CSM EECS 20N Lecture Notes

Authored by Dun-Ming Huang, SID: 303******2

0.1 Preface

These pieces of notes, to be updated irregularly, are meant to support upcoming syllabus changes in coming semesters of EECS 16A towards Signal Processing. The contents of these notes will very likely be a superset of any coming semester‘s syllabus that involves signal processing, and will therefore have extraneous parts out of some semesters‘ scopes. Specifically, these notes will be dedicated to the union of EECS 16A‘s past linear algebra content and EECS 20N‘s 2011 rendition‘s syllabus.

As the volume of this note will naturally exceed a 4-unit course worth of content, it will take a lot of time and effort to complete these notes. Therefore, the project is made open-source for interested parties to help maintaining with. It will be the primary author’s first time reading some of the involved contents in this note, so there may be a lack of upper-division-level insight in signal processing from this note.

The rest of preface will be written when it is finally not Christmas.

Contents

Part I

Linear Algebra is A Really Near Algebruh

Chapter 1

The Fundamentals of Linear Algebra

Hello

1.1 Introduction to Linear Algebra

Hello

1.2 Vectors and Matrices

Hello

1.3 Linearity

Hello

Chapter 2

Arithmetics of Vectors and Matrices

Hello

2.1 Addition and Subtraction

Hello

2.2 Matrix Multiplication

Hello

2.3 Inverse Matrices

Hello

2.4 Determinants

Hello

Chapter 3

Gaussian Elimination

Hello

3.1 Linear Systems of Equations

Hello

3.2 Gaussian Elimination

Hello

3.3 Failure of Gaussian Elimination

Hello

Chapter 4

Linear Dependence

Hello

4.1 Linear Independence and Linear Dependence

Hello

4.2 Geometric Interpretation

Hello

4.3 Rank

Hello

Chapter 5

Vector Spaces

Hello

5.1 Definition of Vector Space

Hello

5.2 Subspaces

Hello

5.3 Columnspaces (Span)

Hello

5.4 Nullspaces (Kernel)

Hello

Chapter 6

Eigen Show You The World: Eigenvalues, Eigenvectors, and Eigenpain

Hello

6.1 Eigenvalues

Hello

6.2 Eigenvectors

Hello

6.3 Eigenspaces

Hello

Chapter 7

Eigenspaces and Change of Basis

Hello

7.1 Cartesian Space and Basis

Hello

7.2 Change of Coordinate Axis

Hello

7.3 Change of Basis

Hello

Chapter 8

Inner Products and Norms, Orthogonality

Hello

8.1 Inner Products of Vectors

Hello

8.2 Inner Product of Matrices

Hello

8.3 Norms of Vectors

8.3.1 Definition of Vector Norms

In prior courses, we have recognized the following symbol:

Symbol 8.3.1. Euclidean Norm

The Euclidiean Norm, otherwise known as a **2-Norm**, is a mathematical quantity for some vector $\vec{x}$:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

which measures the Euclidean distance (Pythagorean Theorem's results) between the starting and terminal point of a vector.

While the Euclidean Norm is an extremely common norm, there exist more norms to vectors.
Particularly, **norms** are functions that satisfy the following properties:

Definition 8.3.1. Norm

A norm is a function $f : \mathcal{X} \to \mathbb{R}$ such that:

1. Non-negativeness: $\forall \vec{x} \in \mathcal{X}, \|\vec{x}\| \geq 0$, and $\|\vec{x}\| = 0 \iff \vec{x} = \vec{0}$
2. Triangle Inequality: $\forall \vec{x}, \vec{y} \in \mathcal{X}, \|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$
3. Scalar Multiplication: $\forall \alpha \in \mathbb{R}, \vec{x} \in \mathcal{X}, \|\alpha\vec{x}\| = \alpha\|\vec{x}\|$

It is noteworthy to mention that, almost every norm demonstrates one property of the vector. For the diversity of characteristics that a vector may have, there certainly exists a diversity of norms for vectors. For example, a general family of norms that satisfy the above properties would be the **LP-norms**:

Definition 8.3.2. LP-Norms

LP-Norms are norm functions defined as:

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

for some natural number p.

Particularly, the Euclidean Norm, otherwise known as the *2-norm* is LP-norm with $p = 2$.
Meanwhile, the 1-norm and ∞-norm are defined as follows:

$$\|\vec{x}\|_1 = \sum_{i=1}^{n} |x_i|$$
$$\|\vec{x}\|_\infty = \max_{i=1,\ldots,n} |x_i|$$

where, the 1-norm of a vector can be stated as the Manhattan distance (Taxicab geometry) of between that vector's starting and terminal point.

8.3.2 Cauchy-Schwartz Inequality

This inequality was active in EECS 16A!

Definition 8.3.3. Cauchy-Schwartz Inequality

The inequality is phrased as:

$$|\vec{x}^T \vec{y}| \leq \|\vec{x}\|_2 \|\vec{y}\|_2$$

And using the property,

$$\forall x \in \mathbb{R}, |\cos(x)| \leq 1$$

This inequality originates from the following algebraic work:

$$\begin{aligned} |\langle \vec{x}, \vec{y} \rangle| &= |\vec{x}^T \vec{y}| = |\vec{y}^T \vec{x}| \\ &= |\|\vec{x}\|_2 \|\vec{y}\|_2 \cos(\theta_{\vec{x},\vec{y}})| \leq |\|\vec{x}\|_2 \|\vec{y}\|_2| \end{aligned}$$

We observe that the above derivation holds statement on equivalence between inner product and product of magnitudes, cosine.
That is justified by the mechanics along which we find the projection of $\vec{y}$ onto $\vec{x}$:

$$\text{proj}_{\vec{x}}(\vec{y}) = \vec{x} \frac{\vec{x}^T \vec{y}}{\|\vec{x}\|_2^2}$$

where, since the projection is a multiple of $\vec{x}$ such that $\text{proj}_{\vec{x}}(\vec{y}) = t\vec{x}$, we also recognize that,

$$\cos\left(\theta_{\vec{x},\vec{y}}\right) = \frac{\|t\vec{x}\|_2}{\|\vec{y}\|_2}$$

Upon matching the two equations, I acquire:

$$t = \cos\left(\theta_{\vec{x},\vec{y}}\right)\frac{\|\vec{y}\|_2}{\|\vec{x}\|_2} = \frac{\vec{x}^T\vec{y}}{\|\vec{x}\|_2^2}$$
$$|\|\vec{x}\|_2\|\vec{y}\|_2| = \langle\vec{x},\vec{y}\rangle$$

In fact, we find that the Cauchy-Schwartz is a general case of this inequality:

Definition 8.3.4. Holder's Inequality

Provided the quantities

$$\vec{x},\vec{y} \in \mathbb{R}^n, \text{and some } p,q \geq 1 \text{ s.t. } \frac{1}{p}+\frac{1}{q} = 1$$

Then,

$$|\vec{x}^T\vec{y}| \leq \|\vec{x}\|_p\|\vec{y}\|_p$$

8.3.3 Application of Cauchy-Schwarz Inequality: Norm Ball

From this concept, let us consider the concept of *"norm ball"*: the geographical object containing all vectors such that their lp-norm is 1.

- For a 2-norm, for example, the norm ball looks like a unit circle (containing all 2D vectors with length 1, hence a circle of radius 1).
- For a 1-norm, similar logic guides us to a diagonally placed circle (rotated $45°$) centered at the origin with side lengths 2.
- For the ∞-norm, has a norm ball of a square centered at origin with side length 2. This embeds the unit ball from 2-norm, which embeds the unit ball from 1-norm.

The last bullet point hints that the area of norm ball is larger for any increase in p such that it embeds the prior norm balls.

Now, let us attempt to solve for some optimization regarding a norm ball:

$$\max_{\|\vec{x}\|_p \leq 1} \vec{x}^T\vec{y}$$

p = 2:

The solution would be, by the Cauchy Schwartz's implication,

$$\vec{x}^* = \frac{\vec{y}}{\|\vec{y}\|_2}$$

p = 1:

The expression of dot product is equivalently

$$x_1y_1 + \cdots + x_ny_n$$

Where the constraint is

$$|x_1| + \cdots + |x_n| \leq 1$$

For each value the components of $\vec{x}$, which is finite and upper bounded by 1, we should allocate maximum contribution to the maximum element of $\vec{y}$ to maximize this dot product (a weighted sum of $\vec{y}$'s component, essentially).
Therefore, let i be the index at which $\vec{y}$ has the component of largest absolute value,

> There is an achievable solution for $\vec{x}^*$, being the unit vector $\vec{e}_i$ multiplied by $sgn(y_i)$, so to counter for cases where $\vec{y}$ is negative.

In turn, we see that

$$\vec{x}^T\vec{y} = \max_i |y_i| = \|\vec{y}\|_\infty$$

Meanwhile, let us use the Holder's Inequality to achieve a more rigorous proof. Holder's Inequality states that,

$$|\vec{x}^T\vec{y}| \leq \|\vec{x}\|_1 \|\vec{y}\|_\infty = \max_i |y_i|$$

> The Holder's Inequality expresses **an upper bound**, and the formulation in above section shows **achievability of upper bound**.

