{k+1}$$

is continuous and k times differentiable at all x. At $x = 0$, however, f is not $(k + 1)$ times differentiable, so f is of class C^k, but not of class C^j where $j > k$.



The function $g(x) = x^2 \sin(1/x)$ for $x > 0$.

Example: differentiable but not continuously differentiable (not C^1) [edit]

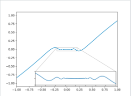
The function

$$g(x) = \begin{cases} x^2 \sin\left(\frac{1}{x}\right) & \text{if } x \neq 0, \\ 0 & \text{if } x = 0 \end{cases}$$

is differentiable, with derivative

$$g'(x) = \begin{cases} -\cos\left(\frac{1}{x}\right) + 2x\sin\left(\frac{1}{x}\right) & \text{if } x \neq 0, \\ 0 & \text{if } x = 0. \end{cases}$$

Because $\cos(1/x)$ oscillates as $x \to 0$, $g'(x)$ is not continuous at zero. Therefore, $g(x)$ is differentiable but not of class C^1.



The function $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = x^2 \sin\left(\frac{1}{x}\right)$ for $x \neq 0$ and $f(0) = 0$ is differentiable. However, this function is not continuously differentiable.

Example: differentiable but not Lipschitz continuous [edit]

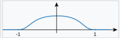
The function

$$h(x) = \begin{cases} x^{4/3} \sin\left(\frac{1}{x}\right) & \text{if } x \neq 0, \\ 0 & \text{if } x = 0 \end{cases}$$

is differentiable but its derivative is unbounded on a compact set. Therefore, h is an example of a function that is differentiable but not locally Lipschitz continuous.

Example: analytic (C^ω) [edit]

The exponential function e^x is analytic, and hence falls into the class C^ω (where ω is the smallest transfinite ordinal). The trigonometric functions are also analytic wherever they are defined, because they are linear combinations of complex exponential functions e^{ix} and e^{-ix}.



A smooth function that is not analytic.

Example: smooth (C^∞) but not analytic (C^ω) [edit]

The bump function

$$f(x) = \begin{cases} e^{-\frac{1}{1-x^2}} & \text{if } |x| < 1, \\ 0 & \text{otherwise} \end{cases}$$

is smooth, so of class C^∞, but it is not analytic at $x = \pm 1$, and hence is not of class C^ω. The function f is an example of a smooth function with compact support.

Positive Definteness

Def. A square, symmetric matrix A is **positive definite** if $\boldsymbol{x}^T A\boldsymbol{x}$ is positive for all points other than the origin: $\boldsymbol{x}^T A\boldsymbol{x} > 0$ for all $x \neq 0$.
Def. A square, symmetric matrix A is **positive semidefinite** if $\boldsymbol{x}^T A\boldsymbol{x}$ is always non-negative: $\boldsymbol{x}^T A\boldsymbol{x} \geq 0$ for all $\boldsymbol{x}$.

A matrix A is positive definite if and only if all its eigenvalues are positive. (This can be used for recognition.)

If the matrix A is positive definite in the function $f(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}$, then f has a unique global minimum.

Recall that the second order Taylor approximation of a twice-differentiable function f at $\boldsymbol{x}_0$ is

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^T(\boldsymbol{x} - \boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^T H_0(\boldsymbol{x} - \boldsymbol{x}_0)$$

where H_0 is the Hessian evaluated at $\boldsymbol{x}_0$. If $(\boldsymbol{x} - \boldsymbol{x}_0)^T H_0(\boldsymbol{x} - \boldsymbol{x}_0)$ has a unique global minimum, then the overall approximation has a unique global minimum.

