Linear Algebra

Chapter 0: The Function (and other mathematical and computational preliminaries)

University of Seoul
School of Computer Science
Minho Kim

Table of contents

Basic mathematical concepts

- Sets
- Sequence (ilsts)
- Functions
- Probability theory

Outline

Set terminology and notation

Cartesian product

The function

Probability

Lab: Introduction to Python – sets, lists, dictionaries, and comprehensions

Lab: Python – modules and control structures – and inverse index

Set terminology and notation

- Definition?
- $\{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$
- $\heartsuit \in \{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$
- $S_1 \subseteq S_2$
- $S_1 = S_2 \Leftrightarrow S_1 \subseteq S_2$ and $S_2 \subseteq S_1$
- $|S|$: **cardinality** of S

Set Expressions

- Set builder notation

$$\{x \in \mathbb{R} : x \geqslant 0\}$$

  - “the set of non-negative real numbers”
  - $\mathbb{R}$: the set of real numbers

- **Set comprehension** in Python

```
a = {x for x in range(1,10) if x%2==0}
```

  - “even numbers less than 10”
  - `range(1,10)`: a sequence type denoting $1, \ldots, 9$ (excluding 10!)
  - %: modulo operator

Outline

Cartesian product

The Cartesian product of two sets A and B is the set of all pairs (a, b) where $a \in A$ and $b \in B$.

$$A \times B := \{(a, b) : a \in A \text{ and } b \in B\}$$

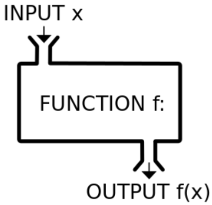
Proposition 0.2.3

For finite sets A and B, $|A \times B| = |A| \cdot |B|$.

Outline

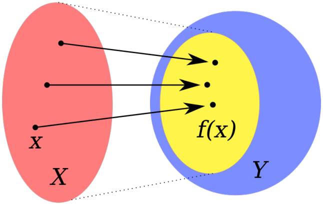
Functions

- "...a rule that, for each element in some set D of possible inputs, assigns a possible output."
- "...a (possibly infinite) set of pairs (a, b) no two of which share the same first entry."
- (Wikipedia) "...a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output."



(Wikipedia)

Functions (cont'd)



$f : X \to Y$

- $r = f(q)$
  - r is the **image** (상 : 像) of q under (the function) f.
  - q **maps** to r under (the function) f
    cf) $q \mapsto r$ "q maps to r"
  - q is the **pre-image** (원상 : 原像) of r under (the function) f.
- $f : X \to Y$
  - X is the **domain** (정의역 : 定義域) of f
  - Y is the **co-domain** (공역 : 共域) for f
  - "a function from X to Y" or "a function that maps X to Y"
- $\operatorname{ran} f = f(X) = \{f(x) : x \in X\} \subset Y$
  - **range** or **image** (치역 : 値域) of f

Procedures vs. Computational Problems

- **Procedures**
  - a precise description of a computation
  - accepts **inputs** (called **arguments**) and produces an output (called the **return value**)

```
def mul(p,q): return p*q
```

- **Computational problems**
  - an input-output specification that a procedure might be required to satisfy

input: a pair (p, q) of integers greater than 1
output: the product pq

input: an integer m greater than 1
output: a pair (p, q) of integers whose product is m

Procedures vs. Computational Problems (cont'd)

- a function or computational problem does not give us any idea how to compute the output from the input
- sometimes the same procedure can be used for different functions
- unlike a function, a computational problem needs not specify a unique output for every input

Functions vs. Computational Problems

- For each function f, there is a corresponding computational problem

The forward problem

Given an element a of f's domain, compute $f(a)$, the image of a under f.

The backward problem

Given an element r of the co-domain of the function compute any pre-image (or report that none exists).

- The backward problem is usually more difficult.
  $\rightarrow$ only for specific functions
  $\rightarrow$ intractability vs. inapplicability
  $\rightarrow$ **linear functions**

Functions in Python

Generally a function is **implemented** as a Python procedure

```
def mul(p,q): return p*q
```

A finite discrete function can be represented as a Python dictionary.

```
>>> CountryCodes = {'South Korea':82, 'United
States':1, 'China':86, 'Japan':81}
```

Set of Functions and Identity Function

- F^D denotes the set of all functions from D to F.

Fact 0.3.9

For any **finite** sets D and F, $|D^F| = |D|^{|F|}$.

- For any domain D, there is a function $\mathrm{id}_D : D \to D$ called **identity function** for D, defined by

$$\mathrm{id}_D(d) = d$$

for every $d \in D$.

