$$\sum_{i=1}^{n} a_i^T S^{-1} a_i = \sum_{i=1}^{n} m_{ii},$$

where m_{ii} - i-th diagonal element of some matrix M.
5. Note, that $M = A^T \left(S^{-1} A\right)$ is the product of 2 matrices, because i-th diagonal element of M is the scalar product of i-th row of the first matrix A^T and i-th column of the second matrix $S^{-1}A$. i-th row of matrix A^T, by definition, is a_i^T, while i-th column of the matrix $S^{-1}A$ is clearly $S^{>}-1a_i$.
Indeed, $m_{ii} = a_i^T S^{-1} a_i$, then we can finish the exercise:

$$\sum_{i=1}^{n} a_i^T S^{-1} a_i = \sum_{i=1}^{n} m_{ii} = \mathrm{tr} M = \mathrm{tr}\left(A^T S^{-1} A\right) = \mathrm{tr}\left(A A^T S^{-1}\right) = >tr\left(SS\right.$$

Eigenvalues, eigenvectors, and the singular-value decomposition

Eigenvalues

A scalar value λ is an eigenvalue of the $n \times n$ matrix A if there is a nonzero vector q such that

$$Aq = \lambda q.$$

EXAMPLE

Consider a 2x2 matrix: $A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$ The eigenvalues of this matrix can be found by solving the characteristic equation: $\det(A - \lambda I) = 0$ For this matrix, the eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = 4$. These eigenvalues tell us about the scaling factors of the matrix along its principal axes.

The vector q is called an eigenvector of A. The matrix A is nonsingular if none of its eigenvalues are zero. The eigenvalues of symmetric matrices are all real numbers, while nonsymmetric matrices may have imaginary eigenvalues. If the matrix is positive definite as well as symmetric, its eigenvalues are all positive real numbers.

THEOREM

$$A \succeq 0 \Leftrightarrow \text{all eigenvalues of } A \text{ are } \geq 0$$

$$A \succ 0 \Leftrightarrow \text{all eigenvalues of } A \text{ are } > 0$$

▼ Proof

We will just prove the first point here. The second one can be proved analogously.

1.$\rightarrow$ Suppose some eigenvalue λ is negative and let x denote its corresponding eigenvector. Then

$$Ax = \lambda x \rightarrow x^T Ax = \lambda x^T x < 0 \rightarrow A \not\succeq 0.$$

2. $\leftarrow$ For any symmetric matrix, we can pick a set of eigenvectors $v_1, \ldots, v_n$ that form an orthogonal basis of $\mathbb{R}^n$. Pick any $x > in\mathbb{R}^n$.

$$x^T Ax = (\alpha_1 v_1 + \cdots + \alpha_n v_n)^T A(\alpha_1 v_1 + \cdots + \alpha_n v_n)$$

$= \sum \alpha_i^2 v_i^T Av_i = \sum \alpha_i^2 \lambda_i v_i^T v_i \geq 0$
here we have used the fact that $v_i^T v_j = 0$, for $i \neq j$.

🤔 **QUESTION**

If a matrix has all positive eigenvalues, what can we infer about the matrix's definiteness?

Suppose $A \in S_n$, i.e., A is a real symmetric $n \times n$ matrix. Then A can be factorized as

$$A = Q\Lambda Q^T$$

where $Q \in \mathbb{R}^{n \times n}$ is orthogonal, i.e., satisfies $Q^T Q = I$, and $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$. The (real) numbers λ_i are the eigenvalues of A, and are the roots of the characteristic polynomial $\det(A - \lambda I)$. The columns of Q form an orthonormal set of eigenvectors of A. The factorization is called the spectral decomposition or (symmetric) eigenvalue decomposition of A. [2]

We usually order the eigenvalues as $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. We use the notation $\lambda_i(A)$ to refer to the ith largest eigenvalue of $A \in S$. We usually write the largest or maximum eigenvalue as $\lambda_1(A) = \lambda_{\max}(A)$, and the least or minimum eigenvalue as $\lambda_n(A) = \lambda_{\min}(A)$.

The largest and smallest eigenvalues satisfy

$$\lambda_{\min}(A) = \inf_{x \neq 0} \frac{x^T Ax}{x^T x}, \qquad \lambda_{\max}(A) = \sup_{x \neq 0} \frac{x^T Ax}{x^T x}$$

and consequently $\forall x \in \mathbb{R}^n$ (Rayleigh quotient):

$$\lambda_{\min}(A)x^Tx \leq x^TAx \leq \lambda_{\max}(A)x^Tx$$

The **condition number** of a nonsingular matrix is defined as

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

Suppose $A \in \mathbb{R}^{m\times n}$ with rank $A = r$. Then A can be factored as

$$A = U\Sigma V^T, \quad (A.12)$$

where $U \in \mathbb{R}^{m\times r}$ satisfies $U^TU = I$, $V \in \mathbb{R}^{n\times r}$ satisfies $V^TV = I$, and Σ is a diagonal matrix with $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r)$, such that

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0.$$

Singular value decomposition

This factorization is called the **singular value decomposition (SVD)** of A. The columns of U are called left singular vectors of A, the columns of V are right singular vectors, and the numbers σ_i are the singular values. The singular value decomposition can be written as

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

where $u_i \in \mathbb{R}^m$ are the left singular vectors, and $v_i \in \mathbb{R}^n$ are the right singular vectors.

