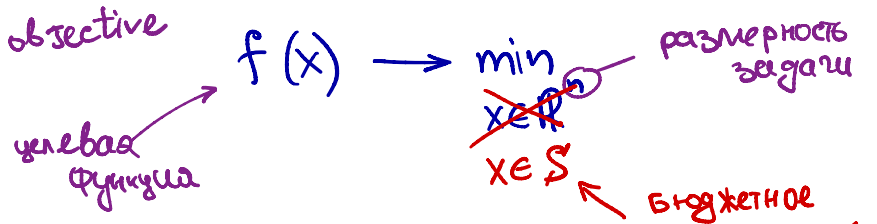


# Optimization problems. CVXPY. Basic linear algebra recap.

Daniil Merkulov

Applied Math for Data Science. Sberuniversity.





### Optimization problems

$$\underline{x \in \mathbb{R}^n}$$

Безусловная  
задача  
оптимизации

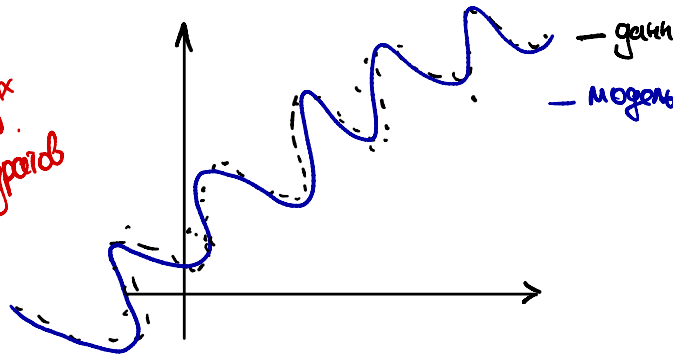
$$x \in S \subseteq \mathbb{R}^n$$

УСЛОВНАЯ  
ЗАДАЧА  
ОПТИМИЗ

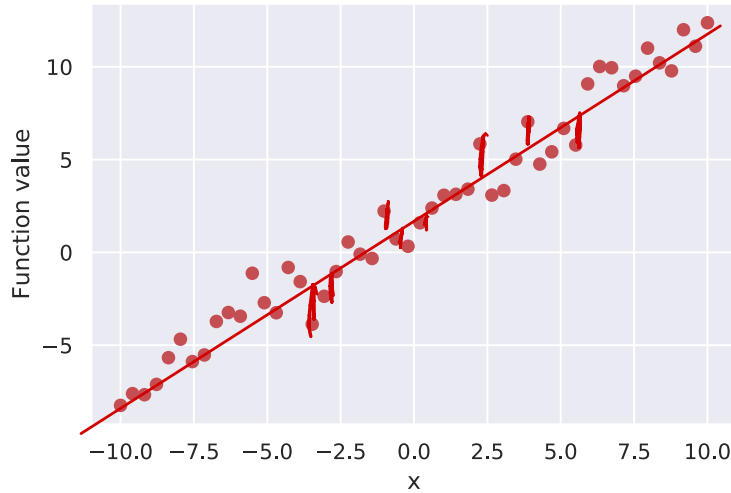
# Linear least squares

## 1 Problem

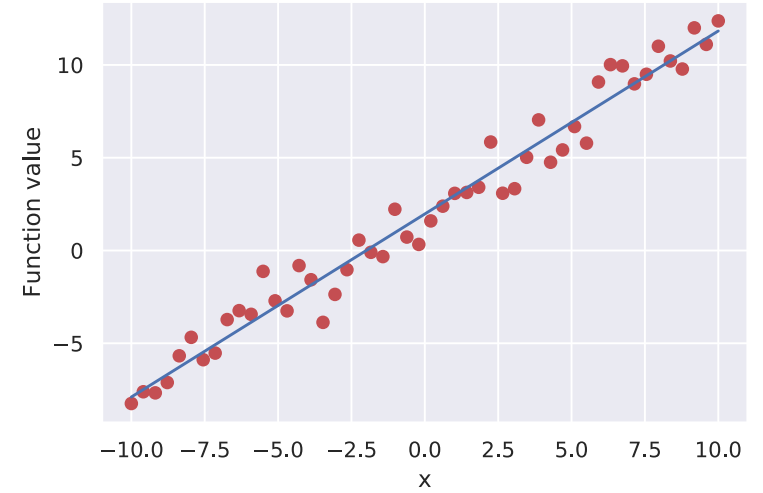
метод  
наименьших  
квадратов



Linear least squares.



Linear least squares.



Illustration

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:

$$\sum_{i=1}^m (x_i^\top \theta - y_i)^2 \rightarrow \min_{\theta \in \mathbb{R}^n}$$

выбор  
ряда  
нулей  $\theta$   
 $y_i$  был похож  
на  $\theta^\top x_i$

also known as the  $l_2$ -norm squared,  $\|X\theta - y\|_2^2 \rightarrow \min_{\theta \in \mathbb{R}^n}$

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  $y$

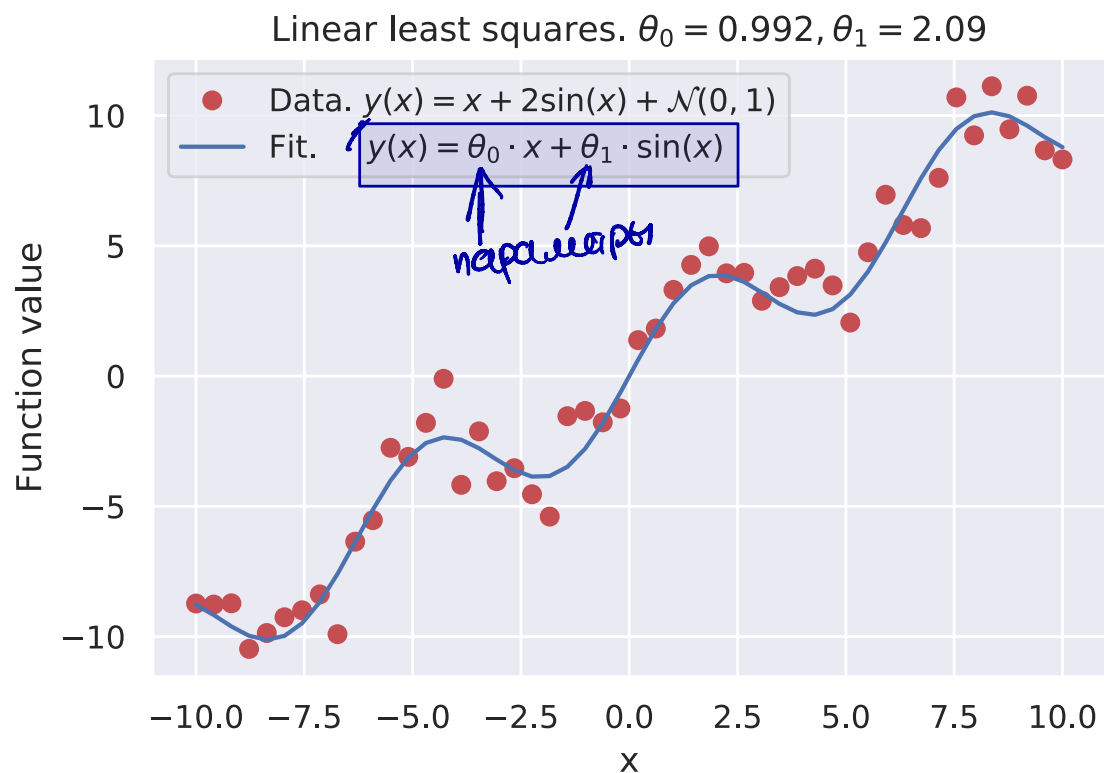
is the measurement we want to predict from  $x_i^\top$ , such as ad spending. The prediction is given by  $x_i^\top \theta$ .

We find the optimal  $\theta$  by solving the optimization problem

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

Let  $\theta^*$  denote the optimal  $\theta$ . 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  $\theta$  that are to be determined in the best fit.



обучение модели  
 $\min L(\theta_0, \theta_1)$   
 $\theta_0, \theta_1 \in \mathbb{R}^2$

## 2 Approaches



## 2.1 Moore–Penrose inverse

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

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

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.