Composition

- Given two functions $f: A \to B$ and $g: B \to C$, the function $g \circ f$, called the **compotision** of f and g, is a function whose domain is A and its co-domain is C and is defined by the rule
$$(g \circ f)(x) = g(f(x))$$
for every $x \in A$.
- (Associativity of composition) For functions f, f, h,
$$h \circ (g \circ f) = (h \circ g) \circ f$$
if the composition are legal.
$\to h \circ g \circ f$

Functional Inverse

Definition 0.3.13

We way that functions f and g are **functional inverses** of each other if

- $f \circ g$ is defined and is the identity function on the domain of g, and
- $g \circ f$ is defined and is the identity function on the domain of f.

Requirement for **invertibility**

Definition 0.3.14

Consider a function $f : D \to F$. We way that f is **one-to-one** (or **injective**) (단사함수:單射函數) if for every $x, y \in D$, $f(x) = f(y)$ implies $x = y$. We say that f is **onto** (or **surjective**) (전사함수:全射函數) if, for every $z \in F$, there exists $x \in D$ such that $f(x) = z$.

Functional Inverse (cont'd)

Lemma 0.3.16 & 0.3.17

An invertible function is one-to-one and onto.

Theorem 0.3.18 (Function Invertibility Theorem)

A function is invertible iff (if and only if) it is one-to-one and onto.

Lemma 0.3.19

Every function has at most one functional inverse.

Lemma 0.3.20

If f and g are invertible functions and $f \circ g$ exists then $f \circ g$ is invertible and $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$.

Outline

Probability

- Probability theory is just about...
  - What **could** happen and
  - **How likely** it is to happen
- Used to make **predictions** about a **hypothetical** experiment.
- cf) **statistics** (통계학) – used to figure out **what it means** once something actually happens.

Probability Distribution

- A **(discrete) probability distribution** $\Pr : \Omega \to \mathbb{R}^+$
  - Ω finite domain
    - **outcomes**: elements of domain Ω
  - $\mathbb{R}^+$ non-negative reals
    - $\Pr(\omega)$: the **probability** of (the outcome) ω
  - $\sum_{\omega \in \Omega} \Pr(\omega) = 1$
- Uniform vs. Non-uniform distributions
  - ```
    >>> PrCoin = {'heads':1/2, 'tails':1/2}
    ```
  - ```
    >>> PrDie = {1: 1/6, 2:1/6, 3:1/6, 4:1/6,
    5:1/6, 6:1/6}
    ```
  - ```
    >>> PrTwoCoins = {'both heads':1/4, 'head and
    tail':1/2, 'both tails':1/4}
    ```
  - letters of a Scrabble game
    ```

Events

Fundamental Principle of Probability Theory

The probability of an event is the sum of probabilities of the outcomes making up the event.

- **event**: a set of outcomes
- Example: The probability of rolling an even number of a dice = $1/6 + 1/6 + 1/6 = 1/2$

Applying a function to a random input

- Example 0.4.6
- Quiz 0.4.7
- Example 0.4.8
- Example 0.4.9

Perfect Secrecy

Cryptosystem requirements

- The intended recipient of an encrypted message must be able to decrypt it.
- Someone for whom the message was not indended should not be able to decrypt it.

Kerckhoffs Doctrine

The security of a cryptosystem should depende only on the secrecy of the key used, not on the secrecy of the system itself.

- Kerckhoffs Doctrine at Wikipedia
- cf) Security through obscurity

Perfect Secrecy: Example

- Sending a one-bit message p.
- p is encrypted by a key k into a cyphertext c according to a (publicly known) coding table.
- Secrecy of two schemes?

scheme 1

p	k	c
0	$\clubsuit$	0
0	$\heartsuit$	1
0	$\spadesuit$	1
1	$\clubsuit$	1
1	$\heartsuit$	0
1	$\spadesuit$	0

scheme 2

p	k	c
0	$\clubsuit$	0
0	$\heartsuit$	1
1	$\clubsuit$	1
1	$\heartsuit$	0

Outline

Set terminology and notation

Cartesian product

The function

Probability

Lab: Introduction to Python – sets, lists, dictionaries, and comprehensions

Lab: Python - modules and control structures - and inverse index

Simple Expressions

- Arithmetic and numbers
  - `**` – exponentiation
  - `//` – truncating integer division
  - `%` – modulo
- Strings
  - enclosed by single or double quotes
- Comparisons, conditions and booleans
  - `==, <, >, <=, >=, !=`
  - `True, False, not`
- Collections – sets, lists, tuples, dictionaries

Assignment Statements

$\langle \textit{variable name} \rangle$ = $\langle \textit{expression} \rangle$

- ```
  >>> mynum = 4+1
  ```
- No declaration / No type specifier
- An assignment statement binds a variable to the *value* of an expression, not to the expression itself.