EXAMPLE

Consider a 2x2 matrix: $B = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$ The singular value decomposition of this matrix can be represented as: $B = U\Sigma V^T$ Where U and V are orthogonal matrices and Σ is a diagonal matrix with the singular values on its diagonal. For this matrix, the singular values are 4 and 2, which are also the eigenvalues of the matrix.

EXAMPLE

Let $A \in \mathbb{R}^{m\times n}$, and let $q := \min m, n$. Show that

$$\|A\|_F^2 = \sum_{i=1}^{q} \sigma_i^2(A),$$

where $\sigma_1(A) \geq \ldots \geq \sigma_q(A) \geq 0$ are the singular values of matrix A. Hint: use

the connection betwwen Frobenius norm and scalar product and SVD.

▼ Solution

🤔 **QUESTION**

Suppose, matrix $A \in \mathbb{S}^n_{++}$. What can we say about the connection between its eigenvalues and singular values?

🤔 **QUESTION**

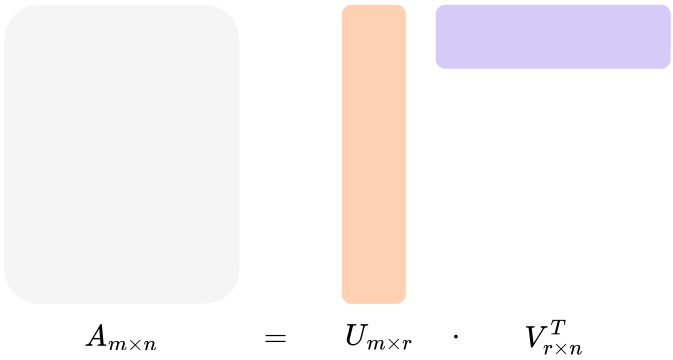
How do the singular values of a matrix relate to its eigenvalues, especially for a symmetric matrix?

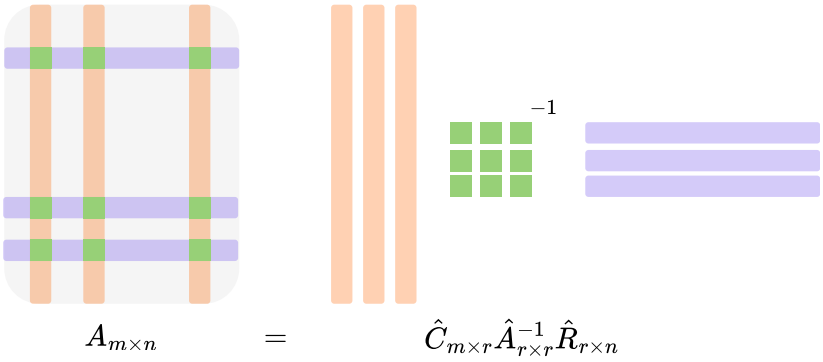
Skeleton decomposition

Simple, yet very interesting decomposition is Skeleton decomposition, which can be written in two forms:

$$A = UV^T \quad A = \hat{C}\hat{A}^{-1}\hat{R}$$

The latter expression refers to the fun fact: you can randomly choose r linearly independent columns of a matrix and any r linearly independent rows of a matrix and store only them with an ability to reconstruct the whole matrix exactly.





