

Background refresher

Complex numbers (\mathbb{C}), Linear algebra with \mathbb{C} , Computability- and Complexity theory



Casper Gyurik

Leiden Institute of Advanced Computer Science
September 5th, 2019



**Universiteit
Leiden**
The Netherlands

Linear algebra with complex numbers

The vector space \mathbb{R}^n

Matrices on \mathbb{R}^n

Inner products, orthogonality and orthonormality

Complex numbers

What are complex numbers?

Polar notation of complex numbers

Unitary and Hermitian operators/matrices

Computability theory

Complexity Theory

The n -dimensional vector space \mathbb{R}^n

vectors, componentwise addition and scalar multiplication

Definition (The n -dimensional vector space \mathbb{R}^n and addition)

► Elements of \mathbb{R}^n are *vectors*: $\vec{v} = \begin{pmatrix} v_0 \\ \vdots \\ v_{n-1} \end{pmatrix}$, where $v_i \in \mathbb{R}$.

► *Componentwise addition*: $\begin{pmatrix} v_0 \\ \vdots \\ v_{n-1} \end{pmatrix} + \begin{pmatrix} w_0 \\ \vdots \\ w_{n-1} \end{pmatrix} = \begin{pmatrix} v_0 + w_0 \\ \vdots \\ v_{n-1} + w_{n-1} \end{pmatrix}$.

► *Scalar multiplication*: $\alpha \begin{pmatrix} v_0 \\ \vdots \\ v_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha v_0 \\ \vdots \\ \alpha v_{n-1} \end{pmatrix}$, where $\alpha \in \mathbb{R}$.

The n -dimensional vector space \mathbb{R}^n

The canonical basis

Definition (Basis of a vector space)

A set of vectors $\{\vec{w}_0, \vec{w}_1, \dots, \vec{w}_{n-1}\}$ is called a *basis* of \mathbb{R}^n if any vector $\vec{v} \in \mathbb{R}^n$ can be *uniquely* written as a sum of scalar multiples of the \vec{w}_i .

The n -dimensional vector space \mathbb{R}^n

The canonical basis

Definition (Basis of a vector space)

A set of vectors $\{\vec{w}_0, \vec{w}_1, \dots, \vec{w}_{n-1}\}$ is called a *basis* of \mathbb{R}^n if any vector $\vec{v} \in \mathbb{R}^n$ can be *uniquely* written as a sum of scalar multiples of the \vec{w}_i .

Example (The canonical basis vectors)

We can write any vector \vec{v} as the following sum of scalar multiples

$$\begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{pmatrix} = v_0 \overbrace{\begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}}^{\vec{e}_0} + v_1 \overbrace{\begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}}^{\vec{e}_1} + \dots + v_{n-1} \overbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}}^{\vec{e}_{n-1}}.$$

The n -dimensional vector space \mathbb{R}^n

The canonical basis

Definition (Basis of a vector space)

A set of vectors $\{\vec{w}_0, \vec{w}_1, \dots, \vec{w}_{n-1}\}$ is called a *basis* of \mathbb{R}^n if any vector $\vec{v} \in \mathbb{R}^n$ can be *uniquely* written as a sum of scalar multiples of the \vec{w}_i .

Example (The canonical basis vectors)

We can write any vector \vec{v} as the following sum of scalar multiples

$$\begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{pmatrix} = v_0 \overbrace{\begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}}^{\vec{e}_0} + v_1 \overbrace{\begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}}^{\vec{e}_1} + \dots + v_{n-1} \overbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}}^{\vec{e}_{n-1}}.$$

The set $\{\vec{e}_0, \vec{e}_1, \dots, \vec{e}_{n-1}\}$ is called the *canonical basis*.

Matrices on \mathbb{R}^n

Matrices and their operations

Definition (Matrices on \mathbb{R}^n)

- ▶ A matrix A is an array of real numbers. For example

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & k \end{pmatrix}, \text{ where } a, b, c, \dots, k \in \mathbb{R}.$$

Matrices on \mathbb{R}^n

Matrices and their operations

Definition (Matrices on \mathbb{R}^n)

- ▶ A matrix A is an array of real numbers. For example

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & k \end{pmatrix}, \text{ where } a, b, c, \dots, k \in \mathbb{R}.$$

- ▶ Addition of two matrices is componentwise

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} + \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix} = \begin{pmatrix} a_1 + a_2 & b_1 + b_2 & c_1 + c_2 \\ d_1 + d_2 & e_1 + e_2 & f_1 + f_2 \\ g_1 + g_2 & h_1 + h_2 & k_1 + k_2 \end{pmatrix}.$$

Matrices on \mathbb{R}^n

Matrices and their operations

Definition (Matrices on \mathbb{R}^n)

- ▶ A matrix A is an array of real numbers. For example

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & k \end{pmatrix}, \text{ where } a, b, c, \dots, k \in \mathbb{R}.$$

- ▶ Addition of two matrices is componentwise

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} + \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix} = \begin{pmatrix} a_1 + a_2 & b_1 + b_2 & c_1 + c_2 \\ d_1 + d_2 & e_1 + e_2 & f_1 + f_2 \\ g_1 + g_2 & h_1 + h_2 & k_1 + k_2 \end{pmatrix}.$$

- ▶ The transpose of a matrix is given by

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & k \end{pmatrix}^T = \begin{pmatrix} a & d & g \\ b & e & h \\ c & f & k \end{pmatrix}$$

- ▶ A matrix M is *symmetric* if $M^T = M$.