Alternative proof of p = 1 via Triangle Inequality:
Via the triangle inequality, we acquire:

$$\begin{aligned}
|\vec{x}^T\vec{y}| &= |\sum_i x_i y_i| \\
&\underset{Triangle\ Inequality}{\leq} \sum_i |x_i y_i| \\
&= \sum_i |x_i||y_i| \\
&\leq \sum_i |x_i| \max_i |y_i| = \max_i |y_i| = \|\vec{x}\|_\infty
\end{aligned}$$

We see the expression $|\vec{x}^T\vec{y}|$ has an **achievable upper bound**, assembling all necessary aspects of deriving the solution for the optimization problem.

p = ∞:

We can find upper bound of the expressionvia Holder's Inequality,

$$|\vec{x}^T\vec{y}| \leq \|\vec{x}\|_\infty \|\vec{y}\|_1 \leq \sum_i |y_i|$$

The infinity norm of $\vec{x}$ shows the constraint that:

$$\max_i |x_i| = 1$$

and we may simply compute:

$$x_i = sgn(y_i)$$

to find and certify the achievability of this upper bound.

8.4 Norms of Matrices

Hello

Chapter 9

Least Squares Algorithm: Where Machine Learning Starts

Hello

9.1 Overdetermined Systems

Hello

9.2 Estimation of Solutions for Overdetermined Systems

Hello

9.3 Summary of Least Squares Algorithm

Hello

Part II

New Mathematical Prerequisites

Chapter 10

An Integral Review for Integration

Eventually, integrals will become a core aspect of this coursework. Therefore, to prepare students for upcoming mathematical prerequisite knowledge, we aim to review some core aspects of integrals here. If you have never learned integrals before, you probably should not be here; and, if you would still like to digest the contents here, you may want to consult other resources along with this one, as this resource will only discuss a summary of integral techniques.

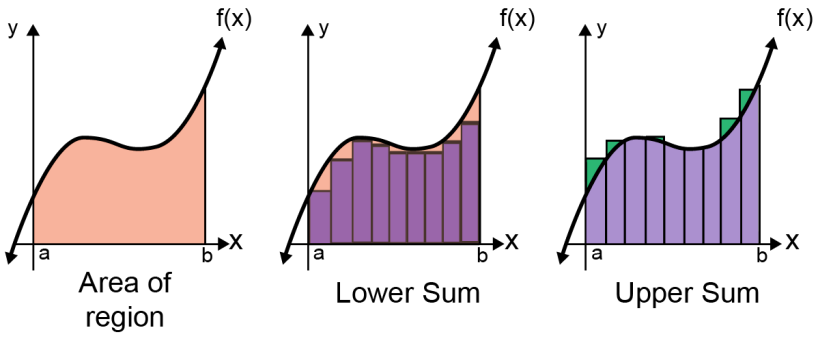
10.1 A Brief Review of An Integral

The most explicit interpretation of an integral is the area under curve: provided some curve that represents a single-variate function f, the area under its curve from $x = a$ to $x = b$ is an integral: $\int_a^b f(x)\,dx$.

Let us discuss this interpretation furthermore by discussing some methodologies within our scope for approximating the area under curve.

In the above figure, the "Lower Sum" and "Upper Sum" techniques (formally known as Reimann Sums) attempt to approximate the area under curve with the following algorithm:

1. Decide the width of the rectangles that will fill the area under curve (or, overfill).
2. Decide the height of the rectangle based on either the curve height at the left or right end of the rectangle.



Calcworkshop.com

Alternatively, we may also choose either the lower one or the higher one (respectively leading to the "Lower Sum" and "Higher Sum" techniques above).

3. Sum the areas of these rectangles.

We consider this as a discrete operation. Discrete here means that the mathematical operations is done on a non-continuous number line that does not contain every possible number. For example, by deciding the width of rectangle, we also decide to skip looking at the function values of any number that is not the rectangle width. However, as you may have noted, discrete operations do not provide a great approximation and may take a lot of computation as the number of rectangles increase.

What if there is, instead, a continuous method of doing so, such that we can consider rectangles of infinitesimal widths and sum up their areas? Such is the operation that we call integrals. In other words, integral operations are the discrete summation techniques we see in the above picture, except we now use infinitesimal-width rectangles such that we can most accurately portray the area under curve with the sum of those rectangles. This shows us that integrals are actually the analogy of summations in a continuous space.

While the summation methods using n rectangles to approximate the area under curve within $[a, b]$ may be simplified as a summation:

$$\sum_{i=0}^{n-1} \frac{b-a}{n} f(a + i \times \frac{b-a}{n})$$

The integral version of such method is instead written as:

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{b-a}{n} f(a + i \times \frac{b-a}{n})$$

We will introduce its mathematical operations in the coming section.

10.2 Computing An Integral

To compute an integral $\int_a^b f(x)\,dx$, we first find the antiderivate of function f (which we would call F), then we would compute the value of integral as $F(b) - F(a)$.
The above very, very rough summary forgets to address several subtleties:

1. The definition of antiderivative and the computations required to find it.
2. The reason why we may claim the value of $\int_a^b f(x)\,dx$ to be the difference $F(b) - F(a)$.
3. What do we do when there exists a point $x' \in [a, b]$ at which the function f is not defined at.

Let us address the above concerns one by one to validify the summarizing statement.

First of all, the antiderivaitve F of a function f is a function such that $\frac{\mathrm{d}F}{\mathrm{d}x} = f(x)$. Provided so, the antiderivative of any function f is not unique. Let us proceed with a proof, or, an argument to prove or disprove a mathematical statement:

Explain 10.2.1. Antiderivative of Function $f(x)$ is Non-Unique

Suppose that the function f already has an antiderivative $F(x)$. Then, let C be a scalar, real constant. We may see that:

$$\begin{aligned} \mathrm{d}/\mathrm{d}x &= f(x) + \mathrm{d}/\mathrm{d}x\ C \\ &= f(x) + 0 = f(x) \end{aligned}$$

Usually, then, we denote the antiderivative of $f(x)$ as $F(x) + C$ for the above reason.

Meanwhile, the computation of integral from antiderivatives is secured by a famous theorem that we introduce below:

Theorem 10.2.1. Fundamental Theorems of Calculus

The first fundamental theorem of calculus. Let f be a continuous real-valued function defined on a closed interval $[a, b]$. Let F be the function defined, for all x in $[a, b]$, by

$$F(x) = \int_a^x f(t)\, dt$$

Then F is uniformly continuous on $[a, b]$ and differentiable on the open interval (a, b), and

$$F'(x) = f(x)$$

for all x in (a, b) so F is an antiderivative of f.
This theorem formally defines what really is an antiderivative.
The second fundamental theorem of calculus. Let f be a real-valued function on a closed interval $[a, b]$ and F a continuous function on $[a, b]$ which is an antiderivative of f in (a, b):

$$F'(x) = f(x)$$

If f is Riemann integrable on $[a, b]$ (or, in other words, if the Reimann sum, approximation of area under curve converges as the rectangle width approaches infinitesimal), then

$$\int_a^b f(x)\, dx = F(b) - F(a)$$

And when there exists a point $x' \in [a, b]$ at which the function f is not defined at, the integral is simply undefined. Usually, when we still want to find the area under curve, we can find a workaround by defining a function:

$$\hat{f}(x) = \begin{cases} f(x), & x \neq x' \\ \text{some other value}, & x = x' \end{cases}$$

and computing its integral instead.

10.3 Integral Tricks

There are several integral tricks that may become prevelant in the upcoming course contents. This section provides a survey of them.

10.3.1 Integral Identities

Let's review some integral identities:

Fundamentals.

1. $\int x^n\, dx = \begin{cases} \frac{x^{n+1}}{n+1} + C, & n \neq -1 \\ \ln|x|, & n = -1 \end{cases}$
2. $\int e^x\, dx = e^x + C$
3. $\int a^x\, dx = \frac{a^x}{\ln(a)} + C$

Trigonometric.

1. $\int \cos(x)\,dx = \sin(x) + C$
2. $\int \sin(x)\,dx = -\cos(x) + C$
3. $\int \sec^2(x)\,dx = \tan(x) + C$
4. $\int \csc^2(x)\,dx = -\cot(x) + C$
5. $\int \sec(x)\tan(x)\,dx = \sec(x) + C$
6. $\int \csc(x)\tan(x)\,dx = -\csc(x) + C$

Inverse Trigonometric.

1. $\int \frac{1}{\sqrt{1-x^2}}\,dx = \sin^{-1}(x) = -\cos^{-1}(x)$
2. $\int \frac{1}{1+x^2}\,dx = \tan^{-1}(x) = -\cot^{-1}(x)$
3. $\int \frac{1}{x\sqrt{x^2-1}}\,dx = \sec^{-1}(x) = -\csc^{-1}(x)$

10.3.2 Partial Fractions

The technique of Partial Fractions concern decomposing a rational function into several more rational functions that are easier to integrate. An example follows:

Explain 10.3.1. Example of Partial Fraction Trick

Compute the following expression:

$$\int \frac{1}{x^4-1}\,dx$$

Let us first attempt to decompose $\frac{1}{x^4-1}$ into a sum of several more rational functions. The first step of doing so is to find a way to factorize the denominator of our function:

$$x^4 - 1 = (x^2-1)(x^2+1) = (x-1)(x+1)(x^2+1)$$

Then, we look for some coefficients that allows for us to express:

$$\frac{1}{x^4-1} = \frac{A}{x-1} + \frac{B}{x+1} + \frac{Cx+D}{x^2+1}$$

How many coefficients we need can be explored upon with rules of thumbs that, for the sake of brevity, should not be discussed here.
Then, we manipulate such that:

$$\begin{aligned}\frac{1}{x^4-1} &= \frac{A}{x-1} + \frac{B}{x+1} + \frac{Cx+D}{x^2+1} \\ &= \frac{1}{x^4-1}\cdot(A(x+1)(x^2+1) + B(x-1)(x^2+1) + (Cx+D)(x^2-1)) \\ &= \frac{1}{x^4-1}\cdot((A+B+C)x^3 + (A-B+D)x^2 + (A+B-C)x + (A-B-D))\end{aligned}$$