Non-symmetric matrices can be positive definite:
Every real matrix A decomposes uniquely as a symmetric matrix and a skew-symmetric matrix:

$$A = S + K, \quad S = \frac{1}{2}(A + A^T), \quad K = \frac{1}{2}(A - A^T).$$

Then:

$$\boldsymbol{x}^T A x = \boldsymbol{x}^T S \boldsymbol{x},$$

because

$$\boldsymbol{x}^T K \boldsymbol{x} = 0 \text{ for all } \boldsymbol{x}.$$

However, all local optimization algorithms are based, explicitly or implicitly, on the second-order Taylor expansion. The matrix governing local curvature is the Hessian and Hessians are necessarily symmetric

LU Decomposition

The LU factorization of a matrix $A \in \mathbf{R}^{n \times n}$ is defined as

$$PA = LU, \tag{A.20}$$

where

P is an $n \times n$ permutation matrix (that is, it is obtained by rearranging the rows of the $n \times n$ identity matrix),

L is unit lower triangular (that is, lower triangular with diagonal elements equal to 1, and

U is upper triangular.

- This factorization can be used to solve efficiently a linear system of the form $Ax = \boldsymbol{b}$ by first solving $Ly = \boldsymbol{b}$ for y and then solving $Ux = y$ for x.
- It can be found by using Gaussian elimination with row partial pivoting, an algorithm that requires approximately $2n^3/3$ floating-point operations when A is dense. Done by standard software LAPACK.

Cholesky Decomposition

When $A \in \mathbf{R}^{n \times n}$ is symmetric positive definite, it is possible to compute a similar but more specialized factorization at about half the cost—about $n^3/3$ operations. This factorization, known as the Cholesky factorization, produces a matrix L such that

$$A = LL^T. \tag{A.23}$$

(If we require L to have positive diagonal elements, it is uniquely defined by this formula.)

- It can also be used to solve systems of linear equations
- The Cholesky factorization can also be used to verify positive definiteness of a symmetric matrix A.

Outline

1. Derivatives
2. Symbolic Differentiation
3. Numerical Differentiation
4. Automatic Differentiation

Symbolic Derivatives

- Symbolic derivatives can give valuable insight into the structure of the problem domain and, in some cases, produce analytical solutions of extrema (e.g., solving for $\frac{\mathrm{d}}{\mathrm{d}x}f(x) = 0$) that can eliminate the need for derivative calculation altogether.

- But they do not lend themselves to efficient runtime calculation of derivative values, as they can get exponentially larger than the expression whose derivative they represent

Outline

Numerical Differentiation

Finite Difference Method

- Neighboring points are used to approximate the derivative

$$f'(x) \approx \underbrace{\frac{f(x+h)-f(x)}{h}}_{\text{forward difference}} \approx \underbrace{\frac{f(x+h/2)-f(x-h/2)}{h}}_{\text{central difference}} \approx \underbrace{\frac{f(x)-f(x-h)}{h}}_{\text{backward difference}}$$

- h too small causes numerical cancellation errors (square root or cube root of the machine precision for floating point values: `sys.float_info.epsilon` difference between 1 and closest representable number)

Derivation

from Taylor series expansion:

$$f(x+h) = f(x) + \frac{f'(x)}{1!}h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \cdots$$

We can rearrange and solve for the first derivative:

$$f'(x)h = f(x+h) - f(x) - \frac{f''(x)}{2!}h^2 - \frac{f'''(x)}{3!}h^3 - \cdots$$

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(x)}{2!}h - \frac{f'''(x)}{3!}h^2 - \cdots$$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

- forward difference has error term $O(h)$, linear error as h approaches zero
- central difference has error term is $O(h^2)$

```python
import sys
import numpy as np

def diff_forward(f, x: float, h: float=np.sqrt(sys.float_info.epsilon)) -> float:
    return (f(x+h) - f(x))/h

def diff_central(f, x: float, h: float=np.cbrt(sys.float_info.epsilon)) -> float:
    return (f(x+h/2) - f(x-h/2))/h

def diff_backward(f, x: float, h: float=np.sqrt(sys.float_info.epsilon)) -> float:
    return (f(x) - f(x-h))/h

# Example usage
def func(x):
    return x**2 + np.sin(x)

x0 = 1.0
print(f"The derivative at x = {x0} is {diff_forward(func, x0)}")
```

finite_diff.py

Numerical Differentiation

Complex step method

Uses one single function evaluation after taking a step in the imaginary direction.

$$f(x+ih) = f(x) + ihf'(x) - h^2\frac{f''(x)}{2!} - ih^3\frac{f'''(x)}{3!} + \cdots$$