Matrices on \mathbb{R}^n and the matrix-vector product

The matrix-vector product and its linearity

Definition (Matrix-vector product)

- ▶ You multiply a vector and a matrix as follows

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & k \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} av_0 + bv_1 + cv_2 \\ dv_0 + ev_1 + fv_2 \\ gv_0 + hv_1 + kv_2 \end{pmatrix}$$

Definition (Linear operations on \mathbb{R}^n)

A map $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is called *linear* if for all $v, w \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$:

$$f(\alpha v + \beta w) = \alpha f(v) + \beta f(w).$$

Matrices on \mathbb{R}^n and the matrix-vector product

The matrix-vector product and its linearity

Definition (Matrix-vector product)

- ▶ You multiply a vector and a matrix as follows

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & k \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} av_0 + bv_1 + cv_2 \\ dv_0 + ev_1 + fv_2 \\ gv_0 + hv_1 + kv_2 \end{pmatrix}$$

Definition (Linear operations on \mathbb{R}^n)

A map $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is called *linear* if for all $v, w \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$:

$$f(\alpha v + \beta w) = \alpha f(v) + \beta f(w).$$

- ▶ Matrix-vector multiplication is linear, i.e., for any $v, w \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$:

$$A(\alpha v + \beta w) = \alpha Av + \beta Aw.$$

Operations on matrices on \mathbb{R}^n

Multiplication of matrices

Definition (Multiplication of matrices)

Multiplication of matrices goes as follows

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix}$$
$$= \begin{pmatrix} a_1a_2 + b_1d_2 + c_1g_2 & a_1b_2 + b_1e_2 + c_1h_2 & a_1c_2 + b_1f_2 + c_1k_2 \\ a_2d_1 + d_2e_1 + f_1g_2 & b_2d_1 + e_2e_1 + f_1h_2 & c_2d_1 + e_1f_2 + f_1k_2 \\ a_2g_1 + d_2h_1 + g_2k_1 & b_2g_1 + e_2h_1 + h_2k_1 & c_2g_1 + f_2h_1 + k_1k_2 \end{pmatrix}$$

Operations on matrices on \mathbb{R}^n

Multiplication of matrices

Definition (Multiplication of matrices)

Multiplication of matrices goes as follows

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix} \\ = \begin{pmatrix} a_1a_2 + b_1d_2 + c_1g_2 & a_1b_2 + b_1e_2 + c_1h_2 & a_1c_2 + b_1f_2 + c_1k_2 \\ a_2d_1 + d_2e_1 + f_1g_2 & b_2d_1 + e_2e_1 + f_1h_2 & c_2d_1 + e_1f_2 + f_1k_2 \\ a_2g_1 + d_2h_1 + g_2k_1 & b_2g_1 + e_2h_1 + h_2k_1 & c_2g_1 + f_2h_1 + k_1k_2 \end{pmatrix}$$

Example (Matrix-matrix multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 3 \end{pmatrix} \\ \cdot = 1 \cdot 1 + 2 \cdot -2 = -3$$

Operations on matrices on \mathbb{R}^n

Multiplication of matrices

Definition (Multiplication of matrices)

Multiplication of matrices goes as follows

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix} \\ = \begin{pmatrix} a_1a_2 + b_1d_2 + c_1g_2 & a_1b_2 + b_1e_2 + c_1h_2 & a_1c_2 + b_1f_2 + c_1k_2 \\ a_2d_1 + d_2e_1 + f_1g_2 & b_2d_1 + e_2e_1 + f_1h_2 & c_2d_1 + e_1f_2 + f_1k_2 \\ a_2g_1 + d_2h_1 + g_2k_1 & b_2g_1 + e_2h_1 + h_2k_1 & c_2g_1 + f_2h_1 + k_1k_2 \end{pmatrix}$$

Example (Matrix-matrix multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 3 \\ -3 & . \\ . & . \\ . & . \end{pmatrix} \\ . = 1 \cdot 2 + 2 \cdot 3 = 8$$

Operations on matrices on \mathbb{R}^n

Multiplication of matrices

Definition (Multiplication of matrices)

