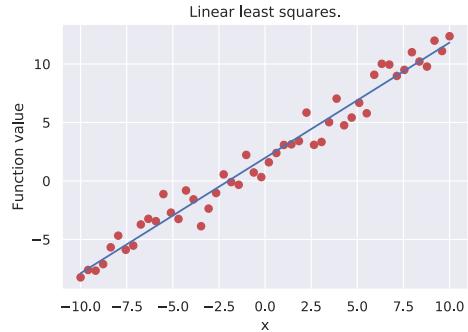
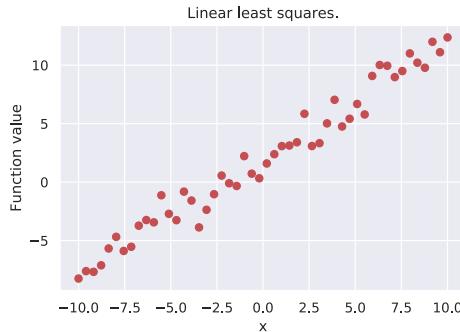


Problem



In a least-squares, or linear regression, problem, we have measurements $X \in \mathbb{R}^{m \times n}$ and $y \in \mathbb{R}^m$ and seek a vector $\theta \in \mathbb{R}^n$ such that $X\theta$ is close to y . Closeness is defined as the sum of the squared differences:

$$\sum_{i=1}^m (x_i^\top \theta - y_i)^2$$

also known as the l_2 -norm squared, $\|X\theta - y\|_2^2$

For example, we might have a dataset of m users, each represented by n features. Each row x_i^\top of X is the features for user i , while the corresponding entry y_i of y is the measurement we want to predict from x_i^\top , such as ad spending. The prediction is given by $x_i^\top \theta$.

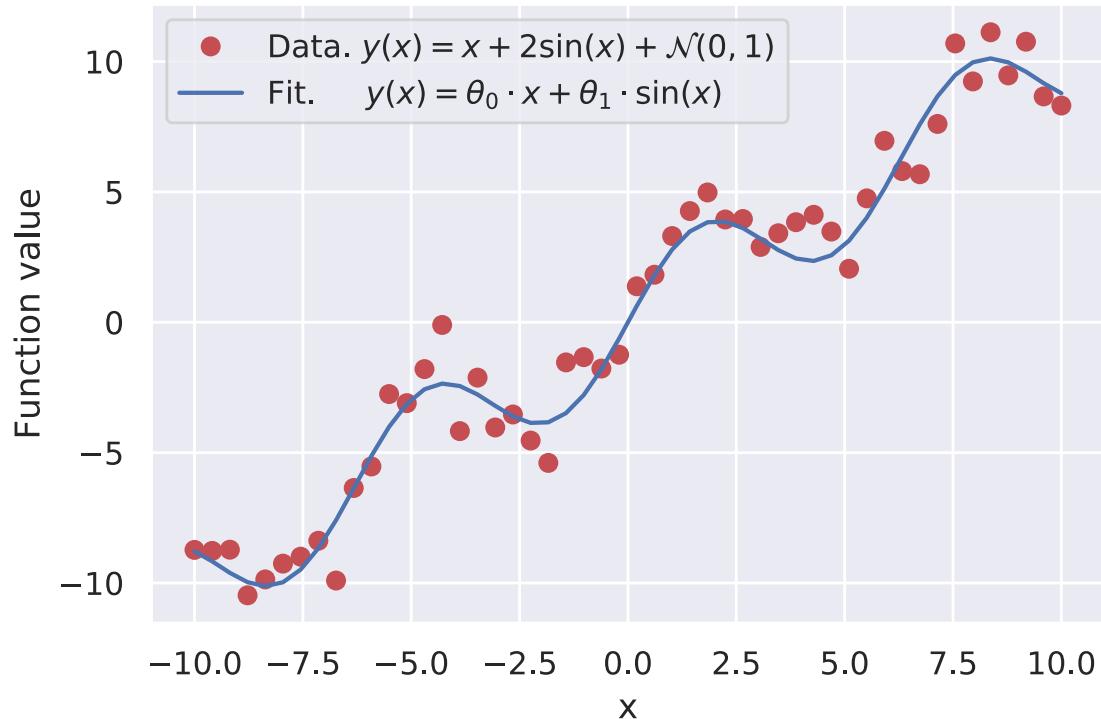
We find the optimal θ by solving the optimization problem

$$\|X\theta - y\|_2^2 \rightarrow \min_{\theta \in \mathbb{R}^n}$$

Let θ^* denote the optimal θ . The quantity $r = X\theta^* - y$ is known as the residual. If $\|r\|_2 = 0$, we have a perfect fit.

Note, that the function needn't be linear in the argument x but only in the parameters θ that are to be determined in the best fit.

Linear least squares. $\theta_0 = 0.992, \theta_1 = 2.09$



Approaches

Moore–Penrose inverse

If the matrix X is relatively small, we can write down and calculate exact solution:

$$\theta^* = (X^\top X)^{-1} X^\top y = X^\dagger y,$$

where X^\dagger is called **pseudo-inverse** matrix. However, this approach squares the condition number of the problem, which could be an obstacle in case of ill-conditioned huge scale problem.