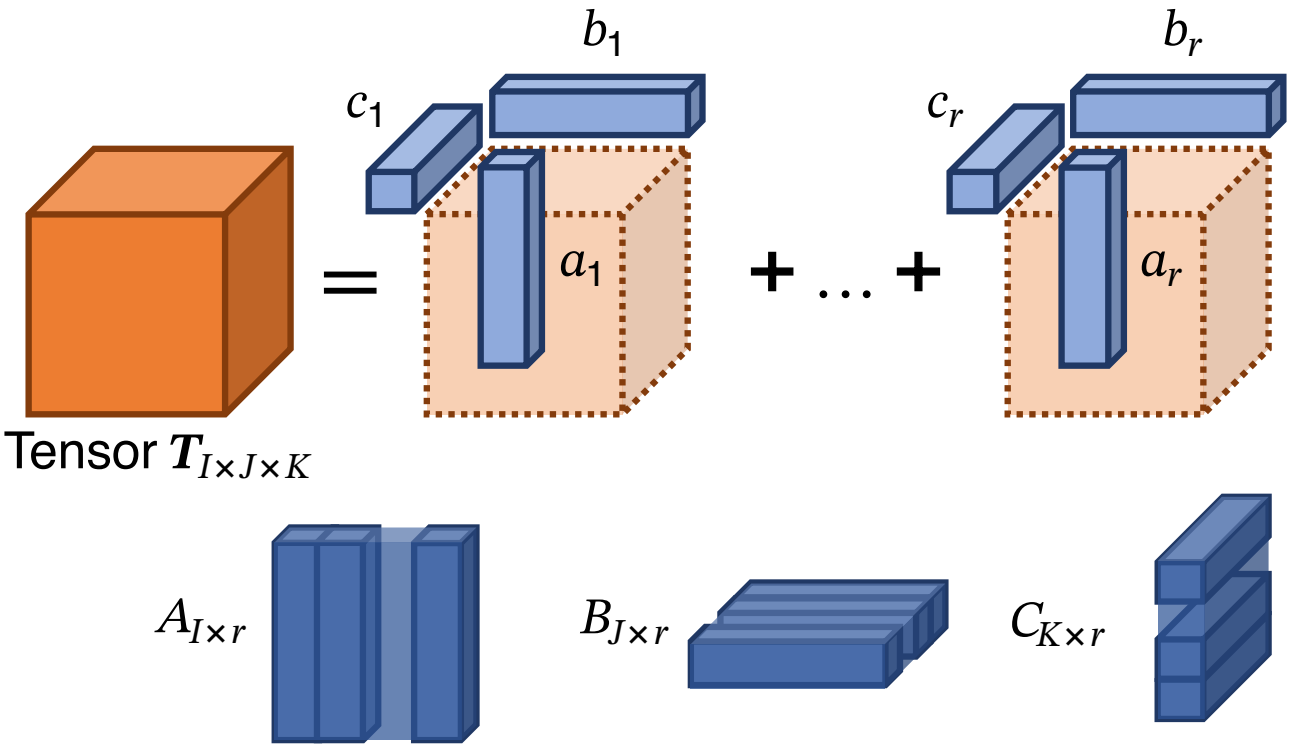
> 🤔 **QUESTION**
> How does the choice of columns and rows in the Skeleton decomposition affect the accuracy of the matrix reconstruction?

Use cases for Skeleton decomposition are:

- Model reduction, data compression, and speedup of computations in numerical analysis: given rank-r matrix with $r \ll n, m$ one needs to store $\mathcal{O}((n+m)r) \ll nm$ elements.
- Feature extraction in machine learning, where it is also known as matrix factorization
- All applications where SVD applies, since Skeleton decomposition can be transformed into truncated SVD form.

Canonical tensor decomposition

One can consider the generalization of Skeleton decomposition to the higher order data structure, like tensors, which implies representing the tensor as sum of r primitive tensors.



EXAMPLE

Note, that there are many tensor decompositions: Canonical, Tucker, Tensor Train (TT), Tensor Ring (TR) and others. In tensor case we do not have the straightforward definition of *rank* for all type of decompositions. For example, for TT decomposition rank is not a scalar, but a vector.

QUESTION

How does the choice of rank in the Canonical tensor decomposition affect the accuracy and interpretability of the decomposed tensor?

Determinant and trace

The determinant and trace can be expressed in terms of the eigenvalues

$$\det A = \prod_{i=1}^{n} \lambda_i, \qquad \operatorname{tr} A = \sum_{i=1}^{n} \lambda_i$$

The determinant has several appealing (and revealing) properties. For instance,

- $\det A = 0$ if and only if A is singular;
- $\det AB = (\det A)(\det B)$;
- $\det A^{-1} = \frac{1}{\det A}$.

Don't forget about the cyclic property of a trace for a square matrices A, B, C, D:

$$\text{tr}(ABCD) = \text{tr}(DABC) = \text{tr}(CDAB) = \text{tr}(BCDA)$$

EXAMPLE

For the matrix:

$$C = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

The determinant is $\det(C) = 6 - 1 = 5$, and the trace is $\text{tr}(C) = 2 + 3 = 5$. The determinant gives us a measure of the volume scaling factor of the matrix, while the trace provides the sum of the eigenvalues.

QUESTION

How does the determinant of a matrix relate to its invertibility?

QUESTION

What can you say about the determinant value of a positive definite matrix?

Optimization bingo

Gradient

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

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

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.

EXAMPLE

For the function $f(x, y) = x^2 + y^2$, the gradient is:

$$\nabla f(x, y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

This gradient points in the direction of steepest ascent of the function.

🤔 **QUESTION**

How does the magnitude of the gradient relate to the steepness of the function?

Hessian

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

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

In fact, 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 $(H(f_1(x)), H(f_2(x)), \ldots, H(f_m(x)))$.

🤦 **EXAMPLE**

For the function $f(x, y) = x^2 + y^2$, the Hessian is:

$$H_f(x, y) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

This matrix provides information about the curvature of the function in different directions.

🤔 **QUESTION**

How can the Hessian matrix be used to determine the concavity or convexity of a function?

Jacobian

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

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

👷 EXAMPLE

For the function

$$f(x, y) = \begin{bmatrix} x + y \\ x - y \end{bmatrix},$$

the Jacobian is:

$$J_f(x, y) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This matrix provides information about the rate of change of the function with respect to its inputs.

🤔 QUESTION

How does the Jacobian matrix relate to the gradient for scalar-valued functions?

🤔 QUESTION

Can we somehow connect those three definitions above (gradient, jacobian and hessian) using a single correct statement?