Multiplication of matrices goes as follows

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix} \\ = \begin{pmatrix} a_1a_2 + b_1d_2 + c_1g_2 & a_1b_2 + b_1e_2 + c_1h_2 & a_1c_2 + b_1f_2 + c_1k_2 \\ a_2d_1 + d_2e_1 + f_1g_2 & b_2d_1 + e_2e_1 + f_1h_2 & c_2d_1 + e_1f_2 + f_1k_2 \\ a_2g_1 + d_2h_1 + g_2k_1 & b_2g_1 + e_2h_1 + h_2k_1 & c_2g_1 + f_2h_1 + k_1k_2 \end{pmatrix}$$

Example (Matrix-matrix multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 3 \\ -3 & 8 \\ \cdot & \cdot \\ \cdot & \cdot \end{pmatrix} \\ \cdot = 3 \cdot 1 + 4 \cdot -2 = -5$$

Operations on matrices on \mathbb{R}^n

Multiplication of matrices

Definition (Multiplication of matrices)

Multiplication of matrices goes as follows

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix} \\ = \begin{pmatrix} a_1a_2 + b_1d_2 + c_1g_2 & a_1b_2 + b_1e_2 + c_1h_2 & a_1c_2 + b_1f_2 + c_1k_2 \\ a_2d_1 + d_2e_1 + f_1g_2 & b_2d_1 + e_2e_1 + f_1h_2 & c_2d_1 + e_1f_2 + f_1k_2 \\ a_2g_1 + d_2h_1 + g_2k_1 & b_2g_1 + e_2h_1 + h_2k_1 & c_2g_1 + f_2h_1 + k_1k_2 \end{pmatrix}$$

Example (Matrix-matrix multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 3 \\ -3 & 8 \\ -5 & . \\ . & . \end{pmatrix} \\ . = 3 \cdot 2 + 4 \cdot 3 = 18$$

Operations on matrices on \mathbb{R}^n

Multiplication of matrices

Definition (Multiplication of matrices)

Multiplication of matrices goes as follows

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix}$$
$$= \begin{pmatrix} a_1a_2 + b_1d_2 + c_1g_2 & a_1b_2 + b_1e_2 + c_1h_2 & a_1c_2 + b_1f_2 + c_1k_2 \\ a_2d_1 + d_2e_1 + f_1g_2 & b_2d_1 + e_2e_1 + f_1h_2 & c_2d_1 + e_1f_2 + f_1k_2 \\ a_2g_1 + d_2h_1 + g_2k_1 & b_2g_1 + e_2h_1 + h_2k_1 & c_2g_1 + f_2h_1 + k_1k_2 \end{pmatrix}$$

Example (Matrix-matrix multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 3 \\ -3 & 8 \\ -5 & 18 \\ \cdot & \cdot \end{pmatrix}$$
$$\cdot = 5 \cdot 1 + 6 \cdot -2 = -7$$

Operations on matrices on \mathbb{R}^n

Multiplication of matrices

Definition (Multiplication of matrices)

Multiplication of matrices goes as follows

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix}$$
$$= \begin{pmatrix} a_1a_2 + b_1d_2 + c_1g_2 & a_1b_2 + b_1e_2 + c_1h_2 & a_1c_2 + b_1f_2 + c_1k_2 \\ a_2d_1 + d_2e_1 + f_1g_2 & b_2d_1 + e_2e_1 + f_1h_2 & c_2d_1 + e_1f_2 + f_1k_2 \\ a_2g_1 + d_2h_1 + g_2k_1 & b_2g_1 + e_2h_1 + h_2k_1 & c_2g_1 + f_2h_1 + k_1k_2 \end{pmatrix}$$

Example (Matrix-matrix multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 3 \\ -3 & 8 \\ -5 & 18 \\ -7 & . \end{pmatrix}$$
$$. = 5 \cdot 2 + 6 \cdot 3 = 28$$

Operations on matrices on \mathbb{R}^n

Multiplication of matrices

Definition (Multiplication of matrices)

Multiplication of matrices goes as follows

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & k_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & k_2 \end{pmatrix} \\ = \begin{pmatrix} a_1a_2 + b_1d_2 + c_1g_2 & a_1b_2 + b_1e_2 + c_1h_2 & a_1c_2 + b_1f_2 + c_1k_2 \\ a_2d_1 + d_2e_1 + f_1g_2 & b_2d_1 + e_2e_1 + f_1h_2 & c_2d_1 + e_1f_2 + f_1k_2 \\ a_2g_1 + d_2h_1 + g_2k_1 & b_2g_1 + e_2h_1 + h_2k_1 & c_2g_1 + f_2h_1 + k_1k_2 \end{pmatrix}$$

Example (Matrix-matrix multiplication)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 3 \\ -3 & 8 \\ -5 & 18 \\ -7 & 28 \end{pmatrix}$$

Operations on matrices on \mathbb{R}^n

Kronecker product of matrices

Definition (Kronecker product of matrices)