Which allows us to define a system of linear equations to solve A, B, C, D for:

$$\begin{cases} A+B+C &= 0 \\ A-B+D &= 0 \\ A+B-C &= 0 \\ A-B-D &= 0 \end{cases}$$

The solution should come out to be $A = \frac{1}{4}$, $B = -\frac{1}{4}$, $C = 0$, $D = -\frac{1}{2}$. Therefore,

$$\begin{aligned} \int \frac{1}{x^4-1}\,dx &= \int (\frac{1}{4}\frac{1}{x-1} - \frac{1}{4}\frac{1}{x+1} - \frac{1}{2}\frac{1}{x^2+1})\,dx \\ &= \frac{1}{4}\int \frac{1}{x-1}\,dx - \frac{1}{4}\int \frac{1}{x+1}\,dx - \frac{1}{2}\int \frac{1}{x^2+1}\,dx \\ &= \frac{1}{4}\ln|x-1| - \frac{1}{4}\ln|x+1| - \frac{1}{2}\tan^{-1}(x) + C \end{aligned}$$

10.3.3 u-Substitution

Write later

10.3.4 Integration by Part

Write later

10.3.5 Trigonometric Substitutions

Write later

Chapter 11

Set!

11.1 Fundamental Properties of Sets

As you may have guessed, in set theory, we mainly discuss a mathematical object called "set".

Definition 11.1.1. Set

A set is a collection of objects.
The mathematical objects that are members of a set are called elements.
Sets are normally expressed in the format of:

$$\{\text{element 1}, \text{element 2}, \ldots, \text{element n}\}$$

The symbol representing membership in a set works as follows:

Symbol 11.1.1. Membership Symbol

If a mathematical object x is a member of a set A, then we may mathematically express that $x \in A$.
Otherwise, for a mathematical object y that does not belong to A, $y \notin A$.

There are several more properties to sets:

- The size of a set is defined as the number of elements in it. This property of set is called **cardinality**. The cardinality of a set A can be mathematically denoted as $|A|$.
- The equality of a set is held when two sets have exactly same elements; order and repetition does not matter, albeit in computers, many implementations of sets already do not allow repetition. We call a set whose elements can be repeated a "multiset".
- Sets whose cardinality are 0, also known as empty sets, are denoted as $\{\}$ as well as $\emptyset$.

To denote a set of mathematical objects that all belong to some other set but suffices some property, we may utilize a notation called **set-builder notation**:

Symbol 11.1.2. Set-Builder Notation

A set A whose members all follow expression exp, and the terms of exp all fit some list of conditions can be written as:

$$\{exp : \text{condition 1}, \ldots, \text{condition n}\}$$

The colon above can be replaced by a vertical bar. Since we will be using vertical bars for other purposes in later chapters, a colon is perhaps the least abusive notation to play with.
But, to conform the formatting of official lecture notes, let us continue with vertical bars.
For example, the list of all rational numbers can be written as:

$$\mathbb{Q} = \left\{ \frac{p}{q} \middle| p, q \in \mathbb{R},\ q \neq 0 \right\}$$

Some useful sets have names for the ease of references:

Notation	Name	Example Contents
$\mathbb{N}$	Natural Numbers	$\{1, 2, \dots\}$
$\mathbb{N}_0$	Non-negative Integers	$\{0, 1, 2, \dots\}$
$\mathbb{Z}$	Integers	$\{\dots, -2, -1, 0, 1, 2, \dots\}$
$\mathbb{R}$	Real Numbers	$(-\infty, \infty)$
$\mathbb{C}$	Complex Numbers	$\{x + \mathrm{i}y \mid x, y \in \mathbb{R}\}$

11.2 Relationships between Sets

We often compare sets in terms of their sizes and elements they contain. Out of these standards for comparison, we can define the relationship between a set and another larger set that contains all elements of the former as follows:

Definition 11.2.1. Subset

If every element of a set A is also in a set B, then A is a subset of B.
Mathematically, we write it as $A \subseteq B$.
Or, stating the equivalent in an opposite direction, $B \supseteq A$, and this would state B as a superset of A.

And a stricter similar relationship follows:

Definition 11.2.2. Strict Subset

If $A \subseteq B$ but A excludes at least an element of B, then we say that A is a strict subset (proper subset) of B.
Mathematically denoted, $A \subset B$.

Utilizing the definitions above, we may form some observations:

- The empty set is a proper subset of any nonempty set.
- The empty set is a subset for any set, including itself's.
- While every set is not a proper subset of itself, every set is a subset of itself.

While we compare sets based on their members, we can also "add", create sets based on the members of two sets. There are two ways of doing so, being **intersection** and **union**.

Definition 11.2.3. Intersection

The intersection of set A and B, written as $A \cap B$, is the set containing all elements that are both in A and B.
In set builder notation:

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

Definition 11.2.4. Union

The union of set A and B, written as $A \cup B$, is the set of all elements contained in either A or B.
In set builder notation, it pronounces very similarly with the notation for intersections:

$$A \cup B = \{x | x \in A \vee x \in B\}$$

Regarding the way empty sets work in the above arithmetic, for an arbitrary set A:

- $A \cap \emptyset = \emptyset$
- $A \cup \emptyset = A$

At last, let us regard the notion of **"complements", the difference between sets**.

Definition 11.2.5. Complement

Imagine the arithmetic of sets to work solely upon their members, and let the difference of sets be defined such that:

$$A - B = A \backslash B = \{x \in A | x \notin B\}$$

This set $A\ B$ is called the set difference between A and B, or alternatively, the relative complement of B in A.

This also indicates that the "subtraction arithmetic" for sets is not commutative. Using empty sets as examples:

- $A \setminus \emptyset = A$
- $\emptyset \setminus A = \emptyset$

And the last arithmetic is the **"multiplication of sets": Cartesian Products**.

Definition 11.2.6. Cartesian Products

The Cartesian Product (cross product) of sets A and B is defined such that:

$$A \times B = \{(a, b) \mid a \in \mathbb{R}, b \in \mathbb{R}\}$$

And last but not least, we have mathematical operations that generate a set of sets (nested) based on other sets:

Definition 11.2.7. Power Set

The power set of S can be written as:

$$\wp(S) = \{P | P \subseteq S\}$$

as one of its various denotations, the one on lecture note using a Weierstrass P.
The power set of a set is the set of all of its possible subsets.

In addition, if the cardinality of S in the above definition is $|S| = k$, then we can state that $|\mathbb{P}(S)| = 2^k$. This will be an explored notion as we discuss a discrete math topic "Counting" in near future.

Chapter 12

Propositional Logic

12.1 Definition of Proposition

To speak the language of mathematics, we must familiarize ourselves with objects of the mathematical language. One fundamental block of this mathematical language is a **proposition**.

Definition 12.1.1. Proposition

A proposition is a logical statement that is either true or false.
In other words, it is a mathematical statement that could be correct, but also could be incorrect.

Propositions need to be either true or false, which means it cannot be statements that can yield some ambiguous result. And because they usually are not known to be true or false, we often need to prove propositions in discrete mathematics.

Exercise Reader! Exercise! 12.1.1: What should a proposition be like?

Which of the following qualifies as a proposition?

(a) $\sqrt{3}$ is rational.

(b) 2 + 2

(c) I often never give up or let down

Option (a) is a statement that has to either hold true or false, because this value $\sqrt{3}$ is either rational or irrational.
Option (b) is a mathematical expression, and there is no true or false aspect to it. It is not a proposition.
Last but not least, option (c) is just a statement, because the word "often" is not properly defined. If propositions are supposed to clearly hold true or false, it should not contain blurrily defined words.
The answer, therefore, is option (a).

12.2 Logical Operations of Proposition

There are approximately four methods of using preexistent propositions to make new propositions, and we do so to produce complicated propositions from simpler subparts.
These four ways being:

- **Conjunction**: $P \wedge Q$, works similarly as the boolean "and".
- **Disjunction**: $P \vee Q$, works similarly as the boolean "or".
- **Negation**: $\neg P$, works similarly as the boolean "not".
- **Implies**: $P \implies Q$, which states a familiar phrase of "if P, then Q". Here, the proposition P is called a hypothesis, and Q a conclusion.

To demonstrate the behaviors of these operations, we may use a **truth table**. A truth table example for the operation "Conjunction" is attached below:

Figure 12.2.1. Truth Table for Conjunction

A truth table is a table that points out the result of an operation on statements P and Q, for a set of given boolean values that P and Q each result in.

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

Now we may discuss some other properties of propositions.
First of all, there is a fundamental principle called **Law of the Excluded Middle**, which states what is as follows:

Axiom 12.2.1: Law of Excluded Middle

For any proposition P, $P \vee \neg P$ always holds true, because P is either true or false (not true).

Here, propositions like $P \vee \neg P$ that always hold true are called **tautologies**.
Meanwhile, propositions that are always false, such as $P \wedge \neg P$ (which cannot be true, since P is either true or false), are called **contradictions**.
Implications also have interesting truth table properties:

Figure 12.2.2. Truth Table for Implication

A truth table for implications is as follows:

P	Q	$P \implies Q$	$\neg P \vee Q$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

While this provides that the implication is only false when P is true and Q is false, this also provides a not so intuitive insight: An implication is trivially true when the hypothesis is false.
This situation is known as **"vacuously true"**.
The reason why this insight seems to hold is because while we cannot confirm the conclusion is false when the hypothesis is false, this is also the nature of how if-then statements are expected to be defined.