## 2.2 QR decomposition

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

$$X = Q \cdot R,$$

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

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

Now, process of finding theta consists of two steps:

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

## 2.3 Cholesky decomposition

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

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

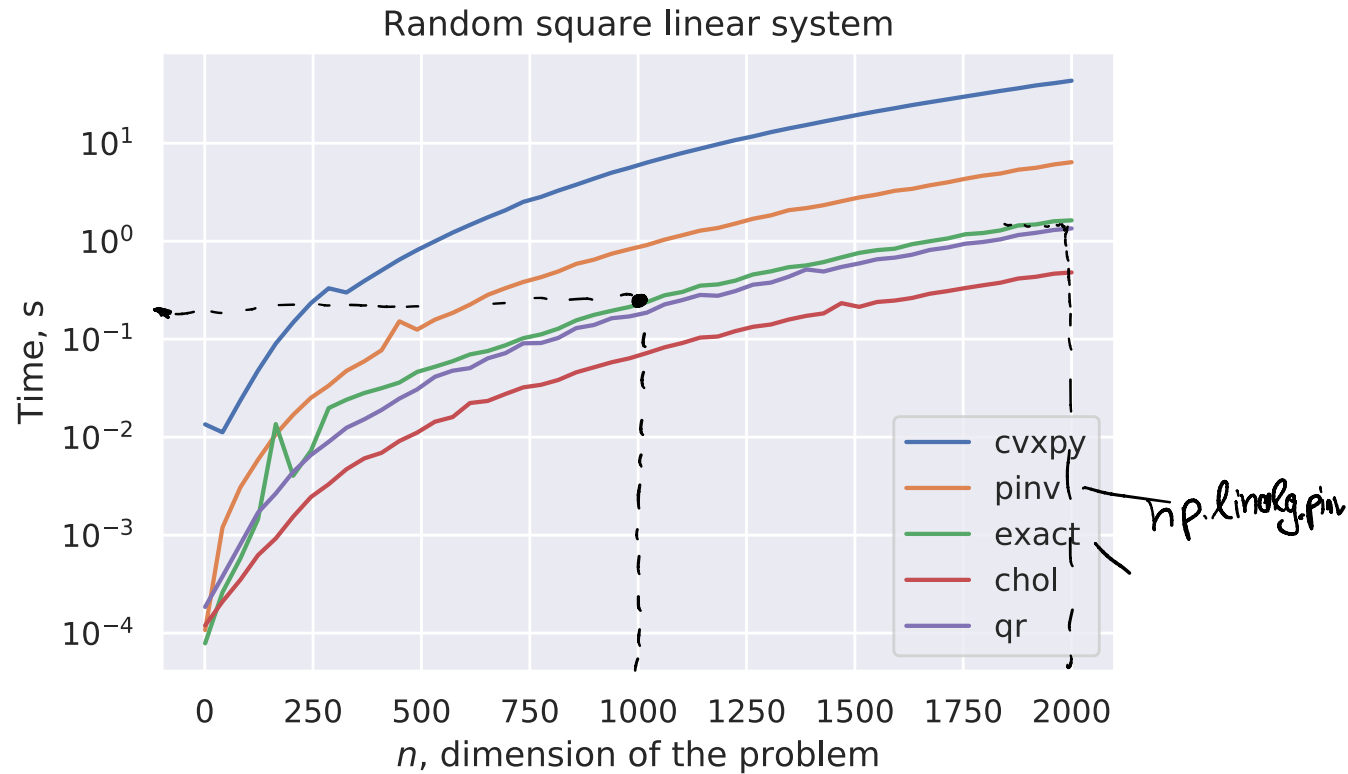
where  $L$  is an lower triangular matrix. We have:

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

Now, process of finding theta consists of two steps:

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

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



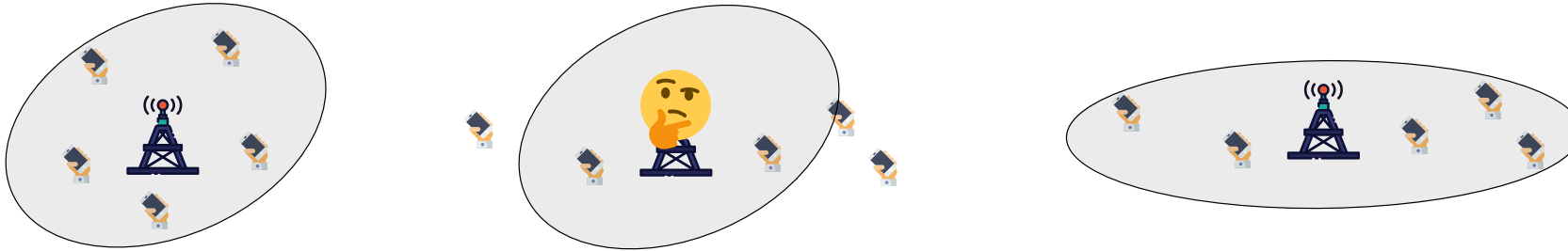
Illustration

### 3 Code

[Open In Colab](#){: .btn } ## References \* [CVXPY documentation](#) \* [Interactive example](#) \* [Jupyter notebook by A. Katrutsa](#)

# Minimum volume ellipsoid

## 1 Problem



Illustration

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:

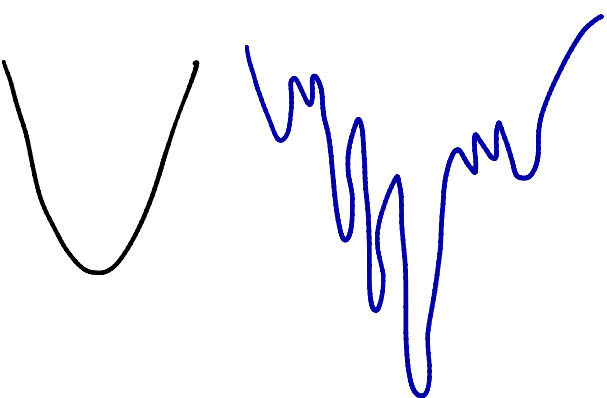
$$S = \{x \in \mathbb{R}^2 \mid u = Ax + b, \|u\|_2^2 \leq 1\}$$

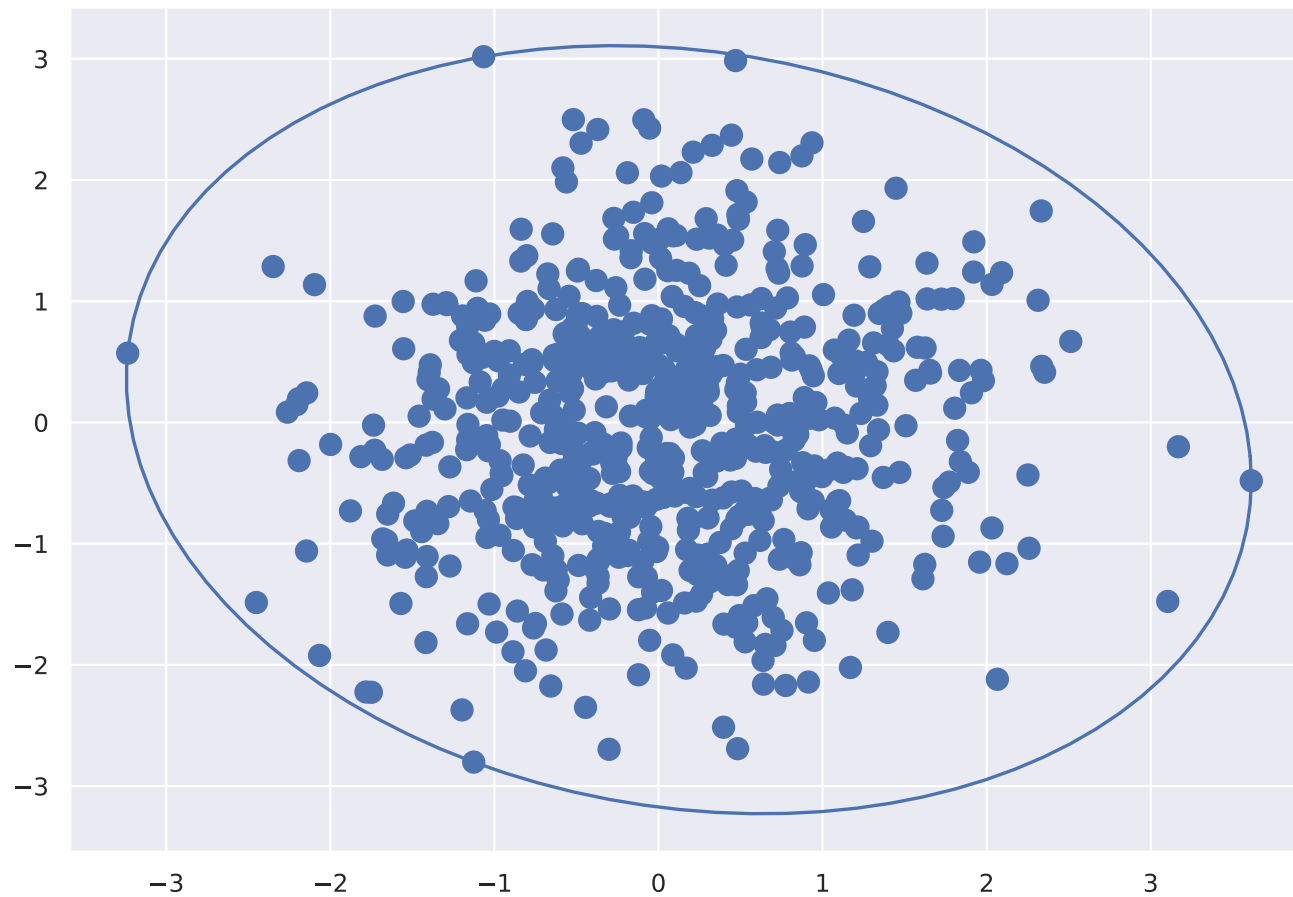
Sadly, the determinant is the function, which is relatively hard to minimize explicitly. However, the function  $\log \det A^{-1} = -\log \det A$  is actually convex, which provides a great opportunity to work with it. As soon as we need to cover all the points with ellipsoid of minimum volume, we pose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