Let A and B be two matrix. Their Kronecker product is given by

$$A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B & a_{0,2}B & \dots & a_{0,n-1}B \\ a_{1,0}B & a_{1,1}B & a_{1,2}B & \dots & a_{1,n-1}B \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0}B & a_{m-1,1}B & a_{m-1,2}B & \dots & a_{m-1,n-1}B \end{pmatrix}$$

Operations on matrices on \mathbb{R}^n

Kronecker product of matrices

Definition (Kronecker product of matrices)

Let A and B be two matrix. Their Kronecker product is given by

$$A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B & a_{0,2}B & \dots & a_{0,n-1}B \\ a_{1,0}B & a_{1,1}B & a_{1,2}B & \dots & a_{1,n-1}B \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0}B & a_{m-1,1}B & a_{m-1,2}B & \dots & a_{m-1,n-1}B \end{pmatrix}$$

Example

Let $A = \begin{pmatrix} 1 & 2 \\ -1 & 0 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Then we have

$$A \otimes B = \begin{pmatrix} 1B & 2B \\ -1B & 0B \end{pmatrix} = \begin{pmatrix} 1 & 2 & 2 & 4 \\ 3 & 4 & 6 & 8 \\ -1 & -2 & 0 & 0 \\ -3 & -4 & 0 & 0 \end{pmatrix}$$

The inner product and the norm

How to compute them

Definition (The inner product and the norm)

► *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.

The inner product and the norm

How to compute them

Definition (The inner product and the norm)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.

The inner product and the norm

How to compute them

Definition (The inner product and the norm)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.

Example

▶ $\left\langle \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} -3 \\ 1 \end{pmatrix} \right\rangle = 1 \cdot -3 + 2 = -1$.

The inner product and the norm

How to compute them

Definition (The inner product and the norm)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.

Example

- ▶ $\left\langle \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} -3 \\ 1 \end{pmatrix} \right\rangle = 1 \cdot -3 + 2 = -1$.
- ▶ $\left\| \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right\| = \sqrt{1^2 + 2^2} = \sqrt{5}$.

The inner product and the norm

When are vectors orthogonal or orthonormal?

Definition (The inner product, the norm and orthogonality)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.

The inner product and the norm

When are vectors orthogonal or orthonormal?

Definition (The inner product, the norm and orthogonality)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.
- ▶ \vec{v} and \vec{w} are called *orthogonal* if $\langle v, w \rangle = 0$.

The inner product and the norm

When are vectors orthogonal or orthonormal?

Definition (The inner product, the norm and orthogonality)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.
- ▶ \vec{v} and \vec{w} are called *orthogonal* if $\langle v, w \rangle = 0$.
- ▶ $\{\vec{v}_0, \dots, \vec{v}_{n-1}\}$ is called *orthonormal* if the \vec{v}_i are pairwise orthogonal and satisfy $\|v_i\| = 1$.

The inner product and the norm

When are vectors orthogonal or orthonormal?

Definition (The inner product, the norm and orthogonality)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.
- ▶ \vec{v} and \vec{w} are called *orthogonal* if $\langle v, w \rangle = 0$.
- ▶ $\{\vec{v}_0, \dots, \vec{v}_{n-1}\}$ is called *orthonormal* if the \vec{v}_i are pairwise orthogonal and satisfy $\|v_i\| = 1$.

The inner product and the norm

When are vectors orthogonal or orthonormal?

Definition (The inner product, the norm and orthogonality)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.
- ▶ \vec{v} and \vec{w} are called *orthogonal* if $\langle v, w \rangle = 0$.
- ▶ $\{\vec{v}_0, \dots, \vec{v}_{n-1}\}$ is called *orthonormal* if the \vec{v}_i are pairwise orthogonal and satisfy $\|v_i\| = 1$.

Example

The vectors $\vec{v} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ and $\vec{w} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$ are orthonormal.

▶ $\langle v, w \rangle = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} \cdot \frac{-1}{\sqrt{2}} = \frac{1}{2} - \frac{1}{2} = 0$.

The inner product and the norm

When are vectors orthogonal or orthonormal?

Definition (The inner product, the norm and orthogonality)

- ▶ *Inner product* of vectors: $\langle v, w \rangle = \sum_{i=0}^{n-1} v_i w_i$.
- ▶ *The norm* of a vector: $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=0}^{n-1} v_i^2}$.
- ▶ \vec{v} and \vec{w} are called *orthogonal* if $\langle v, w \rangle = 0$.
- ▶ $\{\vec{v}_0, \dots, \vec{v}_{n-1}\}$ is called *orthonormal* if the \vec{v}_i are pairwise orthogonal and satisfy $\|v_i\| = 1$.

Example

The vectors $\vec{v} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ and $\vec{w} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$ are orthonormal.

- ▶ $\langle v, w \rangle = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} \cdot \frac{-1}{\sqrt{2}} = \frac{1}{2} - \frac{1}{2} = 0$.
- ▶ $\|\vec{v}\| = \sqrt{|\frac{1}{\sqrt{2}}|^2 + |\frac{1}{\sqrt{2}}|^2} = 1$ and $\|\vec{w}\| = \sqrt{|\frac{1}{\sqrt{2}}|^2 + |-\frac{1}{\sqrt{2}}|^2} = 1$.