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)
$\mathbb{R}^n$	$\mathbb{R}$	$\mathbb{R}^n$	$\frac{\partial f}{\partial x_i}$ (gradient)
$\mathbb{R}^n$	$\mathbb{R}^m$	$\mathbb{R}^{m \times n}$	$\frac{\partial f_i}{\partial x_j}$ (jacobian)

$\mathbb{R}^{m\times n}$	$\mathbb{R}$	$\mathbb{R}^{n\times m}$	$\frac{\partial f}{\partial x_{ij}}$

Taylor approximations

Taylor approximations provide a way to approximate functions locally by polynomials. The idea is that for a smooth function, we can approximate it by its tangent (for first order) or by its parabola (for the second order) at a point.

First order Taylor approximation

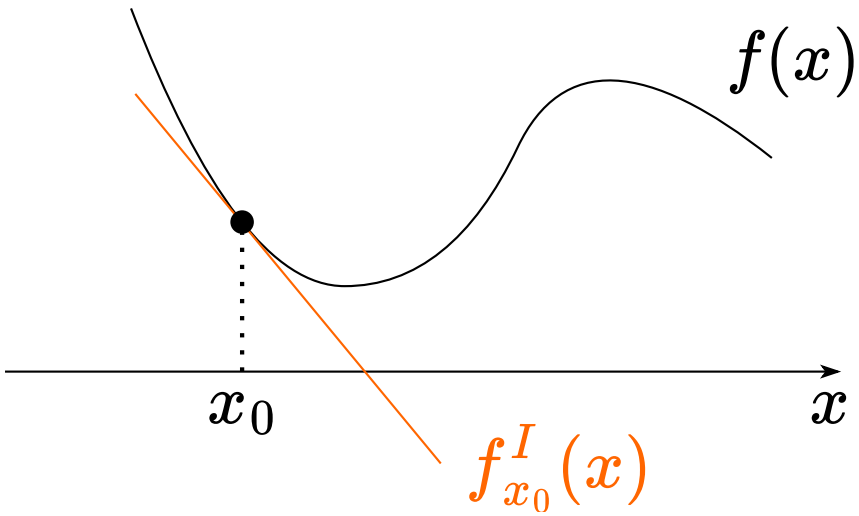
The first order Taylor approximation, also known as the linear approximation, is centered around some point x_0. If $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable function, then its first order Taylor approximation is given by:

$$f^I_{x_0}(x) = f(x_0) + \nabla f(x_0)^T (x - x_0)$$

Where:

- $f(x_0)$ is the value of the function at the point x_0.
- $\nabla f(x_0)$ is the gradient of the function at the point x_0.

It is very usual to replace the $f(x)$ with $f^I_{x_0}(x)$ near the point x_0 for simple analysis of some approaches.



EXAMPLE

For the function $f(x) = e^x$ around the point $x_0 = 0$, the first order Taylor approximation is: $f^I_{x_0}(x) = 1 + x$ And the second order Taylor approximation is: $f^{II}_{x_0}(x) = 1 + x + \frac{x^2}{2}$ These approximations provide polynomial representations of

the function near the point x_0.

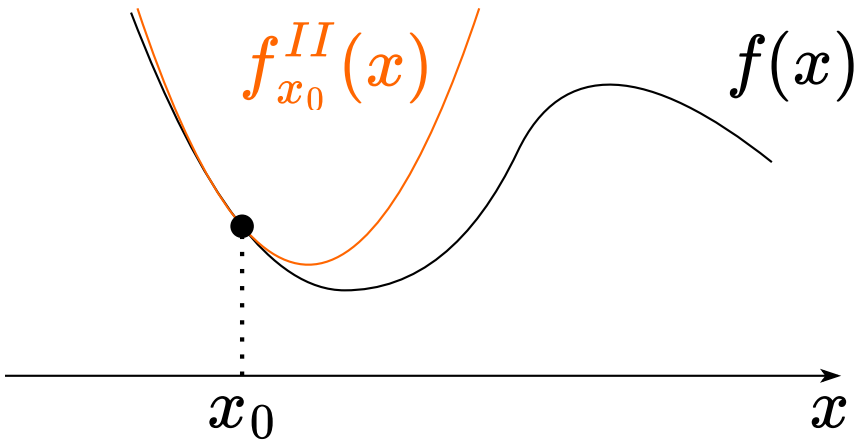
Second order Taylor approximation

The second order Taylor approximation, also known as the quadratic approximation, includes the curvature of the function. For a twice-differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, its second order Taylor approximation centered at some point x_0 is:

$$f_{x_0}^{II}(x) = f(x_0) + \nabla f(x_0)^T (x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

Where:

- $\nabla^2 f(x_0)$ is the Hessian matrix of f at the point x_0.



When using the linear approximation of the function not sufficient one can consider replacing the $f(x)$ with $f_{x_0}^{II}(x)$ near the point x_0. In general, Taylor approximations give us a way to locally approximate functions. The first order approximation is a plane tangent to the function at the point x_0, while the second order approximation includes the curvature and is represented by a parabola. These approximations are especially useful in optimization and numerical methods because they provide a tractable way to work with complex functions.