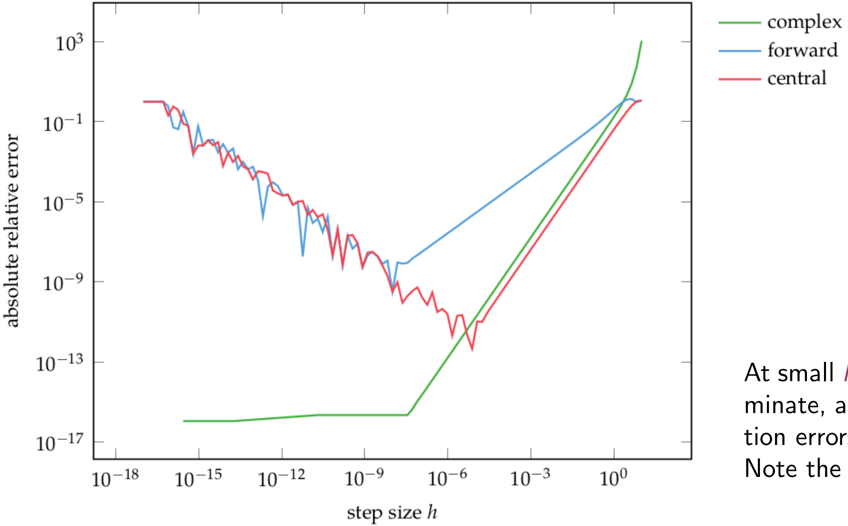
$$\begin{aligned}
\mathrm{Im}(f(x+ih)) &= hf'(x) - h^3\frac{f'''(x)}{3!} + \cdots \\
\Rightarrow f'(x) &= \frac{\mathrm{Im}(f(x+ih))}{h} + h^2\frac{f'''(x)}{3!} - \cdots \\
&= \frac{\mathrm{Im}(f(x+ih))}{h} + O(h^2) \text{ as } h \to 0
\end{aligned}$$

$$\begin{aligned}
\mathrm{Re}(f(x+ih)) &= f(x) - h^2\frac{f''(x)}{2!} + \ldots \\
\Rightarrow f(x) &= \mathrm{Re}(f(x+ih)) + h^2\frac{f''(x)}{2!} - \cdots
\end{aligned}$$

```python
import numpy as np

def diff_complex(f, x: float, h: float=1e-20) -> float:
    return np.imag(f(x + h * 1j)) / h

# Example usage
def func(x):
    return x**2 + np.sin(x)

x0 = 1.0
print(f"The derivative at x = {x0} is {diff_complex(func, x0)}")
```

complex_diff.py

Numerical Differentiation Error Comparison



At small h, round off errors dominate, and at large h, truncation errors dominate.
Note the log transformation.

Numerical Differentiation in ML

- Approximation errors would be tolerated in a deep learning setting thanks to the well-documented error resiliency of neural network architectures (Gupta et al., 2015).

- The $O(n)$ complexity of numerical differentiation for a gradient in n dimensions is the main obstacle to its usefulness in machine learning, where n can be as large as millions or billions in state-of-the-art deep learning models (Shazeer et al., 2017).

Outline

1. Derivatives
2. Symbolic Differentiation
3. Numerical Differentiation
4. Automatic Differentiation

Automatic Differentiation

Automatic differentiation techniques are founded on the observation that any function is evaluated by performing a sequence of simple elementary operations involving just one or two arguments at a time:

- addition
- multiplication
- division
- power operation a^b
- trigonometric functions
- exponential functions
- logarithmic
- chain rule:

Chain Rule

The chain rule is a formula that expresses the derivative of the composition of two differentiable functions. More precisely, if $h = z \circ y$ is the composition such that $h(x) = z(y(x))$ for every x, then the chain rule is:

in Lagrange's notation:

$$h'(x) = z'(y(x))y'(x).$$

in Leibniz's notation:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx},$$

and

$$\left.\frac{dz}{dx}\right|_{x} = \left.\frac{dz}{dy}\right|_{y(x)} \cdot \left.\frac{dy}{dx}\right|_{x},$$

for indicating at which points the derivatives have to be evaluated.