Indicated via the above truth table, as $P \implies Q$ and $/P \vee Q$ have the exact same boolean behavior on truth table, they are logically equivalent. This means they are statements that perform the logical behavior and meaning.

Interestingly, there are also various ways to pronounce implications, such as "if P, then Q", as well as "Q if P", and many more.
Sometimes we also come across words that seem to connect two propositions in a tighter implication, such as "iff".

An **iff** relationship is also standardly pronounced as **if and only if**, which is mathematically expressed as $P \iff Q$. This "iff" holds if both $P \implies Q$ and $Q \implies P$ are true. Consequentially, and as encountered in previous coursework, such as EECS 16A, the proof of an "iff" statement is at its nature the proof of two smaller statements.

12.3 Implications from Implications

For an implication $P \implies Q$, we may define two other implications out of it:

- **Contrapositive**: $\neg Q \implies \neg P$, which is logically equivalent to $P \implies Q$. At its implication: if a statement P's contrapositive is true, so should the statement P per se.
- **Converse**: $Q \implies P$, which does not always hold true. In some cases, converse statements can hold true, but usually accompanies some extra conditions. Some examples can be observed in the contents of MATH 53.

When two propositions are logically equivalent, we may mathematically denote it with the $\equiv$ symbol.
An example of usage would be stating an implication the logical equivalent of its contrapositive, in the forms:

$$(P \implies Q) \equiv (\neg Q \implies \neg P)$$

An application of this is a new approach of performing proofs. To prove a prompt that states a theorem as an implication $P \implies Q$, by proving its contrapositive, which is in some cases easier than proving the prompt directly, we can manage to indirectly prove the prompt.
This application will be introduced in the next chapter as "proof by contrapositive".

12.4 Quantifiers

12.4.1 Use of Quantifiers

In the previous chapter, we introduced the usage of quantifiers as an abbreviation and selector of elements in sets to which a condition holds, in the format of:

<quantifier selecting clause> <second clause containing condition to hold for selected values>

The second clause is effectively a proposition, while the entire sentence containing the first and second clause is also effectively a proposition.

There is a more sophisticated way to express the first clauses of these forms of propositions.
The first clause has a quantifier, and throughout a "universe" we are working with (abstractly, a wide variety of mathematical objects following some conventions), the statement is quantified (holding true) for a selected range of objects in this universe.
So syntactically, the quantifier serves to involve this range of quantification for our proposition.

Let me expound on this abstraction of "universe" further. Say we attempt to qualify a proposition on all natural numbers, such as:

$$\forall x \in \mathbb{N}, x \in \mathbb{Q}$$

Then, the total set of objects I'm working through is the set of all natural numbers, and is therefore the **"universe": a collection of all objects to be qualified in this proposition**. In that sense, universe is a set.

In a finite universe, there is also another way of conceptualizing the quantifiers we know:

Exercise Reader! Exercise! 12.4.1: Propositional Logic in Quantifiers

Let us work with a universe $\mathbb{U} = U_1, \ldots, U_n$, where n is some finite natural number and elements of this universe are some mathematical object.
Then:

- **Universal Quantifiers**: $(\forall x \in \mathbb{U}(P(x))) \equiv (P(U_1) \wedge \cdots \wedge P(U_n))$
- **Existential Quantifiers**: $(\exists x \in \mathbb{U}(P(x))) \equiv (P(U_1) \vee \cdots \vee P(U_n))$

To express more complicated propositions, we must expand beyond the previous first-clause-second-clause format for propositions. This is especially when we are selecting two ranges of values to describe a multi-variable proposition with.

Exercise Reader! Exercise! 12.4.2: How to read complicated propositions?

Say we are dealing with the proposition:

$$(\forall x \in \mathbb{Z})(\exists y \in Z)(x < y)$$

Here, I can observe two quantifying clauses and the last clause that stands for a proposition. In fact, propositions involving quantifiers will always have the last clause as a proposition, due to the syntactical limits and customs of mathematical language.
Now let us try translating each clause into the English language:

1. $\forall x \in \mathbb{Z}$: For all objects x that belongs to $\mathbb{Z}$, the set of all integers.
2. $\exists y \in \mathbb{Z}$: There exists an object y that belongs to $\mathbb{Z}$, the set of all integers.
3. $x < y$: Object x is smaller than y.

Piecing the clauses together: For all objects x that belongs to $\mathbb{Z}$, the set of all integers, there exists an object y that belongs to $\mathbb{Z}$ such that x is smaller than y.
This proposition, in fact, holds true, since we can always find a larger integer.
However, the proposition with a reversed order for quantifier clauses:

$$(\exists y \in Z)(\forall x \in \mathbb{Z})(x < y)$$

has to hold False.

With the above translating process, this statement states that: there is an object y that belongs to $\mathbb{Z}$, the set of all integers, such that for all objects x that belongs to $\mathbb{Z}$ such that x is smaller than y.
In other words, it assumes the existence of one largest integer, which does not exist!

Then, let us look at an example from the Fall 2022 Midterm 1:

Example 12.4.1: Reading Complicated Quantifier Statements

Which logical expression(s) below correspond to the predicate that p is prime?

a. $(\forall b \in \mathbb{N})\big((b \leq 1) \vee \neg(\exists a \in \mathbb{N})(a \neq 0 \wedge ab \neq p)\big)$

b. $\neg\big((\exists a, b \in \mathbb{N})((a \geq 2) \wedge (a < p) \wedge (p = ab))\big)$

For the expression (a), it states that:

> For all natural numbers b, either $b \leq 1$ or there does not exist a natural number a such that a is nonzero and $ab \neq p$.

Therefore, for any positive integer b larger than 2, there does not exist another positive integer a such that $ab \neq p$.

However, let's consider the possibility $p = 2$, which is prime, then for an integer $b = 2$, there exists $a = 1$ such that $ab = 2$.
Therefore, the trueness of expression (a) does not indicate that p is prime.

Now, consider expression (b), which instead states that:

> It is not such that there exists two natural numbers a, b where simulatneously, $a \geq 2$, $a < p$, and $p = ab$.

To simplify the above quote, there does not exist a pair of natural numbers such that when one of them is a natural number $2 \leq a < p$, the other number multiplied by this number is p.
Since $a < p$, to achieve $ab = p$, it is impossible that $b = 1$. This successfully circumvents the limitation encountered in expression (a), qualifying expression (b) as the correct solution. This expression does successfully describe the definition of prime number p:

> The only pair of natural number whose product is prime p is $(1, p)$

By demonstrating that there exist no pair of natural number whose product is p if one of the element is exclusively between 1 and p.

This is an interesting discussion, as the question presented above has in fact been regraded!

12.5 Reading and Negation

12.5.1 Meaning of Negation

If a proposition P is false, its negation is true.
This inspires another approach for proofs (proof by contradiction), which usually work for simpler prompts since their contradictions would only be simpler to prove, if provided that this method is optimal to begin with.
However, when P appears to be a more complicated proposition involving other smaller propositions, we should look for some way to make our understanding of it more concise.
A logical law, called the **De Morgan's Laws**, facilitates us with such matter:

Axiom 12.5.1: De Morgan's Law

The De Morgan's Law states the two following logical equivalences:

$$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$

$$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$$

In fact, negations of propositions involving quantifiers follow analogous laws, due to the similarity of universal quantifier with conjunction and existential quantifier with disjunction:

Axiom 12.5.2: Extension of De Morgan's Law

Based on the similiarity of quantifiers with operations of propositions:

$$\neg(\forall x P(x)) \equiv (\exists x \neg P(x))$$

$$\neg(\exists x P(x)) \equiv (\forall x \neg P(x))$$

These above equivalences can provide flexibility in coming mathematical proofs.

12.5.2 Complex Examples of Negation

Let us discuss the example attached on the lecture notes from Summer 2022.

Prompt 1: Write a proposition that states, "there are at least three distinct integers x that satisfies $P(x)$.

And the proposition our textbook provided was:

$$\exists x \exists y \exists z (x \neq y \neq z \wedge P(x) \wedge P(y) \wedge P(z))$$

Exercise Reader! Exercise! 12.5.1: Proposition for Prompt 1

Let us analyze this proposition via some translation:

$\exists x \exists y \exists z (x \neq y \wedge x \neq z \wedge y \neq z \wedge P(x) \wedge P(y) \wedge P(z))$
$\rightarrow$ "there exists a x, y, z" such that
"x, y, z are not equal to each other, and all of $P(x), P(y), P(z)$ holds".
$\rightarrow$ "there exists at least one possible combination of three distinct integers"
such that "all of $P(x), P(y), P(z)$ holds".
$\rightarrow$ there are at least three distinct integers x that satisfies $P(x)$.

In the above translation process, we first analyzed each phrases independently. Then, having each clauses translated, we move on to combine them together semantically.
For example, the fact that the three integers x, y, z are not equal to each other means they are distinct, and so we may summarize the quantifying clause from "there exists a . . . " into "there exists at least one combination of three integers such that . . . ".
This helps us imply that, if there exists at least one combination, then there are at least three integers to satisfy $P(x)$.
This logic of inter-language and inter-context translation allows us to convert a proposition from its mathematical form to pure English text. It should be familiarized via practice and experience.
For purposes of practicing, let us analyze another proposition and attempt to translate it:

$$\exists x \exists y \exists z \forall d (P(d) \implies d = x \vee d = y \vee d = z)$$

And in the following portion, let us perform again the same procedure of directly translating the proposition from symbols to English, and then rephrase each English clause into more concise descriptions.