```
>>> x = 5 + 4
>>> y = 2 * x
>>> y
18
>>> x = 12
>>> y
18
```
```

Conditional Expressions

$\langle expression \rangle$ `if` $\langle condition \rangle$ `else` $\langle expression \rangle$

- $condition$ should be a boolean expression. cf) C
- ```
  2**(y+1/2) if x+10<0 else 2**(y-1/2)
  ```
  ```

Sets

- Constructed by curly braces

```
>>> {1+2, 3, "a"}
{'a', 3}
```

- Cardinality by `len()`

```
>>> len({1+2,3,"a"})
2
```

- Membership test - `in` and `not in`

```
>>> 2 in {1+2,3,"a"}
False
>>> 2 not in {1+2,3,"a"}
True
```

Sets (cont'd)

- Union - `|`

```
>>> {1,2,3} | {2,3,4}
{1, 2, 3, 4}
```

- Intersection - `&`

```
>>> {1,2,3} & {2,3,4}
{2, 3}
```

- Adding/removing an element - `add`, `remove`

```
>>> S={1,2,3}
>>> S.add(4)
>>> S.remove(2)
>>> S
{1, 3, 4}
```

Sets (cont'd)

- Union with another set - `update`
  "`A.update(B)`" is the same as "`A|= B`" or "`A=A|B`"
- Intersection with another set - `intersection_update`
  "`A.intersection_update(B)`" is the same as "`A&=B`" or "`A=A&B`"

Sets – Copying Sets

- Assigning a set to another does not copy the whole set - only one copy is stored.

```
>>> S={1,2,3}
>>> T=S
>>> T.remove(1)
>>> S
{2,3}
```

- The whole set can be copied by `copy()` function.

```
>>> S={1,2,3}
>>> T=S.copy()
>>> T.remove(1)
>>> S
{1, 2, 3}
```

Sets – Comprehension

- To build a collection out of another collection
- Mimics traditional mathematical notations (Set builder notation)
- A set can be built without any loop.

```
>>> {2*x for x in {1,2,3} }
{2, 4, 6}
```

cf) set builder notation

$$\{2x : x \in \{1, 2, 3\}\}$$

Sets – Comprehension (cont'd)

- with filter – conditional construction

```
>>> {x*x for x in {1,2,3} if x>1 }
{4,9}
```

- double comprehension – interating over the Cartesian product of two sets

```
>>> {x*y for x in {1,2,3} for y in {2,3,4}}
{2,3,4,6,8,9,12}
```

Sets - Remarks

- Empty set - `set()`

```
>>> A=set()
>>> len(A)
0
```

cf) `{}` denotes an empty dictionary.

- Set of sets is not allowed. - only hashable objects are allowed as set elements.

Lists

- Ordered sequence of values
- Mutable – Cannot be elements of sets
- Order is significant and repeated elements are allowed.
- Constructed by square brackets

```
>>> [1,1+1,3,2,3]
[1,2,3,2,3]
```

- A list can contain a set or another list.

```
>>> [[1,1+1,4-1],{2*2,5,6},"yo"]
[[1,2,3], {4,5,6}, 'yo']
```

- Length of a list – `len()`

Lists (cont'd)

- Concatenation

```
>>> [1,2,3]+["my", "word"]
[1, 2, 3, 'my', 'word']
```

```
>>> sum([ [1,2,3], [4,5,6], [7,8,9] ], [])
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Comprehension

```
>>> [2*x for x in {2,1,3,4,5} ]
[2, 4, 6, 8, 10]
```

```
>>> [2*x for x in [2,1,3,4,5] ]
[4, 2, 6, 8, 10]
```

Lists – Accessing Elements

- Indexing

```
>>> ['University','of',"Seoul"][1]
'of'
```

- Slices

```
>>> ['a','b','c','d','e'][2:4]
['c', 'd']
>>> ['a','b','c','d','e'][:4]
['a', 'b', 'c', 'd']
>>> ['a','b','c','d','e'][3:]
['d', 'e']
>>> ['a','b','c','d','e'][0:4:2]
['a', 'c']
```

Lists (cont'd)

- Unpacking

```
>>> [x,y,z] = [3,6,9]
>>> x
3
>>> z
9
```

* "`[x,y,z]`" is NOT a list of variables.