From real to complex

Changing the entries of the vectors and matrices

Why do we need all this linear algebra?

From real to complex

Changing the entries of the vectors and matrices

Why do we need all this linear algebra?

- ▶ Classical computation is usually described as follows:
 - ▶ state of the computer \leftrightarrow bitstring.
 - ▶ operations allowed by computer \leftrightarrow logical gates.

From real to complex

Changing the entries of the vectors and matrices

Why do we need all this linear algebra?

- ▶ Classical computation is usually described as follows:
 - ▶ state of the computer \leftrightarrow bitstring.
 - ▶ operations allowed by computer \leftrightarrow logical gates.
- ▶ In quantum computing we describe these as follows:
 - ▶ state of the quantum computer \leftrightarrow *complex* vector.
 - ▶ operations allowed by quantum computer \leftrightarrow *complex* matrices.

From real to complex

Changing the entries of the vectors and matrices

Why do we need all this linear algebra?

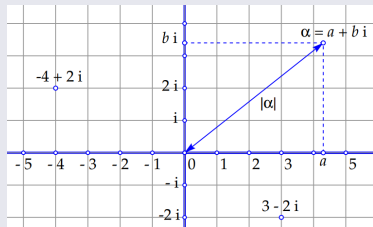
- ▶ Classical computation is usually described as follows:
 - ▶ state of the computer \leftrightarrow bitstring.
 - ▶ operations allowed by computer \leftrightarrow logical gates.
- ▶ In quantum computing we describe these as follows:
 - ▶ state of the quantum computer \leftrightarrow *complex* vector.
 - ▶ operations allowed by quantum computer \leftrightarrow *complex* matrices.

A vector/matrix is complex if their entries are *complex numbers*.

Complex numbers

Points on the complex plane

Complex numbers and the complex plane



► *Complex number: $\alpha = a + bi$.*

Figure: Taken from "An introduction to complex number, Jan van de Craats"

Complex numbers

Points on the complex plane

Complex numbers and the complex plane

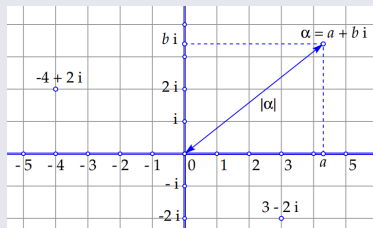


Figure: Taken from "An introduction to complex number, Jan van de Craats"

- ▶ *Complex number:* $\alpha = a + bi$.
- ▶ *Imaginary unit:* $i^2 = -1$.

Complex numbers

Points on the complex plane

Complex numbers and the complex plane

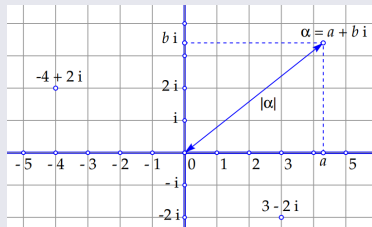


Figure: Taken from "An introduction to complex number, Jan van de Craats"

- ▶ *Complex number:* $\alpha = a + bi$.
- ▶ *Imaginary unit:* $i^2 = -1$.
- ▶ *Conjugate:* $\bar{\alpha} = a - bi$.

Complex numbers

Points on the complex plane

Complex numbers and the complex plane

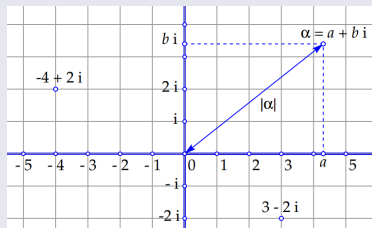


Figure: Taken from "An introduction to complex number, Jan van de Craats"

- ▶ *Complex number:* $\alpha = a + bi$.
- ▶ *Imaginary unit:* $i^2 = -1$.
- ▶ *Conjugate:* $\bar{\alpha} = a - bi$.
- ▶ *Norm:* $|\alpha| = \sqrt{a^2 + b^2}$.

Complex numbers

Points on the complex plane

Complex numbers and the complex plane

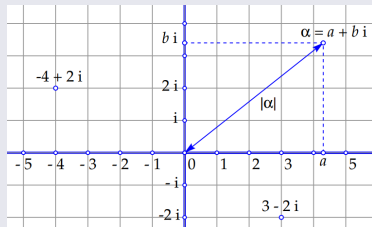


Figure: Taken from "An introduction to complex number, Jan van de Craats"

- ▶ *Complex number:* $\alpha = a + bi$.
- ▶ *Imaginary unit:* $i^2 = -1$.
- ▶ *Conjugate:* $\bar{\alpha} = a - bi$.
- ▶ *Norm:* $|\alpha| = \sqrt{a^2 + b^2}$.

