Optimization problems. CVXPY. Basic linear algebra recap.

Daniil Merkulov

Applied Math for Data Science. Sberuniversity.

Optimization problems

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
Supply [tons]	550 tons	700 tons	n/a

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

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
Supply [tons]	550 tons	700 tons	n/a

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

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

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
Supply [tons]	550 tons	700 tons	n/a

This can be represented in the following graph:

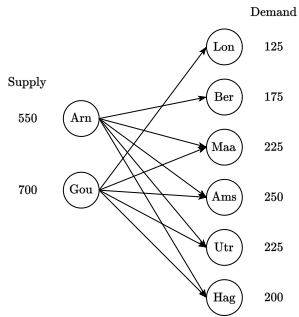

Figure 1: Graph associated with the problem

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

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

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

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

Software

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

$\approx 1.664.510$ x speedup

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

Software

Solving MIP with a modern software on the old hardware

$\approx 2.349.000$ x 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 ≈ 81 speedup on MILP.

[1] R. Bixby report Recent study

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

$\approx 1.664.510$ x speedup

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

Software

Solving MIP with a modern software on the old hardware

$\approx 2.349.000$ x 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 ≈ 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![1]

[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()
```

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).

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. $x + y = 1$ $x - y \geq 1$

```python
import cvxpy as cp

# Define variables
x = cp.Variable()
y = cp.Variable()

# 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: [2]

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

[2] 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: [2]

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

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

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix} \quad A \in \mathbb{R}^{m \times n}, a_{ij} \in \mathbb{R}$$

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

[2] 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.

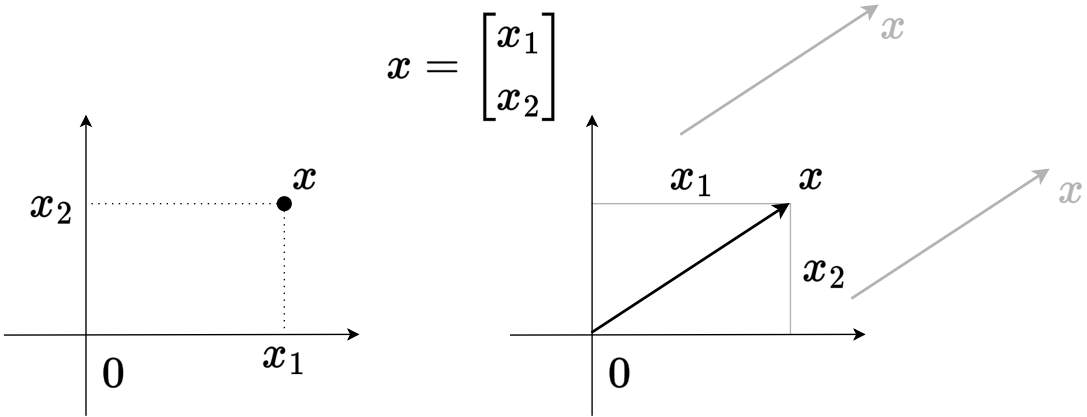

Figure 2: Equivivalent representations of a vector

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.

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 $A \succ (\prec)0$. The set of such matrices is denoted as $\mathbb{S}^n_{++}(\mathbb{S}^n_{--})$

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 $A \succ (\prec)0$. 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?

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 $A \succ (\prec)0$. 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 $x \neq 0 : x^T Ax > (<)0$. We denote this as $A \succ (\prec)0$. 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?

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)

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:

- $C = AB \quad C^T = B^T A^T$
- $AB \neq BA$
- $e^A = \sum\limits_{k=0}^{\infty} \frac{1}{k!} A^k$

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:

- $C = AB \quad C^T = B^T A^T$
- $AB \neq BA$
- $e^A = \sum\limits_{k=0}^{\infty} \frac{1}{k!} A^k$
- $e^{A+B} \neq e^A e^B$ (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:

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

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.

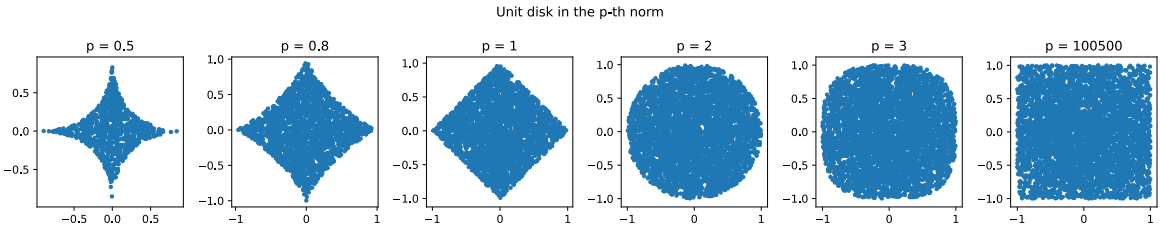

Figure 3: Balls in different norms on a plane

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:

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

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

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

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

Question

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

Eigenvectors and eigenvalues

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

$$Aq = \lambda q.$$

he 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

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

Proof

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

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

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 λ 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 $v_1, \ldots, v_n$ that form an orthogonal basis of $\mathbb{R}^n$. Pick any $x \in \mathbb{R}^n$.