- Forward Accumulation is equivalent to expanding a function using the chain rule and computing the derivatives inside-out
- Requires n-passes to compute n-dimensional gradient
- Example:

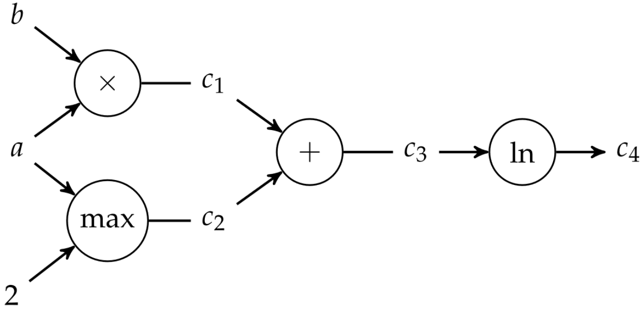
$$f(a, b) = \ln(ab + \max(a, 2))$$

$$\begin{aligned}
\frac{\partial f}{\partial a} &= \frac{\partial}{\partial a} \ln(ab + \max(a, 2)) \\
&= \frac{1}{ab + \max(a, 2)} \frac{\partial}{\partial a} (ab + \max(a, 2)) \\
&= \frac{1}{ab + \max(a, 2)} \left[\frac{\partial (ab)}{\partial a} + \frac{\partial \max(a, 2)}{\partial a} \right] \\
&= \frac{1}{ab + \max(a, 2)} \left[\left(b \frac{\partial a}{\partial a} + a \frac{\partial b}{\partial a} \right) + \left((2 > a) \frac{\partial 2}{\partial a} + (2 < a) \frac{\partial a}{\partial a} \right) \right] \\
&= \frac{1}{ab + \max(a, 2)} [b + (2 < a)]
\end{aligned}$$

Automatic Differentiation

Computational graph: nodes are are operations and the edges are input-output relations. leaf nodes of a computational graph are input variables or constants, and terminal nodes are values output by the function

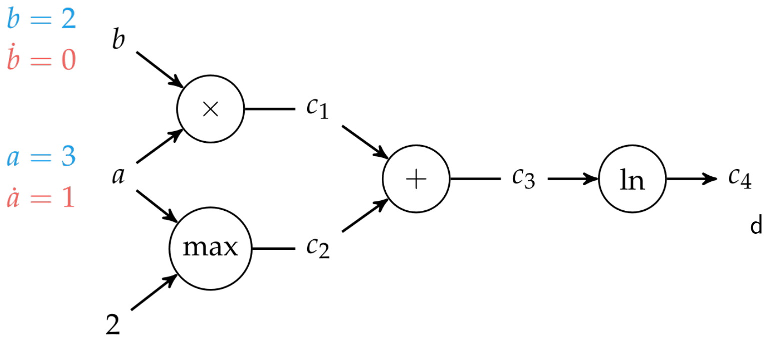
Forward accumulation for $f(a, b) = \ln(ab + \max(a, 2))$



Automatic Differentiation

Computational graph: nodes are are operations and the edges are input-output relations. leaf nodes of a computational graph are input variables or constants, and terminal nodes are values output by the function

Forward accumulation for $f(a, b) = \ln(ab + \max(a, 2))$



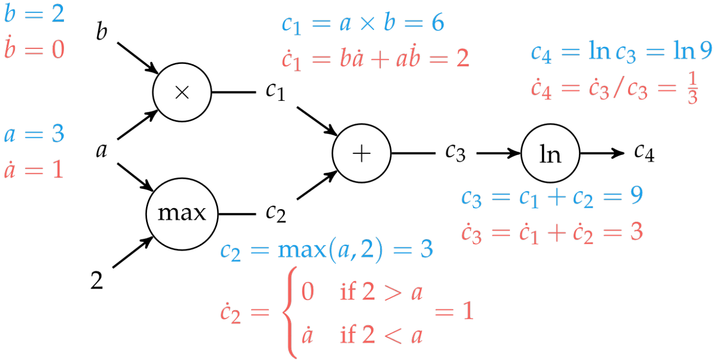
dual numbers

$\frac{\partial b}{\partial a} = \dot{b}$ Newton notation

Automatic Differentiation

Computational graph: nodes are are operations and the edges are input-output relations. leaf nodes of a computational graph are input variables or constants, and terminal nodes are values output by the function