QR decomposition

For any matrix $X \in \mathbb{R}^{m \times n}$ there exists QR decomposition:

$$X = Q \cdot R,$$

where Q is an orthogonal matrix (its columns are orthogonal unit vectors meaning $Q^\top Q = QQ^\top = I$) and R is an upper triangular matrix. It is important to notice, that since $Q^{-1} = Q^\top$, we have:

$$QR\theta = y \quad \longrightarrow \quad R\theta = Q^\top y$$

Now, process of finding theta consists of two steps:

- 1 Find the QR decomposition of X .
- 2 Solve triangular system $R\theta = Q^\top y$, which is triangular and, therefore, easy to solve.

Cholesky decomposition

For any positive definite matrix $A \in \mathbb{R}^{n \times n}$ there exists Cholesky decomposition:

$$X^\top X = A = L^\top \cdot L,$$

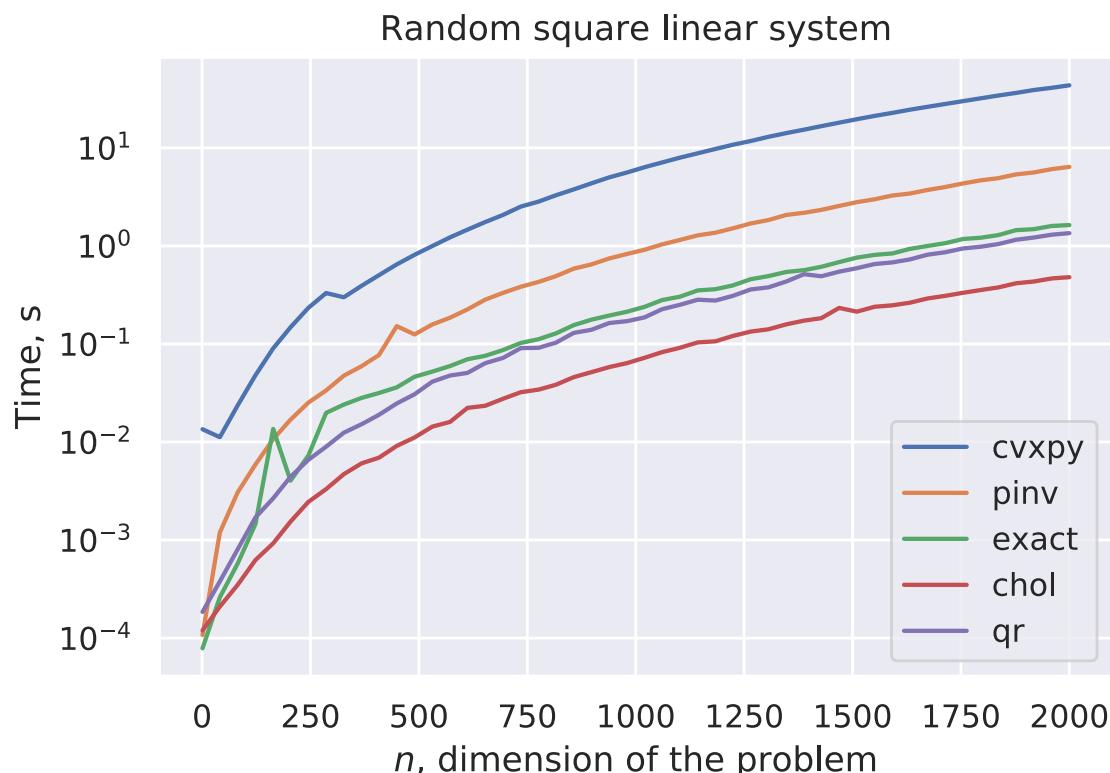
where L is an lower triangular matrix. We have:

$$L^\top L\theta = y \quad \longrightarrow \quad L^\top z_\theta = y$$

Now, process of finding theta consists of two steps:

- 1 Find the Cholesky decomposition of $X^\top X$.
 - 2 Find the $z_\theta = L\theta$ by solving triangular system $L^\top z_\theta = y$
 - 3 Find the θ by solving triangular system $L\theta = z_\theta$
-

Note, that in this case the error stil proportional to the squared condition number.



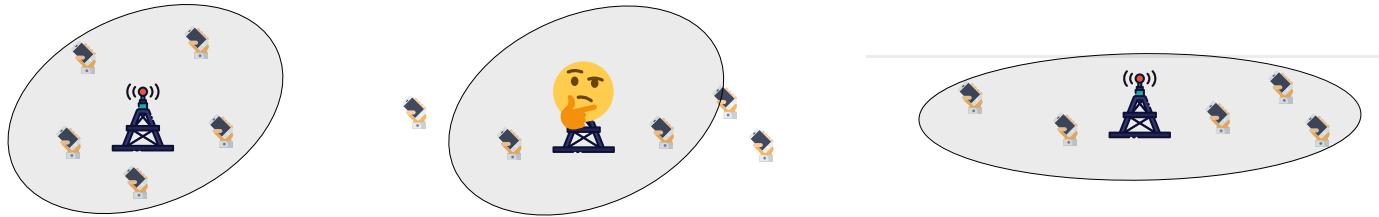
Code

 Open in Colab

References

- [CVXPY documentation](#)
- [Interactive example](#)
- [Jupyter notebook by A. Katrutsa](#)