$$f(A) = -\log \det A$$

ВЫПУКЛАЯ

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





Illustration

## 2 Code

---

[Open In Colab](#){: .btn }

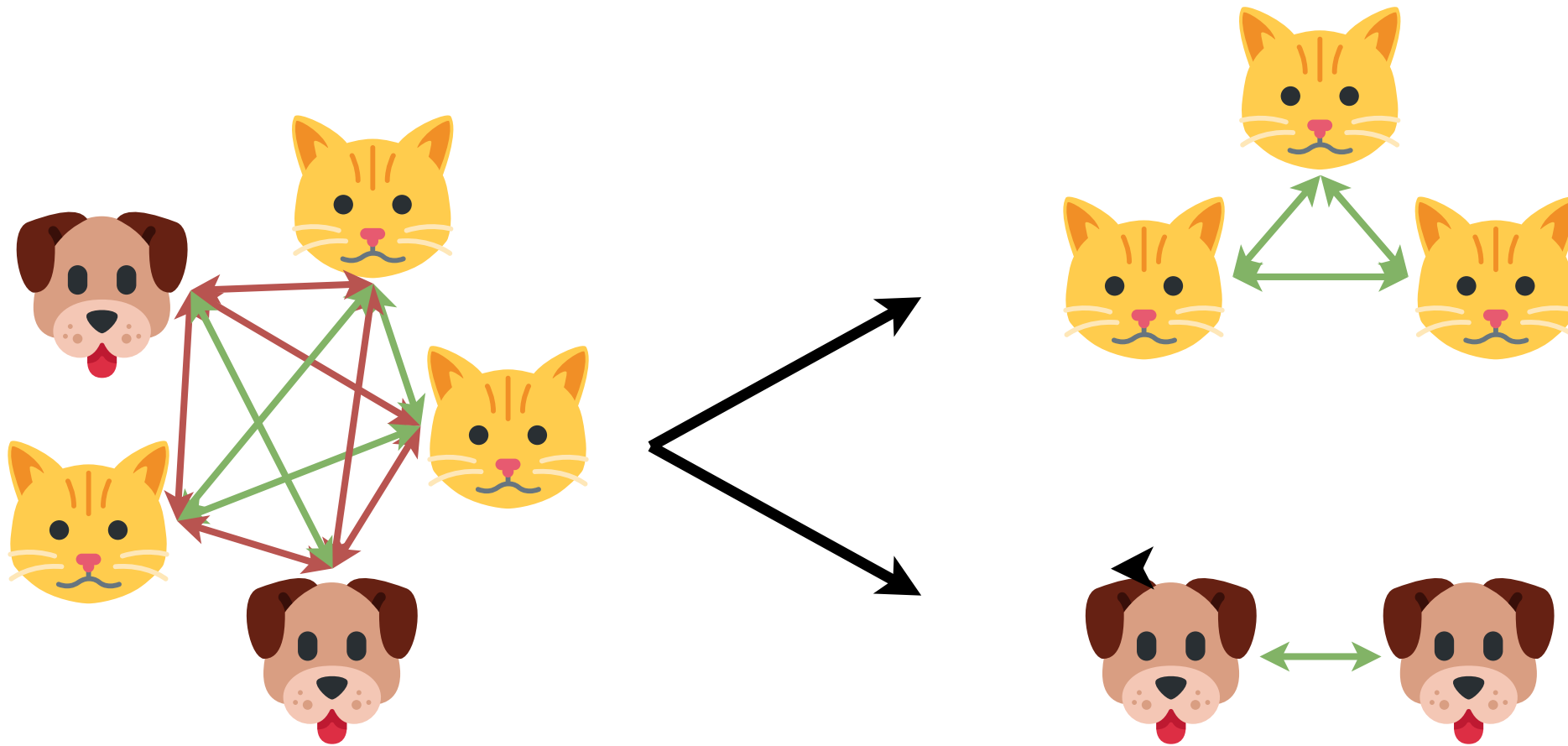
## 3 References

---

- [Jupyter notebook](#) by A. Katrutsa
- <https://cvxopt.org/examples/book/ellipsoids.html>

# Two way partitioning problem

## 1 Intuition

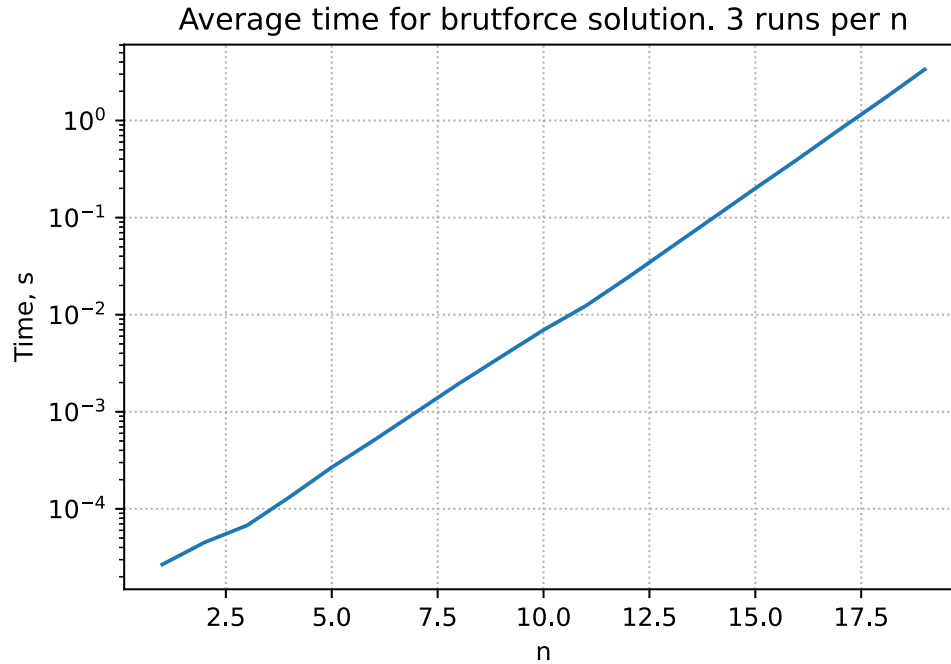


Illustration

Suppose, we have a set of  $n$  objects, which are needed to be split 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

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  $n = 62$ .



Illustration

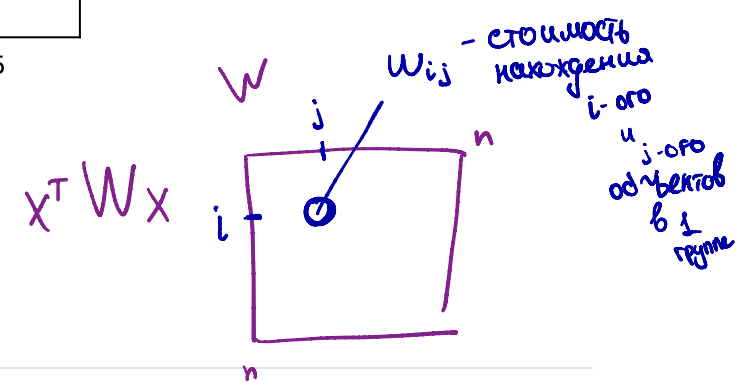
Despite the hardness of the problems, there are several ways to approach it.

## 2 Problem

We consider the (nonconvex) problem

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

$$x \begin{bmatrix} \xrightarrow{n} \\ 1 & -1 & -1 & \dots & 1 \end{bmatrix}$$



$2^n$  вариантов

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  $\pm 1$

where  $W \in \mathbb{R}^{n \times n}$  is the symmetric matrix. The constraints restrict the values of  $x_i$  to  $\pm 1$ , so the problem is equivalent to finding the vector with components  $\pm 1$  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.

## 2.1 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  $x$ :

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

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?

## 3 Code

[Open In Colab](#){: .btn }

## 4 References

- [Convex Optimization](#) book by Stephen Boyd and Lieven Vandenberghe.

# Total variation in-painting

## 1 Problem



Illustration

### 1.1 Grayscale image

A grayscale image is represented as an  $m \times n$  matrix of intensities  $U^{orig}$  (typically between the values 0 and 255). We are given all the values of corrupted picture, but some of them should be preserved as is through the recovering procedure:  $U_{ij}^{corr} \forall (i, j) \in K$ , where  $K \subset \{1, \dots, m\} \times \{1, \dots, n\}$  is the set of indices corresponding to known pixel values. Our job is to in-paint the image by guessing the missing pixel values, i.e., those with indices not in  $K$ . The reconstructed image will be represented by  $U \in \mathbb{R}^{m \times n}$ , where  $U$  matches the known pixels, i.e.  $U_{ij} = U_{ij}^{corr}$  for  $(i, j) \in K$ .