Exercise Reader! Exercise! 12.5.2: Explain this proposition for me, will you?

Translate the proposition to English: $\exists x \exists y \exists z \forall d (P(d) \implies d = x \vee d = y \vee d = z)$

This proposition first has a quantifying clause that states: "there exists a x, y, and z for all values d", or that there exists a set of three numbers x, y and z for all values d.
The last clause is an implication stating: "if $P(d)$ holds true, then $d = x \vee d = y \vee d = z$". That means d is equal to either x, y, or z. So, there exists three numbers out of all possible values in the universe such that if $P(d)$ holds, then d is one of the three numbers we observed before.
Meanwhile, we are not provided that x, y, and z are distinct values, so there are at most three values for d such that $P(d)$ holds.
Let us switch a perspective and view the contrapositive of the later clause and verify our previous interpretation.
If d is equal to neither of x, y, z, then $P(d)$ will not hold.
Assuming they are distinct, this statement would provide that if d is equal to neither of some (up to three) distinct numbers, $P(d)$ will not hold.
Therefore, the proposition is stating "$P(d)$ **holds for at most three distinct integers**".

Here is one demonstration about conjunctions.
By performing conjunction for the propositions above: that $P(d)$ holds for at least and at most 3 integers, we successfully create a new proposition that states "$P(d)$ holds for exactly 3 integers".
Mathematical propositions that come with limits, or quantifying clauses, for some other proposition are useful in helping us locate a range of values for which a condition holds.

If we combine propositions that hold for same conditions but with different quantified limits, we managed to combine the limits together and form some tighter statement providing a proposition with tighter limitations.

Chapter 13

Functions

13.1 Definition of Function

There have been many interpretations of functions in the history of mathematics. Popularly, we mark a function f as a relationship between sets, which the rules of this relationship we will discuss later. With that in mind, the notation of a function is:

$$f : X \to Y$$

where X, Y are sets that the function f concerns. Specifically, the function notes one specific value in Y for every element in X. In other words:

$$\forall x \in X, \exists! y \in Y : y = f(x)$$

(for all x in X, the domain of the function, there exists one and only one y in Y, the codomain of the function, that such x corresponds to. We denote this value as $y = f(x)$.)

Functions can also be mathematically defined by expanding on the above expression. For example, a function that maps every real number x to its squared value would be noted as:

$$\forall x \in \mathbb{R}, f(x) = x^2$$

for which we would also see that the range of this function is $\mathbb{R}$.

However, functions may be multivariate, such that we need a set of sets (a higher-order set) to dictate the domain of our function. For example, for a function that receives two real numbers and returns its sum, we may denote the function as such:

$$\forall (x_1, x_2) \in \mathbb{R} \times \mathbb{R}, g(x_1, x_2) = x_1 + x_2$$

where

$$g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$

13.2 The Properties of Functions

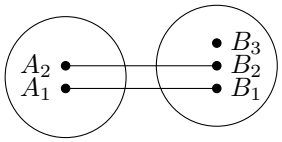
A function is, in its nature, the relationship of sets: it is a relationship of elements from set A to set B. Mathematically expressed,

$$(\forall x \in A)(f(x) \in B), f : A \to B$$

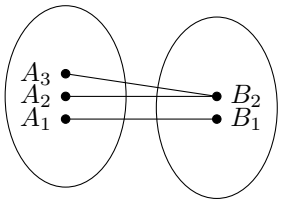
Definition 13.2.1. Injective, Surjective, Bijection

Injective, Surjective, and Bijective are all properties of functions, relationship between two sets A and B. We call the A the set of *pre-images*, and B the set of *images*.

- **Injective**: Each image (output) has at most one pre-image (input). This type of function is thus called a "one-to-one" function.



- **Surjective**: Each image (output) has at least one pre-image (input). This type of function is "onto".



- **Bijective**: A function is Bijective if the function is both injective and surjective. In other words, each image (output) has exactly one pre-image (input) and vice versa.

Part III

Introduction to Signals and Systems

Chapter 14

Introduction to Signals and Systems

14.1 What is A Signal?

14.1.1 Definition of Signals

The "signal" and "system" we study in this course is very prevelant to the design of real-world electronic devices. But, before beginning to study them, we should first strive to understand their conceptual existence.

A signal is a conveyor of information. So, a signal is the transformation between a "thing" and a quantification of it. In a biological perspective, any stimuli is a signal, because it delivers some information to our cognitive devices. However, signals alone cannot alert or notify a cognitive device to perform some task, and we must therefore need another processor that can convert the signal into something else, or elicit information from a signal. For example, just seeing the color on a red sign will not make you think that what lies beyond that sign must be suspicious; or, just seeing the color red will not make you think of the Hit Game Among Us. There must be some other level of processes that picks up this color red, and performs some black-boxed operations to translate the signal of color red into "danger!", "warning!", "Among Us!" In a similar way, just receiving electronic waves is not sufficient for a machine to move. A machine must also be equipped with some other mechanism to interpret this electronic wave and its properties.

The process that generates or transforms signals into forms that we can understand or interpret is called a "system". In the formerly addressed biology case, your brain is a huge system.

14.1.2 Abstraction of Signals

Now that we have understood the layers of abstractions that signals and systems can be understood at, let's attempt at their mathematical representations. Previously we addressed that a signal is a stimuli. Now, imagine a waveform, a sound wave, such as the sound of a violin. The sound of a violin is the result of vibrating air, but that has no detectable mathematical formalism until we attempt to visualize a soundwave as a function. In such case, the signal of violin sound maps the sound of violin per se, which has many original interpretations and is a physical phenomenon, into a waveform that we can represent using a time axis and waveform.

While the signal represents physical existences with physical measures like intensity or waveform, a system maps these signals into other output signals. This is a bit confusing. Let's extend from the violin sound analogy. The violin sound was a physical "thing", and it then gets transformed into a waveform as a signal. Suppose we attempt to record the sound of a violin into an electric device, then the electric device must find a way to further convert that waveform (vibration of air, now sound waves) into another digital format such that it can store the waveform in an electronic operation. This process of mapping a signal (wave form air vibration) into another output signal (electronically storable form of soundwave) is called a system.

So, mathematically, a signal is a mapping (function) from our real world to a physical measure that can be quantified, and a system maps these quantifications into different signal results.

14.2 A Survey of Signals: Audition, Vision, and Vectors

In this section, let's discuss popular forms of signals and how they can be represented with the mathematical abstractions we discussed above.

14.2.1 Audio Signals

Sounds are just rapid variations in air pressure. That means sounds (taco bell sound effects, the sound of that woman defrosting in December, voices inside your head, etc.) are all signals that can be expressed as a function. But, it is not a mapping between existences and measures. The sound signal is a function that maps time to the air pressure detected at that time. In other words:

$$\text{Sounds} : \text{Time} \times \text{Pressure}$$

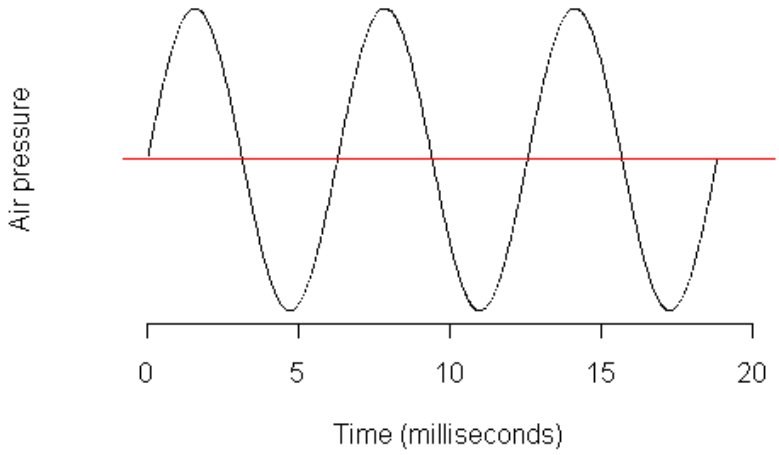


Figure 14.1: Sound waves can be represented as a sequence of $(\text{time}, \text{pressure})$ points.

But actually, a large problem exists. What is "Time"? As in, what is "Time", mathematically? A set of timesteps? If so, doesn't that mean we can't detect the air pressure made by some sound in between the timesteps? What does that question mean? Suppose the music you listen to has one piece of sound information (one time-pressure point) sampled every 1ms starting at the 0th millisecond of a music, then what happened to the information of sound at time 1.5ms? It simply isn't recorded. In other words, the sound signals that we usually encounter, or listen to at the very least, are discrete, not continuous. Discrete means that the number line our metric works on does not contain every single number; for example, if the sound signals we listen to on Spotify are sets of time-pressure points at $1ms$ intervals. A continuous signal would be a sound signal that has the air pressure of every single granularity of seconds (in other words, for every real number worth of miliseconds, up to infinite floating point precision), which would be impossible to directly store in a computer as it contains infinite information.

As such, in computer, a sound is represented as an array of numbers. Spotify's sample rate, for example, is 44.1kHz. That is, 44100 time-pressure points, or samples, are beign collected per second. This unit Hz stands for the frequency at which sampling is performed, where 1 Hz (Hertz) is equivalent to 1 cycle (of sampling) per second.