Problem



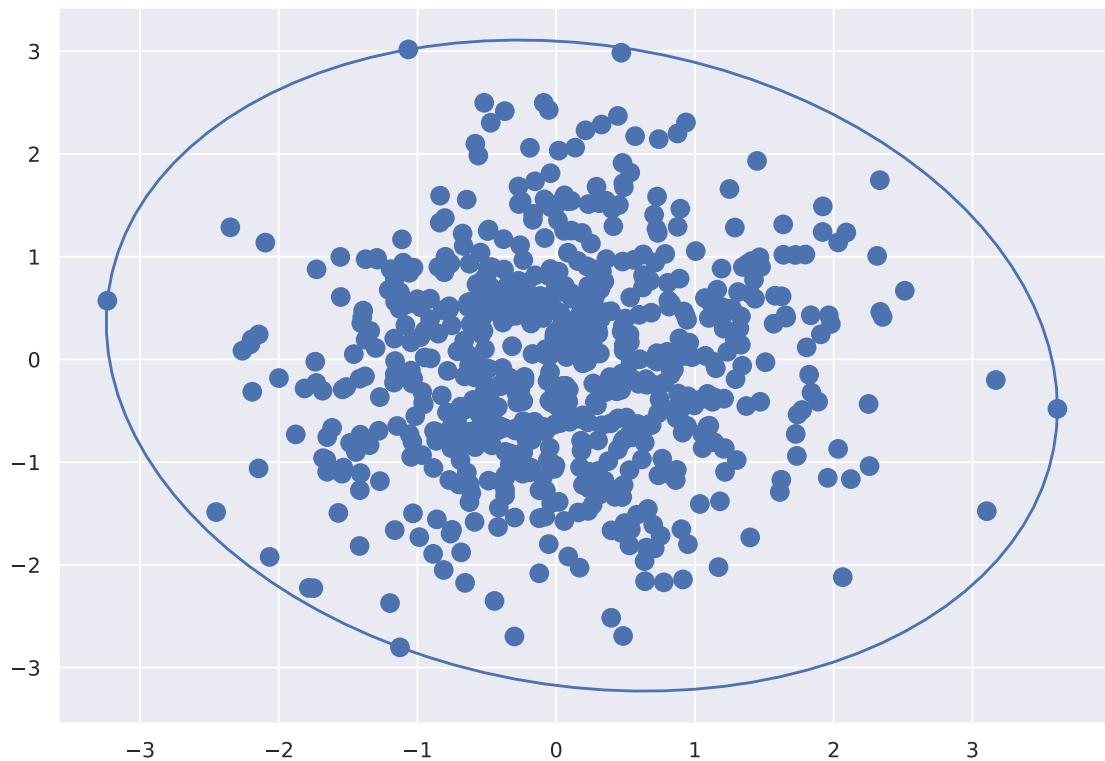
Let x_1, \dots, x_n be the points in \mathbb{R}^2 . Given these points we need to find an ellipsoid, that contains all points with the minimum volume (in 2d case volume of an ellipsoid is just the square).

An invertible linear transformation applied to a unit sphere produces an ellipsoid with the square, that is $\det A^{-1}$ times bigger, than the unit sphere square, that's why we parametrize the interior of ellipsoid in the following way:

$$S = \{x \in \mathbb{R}^2 \mid u = Ax + b, \|u\|_2^2 \leq 1\}$$

Sadly, the determinant is the function, which is relatively hard to minimize explicitly. However, the function $\log \det A^{-1} = -\log \det A$ is actually convex, which provides a great opportunity to work with it. As soon as we need to cover all the points with ellipsoid of minimum volume, we pose an optimization problem on the convex function with convex restrictions:

$$\begin{aligned} & \min_{A \in \mathbb{R}^{2 \times 2}, b \in \mathbb{R}^2} -\log \det(A) \\ \text{s.t. } & \|Ax_i + b\| \leq 1, i = 1, \dots, n \\ & A \succ 0 \end{aligned}$$



Code

Open in Colab

References

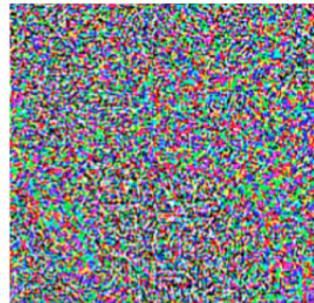
- Jupyter notebook by A. Katrutsa
- <https://cvxopt.org/examples/book/ellipsoids.html>

Lipschitz constant of a convolutional layer in neural network

It was observed, that small perturbation in Neural Network input could lead to significant errors, i.e. misclassifications.