The reconstruction  $U$  is found by minimizing the total variation of  $U$ , subject to matching the known pixel values. We will use the  $l_2$  total variation, defined as

$$TV(U) = \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \| [ U_{i+1,j} - U_{ij} ] \|$$

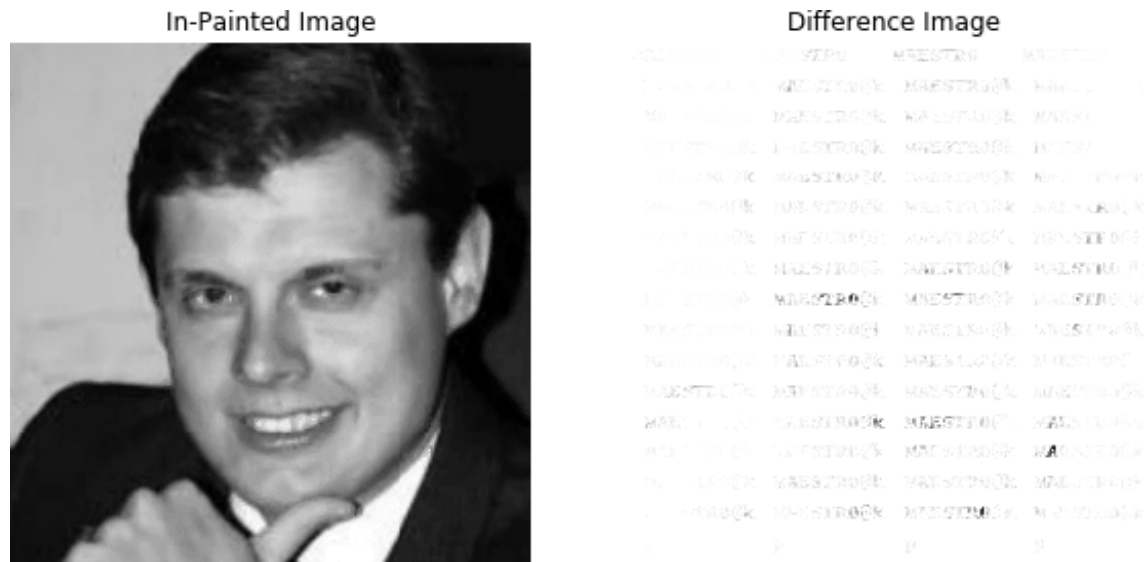


$$\text{tv}(U) = \sum_{i=1}^m \sum_{j=1}^{n-1} \left\| \begin{bmatrix} U_{i,j+1} - U_{ij} \end{bmatrix} \right\|_2.$$

So, the final optimization problem will be written as follows:

$$\begin{aligned} \text{tv}(U) &\rightarrow \min_{U \in \mathbb{R}^{m \times n}} \\ \text{s.t. } U_{ij} &= U_{ij}^{\text{corr}}, \quad (i, j) \in K \end{aligned}$$

The crucial thing about this problem is defining set of known pixels  $K$ . There are some heuristics: for example, we could state, that each pixel with color similar (or exactly equal) to the color of text is unknown. The results for such approach are presented below:



Illustration

## 1.2 Color image

For the color case we consider in-painting problem in a slightly different setting: destroying some random part of all pixels. In this case the image itself is 3d tensor (we convert all others color schemes to the RGB). As it was in the grayscale case, we construct the mask  $K$  of known pixels for all color channels uniformly, based on the principle of similarity of particular 3d pixel to the vector  $[0, 0, 0]$  (black pixel). The results are quite promising - note, that we have no information about the original picture, but assumption, that corrupted pixels are black. For the color picture we just sum all tv's on the each channel:

$$\sum_{k=1}^3 \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \left\| \begin{bmatrix} T^k_{i,j+1} - T^k_{ij} \end{bmatrix} \right\|_2$$

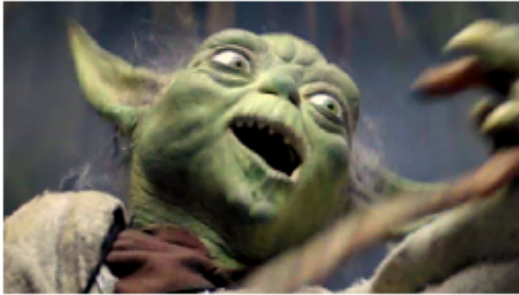
$$\mathbf{tv}(U) = \sum_{k=1} \sum_{i=1} \sum_{j=1} \left\| \begin{bmatrix} \tilde{U}_{i,j+1}^{k,i+1,j} - \tilde{U}_{ij}^k \end{bmatrix} \right\|_2.$$

Then, we need to write down optimization problem to be solved:

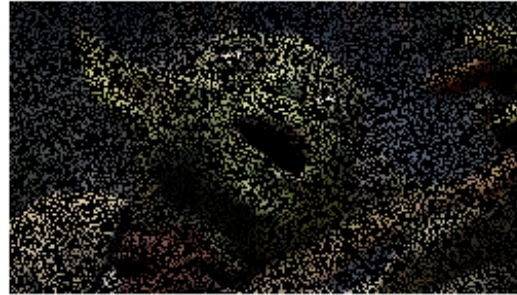
$$\begin{aligned} \mathbf{tv}(U) &\rightarrow \min_{U \in \mathbb{R}^{m \times n \times 3}} \\ \text{s.t. } U_{ij}^k &= U_{ij}^{\text{corr},k}, \quad (i,j) \in K, \quad k = 1, 2, 3 \end{aligned}$$

Results are presented below (these computations really take time):

Original Image

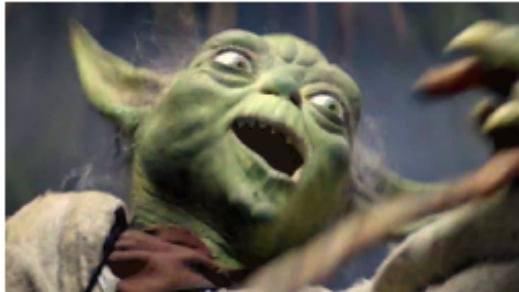


Corrupted Image. Corruption level 0.707

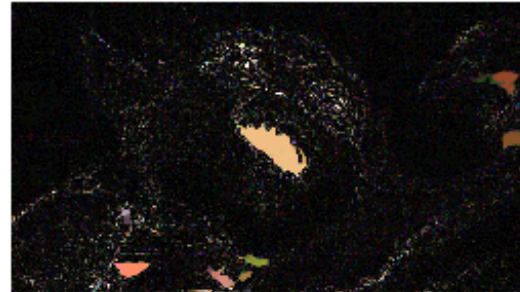


Illustration

In-Painted Image



Difference Image



Illustration

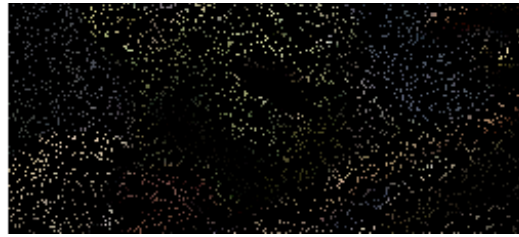
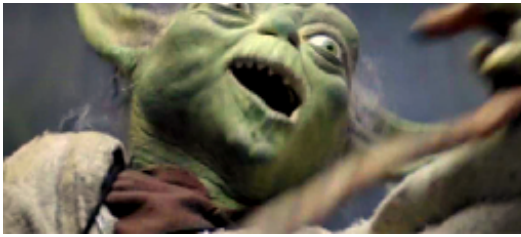
It is not that easy, right?

Original Image



Corrupted Image. Corruption level 0.9



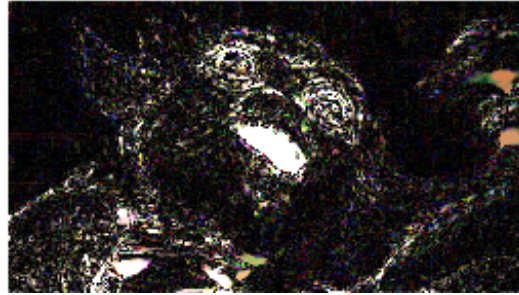


Illustration

In-Painted Image



Difference Image



Illustration

Only 5% of all pixels are left:

Original Image



Corrupted Image. Corruption level 0.95



Illustration

In-Painted Image



Difference Image

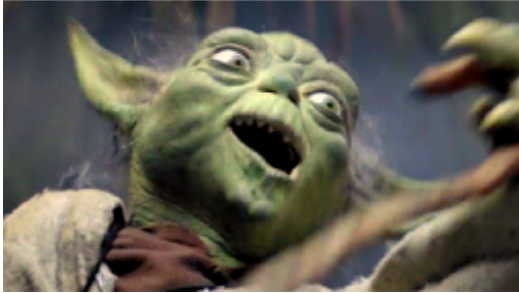




Illustration

What about 1% of all pixels?