Definition (Addition and multiplication of complex numbers)

Let $\alpha = a + bi \in \mathbb{C}$ and $\beta = c + di \in \mathbb{C}$.

- ▶ $\alpha + \beta = (a + c) + (b + d)i$.
- ▶ $\alpha \cdot \beta = (ac - bd) + (ad + bc)i$.

Polar notation of complex numbers

A different way of expressing a complex number

The complex plane and polar notation

Consider $z = x + iy \in \mathbb{C}$.

We can rewrite z in *polar notation* as

$$z = r e^{i\varphi},$$

where we have

$$r = |z| = \sqrt{x^2 + y^2}$$

and

$$\varphi = \tan^{-1}\left(\frac{y}{x}\right) \in [0, 2\pi).$$

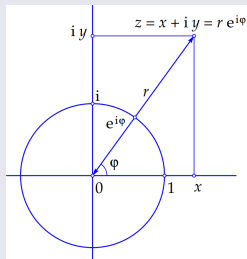


Figure: Taken from “An introduction to complex number, Jan van de Craats”

Polar notation of complex numbers

A different way of expressing a complex number

The complex plane and polar notation

Consider $z = x + iy \in \mathbb{C}$.

We can rewrite z in *polar notation* as

$$z = r e^{i\varphi},$$

where we have

$$r = |z| = \sqrt{x^2 + y^2}$$

and

$$\varphi = \tan^{-1}\left(\frac{y}{x}\right) \in [0, 2\pi).$$

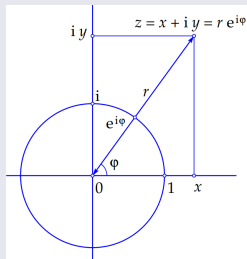


Figure: Taken from “An introduction to complex number, Jan van de Craats”

Example

Let $x = 1 + i$, we can rewrite it in polar notation as

$$\blacktriangleright x = \sqrt{1^2 + 1^2} \cdot e^{i \tan^{-1}(1)} = \sqrt{2} e^{i\pi/4}.$$

References for Complex Linear Algebra

Where can I find more?

For more on complex linear algebra see for example

- ▶ https://cds.cern.ch/record/1522001/files/978-1-4614-6336-8_BookBackMatter.pdf
- ▶ <https://pdfs.semanticscholar.org/6dc6/18f0041a20d8c722ed2cdd8c4057d6e92a0b.pdf>
- ▶ <http://home.iitk.ac.in/~aralal/book/nptel/pdf/booklinear.html>

Unitary and Hermitian matrices

A special kind of matrices

In quantum computing we often deal with special kinds of matrices.

Definition (Unitary and Hermitian matrices)

- ▶ A complex matrix U is called unitary if it is *norm-preserving*, i.e.,

$$\|A\vec{v}\| = \|\vec{v}\|, \text{ for any } \vec{v}.$$

Unitary and Hermitian matrices

A special kind of matrices

In quantum computing we often deal with special kinds of matrices.

Definition (Unitary and Hermitian matrices)

- ▶ A complex matrix U is called unitary if it is *norm-preserving*, i.e.,

$$\|A\vec{v}\| = \|\vec{v}\|, \text{ for any } \vec{v}.$$

- ▶ A complex matrix H is called Hermitian if $\overline{H}^T = H$.

What is computation?

bits and logical gates

- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.

What is computation?

bits and logical gates

- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.
 - C-program → Assembly → Bit manipulations via logical gates.

What is computation?

bits and logical gates

- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.
 - C-program → Assembly → Bit manipulations via logical gates.
- ▶ These bit manipulations compute functions from bitstrings to bitstrings.

What is computation?

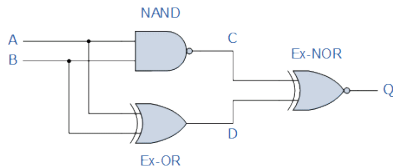
bits and logical gates

- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.
 - C-program \rightarrow Assembly \rightarrow Bit manipulations via logical gates.
- ▶ These bit manipulations compute functions from bitstrings to bitstrings.
 - These $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ are called *Boolean functions*.

What is computation?

bits and logical gates

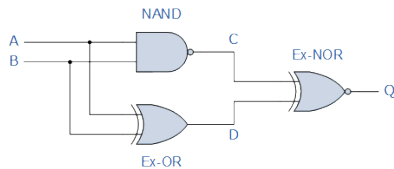
- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.
 - C-program \rightarrow Assembly \rightarrow Bit manipulations via logical gates.
- ▶ These bit manipulations compute functions from bitstrings to bitstrings.
 - These $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ are called *Boolean functions*.
- ▶ These sequences of logical gates are called *Boolean circuits*.



What is computation?

bits and logical gates

- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.
 - C-program \rightarrow Assembly \rightarrow Bit manipulations via logical gates.
- ▶ These bit manipulations compute functions from bitstrings to bitstrings.
 - These $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ are called *Boolean functions*.
- ▶ These sequences of logical gates are called *Boolean circuits*.