+ 0.001 ×



=

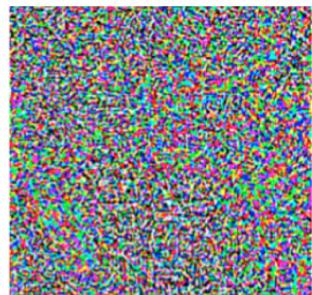


Bagle

piano



+ 0.001 ×



=



stop sign

teddy bear

Lipschitz constant bounds the magnitude of the output of a function, so it cannot change drastically with a slight change in the input

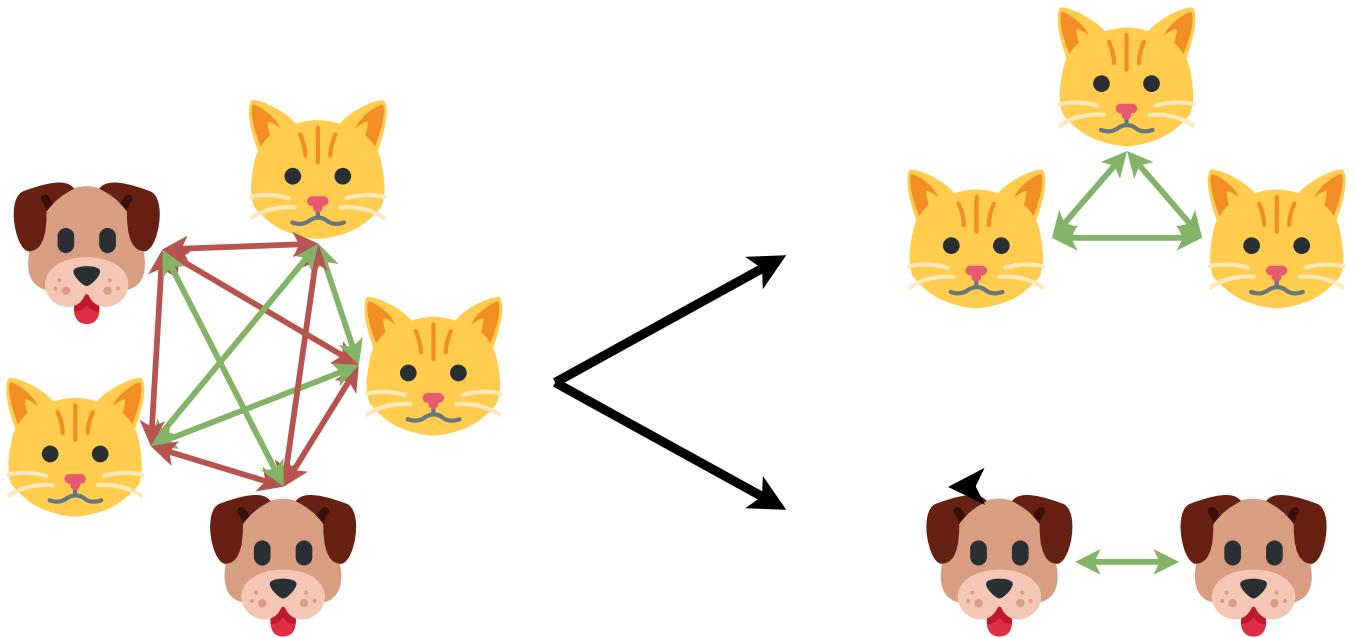
$$\|NN(image) - NN(image + \varepsilon)\| \leq L\|\varepsilon\|$$

In this notebook we will try to estimate Lipschitz constant of some convolutional layer of a Neural Network.