Original Image

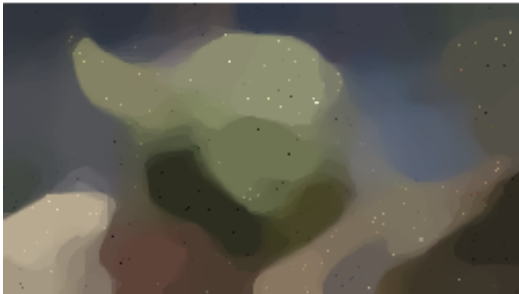


Corrupted Image. Corruption level 0.99



Illustration

In-Painted Image



Difference Image



Illustration

## 2 Code

[Open In Colab](#){: .btn } ## References

- [CVXPY documentation](#)
- [Interactive demo](#)

## Example: Transportation problem

Customer / Source	Arnhem [€/ton]	Gouda [€/ton]	Demand [tons]
London	n/a	2.5	125
Berlin	2.5	n/a	175
Maastricht	1.6	2.0	225
Amsterdam	1.4	1.0	250
Utrecht	0.8	1.0	225
The Hague	1.4	0.8	200
<b>Supply [tons]</b>	550 tons	700 tons	

$$\text{minimize: Cost} = \sum_{c \in \text{Customers}} \sum_{s \in \text{Sources}} T[c, s] x[c, s]$$

## Example: Transportation problem

Customer / Source	Arnhem [€/ton]	Gouda [€/ton]	Demand [tons]
London	n/a	2.5	125
Berlin	2.5	n/a	175
Maastricht	1.6	2.0	225
Amsterdam	1.4	1.0	250
Utrecht	0.8	1.0	225
The Hague	1.4	0.8	200
<b>Supply [tons]</b>	550 tons	700 tons	

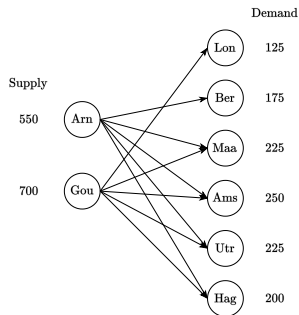
$$\text{minimize: Cost} = \sum_{c \in \text{Customers}} \sum_{s \in \text{Sources}} T[c, s] x[c, s]$$

$$\sum_{c \in \text{Customers}} x[c, s] \leq \text{Supply}[s] \quad \forall s \in \text{Sources}$$

## Example: Transportation problem

Customer / Source	Arnhem [€/ton]	Gouda [€/ton]	Demand [tons]
London	n/a	2.5	125
Berlin	2.5	n/a	175
Maastricht	1.6	2.0	225
Amsterdam	1.4	1.0	250
Utrecht	0.8	1.0	225
The Hague	1.4	0.8	200
<b>Supply [tons]</b>	550 tons	700 tons	

This can be represented in the following graph:



$$\text{minimize: Cost} = \sum_{c \in \text{Customers}} \sum_{s \in \text{Sources}} T[c, s] x[c, s]$$

$$\sum_{c \in \text{Customers}} x[c, s] \leq \text{Supply}[s] \quad \forall s \in \text{Sources}$$


$$\sum_{s \in \text{Sources}} x[c, s] = \text{Demand}[c] \quad \forall c \in \text{Customers}$$


Figure 1: Graph associated with the problem

## Hardware progress vs Software progress

# Mixed Integer Programming

What would you choose, assuming, that the question posed correctly (you can compile software for any hardware and the problem is the same for both options)? We will consider the time period from 1992 to 2023.

 Hardware **PROGRESS I**  
Solving MIP with an old software on the modern hardware

 Software **PROGRESS II**  
Solving MIP with a modern software on the old hardware



# Hardware progress vs Software progress

What would you choose, assuming, that the question posed correctly (you can compile software for any hardware and the problem is the same for both options)? We will consider the time period from 1992 to 2023.

## 🔥 Hardware

Solving MIP with an old software on the modern hardware

$\approx 1.664.510 \times \text{speedup}$

Moore's law states, that computational power doubles every 18 months.

## 🔥 Software

Solving MIP with a modern software on the old hardware

$\approx 2.349.000 \times \text{speedup}$

R. Bixby conducted an intensive experiment with benchmarking all CPLEX software version starting from 1992 to 2007 and measured overall software progress (29000 times), later (in 2009) he was a cofounder of Gurobi optimization software, which gives additional  $\approx 81$  speedup on MILP.



## Hardware progress vs Software progress

What would you choose, assuming, that the question posed correctly (you can compile software for any hardware and the problem is the same for both options)? We will consider the time period from 1992 to 2023.

### Hardware

Solving MIP with an old software on the modern hardware

$\approx 1.664.510 \times$  speedup

### Software

Solving MIP with a modern software on the old hardware

$\approx 2.349.000 \times$  speedup

Moore's law states, that computational power doubles every 18 monthes.

R. Bixby conducted an intensive experiment with benchmarking all CPLEX software version starting from 1992 to 2007 and measured overall software progress (29000 times), later (in 2009) he was a cofounder of Gurobi optimization software, which gives additional  $\approx 81$  speedup on **MILP**.

It turns out that if you need to solve a MILP, it is better to use an old computer and modern methods than vice versa, the newest computer and methods of the early 1990s!<sup>1</sup>

1

[R. Bixby report](#)

[Recent study](#)

# CVXPY Library

## CVXPY python library overview

**CVXPY** is an open-source Python library designed for solving convex optimization problems. It provides a high-level interface for defining and solving a wide range of optimization problems, making it a powerful tool for researchers, engineers, and data scientists.

# CVXPY python library overview

**CVXPY** is an open-source Python library designed for solving convex optimization problems. It provides a high-level interface for defining and solving a wide range of optimization problems, making it a powerful tool for researchers, engineers, and data scientists.

- **User-Friendly Syntax.** CVXPY uses a natural mathematical syntax, making it easy to formulate optimization problems.

```
import cvxpy as cp
x = cp.Variable()
objective = cp.Minimize(x**2)
constraints = [x >= 1]
prob = cp.Problem(objective, constraints)
result = prob.solve()
```

указав solver

$$x^2 \rightarrow \min \\ x \geq 1$$

решение:  $x^* = 1$   
от.  
 $f^* = f(x^*) = 1$

**CVXPY** is an open-source Python library designed for solving convex optimization problems. It provides a high-level interface for defining and solving a wide range of optimization problems, making it a powerful tool for researchers, engineers, and data scientists.

- **User-Friendly Syntax.** CVXPY uses a natural mathematical syntax, making it easy to formulate optimization problems.

```
import cvxpy as cp
x = cp.Variable()
objective = cp.Minimize(x**2)
constraints = [x >= 1]
prob = cp.Problem(objective, constraints)
result = prob.solve()
```

- **Wide Range of Applications.** CVXPY can be used in various fields such as finance, machine learning, control theory, and more. It supports linear programs (LP), quadratic programs (QP), second-order cone programs (SOCP), and semidefinite programs (SDP).

## CVXPY python library overview

**CVXPY** is an open-source Python library designed for solving convex optimization problems. It provides a high-level interface for defining and solving a wide range of optimization problems, making it a powerful tool for researchers, engineers, and data scientists.

- **User-Friendly Syntax.** CVXPY uses a natural mathematical syntax, making it easy to formulate optimization problems.

```
import cvxpy as cp
x = cp.Variable()
objective = cp.Minimize(x**2)
constraints = [x >= 1]
prob = cp.Problem(objective, constraints)
result = prob.solve()
```

- **Wide Range of Applications.** CVXPY can be used in various fields such as finance, machine learning, control theory, and more. It supports linear programs (LP), quadratic programs (QP), second-order cone programs (SOCP), and semidefinite programs (SDP).
- **Integration with Scientific Libraries.** CVXPY integrates seamlessly with other scientific libraries like NumPy, SciPy, and Pandas, allowing for easy manipulation of data and embedding of optimization problems within larger scientific workflows.

## CVXPY Example

Minimize the function  $f(x, y) = x^2 + y^2$  s.t.  $x + y = 1$   $x - y \geq 1$

```
import cvxpy as cp
```

```
# Define variables
```

```
x = cp.Variable()
```

```
y = cp.Variable()
```

=cp.Variable(2)

```
# Define objective
```

```
objective = cp.Minimize(x**2 + y**2)
```

```
# Define constraints
```

```
constraints = [x + y == 1, x - y >= 1]
```

```
# Form and solve problem
```

```
prob = cp.Problem(objective, constraints)
```

```
result = prob.solve()
```

```
print(f"Optimal value: {result}")
```


```
print(f"Optimal variables: x = {x.value}, y = {y.value}")
```



# CVXPY exercises

- CVXPY Examples

# CVXPY exercises

- CVXPY Examples
-  Exercise

## Basic linear algebra background

## Vectors and matrices

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  $\mathbb{R}^{m \times n}$ . That's it: <sup>2</sup>

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad x^T = [x_1 \quad x_2 \quad \dots \quad x_n] \quad x \in \mathbb{R}^n, x_i \in \mathbb{R} \quad (1)$$

*Handwritten notes:* A blue arrow points to the column vector  $x$ . Below the vector  $x$  is the handwritten text "n x 1". Below the matrix  $x^T$  is the handwritten text "1 x n". To the right of the equation is the handwritten text "x ∈ ℝ<sup>n</sup>".