14.2.2 Images

Let us look at this following picture:

Notice that this picture is made of many little squares, which we call pixels. Every picture is made of many little pixels, and when pieced together into one large rectangular array, presents any image that we would usually see in our computer. That means, to produce a high resolution picture, we must prepare a lot of pixels, and therefore a lot of computer storage.

For simplicity, let us represent each pixel with a number that tells us how intense the pixel's color is on a black-to-white scale (with black colors being 0, white being 1). Then, suppose we have this little 20×20 picture above that shows a bird, then the image can be described as a signal that reports a greyscale intensity provided a target pixel:

$$\text{Image} : \{1, 2, \ldots, 20\} \times \{1, 2, \ldots, 20\} \to \{0, \frac{1}{256}, \ldots, 1\}$$

or in a literal sense,

$$\text{Image} : \text{Horizontal} \times \text{Vertical} \to \text{Gresycale Intensity}$$

To make the picture more colorful, we can instead represent colors as RGB triplets (Triplet of redness, greenness, and blueness of a color) in place of the original discrete greyscale intensity. A colored version of the picture would then be a signal of the following form:

$$\text{Image} : \{1, 2, \ldots, 20\} \times \{1, 2, \ldots, 20\} \to \{0, \frac{1}{256}, \ldots, 1\}^3$$

or in a literal sense,

$$\text{Image} : \text{Horizontal} \times \text{Vertical} \to \text{RGB Triplet}$$

14.2.3 Physical Attributes

The positions of a robot device, such as a plane, can also be expressed as a signal, as they are also information that map the physical status of a device into quantifiable terms. The easiest representation of robot positions can be a three-dimensional coordinate on Cartesian space, or a spherical coordinate with respect to Earth's center.

$$\text{Position} : \text{Time} \to \mathbb{R}^3$$

In such way, many vector representations of physical objects that we will from now on study can also be treated as a signal.

14.3 Event Stream and Sampling

As spoken before, information are usually computationally represented as a sequence of values across specific discrete points of time and/or space. These sequences of points that represent a signal are otherwise known as an event stream. For example, we have discussed an image as a sequence of pixel color values across the coordinates of its pixels. For

those represented information, such as the RGB triplets of pixels or pressure value of sound waves at timepoints, we call them "symbols".

Now that we have decided how to represent a signal in our computers discretely, we should also discuss how to obtain a discrete representation of some continuous waveform, such as an ongoing voice of violins. In such case, let us take inspiration from our proposed form of audio storage before, that we only record 44100 samples of the violin voice each second. In such case, we measure soundwaves discretely at intervals of $\frac{1}{44100}$ seconds. This interval at which we obtain samples of the audio is known as a sampling period. Meanwhile, to make the pressure value (which was originally on a continuous real number line in physical world) discrete, we also let the pressure value in soundwaves be approximated to one specific number on a number line of 256 uniformly spaced values. All of these are examples of converting an analog signal, a signal with continuous domain and range, into a digital signal, a signal with discrete domain and range such that it may be computationally processed.

14.4 Introduction to Systems

14.4.1 Definition of Systems

Systems transform signals into other signals. On top of that, there are a couple of things systems can do to help with this main mission:

- A system can **store** representations of some information for preservation purposes. This can later help with many other things.
- A system may be purposed for **transmission**, which is the delivery of one information into another system, so a system may want to preserve or transfer the information in different forms.

  In such case, a system **encodes** an information by coming up with a different representation of the original information (electronic storage of violin sounds). Meanwhile, after this component, some other system will come up with a **decoder** that decodes the information into its unencoded form.

  The encoder and decoder are together called a **codec**.
- A system can also instead transform a signal for the difficulty of reading (instead of making it easier to read). Or, to protect information, a machine can turn an information into another form that requires access to read. This is called **encryption**. The opposite is **decryption**.
- A system can also emphasize or annotate a signal to make the important parts of its information more visible. This is called **enchancing** a signal.
- Extraction of digital information from a signal is known as **detection**.
- Systems can also help **control** a specific state, such as room temperature or plane position. Such system is usually inputted the desired state via a **controller**, and collects signal to decide what to do to help with the control mission.
- Systems may also **translate** signals from a form into another (such as Google Translate).

14.4.2 Abstraction of Systems

We have spoken many possible purposes of a system, a signal-to-signal function, in the above bullet points. Then, systems can indeed be formaliezd as a function from a set of signals to another set of signals. For example, a system S that maps a signal x into another output signal y presents that $y = S(x)$.

Let us now concern the definition of systems at the level of not a function, but functions:

$$X = [D \to R] = \{x | x : D \to R\}$$

which states that X, mapping from a domain D to a range R, is the set of all x that is a function mapping D to R. The

set-builder notation is more elucidating. This set is known as a signal space, or function space.

Then, provided that systems map an input signal onto an output signal, we may discuss systems as a function whose domain and range are both signal spaces (a space of signals), such that for $y = S(x)$ and an input signal function x, the output signal y is similarly a function that takes in some space-time related information and outputs a symbol. Therefore, the following expression:

$$S(x)(z)$$

can be interpreted as the signal (function) $y = S(x)$ taking in a space-time information z and outputting the corresponding symbol or sample.

Chapter 15

Mathematical Definitions for Functions

15.1 Assignment and Declaration of Function

In the realm of mathematics, we seek for a unified approach to formally define mathematical objects like functions. Mathematical objects are objects with mathematical properties, or, "things" that exist in mathematics. Previously, we discussed a few way to define and denote a function. In this section, let us discuss these idea more.

As spoken before, a function is fundamentally a relationship between sets. A function denoted as $f : X \to Y$ has a name f, domain X, and codomain Y; furthermore, it matches each element of X to one and only one element in Y. And once again, this has two implications:

1. Each element x of X must have a corresponding value in the codomain of function Y.
2. Each element x of X cannot have multiple corresponding value in the codomain of function Y.

Previously, in Algebra II, you may have learned such a thing called "Vertical Line Test", which asserts that the graph of a function on a 2-dimensional space, with the horizontal axis hosting the domain and vertical axis the codomain, must have no more than one intersection than any vertical line. In the sense of usual single-variate functions $y = f(x)$, it is that one value of x can only have one value of corresponding y. For example, $f(x) = x^2$ is a function that only has one intersection with any vertical line.

To declare the relationship that a function hosts between its domain and codomain: we define a function with the following prototype:

$$\forall x \in X, f(x) = \text{expression in x}$$

This definition of the function is also said to be **declarative**, because it declares properties of the function without explaining the precise algorithms (or computer instructions) of doing so.

15.2 Different Expressions of A Function: Graph, Table, Procedure

Functions may also be expressed in several different ways.

15.2.1 Graph

For a function $f : X \to Y$, its relationship provides us as many elements in X as the number of corresponding values across domain and codomain: (x, y), or in other words $(x, f(x))$. Meanwhile, we may notice that the Cartesian product $X \times Y$ hosts the set of all possible pairings between elements of X and elements of Y:

$$X \times Y = \{(x, y) | x \in X, y \in Y\}$$

And the function f instead hosts such set of tuples:

$$\{(x, f(x))|x \in X\} \subset X \times Y$$

We call this above set of tuples containing the function's mapping the **graph** of f, or, $\text{graph}(f)$. Therefore, a function can also be defined as a subset of the Cartesian product of its domain and codomain.

15.2.2 Table

We may also express a function in a table that maps an input value to an output value. This is perhaps the most primordial, non-mathematical method of exhibiting a function. For example, for a function $f : \mathbb{R} \to$ Set of student name where the input value is a Student ID, we may write such function in a table as follows:

Domain: Student ID	Codomain: Student Names
3036594561	John Doe
3036594562	Jonathon Doe
⋮	⋮
3036595000	Joe Dohn

15.2.3 Procedure

A procedure is an algorithm that notes how a function can be computed using algorithmic notations. For example, to compute the factorial function, we may use the following instruction: Such set of instruction also defines our function:

Algorithm 1 A set of instructions for computing factorials

Require: $n \geq 0$
Ensure: $y = x!$
  $y \leftarrow 1$
  $N \leftarrow n$
  while $N > 0$ **do**
  $y \leftarrow y \times N$
  $N \leftarrow N - 1$
  end while

instead of providing the mathematical abstractions,

$$\forall x \in \mathbb{N}_0, f(x) = x!$$

it provides the precise instructions required for anyone (even a computer) to compute the function. Using procedures to define functions is otherwise known as an imperative assignment.

15.3 A Review of Composition

We can combine functions to create new functions. This mechanism is known as composition. Without assuming anything about the correspondence of different functions' domains and codomains, for two functions f_1 and f_2, we may write the composition of these functions as:

$$(f_2 \circ f_1)(x) = f_2(f_1(x))$$

such that $f_2 \circ f_1$ becomes a function that accepts any element x in the domain of f_1 and computes the above value.

Note that, the fundamental requirement for such composition to be held is that the range of f_1 must eb a subset of the domain of f_2, such that the output of the first function f_1 is always within the domain of the second function f_2. So, to end with a more precise definition of composition:

Definition 15.3.1. Composition of Function

Assume $f_1 : X \to Y$ and $f_2 : X' \to Y'$, then a composition $f_3 = f_2 \circ f_1$ would be defined as a function $f_3 : X \to Y'$ and declared as:

$$\forall x \in X, f_3(x) = f_2(f_1(x))$$

Chapter 16

Mathematical Definitions for Signals and Systems

16.1 Defining Signals

Because a signal is fundamentally a function, the conventions of defining a signal closely follow the conventions of defining a function. For example, suppose we would like to define an audio signal as a function of time that returns the soundwave air pressure of some specific point, then we can simply define this function declaratively as:

$$\forall t \in \text{Time}, s(t) = \sin(440 \times 2\pi t)$$

which states that the air pressure of our sound signal is a sinusoidal function with respect to time.