Code

Open in Colab

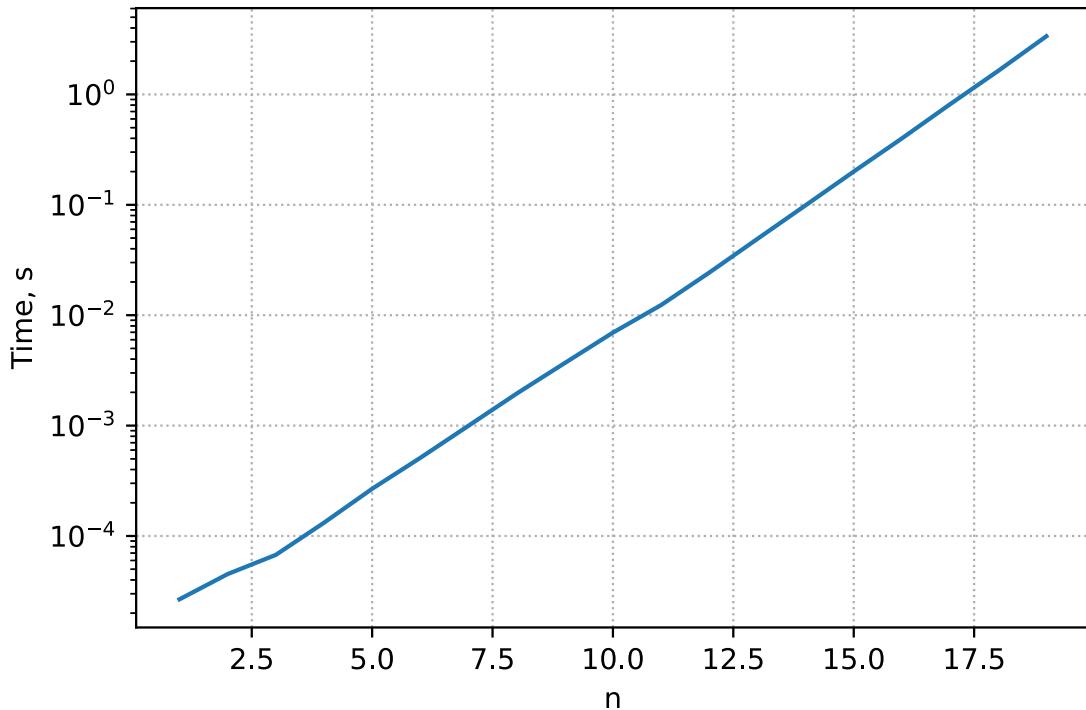
Intuition



Suppose, we have a set of n objects, which are needed to be splitted into two groups. Moreover, we have information about the preferences of all possible pairs of objects to be in the same group. this information could be presented in the matrix form: $W \in \mathbb{R}^{n \times n}$, where $\{w_{ij}\}$ is the cost of having i -th and j -th object in the same partitions. It is easy to see, that the total number of partitions is finite and equals to 2^n . So this problem can in principle be solved by simply checking the objective value of each feasible point. Since the number of feasible points grows exponentially, however, this is possible only for small problems (say, with $n \leq 30$). In general (and for n larger than, say, 50) the problem is very difficult to solve.

For example, bruteforce solution on MacBook Air with M1 processor without any explicit parallelization will take more, than a universe lifetime for $n = 62$.

Average time for bruteforce solution. 3 runs per n



Despite the hardness of the problems, there are several ways to approach it.

Problem

We consider the (nonconvex) problem

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} x^\top W x, \\ & \text{s.t. } x_i^2 = 1, \quad i = 1, \dots, n \end{aligned}$$

where $W \in \mathbb{R}^n$ is the symmetric matrix. The constraints restrict the values of x_i to 1 or -1 , so the problem is equivalent to finding the vector with components ± 1 that minimizes $x^\top W x$. The feasible set here is finite (it contains 2^n points), thus, is non-convex.

The objective is the total cost, over all pairs of elements, and the problem is to find the partition with least total cost.

Simple lower bound with duality

We now derive the dual function for this problem. The Lagrangian is

$$L(x, \nu) = x^\top W x + \sum_{i=1}^n \nu_i (x_i^2 - 1) = x^\top (W + \text{diag}(\nu)) x - \mathbf{1}^\top \nu.$$

We obtain the Lagrange dual function by minimizing over x :

$$\begin{aligned} g(\nu) &= \inf_{x \in \mathbb{R}^n} x^\top (W + \text{diag}(\nu)) x - \mathbf{1}^\top \nu = \\ &= \begin{cases} \mathbf{1}^\top \nu, & W + \text{diag}(\nu) \succeq 0 \\ -\infty, & \text{otherwise} \end{cases} \end{aligned}$$

so

This dual function provides lower bounds on the optimal value of the difficult problem. For example, we can take any specific value of the dual variable

$$\nu = -\lambda_{\min}(W)\mathbf{1},$$

This yields the bound on the optimal value p^* :

$$p^* \geq g(\nu) \geq -\mathbf{1}^\top \nu = n\lambda_{\min}(W)$$

Question Can you obtain the same lower bound without knowledge of duality, but using the idea of eigenvalues?

Code

 Open in Colab

References

- Convex Optimization book by Stephen Boyd and Lieven Vandenberghe.

General formulation

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ \text{s.t. } & g_i(x) \leq 0, i = 1, \dots, m \\ & h_j(x) = 0, j = 1, \dots, k \end{aligned}$$

Some necessary or/and sufficient conditions are known (See [Optimality conditions. KKT](#) and [Convex optimization problem](#))

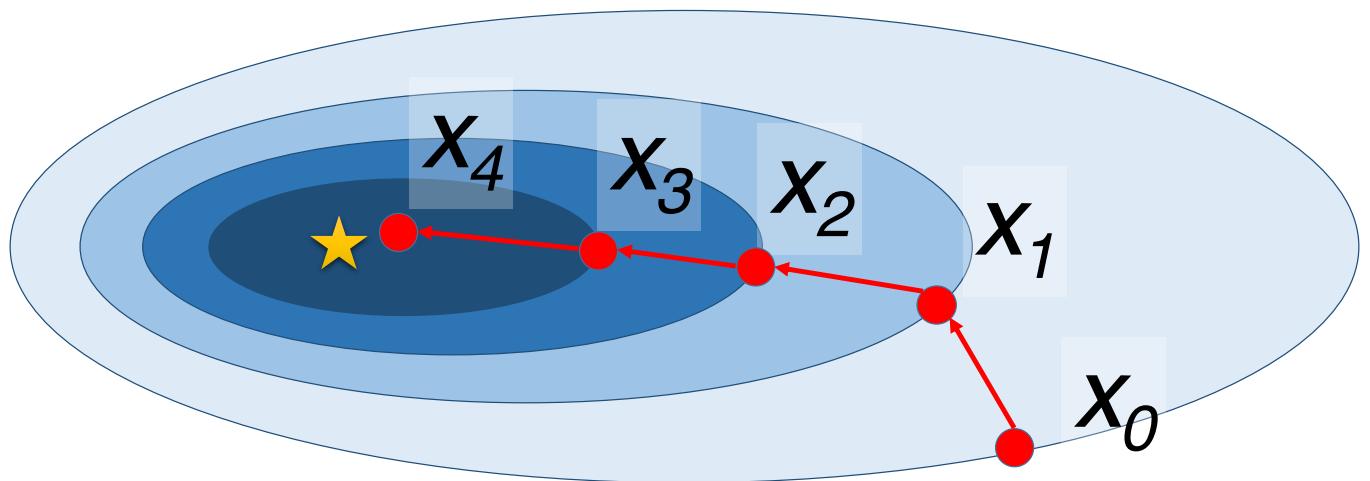
- In fact, there might be very challenging to recognize the convenient form of optimization problem.
- Analytical solution of KKT could be inviable.