EXAMPLE

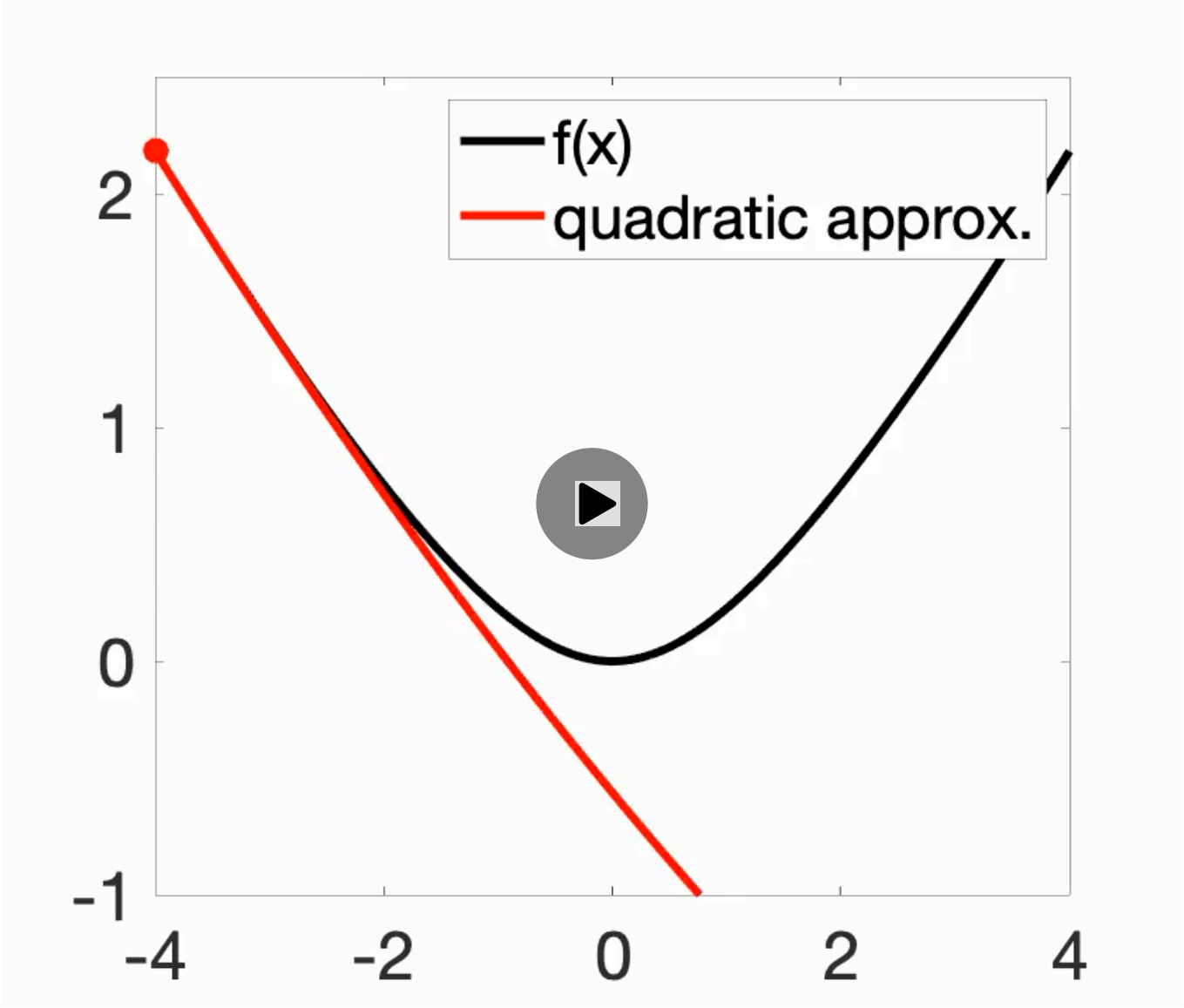
Calculate first and second order Taylor approximation of the function $f(x) = \frac{1}{2}x^T Ax - b^T x + c$

▼ Solution

🤔 **QUESTION**

Why might one choose to use a Taylor approximation instead of the original function in certain applications?