Forward accumulation for $f(a,b) = \ln(ab + \max(a,2))$



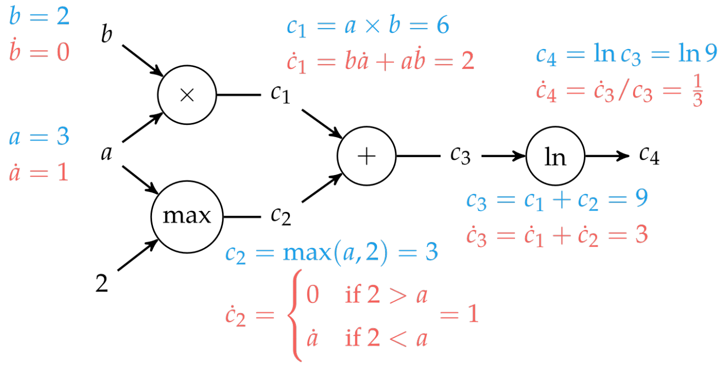
dual numbers

$\frac{\partial b}{\partial a} = \dot{b}$ Newton notation

Automatic Differentiation

Computational graph: nodes are are operations and the edges are input-output relations. leaf nodes of a computational graph are input variables or constants, and terminal nodes are values output by the function

Forward accumulation for $f(a, b) = \ln(ab + \max(a, 2))$



dual numbers

$\frac{\partial b}{\partial a} = \dot{b}$ Newton notation

for $\frac{\partial f}{\partial b}$ set $\dot{a} = 0, \dot{b} = 1$

Dual numbers

- Dual numbers can be expressed mathematically by including the abstract quantity ϵ, where ϵ^2 is defined to be 0.

- Like a complex number, a dual number is written $a + b\epsilon$ where a and b are both real values.

- $(a + b\epsilon) + (c + d\epsilon) = (a + c) + (b + d)\epsilon$
  $(a + b\epsilon) \times (c + d\epsilon) = (ac) + (ad + bc)\epsilon$

- by passing a dual number into any smooth function f, we get the evaluation and its derivative. We can show this using the Taylor series:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

$$f(a+b\epsilon) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(a+b\epsilon-a)^k$$

$$= \sum_{k=0}^{\infty} \frac{f^{(k)}(a)b^k\epsilon^k}{k!}$$

$$= f(a) + bf'(a)\epsilon + \epsilon^2 \sum_{k=2}^{\infty} \frac{f^{(k)}(a)b^k}{k!}\epsilon^{(k-2)}$$

$$= f(a) + bf'(a)\epsilon$$

Note that

$$
\begin{aligned}
(v + \dot{v}\epsilon) + (u + \dot{u}\epsilon) &= (v + u) + (\dot{v} + \dot{u})\epsilon \\
(v + \dot{v}\epsilon)(u + \dot{u}\epsilon) &= (vu) + (v\dot{u} + \dot{v}u)\epsilon \ ,
\end{aligned}
$$

satisfies the rules of differentiation

Setting:

$$
f(v + \dot{v}\epsilon) = f(v) + f'(v)\dot{v}\epsilon
$$

The chain rule follows:

$$
\begin{aligned}
f(g(v + \dot{v}\epsilon)) &= f(g(v) + g'(v)\dot{v}\epsilon) \\
&= f(g(v)) + f'(g(v))g'(v)\dot{v}\epsilon \ .
\end{aligned}
$$

Automatic Differentiation

- **Reverse accumulation** is performed in a single run using two passes $O(m \cdot \mathrm{ops}(f))$ (forward and back) for $f : \mathbb{R}^n \to \mathbb{R}^m$
- Note: this is central to the back-propagation algorithm used to train neural networks because it needs only one pass for the n-dimensional function to find the gradient.
- implemented through two different operation overloading functions (for forward and backward)
- Many open-source software implementations are available: eg, Tensorflow