Iterative methods

Typically, the methods generate an infinite sequence of approximate solutions

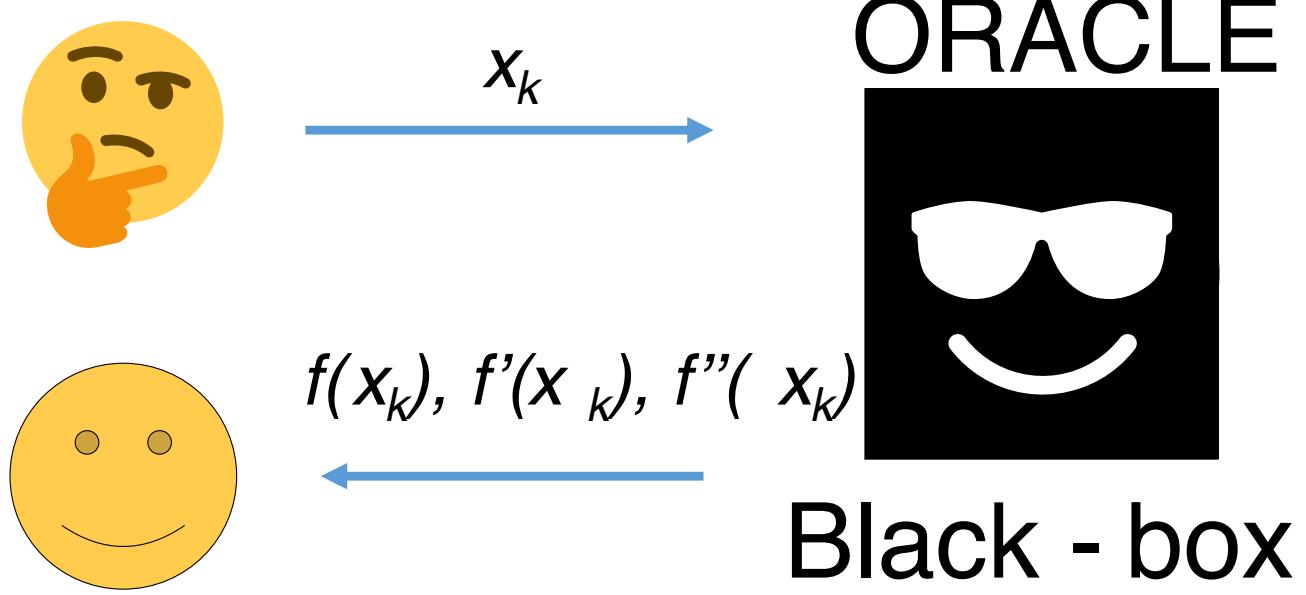
$$\{x_t\},$$

which for a finite number of steps (or better - time) converges to an optimal (at least one of the optimal) solution x_* .



```
def GeneralScheme(x, epsilon):
    while not StopCriterion(x, epsilon):
        OracleResponse = RequestOracle(x)
        x = NextPoint(x, OracleResponse)
    return x
```

Oracle conception



Complexity

Challenges

Unsolvability

In general, **optimization problems are unsolvable.** $\neg(\forall)/\neg$

Consider the following simple optimization problem of a function over unit cube:

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ \text{s.t. } & x \in \mathbb{B}^n \end{aligned}$$

We assume, that the objective function $f(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}$ is Lipschitz continuous on \mathbb{B}^n :

$$|f(x) - f(y)| \leq L \|x - y\|_\infty \quad \forall x, y \in \mathbb{B}^n,$$

with some constant L (Lipschitz constant). Here \mathbb{B}^n - the n -dimensional unit cube

$$\mathbb{B}^n = \{x \in \mathbb{R}^n \mid 0 \leq x_i \leq 1, i = 1, \dots, n\}$$

Our goal is to find such $\tilde{x} : |f(\tilde{x}) - f^*| \leq \varepsilon$ for some positive ε . Here f^* is the global minima of the problem. Uniform grid with p points on each dimension guarantees at least this quality:

$$\|\tilde{x} - x_*\|_\infty \leq \frac{1}{2p},$$

which means, that

$$|f(\tilde{x}) - f(x_*)| \leq \frac{L}{2p}$$

Our goal is to find the p for some ε . So, we need to sample $\left(\frac{L}{2\varepsilon}\right)^n$ points, since we need to measure function in p^n points. Doesn't look scary, but if we'll take $L = 2, n = 11, \varepsilon = 0.01$, computations on the modern personal computers will take 31,250,000 years.