- Mutating a list

```
>>> mylist = [30,20,10]
>>> mylist[1] = 0
>>> mylist
[30, 0, 10]
```

Tuples

- Ordered sequence of elements
- Immutable – Can be elements of sets
- Constructed by parantheses

```
>>> (1,1+1,3)
(1,2,3)
```

- Indexing

```
>>> ("University", "of", 'Seoul')[2]
'Seoul'
```

Tuples (cont'd)

- Unpacking

```
>>> (a,b) = (1,5-3)
>>> a
1
```

- Comprehension

```
>>> tuple(i for i in [1,2,3])
(1, 2, 3)
```

cf) "`(i for i in [1,2,3])`" is a generator!

Collections Conversion

- `set()`, `list()`, `tuple()`

```
>>> set([1,2,3])
{1, 2, 3}
>>> set((1,2,3))
{1, 2, 3}
>>> tuple([1,2,3])
(1, 2, 3)
>>> list({1,2,3})
[1, 2, 3]
```

Iterable Sequences

- Ranges – not a list

```
>>> sum({i for i in range(2,8)})
27
>>> list(range(3,21,3))
[3, 6, 9, 12, 15, 18]
>>> tuple(range(5))
(0, 1, 2, 3, 4)
```

- Zip – from other collections all of the same length

```
>>> list(zip([1,3,5],[2,4,6]))
[(1, 2), (3, 4), (5, 6)]
```

- `reversed()`

```
>>> [x for x in reversed(range(10))]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Dictionaries

- Suitable to represent functions with finite domain
- A set of key-value pairs

```
>>> CountryCodes = {'South Korea':82, 'United
States':1, 'China':86, 'Japan':81}
```

- The keys must be immutable
- Only one key is allowed

```
>>> {0:'zero', 0:'nothing'}
{0: 'nothing'}
```

Dictionaries (cont'd)

- Indexing

```
>>> {'South Korea':82, 'United States':1,
'China':86, 'Japan':81}['South Korea']
82
```

- Membership test

```
>>> 'Korea' in {'South Korea':82, 'United
States':1, 'China':86, 'Japan':81}
False
```

Dictionaries (cont'd)

- Mutating

```
>>> CountryCodes = {'South Korea':82, 'United
States':1, 'China':86}
>>> CountryCodes['Japan'] = 81
>>> CountryCodes
{'China':  86, 'United States':  1, 'Japan':
81, 'South Korea':  82}
```

- Comprehension

```
>>> {k:v for (k,v) in zip(['KR', 'US', 'JP'],
[82, 1, 81])}
{'US':  1, 'KR':  82, 'JP':  81}
```

Dictionaries – Iterating Over Dictionaries

- `keys()`, `values()`, `items()`

```
>>> CountryCodes = {'KR':82, 'US':1, 'JP':81,
'CN':86}
>>> [k for k in CountryCodes]
['CN', 'US', 'KR', 'JP']
>>> [k for k in CountryCodes.keys()]
['CN', 'US', 'KR', 'JP']
>>> [v for v in CountryCodes.values()]
[86, 1, 82, 81]
>>> [(k,v) for (k,v) in CountryCodes.items()]
[('CN', 86), ('US', 1), ('KR', 82), ('JP',
81)]
```

One-Line Procedure

```
>>> def twice(z): return 2*z
...
>>> twice(1+2)
6
```

Outline

Set terminology and notation

Cartesian product

The function

Probability

Lab: Introduction to Python – sets, lists, dictionaries, and comprehensions

Lab: Python – modules and control structures – and inverse index

Modules

- Importing modules

```
>>> import math
>>> math.cos(math.pi/3)
0.5000000000000001
```

```
>>> from math import cos, pi
>>> cos(pi/3)
0.5000000000000001
```

Creating Own Modules

1. Dashboard → Files → Enter file name (ex, `test.py`) in "Enter new file name" and click "New"
2. Edit file

```
def my_func(x): return x**2
```

3. Save by clicking "Save" button
4. Call from the console

```
>>> import test
>>> test.my_func(3)
9
```

Reloading My Module

1. Edit `tests.py` and save it

```
def my_func(x): return x**3
```

2. Reload it

```
>>> test.my_func(3)
9
>>> from imp import reload
>>> reload(test)
>>> test.my_func(3)
27
```

Python Blocks

- Python blocks are specified by indentations!
- In `pythonanywhere.com` console, use SPACE for indentation. (4 spaces are recommended)

```
>>> for i in range(3):
...     print(i**2)
...
0
1
4
```

- `pythonanywhere.com` text editor support auto indentation. Use TAB key for manual indentation.

```
def my_func(x):
    a = 3
    return x**3
```

Reading From a Text File

- `open()`

```
>>> f = open('stories_big.txt')
>>> for line in f:
...     print(line)
```

```
>>> f = open('stories_small.txt')
>>> stories = list(f)
>>> len(stories)
50
```