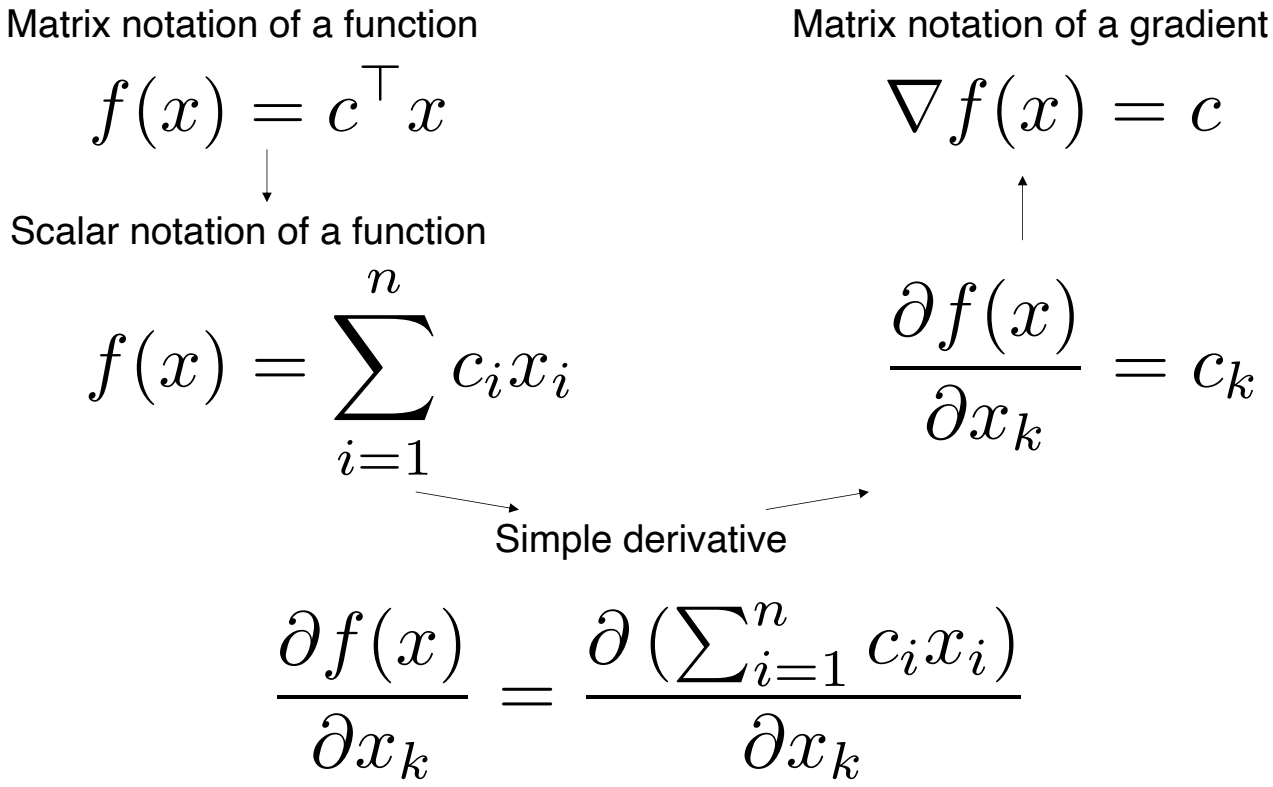
Note, that even the second order approximation could become inaccurate very quickly:



Derivatives

Naive approach

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



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

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

Differential 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. [3]

🤓 **THEOREM**

Let $x \in S$ be an interior point of the set S, and let $D : U \to V$ be a linear operator. We say that the function f is differentiable at the point x with derivative D if for all sufficiently small $h \in U$ the following decomposition holds:

$$f(x + h) = f(x) + D[h] + o(\|h\|)$$

If for any linear operator $D : U \to V$ the function f is not differentiable at the point x with derivative D, then we say that f is not differentiable at the point x.

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 $d(df) = d^2 f(x)$

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

Properties

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

- $dA = 0$
- $d(\alpha X) = \alpha(dX)$
- $d(AXB) = A(dX)B$
- $d(X + Y) = dX + dY$
- $d(X^T) = (dX)^T$
- $d(XY) = (dX)Y + X(dY)$
- $d\langle X, Y \rangle = \langle dX, Y \rangle + \langle X, dY \rangle$
- $d\left(\frac{X}{\phi}\right) = \frac{\phi dX - (d\phi)X}{\phi^2}$
- $d\left(\det X\right) = \det X \langle X^{-\top}, dX \rangle$
- $d\left(\operatorname{tr} X\right) = \langle I, dX \rangle$
- $df(g(x)) = \frac{df}{dg} \cdot dg(x)$
- $H = (J(\nabla f))^T$
- $d(X^{-1}) = -X^{-1}(dX)X^{-1}$

> 👷 **EXAMPLE**
>
> Find $\nabla^2 f(x)$, if $f(x) = \frac{1}{2}\langle Ax, x \rangle - \langle b, x \rangle + c$.
>
> ▼ Solution

EXAMPLE

Find $df, \nabla f(x)$, if $f(x) = \ln\langle x, Ax\rangle$.

▼ Solution

1. It is essential for A to be positive definite, because it is a logarithm argument. So, $A \in \mathbb{S}^n_{++}$Let's find the differential first:

$$df = d\left(\ln\langle x, Ax\rangle\right) = \frac{d\left(\langle x, Ax\rangle\right)}{\langle x, Ax\rangle} = \frac{\langle dx, Ax\rangle + \langle x, d(Ax)\rangle}{\langle x, Ax\rangle} =$$
$$= \frac{\langle Ax, dx\rangle + \langle x, Adx\rangle}{\langle x, Ax\rangle} = \frac{\langle Ax, dx\rangle + \langle A^T x, dx\rangle}{\langle x, Ax\rangle} = \frac{\langle (A + A^T)x, dx\rangle}{\langle x, Ax\rangle}$$

2. Note, that our main goal is to derive the form $df = \langle \cdot, dx\rangle$

$$df = \left\langle \frac{2Ax}{\langle x, Ax\rangle}, dx \right\rangle$$

Hence, the gradient is $\nabla f(x) = \dfrac{2Ax}{\langle x, Ax\rangle}$

EXAMPLE

Find $df, \nabla f(X)$, if $f(X) = \|AX - B\|_F$.