Stopping rules

- Argument closeness:

$$\|x_k - x_*\|_2 < \varepsilon$$

- Function value closeness:

$$\|f_k - f^*\|_2 < \varepsilon$$

- Closeness to a critical point

$$\|f'(x_k)\|_2 < \varepsilon$$

But x_* and $f^* = f(x_*)$ are unknown!

Sometimes, we can use the trick:

$$\|x_{k+1} - x_k\| = \|x_{k+1} - x_k + x_* - x_*\| \leq \|x_{k+1} - x_*\| + \|x_k - x_*\| \leq 2\varepsilon$$

Note: it's better to use relative changing of these values, i.e. $\frac{\|x_{k+1} - x_k\|_2}{\|x_k\|_2}$.

Local nature of the methods

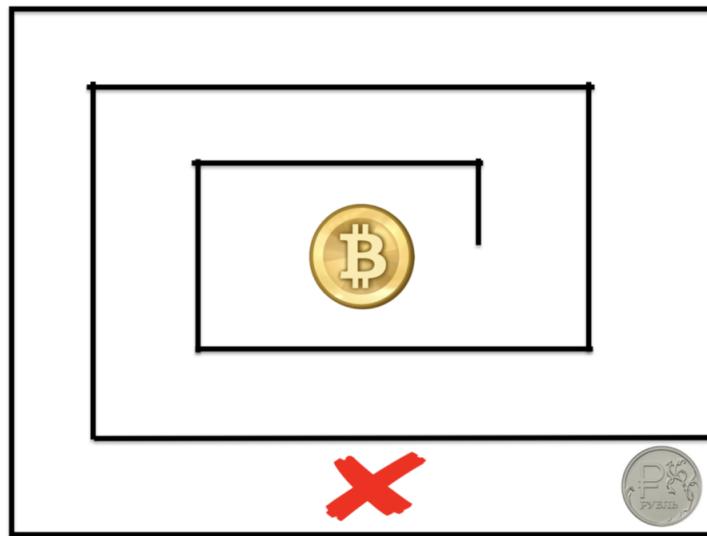


TABLE OF CONTENTS

- Line search
- Zero order methods
- First order methods
- Adaptive metric methods
- LP and simplex algorithm
- Automatic differentiation

Speed of convergence

In order to compare performance of algorithms we need to define a terminology for different types of convergence. Let $\{x_k\}$ be a sequence in \mathbb{R}^n that converges to some point x^*

Linear convergence

We can define the *linear* convergence in a two different forms:

$$\|x_{k+1} - x^*\|_2 \leq Cq^k \quad \text{or} \quad \|x_{k+1} - x^*\|_2 \leq q\|x_k - x^*\|_2,$$

for all sufficiently large k . Here $q \in (0, 1)$ and $0 < C < \infty$. This means that the distance to the solution x^* decreases at each iteration by at least a constant factor bounded away from 1. Note, that sometimes this type of convergence is also called *exponential* or *geometric*.

Superlinear convergence

The convergence is said to be *superlinear* if:

$$\|x_{k+1} - x^*\|_2 \leq Cq^{k^2} \quad \text{or} \quad \|x_{k+1} - x^*\|_2 \leq C_k\|x_k - x^*\|_2,$$

where $q \in (0, 1)$ or $0 < C_k < \infty$, $C_k \rightarrow 0$. Note, that superlinear convergence is also linear convergence (one can even say, that it is linear convergence with $q = 0$).

Sublinear convergence

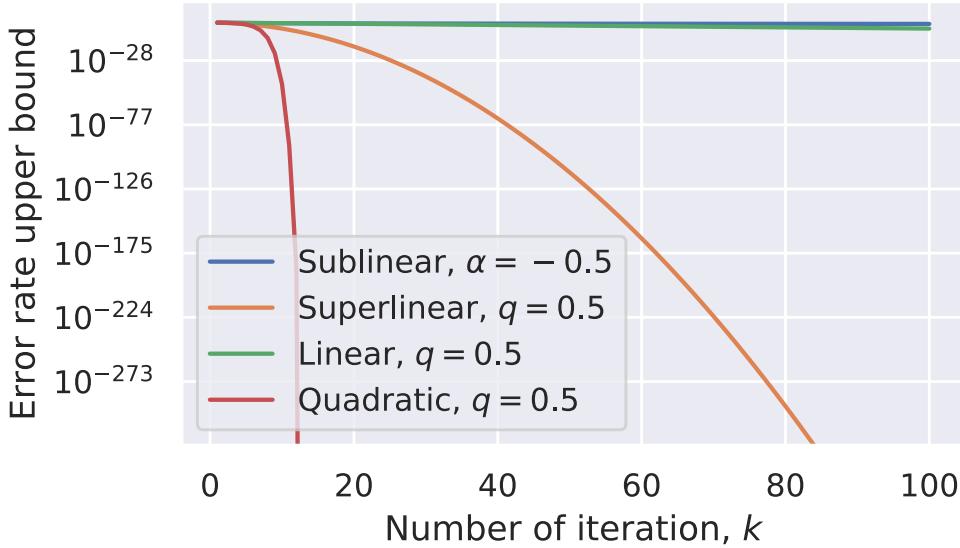
$$\|x_{k+1} - x^*\|_2 \leq Ck^q,$$

where $q < 0$ and $0 < C < \infty$. Note, that sublinear convergence means, that the sequence is converging slower, than any geometric progression.