Forward implements:

$$\frac{df}{dx} = \frac{df}{dc_4}\frac{dc_4}{dx} = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\frac{dc_3}{dx}\right) = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\left(\frac{dc_3}{dc_2}\frac{dc_2}{dx} + \frac{dc_3}{dc_1}\frac{dc_1}{dx}\right)\right)$$

Backward implements:

$$\frac{df}{dx} = \frac{df}{dc_4}\frac{dc_4}{dx} = \left(\frac{df}{dc_3}\frac{dc_3}{dc_4}\right)\frac{dc_4}{dx} = \left(\left(\frac{df}{dc_2}\frac{dc_2}{dc_3} + \frac{df}{dc_1}\frac{dc_1}{dc_3}\right)\frac{dc_3}{dc_4}\right)\frac{dc_4}{dx}$$

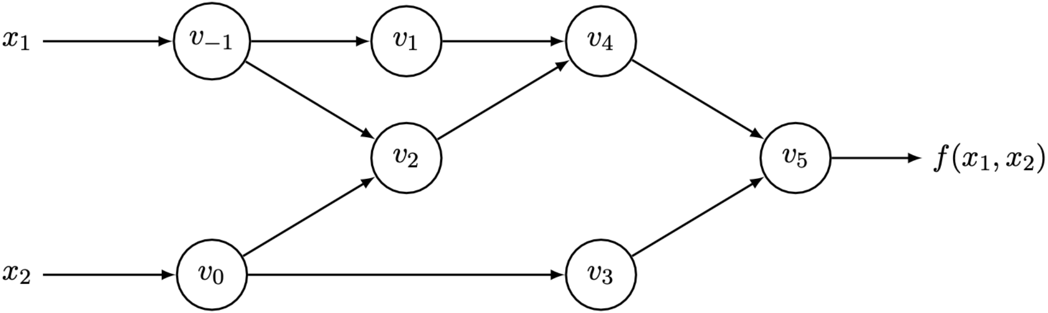
Complementing each intermediate variable v_i with an **adjoint**

$$\bar{v}_i = \frac{\partial y_j}{\partial v_i}$$

which represents the sensitivity of a considered output y_j with respect to changes in v_i.

Example

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Example: Forward Accumulation

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Forward Primal Trace		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

Forward Tangent (Derivative) Trace		
$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
$\dot{v}_0$	$= \dot{x}_2$	$= 0$
$\dot{v}_1$	$= \dot{v}_{-1}/v_{-1}$	$= 1/2$
$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
$\boldsymbol{\dot{y}}$	$\boldsymbol{= \dot{v}_5}$	$\mathbf{= 5.5}$

$O(n \cdot \mathrm{ops}(f))$

Example: Reverse Accumulation

Forward Primal Trace		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

Reverse Adjoint (Derivative) Trace			
$\boldsymbol{\bar{x}_1}$	$\boldsymbol{= \bar{v}_{-1}}$		$\mathbf{= 5.5}$
$\boldsymbol{\bar{x}_2}$	$\boldsymbol{= \bar{v}_0}$		$\mathbf{= 1.716}$
$\bar{v}_{-1}$	$= \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
$\bar{v}_0$	$= \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
$\bar{v}_{-1}$	$= \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$
$\bar{v}_0$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$
$\bar{v}_2$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_1$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_3$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$
$\bar{v}_4$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$	$= 1$
$\bar{v}_5$	$= \bar{y}$	$= 1$	

$O(m \cdot \mathrm{ops}(f))$

Summary

- Derivatives are useful in optimization because they provide information about how to change a given point in order to improve the objective function
- For multivariate functions, various derivative-based concepts are useful for directing the search for an optimum, including the gradient, the Hessian, and the directional derivative
- computation of derivatives in computer programs can be classified into four categories:
  1. manually working out derivatives and coding them (error prone and time consuming)
  2. numerical differentiation using finite difference approximations
     Complex step method can eliminate the effect of subtractive cancellation error when taking small steps
  3. symbolic differentiation using expression manipulation in computer algebra systems
  4. automatic differentiation, (aka algorithmic differentiation)
     forward and reverse accumulation on computational graphs