▼ Solution

EXAMPLE

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

▼ Solution

EXAMPLE

Find the gradient $\nabla f(x)$ and hessian $\nabla^2 f(x)$, if $f(x) = \ln\left(1 + \exp\langle a, x\rangle\right)$

▼ Solution

References

- Convex Optimization book by S. Boyd and L. Vandenberghe - Appendix A. Mathematical background.
- Numerical Optimization by J. Nocedal and S. J. Wright. - Background Material.
- Matrix decompositions Cheat Sheet.
- Good introduction
- The Matrix Cookbook
- MSU seminars (Rus.)
- Online tool for analytic expression of a derivative.

- Determinant derivative
- Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares - book by Stephen Boyd & Lieven Vandenberghe.
- Numerical Linear Algebra course at Skoltech

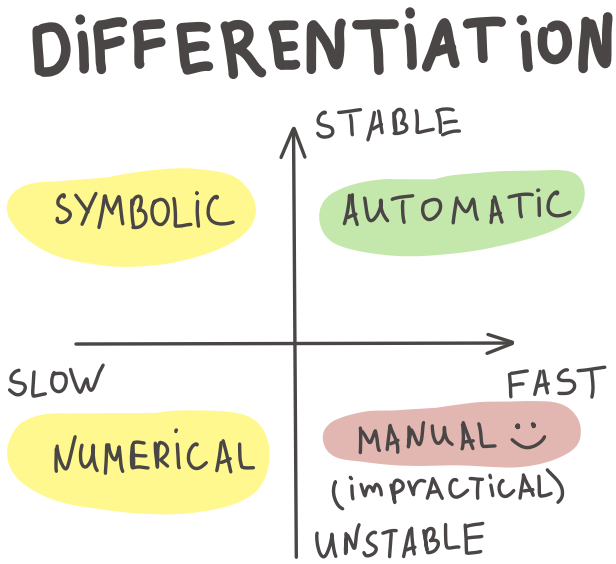
Footnotes

1 Full introduction to applied linear algebra could be found in Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares - book by Stephen Boyd & Lieven Vandenberghe, which is indicated in the source. Also, useful refresher for linear algebra is in the appendix A of book Numerical Optimization by Jorge Nocedal Stephen J. Wright. ↩

2 Good cheatsheet with matrix decomposition is available at NLA course website. ↩

3 The most comprehensive and intuitive guide about the theory of taking matrix derivatives is presented in these notes by Dmitry Kropotov team. ↩

Idea



Automatic differentiation is a scheme, that allows 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 $f : \mathbb{R}^n \to \mathbb{R}$:

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

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

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

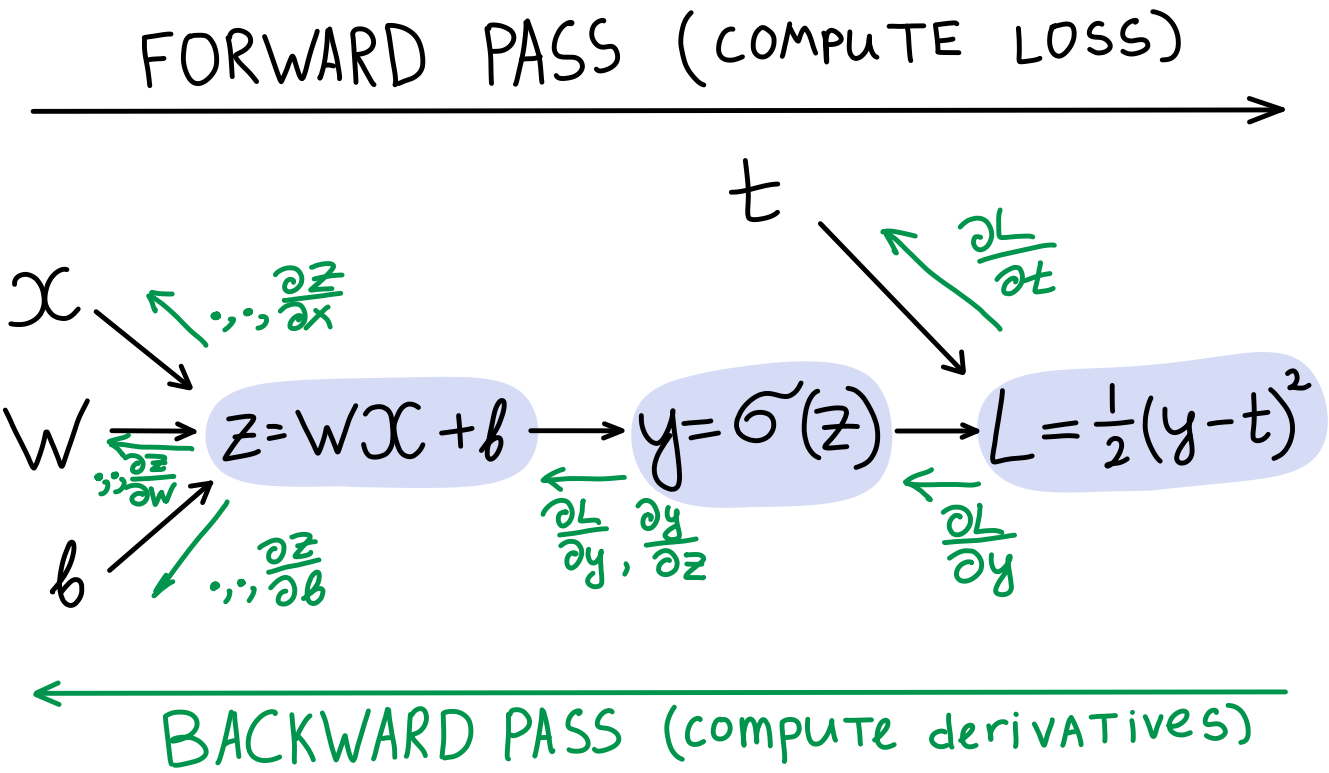
where matrix $J \in \mathbb{R}^{m \times n}$ is the jacobian of the f. Hence, we could write it in a vector way:

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

Backpropagation

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

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



$$z = wx + b \qquad \frac{\partial z}{\partial w} = x, \frac{\partial z}{\partial x} = w, \frac{\partial z}{\partial b} = 0$$

$$y = \sigma(z) \qquad \frac{\partial y}{\partial z} = \sigma'(z)$$

$$L = \frac{1}{2}(y - t)^2 \qquad \frac{\partial L}{\partial y} = y - t, \frac{\partial L}{\partial t} = t - y$$

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 $v_1, ..., v_N$ 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:

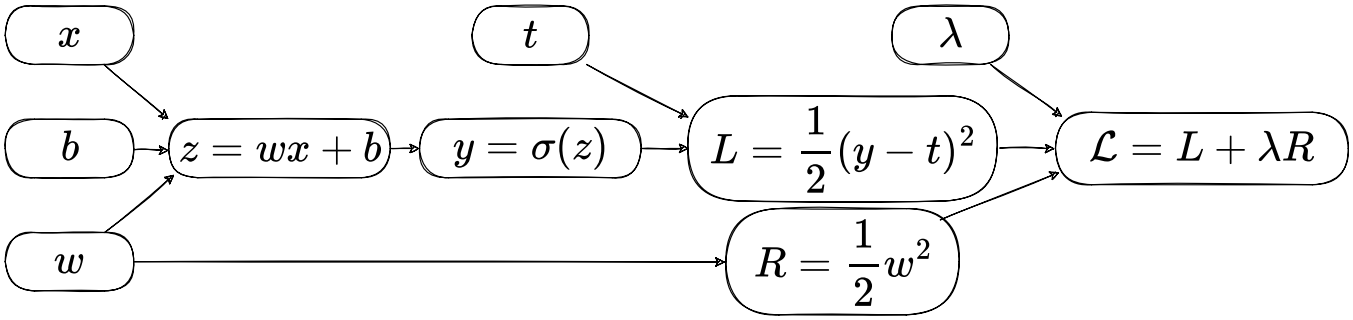
- For $i = 1, \ldots, N$:
  - Compute v_i as a function of its parents.

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.

Univariate logistic least squares regression



Forward pass

$$
\begin{aligned}
z &= wx + b \\
y &= \sigma(z) \\
L &= \frac{1}{2}(y - t)^2 \\
R &= \frac{1}{2}w^2 \\
\mathcal{L} &= L + \lambda R
\end{aligned}
$$

Backward pass

$$
\begin{aligned}
\overline{\mathcal{L}} &= 1 \\
\overline{R} &= \overline{\mathcal{L}}\frac{d\mathcal{L}}{dR} = \overline{\mathcal{L}}\lambda \\
\overline{L} &= \overline{\mathcal{L}}\frac{d\mathcal{L}}{dL} = \overline{\mathcal{L}} \\
\overline{y} &= \overline{L}\frac{dL}{dy} = \overline{L}(y - t)
\end{aligned}
\qquad
\begin{aligned}
\overline{z} &= \overline{y}\frac{dy}{dz} = \overline{y}\sigma'(z) \\
\overline{w} &= \overline{z}\frac{dz}{dw} + \overline{R}\frac{dR}{dw} = \overline{z}x + \overline{R}w \\
\overline{b} &= \overline{z}\frac{dz}{db} = \overline{z} \\
\overline{x} &= \overline{z}\frac{dz}{dx} = \overline{z}w
\end{aligned}
$$

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

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

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 vector-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

Open In Colab

Materials

- Autodidact - a pedagogical implementation of Autograd
- CSC321 Lecture 6

- CSC321 Lecture 10
- Why you should understand backpropagation :)
- JAX autodiff cookbook
- Materials from CS207: Systems Development for Computational Science course with very intuitive explanation.