On the other hand, we may also define our function imperatively when programming the signal, which would result in the portrayal of abbove function using code:

```
import numpy as np
signal = lambda t: np.sin(440 * 2 * np.pi * t)
```

You will be frequently using imperative definitions in lab assignments, and declarative definitions when working with the mathematical representation of signal in homework assignments or exams.

16.2 Defining Systems

16.2.1 A Review of Systems

Remember that we define systems as functions that map an input signal space (a space of all possible input signals) onto an output signal space. Therefore, systems are functions whose domains and ranges can be expressed in: $S : [D \to R] \to [D' \to R']$. Alternatively, we can describe a system S with the graph of its corresponding function:

$$\{(x, S(x)) | x \in [D \to R]\}$$

Such graph fundamentally describes every possible input-output signal pair, which directly depicts the system's behavior upon any provided input. Therefore, we also address the graph of a system's function S as its behavior.

What about using a table instead of a graph? Unfortunately, there is usually too many input signals to consider for a system. The construction of tables assume a finite input domain and finite output domain, and needs to exhaustively list all input-output correspondences. However, systems very often encounter an infinite input domain (which means

there can be infinite possible inputs), if not an extremely large input domain (eg., the set of all possible 1920×1080 pictures). Therefore, tables are less practical choices for mathematical representations of system behavior than sets are.

16.2.2 Systems and Its Memories

A system is **memoryless** when it only depends on the current input state. We call such systems "memoryless" by the fact that it only depends on the present input, but neither past input nor information regarding timestep.

Definition 16.2.1. Memoryless Systems

To ensure that a system $F : [\mathbb{R} \to Y] \to [\mathbb{R} \to Y]$ is memoryless, we require that there exists a function $f : Y \to Y$ such that:

$$\forall t \in \mathbb{R}, x \in [\mathbb{R} \to Y], (F(x))(t) = f(x(t))$$

such that the system F can be portrayed as a function that solely depends on the current input $x(t)$.

On that note, a system with memory would be any system whose expression $(F(x))(t)$ relies on information outside of $x(t)$. Let us refine our understanding with a couple of examples below:

Example 16.2.1: An Example of Memoryless System

Suppose our system S receives an input signal x and outputs signal y such that:

$$\forall t \in \mathbb{R}, y(t) = x(t)^2$$

In such case, we can model the system's relation solely upon the value of $x(t)$, so the system is memoryless.

Example 16.2.2: An Example of System with Memory

Suppose our system S receives an input signal x and outputs signal y such that:

$$\forall t \in \mathbb{R}, y(t) = \sum_{t'=0}^{t-1} (0.99)^{t-t'} x(t')$$

Becuase this system's relation depends on information outside of current input, such as past input signals and timestep information, this system has and requires memory.

16.2.3 Using Difference and Differential Equations

Systems can also be defined using equations that model the inner dynamics of the system. For continuous signals, such equations are differential equations, which are equations that describe the rate of change of a variable as an expression of that variable. These equations will be frequently encountered at the end of MATH 1B, as well as some aspects of this course.

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = x(t) + \lambda y(t)$$

Meanwhile, discrete signals use difference equations because differential equations are not fitting for non-continuous functions like discrete-time functions. Difference equations are equations that relate the output signal with the input signal; essentially, it is the expression of the system function. For example:

$$y(n) = \frac{x(n) + x(n-1)}{2}$$

16.2.4 Block Diagrams as Schematics

A block diagram is a visual summary of system using blocks to represent the components of a complicated larger system. Usually, block diagrams are presented on design blueprints and research papers to help readers visually summarize the schematics of a large system. Let's use the following block diagram, which protrays the system:

$$\forall t \in \mathbb{R}, S_1(x)(t) = y_1(t) = x(t)^2 + 2x(t) + \frac{1}{x(t)}$$

as an example:

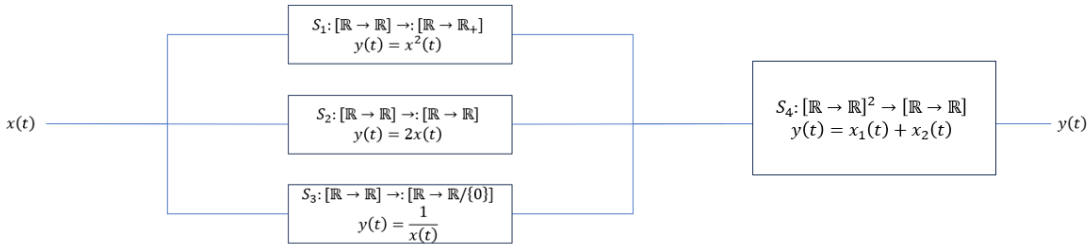


Figure 16.1: Example block diagram of the above memoryless system.

In a block diagram, each block resembles one smaller component of the system (which can also be a system) that maps the input signal into some intermediate form (which can also be an output signal). Ideally, each block only resembles one function. Then, throughout the block diagram, the outputs of each block become the input of its connected blocks, flowing from left to right. It provides us a graphical comprehension of the system we deal with.

Occassionally, our systems would require memory, which means that we may rely on information about the past input signals. For example, such system:

$$\forall t \in \mathbb{R}, S_2(x)(t) = y_2(t) = y_2(t-1) + \frac{1}{x(t)}$$

The connection from a previous output state into a current output state's computation is known as a **feedback**. This name is intuitive in the sense that whenever a feedback occurs, it is that a previous output state feeds back to another current output state (and this can be either additive or subtractive, sometimes even multiplicative). Let us see how a block diagram would work with this information:

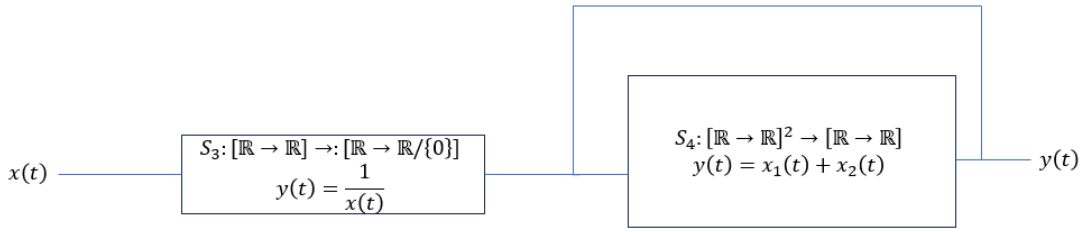


Figure 16.2: Example block diagram of the above memory system.

In a block diagram, feedbacks are portrayed by connections between the output state per se and the associated computing blocks. In the above diagram, the output signal's output value, $y(t)$, will be fed into the system's adder block in the next timestep $t + 1$ to achieve the recrusive formula above.

Chapter 17

State Machines

17.1 Structure of State Machines

We refer to the current status of a system using the word "state". In daily English usage, we may say, "the status of Ryan is 'drunk' (albeit that rarely occurs)", "the state of Winston is that he is insane", etc. In systems, we may instead say the state of a system is represented by some vector. For example, the state of a robot dog may be a vector involving the position of its legs, head, body, etc. Such characterizations are called "state-space models": that is, models' current situation are specified by a vector-representable "state".

Let us summarize all of these systems, models, robots, planes... anything that uses a state to represent its current situation as a "state machine". A state machine has a mathematical prototype just as any systems. The input and output signals of a state machine are called "event streams", or put blatantly it is a sequence of signals that have occurred (and therefore a stream of events, event stream)

$$EventStream : \mathbb{N}_0 \to Symbols$$

A stream of event is a numerically ordered sequence of inputs and outputs that occur throughout the specified timesteps. In this case it could be represented like:

$$\begin{aligned} EventStream(0) &= \text{Sleeping} \\ EventStream(1) &= \text{Awake} \\ EventStream(2) &= \text{Sleeping} \\ EventStream(3) &= \text{Sleeping} \\ &\vdots \end{aligned}$$

where "Sleeping" and "Awake" are two different symbols that represent the occuring event in the event stream at the input timestep.

A state machine takes the mathematical prototype of:

$$StateMachine = (States, Inputs, Outputs, update, initalState)$$

meaning, a state machine is composed of five things:

Component Name	Component Item
States	A set of all possible states for the machine
Inputs	A set of all possible inputs for the machine
Outputs	A set of all possible outputs for the machine
update	A function about changes in state and output of the machine given input and current state
initialState	The initial state of the machine; how the machine starts up

This mathematical prototype uses sets and functions to model (represent) a state machine. Then, the event streams of a state machine are defined as follows:

$$InputSignals = [\mathbb{N}_0 \to Inputs]$$
$$OutputSignals = [\mathbb{N}_0 \to OutputSignalsputs]$$

One other interesting subtlety here is that we don't necessarily say the StateMachine moves through time. Instead, we say that the machine "evolves" (changes its states, outputs) over steps (unit of time for new changes in input). Alternatively, we say the machine evolves through timesteps. This bodes similarity to the pagerank (or pump system) eigenanalysis we processed in previous chapters.