---

<sup>2</sup>A full introduction to applied linear algebra can 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, a useful refresher for linear algebra is in Appendix A of the book Numerical Optimization by Jorge Nocedal Stephen J. Wright.

## Vectors and matrices

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  $\mathbb{R}^{m \times n}$ . That's it:<sup>2</sup>

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad x^T = [x_1 \quad x_2 \quad \dots \quad x_n] \quad x \in \mathbb{R}^n, x_i \in \mathbb{R} \quad (1)$$

*вектор* →  $x = 1$  ← *вектор*
 $x \geq 0$

$x \succeq 0$   
 $A_{m \times n} \rightarrow A^T_{n \times m}$

Similarly, if  $A \in \mathbb{R}^{m \times n}$  we denote transposition as  $A^T \in \mathbb{R}^{n \times m}$ :

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix}$$

$A \in \mathbb{R}^{m \times n} \quad a_{ij} \in \mathbb{R}$

We will write  $x \geq 0$  and  $x \neq 0$  to indicate componentwise relationships

<sup>2</sup>A full introduction to applied linear algebra can 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, a useful refresher for linear algebra is in Appendix A of the book Numerical Optimization by Jorge Nocedal Stephen J. Wright.

$$\begin{pmatrix} 4 \\ 3 \end{pmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

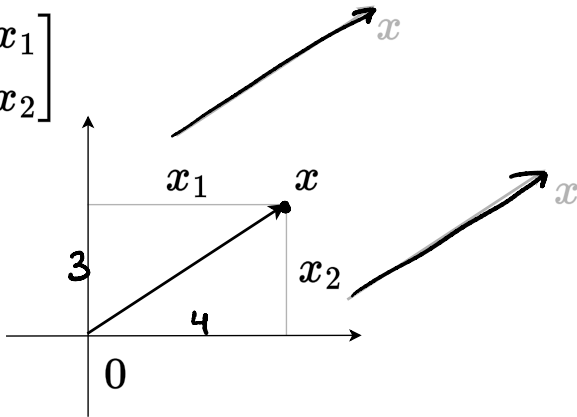
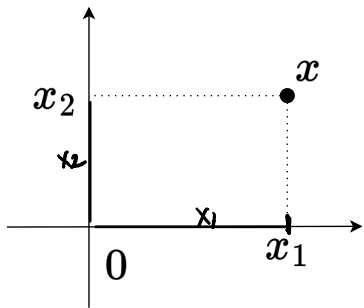


Figure 2: Equivalent representations of a vector

A matrix is symmetric if  $A = A^T$ . It is denoted as  $A \in \mathbb{S}^n$  (set of square symmetric matrices of dimension  $n$ ). Note, that only a square matrix could be symmetric by definition.

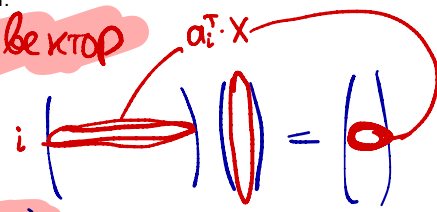
**Умножение матрицы на вектор**

Матрица  $O(n^{\log_2 7})$

$$\log_2 7 < \frac{\log_2 8}{3}$$

$$A x = b$$

$n \times m$     $n \times 1$     $n \times 1$

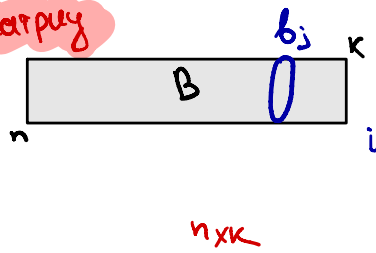
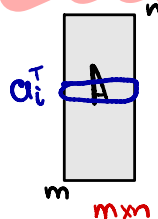


matrix MATRICK  
 $b_i = a_i^T x$   
 $\sum_{j=1}^n a_{ij} x_j$

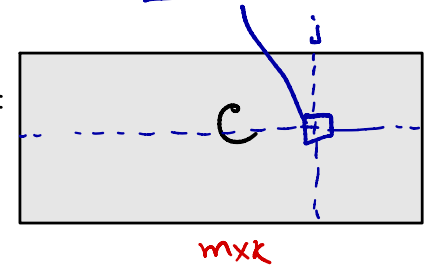
сложность:  $O(n^2)$

**Умножение матриц**

$O(n^3)$



$$c_{ij} = a_i^T \cdot b_j$$



A matrix is symmetric if  $A = A^T$ . It is denoted as  $A \in \mathbb{S}^n$  (set of square symmetric matrices of dimension  $n$ ). Note, that only a square matrix could be symmetric by definition.

A matrix  $A \in \mathbb{S}^n$  is called **positive (negative) definite** if for all  $x \neq 0 : x^T A x > (<) 0$ . We denote this as  $A \succ (<) 0$ . The set of such matrices is denoted as  $\mathbb{S}_{++}^n (\mathbb{S}_{--}^n)$

$$\forall x \in \mathbb{R}^n$$

$$x^T A x > 0$$

$1 \times n$     $n \times n$     $n \times 1$



A matrix is symmetric if  $A = A^T$ . It is denoted as  $A \in \mathbb{S}^n$  (set of square symmetric matrices of dimension  $n$ ). Note, that only a square matrix could be symmetric by definition.

A matrix  $A \in \mathbb{S}^n$  is called **positive (negative) definite** if for all  $x \neq 0 : x^T A x > (<) 0$ . We denote this as  $A \succ (<) 0$ . The set of such matrices is denoted as  $\mathbb{S}_{++}^n (\mathbb{S}_{--}^n)$

A matrix  $A \in \mathbb{S}^n$  is called **positive (negative) semidefinite** if for all  $x : x^T A x \geq (\leq) 0$ . We denote this as  $A \succeq (\preceq) 0$ . The set of such matrices is denoted as  $\mathbb{S}_+^n (\mathbb{S}_-^n)$

Question

Is it correct, that a positive definite matrix has all positive entries?

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

АНАЛОГ ПОЛОЖИТЕЛЬНЫХ ЧИСЕЛ

$$x \underbrace{a}_{1 \times 1} x = ax^2 > 0$$

$a > 0$

A matrix is symmetric if  $A = A^T$ . It is denoted as  $A \in \mathbb{S}^n$  (set of square symmetric matrices of dimension  $n$ ). Note, that only a square matrix could be symmetric by definition.

A matrix  $A \in \mathbb{S}^n$  is called **positive (negative) definite** if for all  $x \neq 0 : x^T Ax > (<)0$ . We denote this as  $A \succ (<)0$ . The set of such matrices is denoted as  $\mathbb{S}_{++}^n (\mathbb{S}_{--}^n)$

A matrix  $A \in \mathbb{S}^n$  is called **positive (negative) semidefinite** if for all  $x : x^T Ax \geq (\leq)0$ . We denote this as  $A \succeq (\preceq)0$ . The set of such matrices is denoted as  $\mathbb{S}_+^n (\mathbb{S}_-^n)$

Question

Is it correct, that a positive definite matrix has all positive entries?

Question

Is it correct, that if a matrix is symmetric it should be positive definite?



A matrix is symmetric if  $A = A^T$ . It is denoted as  $A \in \mathbb{S}^n$  (set of square symmetric matrices of dimension  $n$ ). Note, that only a square matrix could be symmetric by definition.

A matrix  $A \in \mathbb{S}^n$  is called **positive (negative) definite** if for all  $x \neq 0 : x^T A x > (<) 0$ . We denote this as  $A \succ (<) 0$ . The set of such matrices is denoted as  $\mathbb{S}_{++}^n (\mathbb{S}_{--}^n)$

A matrix  $A \in \mathbb{S}^n$  is called **positive (negative) semidefinite** if for all  $x : x^T A x \geq (\leq) 0$ . We denote this as  $A \succeq (\preceq) 0$ . The set of such matrices is denoted as  $\mathbb{S}_+^n (\mathbb{S}_-^n)$

Question

Is it correct, that a positive definite matrix has all positive entries?

Question

Is it correct, that if a matrix is symmetric it should be positive definite?

Question

Is it correct, that if a matrix is positive definite it should be symmetric?

## Matrix product (matmul)

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:

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

This operation in a naive form requires  $\mathcal{O}(n^3)$  arithmetical operations, where  $n$  is usually assumed as the largest dimension of matrices.

## Matrix product (matmul)

matree gas type  
 $n \cdot \log n$

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:

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

This operation in a naive form requires  $\mathcal{O}(n^3)$  arithmetical operations, where  $n$  is usually assumed as the largest dimension of matrices.

### Question

Is it possible to multiply two matrices faster, than  $\mathcal{O}(n^3)$ ? How about  $\mathcal{O}(n^2)$ ,  $\mathcal{O}(n)$ ?

## Matrix by vector product (matvec)

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:

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

This operation in a naive form requires  $\mathcal{O}(n^2)$  arithmetical operations, where  $n$  is usually assumed as the largest dimension of matrices.