Quadratic convergence

$$\|x_{k+1} - x^*\|_2 \leq Cq^{2^k} \quad \text{or} \quad \|x_{k+1} - x^*\|_2 \leq C\|x_k - x^*\|_2^2,$$

where $q \in (0, 1)$ and $0 < C < \infty$.



Quasi-Newton methods for unconstrained optimization typically converge superlinearly, whereas Newton's method converges quadratically under appropriate assumptions. In contrast, steepest descent algorithms converge only at a linear rate, and when the problem is ill-conditioned the convergence constant q is close to 1.

How to determine convergence type

Root test

Let $\{r_k\}_{k=m}^{\infty}$ be a sequence of non-negative numbers, converging to zero, and let

$$q = \limsup_{k \rightarrow \infty} r_k^{1/k}$$

- If $0 \leq q < 1$, then $\{r_k\}_{k=m}^{\infty}$ has linear convergence with constant q .
- In particular, if $q = 0$, then $\{r_k\}_{k=m}^{\infty}$ has superlinear convergence.
- If $q = 1$, then $\{r_k\}_{k=m}^{\infty}$ has sublinear convergence.
- The case $q > 1$ is impossible.

Ratio test

Let $\{r_k\}_{k=m}^{\infty}$ be a sequence of strictly positive numbers converging to zero. Let

$$q = \lim_{k \rightarrow \infty} \frac{r_{k+1}}{r_k}$$

- If there exists q and $0 \leq q < 1$, then $\{r_k\}_{k=m}^{\infty}$ has linear convergence with constant q .
- In particular, if $q = 0$, then $\{r_k\}_{k=m}^{\infty}$ has superlinear convergence.
- If q does not exist, but $q = \limsup_{k \rightarrow \infty} \frac{r_{k+1}}{r_k} < 1$, then $\{r_k\}_{k=m}^{\infty}$ has linear convergence with a constant not exceeding q .
- If $\liminf_{k \rightarrow \infty} \frac{r_{k+1}}{r_k} = 1$, then $\{r_k\}_{k=m}^{\infty}$ has sublinear convergence.
- The case $\liminf_{k \rightarrow \infty} \frac{r_{k+1}}{r_k} > 1$ is impossible.
- In all other cases (i.e., when $\liminf_{k \rightarrow \infty} \frac{r_{k+1}}{r_k} < 1 \leq \limsup_{k \rightarrow \infty} \frac{r_{k+1}}{r_k}$) we cannot claim anything concrete about the convergence rate $\{r_k\}_{k=m}^{\infty}$.

References

- Code for convergence plots - [!\[\]\(bd416a8536ad6d035be85a90bc470d6b_img.jpg\) Open in Colab](#)
- [CMC seminars \(ru\)](#)
- Numerical Optimization by J.Nocedal and S.J.Wright

Useful definitions and notations

We will treat all vectors as column vectors by default. The space of real vectors of length n is denoted by \mathbb{R}^n , while the space of real-valued $m \times n$ matrices is denoted by $\mathbb{R}^{m \times n}$.

Basic linear algebra background

The standard **inner product** between vectors x and y from \mathbb{R}^n is given by

$$\langle x, y \rangle = x^\top y = \sum_{i=1}^n x_i y_i = y^\top x = \langle y, x \rangle$$

Here x_i and y_i are the scalar i -th components of corresponding vectors.

The standard **inner product** between matrices X and Y from $\mathbb{R}^{m \times n}$ is given by

$$\langle X, Y \rangle = \text{tr}(X^\top Y) = \sum_{i=1}^m \sum_{j=1}^n X_{ij} Y_{ij} = \text{tr}(Y^\top X) = \langle Y, X \rangle$$

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

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

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

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

The largest and smallest eigenvalues satisfy

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

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

$$\lambda_{\min}(A)x^\top x \leq x^\top Ax \leq \lambda_{\max}(A)x^\top x$$

A matrix $A \in \mathbb{S}^n$ (set of square symmetric matrices of dimension n) is called **positive (semi)definite** if for all $x \neq 0$ (for all x) : $x^\top Ax > (\geq) 0$. We denote this as

$$A \succ (\succeq) 0.$$

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

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

Matrix and vector multiplication

Let A be a matrix of size $m \times n$, and B be a matrix of size $n \times p$, 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 a matrix of shape $m \times n$, and x be $n \times 1$ vector, then the i -th component of the product:

$$z = Ax$$

is given by:

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

Finally, just to remind:

- $C = AB \quad C^\top = B^\top A^\top$
- $AB \neq BA$
- $e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k$
- $e^{A+B} \neq e^A e^B$ (but if A and B are commuting matrices, which means that $AB = BA, e^{A+B} = e^A e^B$)
- $\langle x, Ay \rangle = \langle A^\top x, y \rangle$

Gradient

Let $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$, 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.

Hessian

Let $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$, 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: ($f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$) 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 $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is the following matrix:

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

Summary

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

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

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

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

Simple derivative

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

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

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.

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).

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

$$\bullet \quad d(X^{-1}) = -X^{-1}(dX)X^{-1}$$

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](#)