17.1.1 Updates in State Machine

An update in a state machine is a function that provides the output signal and next state of a machine provided the input signal and the current state of a machine. Particularly, we can also mathematically state that:

$$(s(n+1), y(n)) = update(s(n), x(n))$$

How do we piece these components in the state machine together? Or, how does a state machine function and why is the update function necessary? Remember that state machines, or systems, are abstractions of a process that occurs in real life and changes upon occurring things. For example, the human brain can be a state machine whose states are the chemicals in the brain, and outputs are the feelings we usually perceive. State machines operate along the following procedure:

1. State machine starts up (we are born into this cruel world). The initial state of the brain is not known to science, but we called it the $InitialState$
2. Now, forever in your life, at the unit of miliseconds:

   Your brain perceives an input signal (can be a color, a noise, or memes)

   Your brain's inner systems process this signal, and eventually, the state of your brain's chemicals change due to this input. Your brain also outputs a reaction. This is what we called an update.

   Your brain outputs, and the state of your brain changes. You continue within this loop as input signals continue to trigger updates in your brain.
3. You will never escape the circle of life until you die.

The indented aspects of the above workflow that is associated with an input-update pair is also called a "reaction". Additionally, whenever there are no input signals (which occurs not for the human brain but for other machines), there would of course not be any output signals. We call this a stutter.

17.1.2 Minecraft as A State Machine

Let's reinforce the above logic further with Minecraft as an example of a state machine. First, let's briefly define the components of Minecraft as aspects of a state machine:

States: The current state of creatures in blocks in your Minecraft world, player status, object durabilities

Inputs: Keyboard inputs

Outputs: Game-player interactions that are not state of the game, such as sound effects.

initialState: The state of the game when world starts

update: Minecraft's code about how the state of your game changes whenever you do something.

Now, let's do an input signal that mines a diamond. Then the following update loop occurs:

1. The input signal and the current state is considered by Minecraft's code.
2. A diamond is digged, so the current state changes based on that (player gets a diamond, some experience values, and a diamong block is removed from the game).
3. A diamond digging sound effect and experience gaining sound effect is outputted. This is the output signal

This update loop occurs for any input signal we provide in Minecraft.

17.2 Finite State Machines

Finite State Machines, otherwise abbreviated as FSM, is a state machine whose amount of state is finite. For example, if we use a state machine to represent the status of a student whether the student is sleeping or not, then our state machine has a finite amount of state (either the student is "Awake", or "Asleep"). Then, this state machine is a finite state machine. Although we discuss finite state machines here, given that the interpretation of finite state machines is in CS61C's syllabus, not EE20N's, we will not go over the construction of a FSM diagram.

17.2.1 Formalizing Updates as State Transition

Previously, we mentioned the process at which machine state changes and output is generated to be an "update". This state-to-state change is otherwise called a "state transition", which is by its name the transition of states. In certain fields, we would state the transition in a tuple:

$$(CurrentState, InputSignal, NextState, OutputSignal)$$

alternatively, you may think of the InputSignal as an action done to the system, and the OutputSignal as some feedback from the system that talks about the transition (for example, is this transition good? harmful to the system?).

Several things may occur with a transition. For example, we may have a transition whose $CurrentState$ and resulting $NextState$ is the same. In Minecraft, this can simply be pausing the game, where the $InputSigna$ doesn't change the gamestate. This type of transition is called a "loop", because the provided input signal has us stuck in the same state despite having performed an action.

Chapter 18

Infinite State Machine

18.1 Definition of ISM

18.1.1 Mathematical Abstraction of ISM

An infinite state machine is fundamentally a state machine that has infinite possible states. These machines have just the same components as a usual state machine does, except the spaces $States$, $Inputs$, $Outputs$ now become:

$$\begin{cases} States & = \mathbb{R}^n \\ Inputs & = \mathbb{R}^m \\ Outputs & = \mathbb{R}^k \end{cases}$$

for some arbitrary n, m, k.

For the sake of terminology, let us also discuss the dimensionality of such systems. A system that receives one-dimensional inputs and provide one-dimensional outputs is known as a single-input, single-output (SISO) system. On the opposite side is a multiple-input, multiple-output (MIMO) system, which is when $m > 1$ and $k > 1$. This knowledge wouldn't really contribute a lot to your knowledge of the syllabus content.

18.1.2 Composition of ISM

Remember that an infinite state machine is still the 5-components tuple of:

$$M = (States, Inputs, Outputs, update, initialState)$$

then, besides the above listed change, we should also observe that the $update$ function can now be defined as follows:

$$update : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n \times \mathbb{R}^k$$

as it takes in an input signal, a current state, and outputs an output signal and a current state.

So, just as any system, an infinite state machine has the necessary components to build an update loop. That is, an equation that decides the next state of the system:

$$\forall n \in \mathbb{Z}, n \geq 0, s(n+1) = nextState(s(n), x(n))$$

and an equation that determines the output system

$$\forall n \in \mathbb{Z}, n \geq 0, y(n) = output(s(n), x(n))$$

The composition of the output loop builds what we call the "state-space model" of our system.

18.1.3 Involvement of Time Measurement

Note that the unit of time in our machine's state evolution is still "step". Across each "step", some input is provided to the machine for a stepwide update. And, because step is a discrete unit of time (you can only have a nonnegative integer amount of steps passed in a system), step becomes what we call the "time index" of the system, which is the identifier of time for in-machine events. When the update rules ($nextState$ and $output$) of the system doesn't change across steps, we call it a **time-invariant** system.

Chapter 19

Linear Systems

Hello

19.1 Definition of Linear Systems

Hello

19.1.1 Impulse Response

Hello

19.2 One-dimensional SISO Systems

Hello

19.3 Multidimensional SISO Systems

Hello

19.4 Multidimensional MIMO Systems

Hello

19.5 Linear Input-output Function

Hello

Part IV

Signal Processing

Chapter 20

It's Time to Get a Little Complex

Hello

20.1 Review of Imaginary Numbers

20.1.1 Definition of Imaginary Numbers

Hello

20.1.2 Arithmetics of Imaginary Numbers

Hello

20.2 Review of Complex Numbers

20.2.1 Definition of Complex Numbers

Hello

20.2.2 Arithmetics of Complex Numbers

Hello

20.3 Trigonometric Representation of Complex Numbers

20.3.1 Exponentials

Hello

20.3.2 Polar Coordinates

Hello

Chapter 21

Frequency, Phase, Domain

21.1 Frequency Decomposition

Hello.

21.2 Phase

Hello.

21.3 Spatial Frequency

Hello.

21.4 Periodic and Finite Signals

Hello.

Chapter 22

Fourier Expansion: Infinite Terms

22.1 Fourier Series

22.1.1 Purpose and Definition

Hello

22.1.2 Fourier Series Approximation and Convergence

Hello

22.1.3 Uniqueness of Fourier Series

Hello

22.1.4 Approximations to Images

Hello

22.2 Discrete-time Signals

Hello

22.2.1 Periodicity

Hello

22.2.2 Discrete-Time Fourier Series

Hello

Chapter 23

Linear Time-Invariant Systems

23.1 Time Invariance

Hello

23.2 Linearity of System

Hello

23.3 Linearity and Time-Invariance

Hello

Chapter 24

Frequency Response and Fourier Series

24.1 Finding and Using Frequency Response

24.1.1 Introduction

Hi

24.1.2 Linear Difference and Differential Equations

Hi

24.1.3 Fourier Series with Complex Exponentials

Hi

24.1.4 Solving Coefficients of Fourier Series

Hello

24.1.5 Frequency Response and Fourier Series

Hello

24.2 Frequency Response of Composite Systems

24.2.1 Cascade Connection

Hello

24.2.2 Feedback Connection

Hello

Chapter 25

Introduction to Filtering

25.1 Convolution

25.1.1 Convolution Sum and Convolution Integral

Hello

25.1.2 Impulses

Hello

25.1.3 Weighted Sum of Delta Functions

Hello

25.2 Impulse Response

25.2.1 Impulse Response and Convolution

Hello

25.2.2 Impulse Response and Frequency Response

Hello

Chapter 26

Impulse Resopnse Filters

26.1 Causality

Hi

26.2 Finite Impulse Response Filters

26.2.1 Definition of FIR Filters

Hello

26.2.2 Design of FIR Filters

Hello

26.2.3 Decibels

Hello

26.3 Infinite Impulse Response Filters

26.3.1 Definition of IIR Filters

Hello

26.3.2 Design of IIR Filters

Hello

26.4 Implementation of Impulse Response Filters

26.4.1 Fundamental Policies and Terminologies

Hello

26.4.2 Signal Flow Graphs

Hello

Part V

Sampling and Fourier Transform

Chapter 27

The Four Fourier Transforms, Part I

Notations

Hello

27.1 Continuous-Time Fourier Series

Hello

27.2 Discrete Fourier Transform

Hello

Chapter 28

The Four Fourier Transforms, Part II

28.1 Discrete-Time Fourier Transform

Hello

28.2 Continuous-Time Fourier Transform

Hello

Chapter 29

A Discussion of Fourier Transform

29.1 Fourier Transform vs. Fourier Series

Hello

29.2 Properties of Fourier Transform

Hello

Chapter 30

Sampling and Reconstruction

30.1 Sampling

30.1.1 Sampling and Aliasing

Hello

30.1.2 Perceived Pitch Experiment

Hello

30.1.3 Aliasing Ambiguities

Hello

30.2 Reconstruction

30.2.1 Survey and Introduction

Hello

30.2.2 Mathematical Assumptions of Reconstruction

Hello

Chapter 31

The Nyquist-Shannon Sampling Theorem

31.1 The Statement of Theorem

Hello

31.2 A Scratch of Its Proof

Hello