Remember, that:

- $C = AB \quad C^T = B^T A^T$

## Matrix by vector product (matvec)

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:

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

This operation in a naive form requires  $\mathcal{O}(n^2)$  arithmetical operations, where  $n$  is usually assumed as the largest dimension of matrices.

Remember, that:

- $C = AB \quad C^T = B^T A^T$
- $AB \neq BA$

## Matrix by vector product (matvec)

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:

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

This operation in a naive form requires  $\mathcal{O}(n^2)$  arithmetical operations, where  $n$  is usually assumed as the largest dimension of matrices.

Remember, that:

- $C = AB \mid C^T = B^T A^T$
- $AB \neq BA$   $m \times k \quad m \times n \quad n \times k \quad k \times m \quad k \times n \quad n \times m$
- $e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k$



## Matrix by vector product (matvec)

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:

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

This operation in a naive form requires  $\mathcal{O}(n^2)$  arithmetical operations, where  $n$  is usually assumed as the largest dimension of matrices.

Remember, that:

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

## Matrix by vector product (matvec)

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:

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

This operation in a naive form requires  $\mathcal{O}(n^2)$  arithmetical operations, where  $n$  is usually assumed as the largest dimension of matrices.

Remember, that:

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

$$\langle x, y \rangle = \sum_{i=1}^n x_i y_i = x^T \cdot y$$
$$\langle x, Ay \rangle = \sum_{i=1}^m x_i (Ay)_i = (x^T A) y = (A^T x)^T y = \langle A^T x, y \rangle$$

$$\langle x, zB \rangle = \langle xB^T, z \rangle$$

# Norms

Norm is a **qualitative measure of the smallness of a vector** and is typically denoted as  $\|x\|$ .

The norm should satisfy certain properties:

1.  $\|\alpha x\| = |\alpha| \|x\|, \alpha \in \mathbb{R}$

# Norms

Norm is a **qualitative measure of the smallness of a vector** and is typically denoted as  $\|x\|$ .

The norm should satisfy certain properties:

1.  $\|\alpha x\| = |\alpha| \|x\|$ ,  $\alpha \in \mathbb{R}$
2.  $\|x + y\| \leq \|x\| + \|y\|$  (triangle inequality)

# Norms

Norm is a **qualitative measure of the smallness of a vector** and is typically denoted as  $\|x\|$ .

The norm should satisfy certain properties:

1.  $\|\alpha x\| = |\alpha| \|x\|$ ,  $\alpha \in \mathbb{R}$
2.  $\|x + y\| \leq \|x\| + \|y\|$  (triangle inequality)
3. If  $\|x\| = 0$  then  $x = 0$

# Norms

Norm is a **qualitative measure of the smallness of a vector** and is typically denoted as  $\|x\|$ .

The norm should satisfy certain properties:

1.  $\|\alpha x\| = |\alpha| \|x\|$ ,  $\alpha \in \mathbb{R}$
2.  $\|x + y\| \leq \|x\| + \|y\|$  (triangle inequality)
3. If  $\|x\| = 0$  then  $x = 0$

# Norms

Norm is a **qualitative measure of the smallness of a vector** and is typically denoted as  $\|x\|$ .

The norm should satisfy certain properties:

1.  $\|\alpha x\| = |\alpha| \|x\|$ ,  $\alpha \in \mathbb{R}$
2.  $\|x + y\| \leq \|x\| + \|y\|$  (triangle inequality)
3. If  $\|x\| = 0$  then  $x = 0$

The distance between two vectors is then defined as

$$d(x, y) = \|x - y\|.$$

The most well-known and widely used norm is **Euclidean norm**:

$$\|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2},$$

which corresponds to the distance in our real life. If the vectors have complex elements, we use their modulus. Euclidean norm, or 2-norm, is a subclass of an important class of  $p$ -norms:

$$\|x\|_p = \left( \sum_{i=1}^n |x_i|^p \right)^{1/p}.$$

## $p$ -norm of a vector

There are two very important special cases. The infinity norm, or Chebyshev norm is defined as the element of the maximal absolute value:

$$\|x\|_{\infty} = \max_i |x_i|$$



## $p$ -norm of a vector

There are two very important special cases. The infinity norm, or Chebyshev norm is defined as the element of the maximal absolute value:

$$\|x\|_{\infty} = \max_i |x_i|$$

$L_1$  norm (or **Manhattan distance**) which is defined as the sum of modules of the elements of  $x$ :

$$\|x\|_1 = \sum_i |x_i|$$

## $p$ -norm of a vector

There are two very important special cases. The infinity norm, or Chebyshev norm is defined as the element of the maximal absolute value:

$$\|x\|_{\infty} = \max_i |x_i|$$

$L_1$  norm (or **Manhattan distance**) which is defined as the sum of modules of the elements of  $x$ :

$$\|x\|_1 = \sum_i |x_i|$$

$L_1$  norm plays a very important role: it all relates to the **compressed sensing** methods that emerged in the mid-00s as one of the most popular research topics. The code for the picture below is available *here*. Check also *this* video.

TO HE HOOMA

Unit disk in the  $p$ -th norm

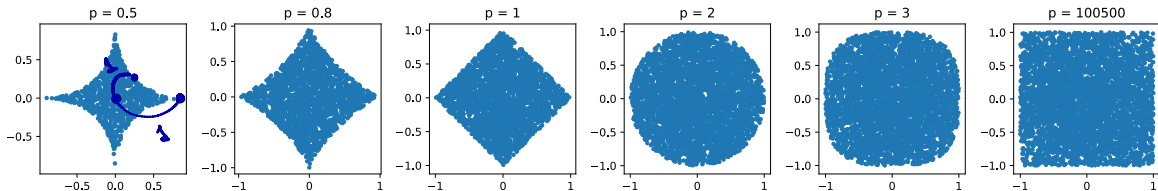


Figure 3: Balls in different norms on a plane

## Matrix norms

In some sense there is no big difference between matrices and vectors (you can vectorize the matrix), and here comes the simplest matrix norm **Frobenius** norm:

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

если векторизовать  
матрицу

$$a = \text{vec}(A)$$

$$\|a\|_2 = \|A\|_F$$

## Matrix norms

In some sense there is no big difference between matrices and vectors (you can vectorize the matrix), and here comes the simplest matrix norm **Frobenius** norm:

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



Spectral norm,  $\|A\|_2$  is one of the most used matrix norms (along with the Frobenius norm).

$$\|A\|_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2},$$

It can not be computed directly from the entries using a simple formula, like the Frobenius norm, however, there are efficient algorithms to compute it. It is directly related to the **singular value decomposition** (SVD) of the matrix. It holds

$$\|A\|_2 = \sigma_1(A) = \sqrt{\lambda_{\max}(A^T A)}$$



where  $\sigma_1(A)$  is the largest singular value of the matrix  $A$ .

## Scalar product

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

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

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

### Example

Prove, that you can switch the position of a matrix inside a scalar product with transposition:  $\langle x, Ay \rangle = \langle A^T x, y \rangle$  and  $\langle x, yB \rangle = \langle xB^T, y \rangle$

# Matrix scalar product

*tr(A) сумма диагональных элементов матрицы*

$$\langle X, X \rangle = \|X\|_F^2$$

$$\left\langle \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix} \right\rangle =$$

$$= \text{tr} \left[ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix} \right] =$$

$$= \text{tr} \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = 5$$

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

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

*↑ матрицы*

## Question

Is there any connection between the Frobenius norm  $\|\cdot\|_F$  and scalar product between matrices  $\langle \cdot, \cdot \rangle$ ?

*для векторов*

$$\langle X, X \rangle = \|X\|_2^2$$

*для векторов*

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} -1 \\ 0 \\ -3 \end{pmatrix}$$

*a*                      *b*

$$\left\langle \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \right\rangle = 1^2 + 2^2 + 3^2 = 14 = \left\| \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \right\|_2^2$$

$$\langle a, b \rangle = 1 \cdot (-1) + 2 \cdot 0 + 3 \cdot (-3) = -10$$

## Eigenvectors and eigenvalues

у  $V$  матрицы есть  $n$   
спектр матрицы  $\rightarrow$  собственных значений  
 $\lambda_1 \dots \lambda_n$

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

$$Aq = \lambda q.$$

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.

# Eigenvectors and eigenvalues

## Theorem

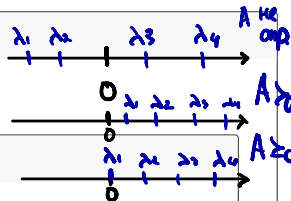
$A \succeq (\succ) 0 \Leftrightarrow$  all eigenvalues of  $A$  are  $\geq (>) 0$

## Proof

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

$$Ax = \lambda x \rightarrow x^T Ax = \lambda x^T x < 0$$

which contradicts the condition of  $A \succeq 0$ .





# Eigenvectors and eigenvalues

## Theorem

$$A \succeq (\succ) 0 \Leftrightarrow \text{all eigenvalues of } A \text{ are } \geq (>) 0$$

## Proof

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

$$Ax = \lambda x \rightarrow x^T Ax = \lambda x^T x < 0$$

which contradicts the condition of  $A \succeq 0$ .