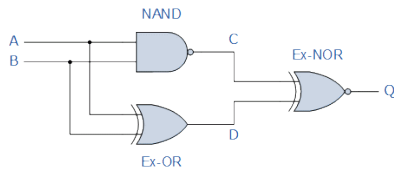


- Boolean circuits are an example of a *model of computation*.

What is computation?

bits and logical gates

- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.
 - C-program \rightarrow Assembly \rightarrow Bit manipulations via logical gates.
- ▶ These bit manipulations compute functions from bitstrings to bitstrings.
 - These $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ are called *Boolean functions*.
- ▶ These sequences of logical gates are called *Boolean circuits*.

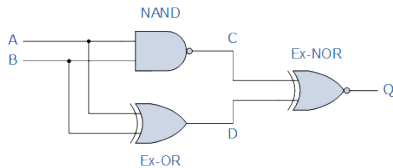


- Boolean circuits are an example of a *model of computation*.
- Others include Turing machines and automata.

What is computation?

bits and logical gates

- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.
 - C-program \rightarrow Assembly \rightarrow Bit manipulations via logical gates.
- ▶ These bit manipulations compute functions from bitstrings to bitstrings.
 - These $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ are called *Boolean functions*.
- ▶ These sequences of logical gates are called *Boolean circuits*.

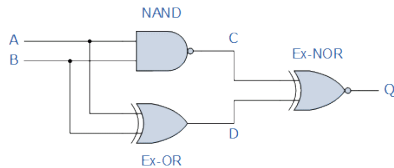


- Boolean circuits are an example of a *model of computation*.
 - Others include Turing machines and automata.
- ▶ Conclusion: classical computation is the use of Boolean circuits to compute Boolean functions.

What is computation?

bits and logical gates

- ▶ Down the line: computation on your laptop is a sequence of bit manipulations via logical gates.
 - C-program \rightarrow Assembly \rightarrow Bit manipulations via logical gates.
- ▶ These bit manipulations compute functions from bitstrings to bitstrings.
 - These $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ are called *Boolean functions*.
- ▶ These sequences of logical gates are called *Boolean circuits*.



- Boolean circuits are an example of a *model of computation*.
 - Others include Turing machines and automata.
- ▶ Conclusion: classical computation is the use of Boolean circuits to compute Boolean functions.
 - ▶ What happens if we change the model to quantum computers?
 - ▶ Do things change for computability and complexity theory?

Computability theory

What can a computer is compute?

Computability theory: What kind of problems can a computer solve?

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.
 - E.g., does this sudoku have a solution if we fill in 6 in the (i, j) th square?

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.
 - E.g., does this sudoku have a solution if we fill in 6 in the (i, j) th square?
 - Using this decision problem we can still find the solution to the Sudoku.

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.
 - E.g., does this sudoku have a solution if we fill in 6 in the (i, j) th square?
 - Using this decision problem we can still find the solution to the Sudoku.
- ▶ Computer can solve a sudoku. Is there a problem a computer can't solve?

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.
 - E.g., does this sudoku have a solution if we fill in 6 in the (i, j) th square?
 - Using this decision problem we can still find the solution to the Sudoku.
- ▶ Computer can solve a sudoku. Is there a problem a computer can't solve?
 - ▶ The answer is yes. Namely, deciding whether a C-program halts or not.

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.
 - E.g., does this sudoku have a solution if we fill in 6 in the (i, j) th square?
 - Using this decision problem we can still find the solution to the Sudoku.
- ▶ Computer can solve a sudoku. Is there a problem a computer can't solve?
 - ▶ The answer is yes. Namely, deciding whether a C-program halts or not.
- ▶ Transitioning to a quantum computer changes the model of computation.

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.
 - E.g., does this sudoku have a solution if we fill in 6 in the (i, j) th square?
 - Using this decision problem we can still find the solution to the Sudoku.
- ▶ Computer can solve a sudoku. Is there a problem a computer can't solve?
 - ▶ The answer is yes. Namely, deciding whether a C-program halts or not.
- ▶ Transitioning to a quantum computer changes the model of computation.
 - ▶ Instead of logical gates we have complex matrices and instead of bits we have qubits.

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.
 - E.g., does this sudoku have a solution if we fill in 6 in the (i, j) th square?
 - Using this decision problem we can still find the solution to the Sudoku.
- ▶ Computer can solve a sudoku. Is there a problem a computer can't solve?
 - ▶ The answer is yes. Namely, deciding whether a C-program halts or not.
- ▶ Transitioning to a quantum computer changes the model of computation.
 - ▶ Instead of logical gates we have complex matrices and instead of bits we have qubits.
- ▶ Is there a problems a classical computer can't solve but a quantum computer can?

Computability theory

What can a computer compute?

Computability theory: What kind of problems can a computer solve?