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

   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

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

[3] 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 $Q^TQ = I$, and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. 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. [3]

[3] 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 $\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_n)$. 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. [3]

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 i-th largest eigenvalue of $A \in S$. We usually write the largest or maximum eigenvalue as $\lambda_1(A) = \lambda_{\mathsf{max}}(A)$, and the least or minimum eigenvalue as $\lambda_n(A) = \lambda_{\mathsf{min}}(A)$.

[3] A good cheat sheet with matrix decomposition is available at the NLA course website.

Eigenvalues

The largest and smallest eigenvalues satisfy

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

Eigenvalues

The largest and smallest eigenvalues satisfy

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

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

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

Eigenvalues

The largest and smallest eigenvalues satisfy

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

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

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

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, $A \in \mathbb{S}^n_{++}$: $\kappa(A) = \dfrac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$

Singular value decomposition

Suppose $A \in \mathbb{R}^{m \times n}$ with 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 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 Σ is a diagonal matrix with $\Sigma = \text{diag}(\sigma_1, ..., \sigma_r)$, such that

Singular value decomposition

Suppose $A \in \mathbb{R}^{m \times n}$ with 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 Σ is a diagonal matrix with $\Sigma = \text{diag}(\sigma_1, ..., \sigma_r)$, such that

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

Singular value decomposition

Suppose $A \in \mathbb{R}^{m \times n}$ with 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 Σ is a diagonal matrix with $\Sigma = \text{diag}(\sigma_1, ..., \sigma_r)$, such that

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

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 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.

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

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.

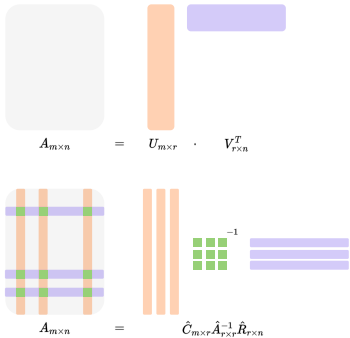

Figure 4: Illustration of Skeleton decomposition

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

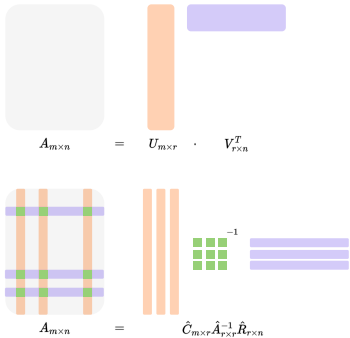

Figure 4: Illustration of Skeleton decomposition

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.

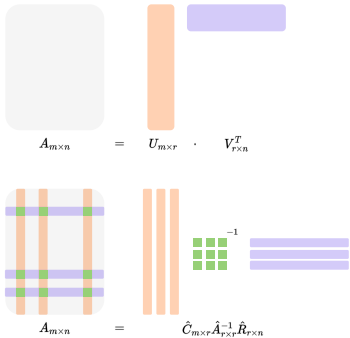

Figure 4: Illustration of Skeleton decomposition

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.

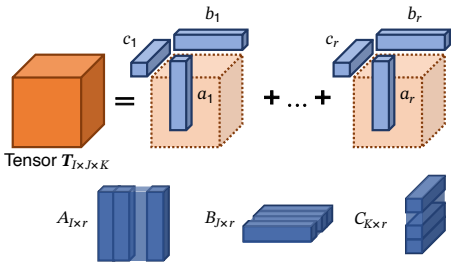

Figure 5: Illustration of Canonical Polyadic decomposition

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

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

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

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

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

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

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

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

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;
- $\det AB = (\det A)(\det B)$;
- $\det A^{-1} = \frac{1}{\det A}$.

Determinant and trace

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

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

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):

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

Determinant and trace

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

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

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 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 $f : \mathbb{R}^n \to \mathbb{R}$ 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.

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.

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.

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.

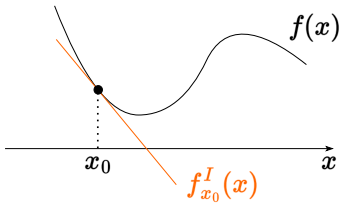

Figure 6: First order Taylor approximation near the point x_0

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 $f : \mathbb{R}^n \to \mathbb{R}$, 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.

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 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.

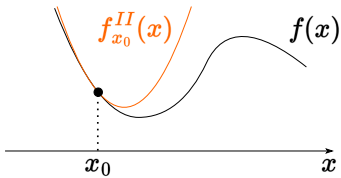

Figure 7: Second order Taylor approximation near the point x_0

Exercises

- Linear Least Squares

Exercises

- 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?