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

$$\begin{aligned} x^T Ax &= (\alpha_1 v_1 + \dots + \alpha_n v_n)^T A (\alpha_1 v_1 + \dots + \alpha_n v_n) \\ &= \sum \alpha_i^2 v_i^T A v_i = \sum \alpha_i^2 \lambda_i v_i^T v_i \geq 0 \end{aligned}$$

here we have used the fact that  $v_i^T v_j = 0$ , for  $i \neq j$ .

## Eigendecomposition (spectral decomposition)

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

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

---

<sup>3</sup>A good cheat sheet with matrix decomposition is available at the NLA course website.

## Eigendecomposition (spectral decomposition)

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  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ . The (real) numbers  $\lambda_i$  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  $A$ .<sup>3</sup>

---

<sup>3</sup>A good cheat sheet with matrix decomposition is available at the NLA course website.

## Eigendecomposition (spectral decomposition)

$$Q^T Q = I$$

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

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

$$(Q) (\Lambda) (Q^T)$$

where  $Q \in \mathbb{R}^{n \times n}$  is orthogonal, i.e., satisfies  $Q^T Q = I$ , and  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ . The (real) numbers  $\lambda_i$  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  $A$ .<sup>3</sup>

We usually order the eigenvalues as  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ . We use the notation  $\lambda_i(A)$  to refer to the  $i$ -th 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  $\lambda_n(A) = \lambda_{\min}(A)$ .

<sup>3</sup>A good cheat sheet with matrix decomposition is available at the NLA course website.

# Eigenvalues

The largest and smallest eigenvalues satisfy

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

# Eigenvalues

The largest and smallest eigenvalues satisfy

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

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

Εστὶν ματρίτσα

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

Καὶ ἐπιζητοῦνται οἱ ἀξιοί  $\lambda(A) = ?$   
ὁποῦντες:

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

ΧΑΡΑΚΤΗΡΙΣΤΙΚΕΣ

ὑποθέτουμε

$$\det(A - \lambda \cdot I) = 0$$

$$\det \left( \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} - \lambda \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right) = 0$$

$$\begin{pmatrix} 1-\lambda & 0 \\ 0 & 2-\lambda \end{pmatrix} = 0 \Rightarrow$$

$$(1-\lambda)(2-\lambda) = 0$$
$$\begin{cases} \lambda = 1 \\ \lambda = 2 \end{cases}$$

## Eigenvalues

The largest and smallest eigenvalues satisfy

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

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

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

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

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

# Eigenvalues

The largest and smallest eigenvalues satisfy

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

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

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

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

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

If we use spectral matrix norm, we can get:

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

If, moreover,  $A \in \mathbb{S}_{++}^n$ :  $\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$



# Singular value decomposition (SVD)

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

не для всех матриц  $A$

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

$$A = U \Sigma V^T$$

## Singular value decomposition

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

$$A = U \Sigma V^T$$

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

$$\begin{aligned} A \cdot x &= b \\ U \Sigma V^T x &= b \quad | \cdot U^T \\ \underbrace{U^T U}_{I} \Sigma V^T x &= U^T b \\ \Sigma V^T x &= U^T b \quad | \Sigma^{-1} \\ \Sigma^{-1} \Sigma V^T x &= \Sigma^{-1} U^T b \\ V^T x &= \Sigma^{-1} U^T b \quad | \cdot V \\ \underbrace{(V V^T)}_I x &= V \Sigma^{-1} U^T b \\ \Rightarrow x &= V \Sigma^{-1} U^T b \end{aligned}$$

## Singular value decomposition

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

$$A = U\Sigma V^T$$

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

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

# Singular value decomposition

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

$$A = U \Sigma V^T$$

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

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

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  $\sigma_i$  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.

Handwritten notes:

- $A = U \Sigma V^T$  (with dimensions  $m \times n$ ,  $m \times r$ ,  $r \times r$ ,  $r \times n$ )
- $A^T = V \Sigma U^T$
- $A \cdot A^T = U \Sigma V^T \cdot V \Sigma U^T = U \Sigma^2 U^T$  (where  $V^T V = I$ )
- $A = Q \Lambda Q^T$
- Сингулярное разложение  $AA^T$
- $\lambda(AA^T) = \sigma^2(A)$
- $a^T b = \text{скаляр}$  (scalar)
- $a b^T = \text{матрица}$  (matrix)

# Singular value decomposition

## Question

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

## Singular value decomposition

$$\sigma(A) = \sqrt{\lambda(A^T A)}$$

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:

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

## 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 the ability to reconstruct the whole matrix exactly.



# 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 the ability to reconstruct the whole matrix exactly.

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.

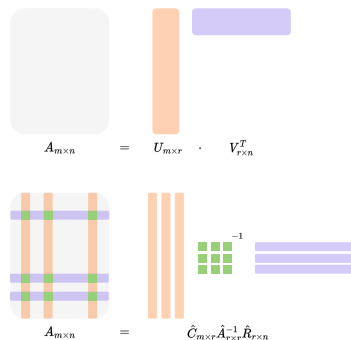


Figure 4: Illustration of Skeleton decomposition

# 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 the ability to reconstruct the whole matrix exactly.

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

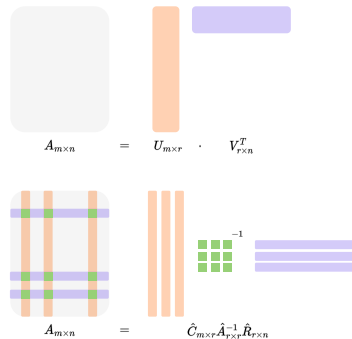


Figure 4: Illustration of Skeleton decomposition

# 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 the ability to reconstruct the whole matrix exactly.

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.

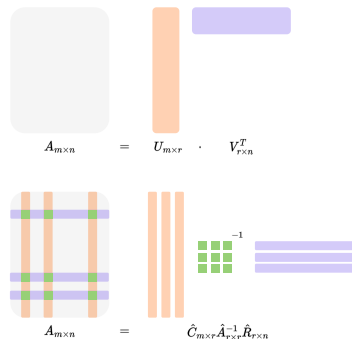


Figure 4: Illustration of Skeleton decomposition

## 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 a sum of  $r$  primitive tensors.

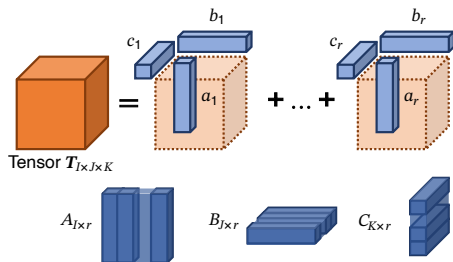


Figure 5: Illustration of Canonical Polyadic decomposition

### Example

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

## Determinant and trace

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

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

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

- $\det A = 0$  if and only if  $A$  is singular;

## Determinant and trace

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

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

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

- $\det A = 0$  if and only if  $A$  is singular;
- $\det AB = (\det A)(\det B)$ ;

## Determinant and trace

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

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

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

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

## Determinant and trace

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

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

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

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



## Determinant and trace

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

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

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 arbitrary matrices  $A, B, C, D$  (assuming, that all dimensions are consistent):

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

## Determinant and trace

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

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

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 arbitrary matrices  $A, B, C, D$  (assuming, that all dimensions are consistent):

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

### Question

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

## First-order Taylor approximation

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

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

Where:

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

## First-order Taylor approximation

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

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

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  $x_0$ .

## First-order Taylor approximation

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

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

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  $x_0$ .

## First-order Taylor approximation

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

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

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  $f_{x_0}^I(x)$  near the point  $x_0$  for simple analysis of some approaches.

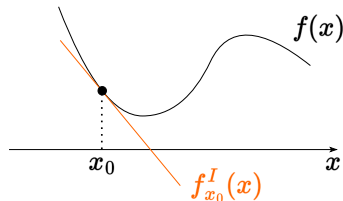


Figure 6: First order Taylor approximation near the point  $x_0$

## 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  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , its second-order Taylor approximation centered at some point  $x_0$  is:

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

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

## 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  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , 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 is 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.

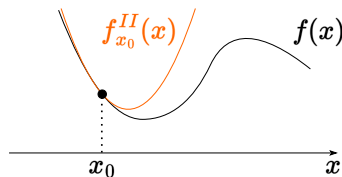


Figure 7: Second order Taylor approximation near the point  $x_0$



# Exercises

- Linear Least Squares

## Exercises

$$A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5 \cdot X = \text{вектор}$$

$n \times n$



- Linear Least Squares
- 🤖 Stupid, but important idea on matrix multiplication

# Exercises

- Linear Least Squares
- 🧠 Stupid, but important idea on matrix multiplication
- 🧠 Problems

# Exercises

- Linear Least Squares
- 🧩 Stupid, but important idea on matrix multiplication
- 🧩 Problems
- How to calculate minimum and maximum eigenvalue of the hessian matrix of linear least squares problem?  
What about binary logistic regression?