- ▶ An example of a problem one might be interested in is solving Sudokus.
- ▶ In theoretical computer science, we often consider *decision problems*.
 - E.g., does this sudoku have a solution if we fill in 6 in the (i, j) th square?
 - Using this decision problem we can still find the solution to the Sudoku.
- ▶ Computer can solve a sudoku. Is there a problem a computer can't solve?
 - ▶ The answer is yes. Namely, deciding whether a C-program halts or not.
- ▶ Transitioning to a quantum computer changes the model of computation.
 - ▶ Instead of logical gates we have complex matrices and instead of bits we have qubits.
- ▶ Is there a problems a classical computer can't solve but a quantum computer can?
 - ▶ You will know the answer within 2 lectures!

Complexity theory

Dividing problems into complexity classes

- ▶ Another field where changing the model of computation to quantum computers might change things is *complexity theory*.

Complexity theory

Dividing problems into complexity classes

- ▶ Another field where changing the model of computation to quantum computers might change things is *complexity theory*.
- ▶ Complexity theory *classifies problems* that can/can't be solved with restricted resources within a model of computation in order to capture their *hardness*.

Complexity theory

Dividing problems into complexity classes

- ▶ Another field where changing the model of computation to quantum computers might change things is *complexity theory*.
- ▶ Complexity theory *classifies problems* that can/can't be solved with restricted resources within a model of computation in order to capture their *hardness*.

Definition (Complexity class)

A complexity class is defined by a triple **resource**, **model**, **scaling** and is the set of all the problems that can be solved within that specified **model** using an amount of **resources** that scales according to the given **scaling**.

Complexity theory

Dividing problems into complexity classes

- ▶ Another field where changing the model of computation to quantum computers might change things is *complexity theory*.
- ▶ Complexity theory *classifies problems* that can/can't be solved with restricted resources within a model of computation in order to capture their *hardness*.

Definition (Complexity class)

A complexity class is defined by a triple **resource**, **model**, **scaling** and is the set of all the problems that can be solved within that specified **model** using an amount of **resources** that scales according to the given **scaling**.

Two resources that are often studied are *time* and *space*.

Complexity theory

Dividing problems into complexity classes

Example

- ▶ P is the complexity class containing all problems that can be solved in polynomial time on a deterministic Turing machine.

Complexity theory

Dividing problems into complexity classes

Example

- ▶ P is the complexity class containing all problems that can be solved in polynomial time on a deterministic Turing machine.
- ▶ NP is the complexity class containing all problems that can be solved in polynomial time on a nondeterministic Turing machine.

Complexity theory

Dividing problems into complexity classes

Example

- ▶ P is the complexity class containing all problems that can be solved in **polynomial time** on a **deterministic Turing machine**.
- ▶ NP is the complexity class containing all problems that can be solved in **polynomial time** on a **nondeterministic Turing machine**.
- ▶ PSPACE is the complexity class containing all problems that can be solved in **polynomial space** on a **deterministic Turing machine**.

Complexity theory

Dividing problems into complexity classes

Example

- ▶ P is the complexity class containing all problems that can be solved in **polynomial time** on a **deterministic Turing machine**.
- ▶ NP is the complexity class containing all problems that can be solved in **polynomial time** on a **nondeterministic Turing machine**.
- ▶ PSPACE is the complexity class containing all problems that can be solved in **polynomial space** on a **deterministic Turing machine**.
- ▶ BPP is the complexity class containing all problems that can be solved in **polynomial time** on a **probabilistic Turing machine**.

Complexity theory

Dividing problems into complexity classes

Example

- ▶ P is the complexity class containing all problems that can be solved in **polynomial time** on a **deterministic Turing machine**.
- ▶ NP is the complexity class containing all problems that can be solved in **polynomial time** on a **nondeterministic Turing machine**.
- ▶ PSPACE is the complexity class containing all problems that can be solved in **polynomial space** on a **deterministic Turing machine**.
- ▶ BPP is the complexity class containing all problems that can be solved in **polynomial time** on a **probabilistic Turing machine**.
- ▶ BQP is the complexity class containing all problems that can be solved in **polynomial time** on a **quantum computer**.

Complexity theory

Dividing problems into complexity classes

Example

- ▶ P is the complexity class containing all problems that can be solved in **polynomial time** on a **deterministic Turing machine**.
- ▶ NP is the complexity class containing all problems that can be solved in **polynomial time** on a **nondeterministic Turing machine**.
- ▶ PSPACE is the complexity class containing all problems that can be solved in **polynomial space** on a **deterministic Turing machine**.
- ▶ BPP is the complexity class containing all problems that can be solved in **polynomial time** on a **probabilistic Turing machine**.
- ▶ BQP is the complexity class containing all problems that can be solved in **polynomial time** on a **quantum computer**.

A problem is called *tractable/efficiently solvable* if there is an algorithm that solves it in **polynomial time** within that model of computation (e.g., P and BQP)