English Consonants and Vowels

모든 consonants and vowels는 유성음과 무성음으로 구분된다.

목이 떠는 음: 유성음(voice sound): 모음 전체, b, d, g, m, n, ŋ, v, ð, z, ž, l, w, r, j(y)

목이 떨지 않는 음: 무성음(voiceless sound): p, t, k, f, s, h, sh, ch, θ

모음에는 단모음인 monophthongs 와 복모음인 diphthongs 가 있다.

Phonetics

Phonology: 음운론 Phonetics: 음성학 Speech: 사람의 말

Articulatory phonetics: 사람이 만들어내는 원리, 성대가 펄럭이며 소리를 만들어 냄.

성대 펄럭임: 남자 100/s, 여자 200~250/s

Acoustic phonetics: 소리와 공기의 반응, 사람이 개입되지 않음.

Auditory: 어떻게 듣는가에 대한, 다시 사람이 수반된 매커니즘; 고막, 귓바퀴로 인한 소리의 증폭

Articulation

The vocal tract(관)

Upper vocal tract

구성 부위: Palate(구개), Alveolar(윗니 뒤쪽 턱처럼 나온 부분), Velum/Soft Palate(연구개)

Uvula(목젖), Upper Lip, Upper Teeth, etc.

Lower vocal tract

구성 부위: Epiglottis(성대 뚜껑), Glottis(성대), Lower Lip, Tongue, etc.

Nasal track: Open for nasal sounds: closed by Velum; Velum goes up: nasal track is closed

Velum goes down: nasal track is open.

Nasal track이 막히면 모든 모음과 비음을 제외한 자음은 발은 가능하다.

5 speech organs = constrictors = articulators: lips, tongue tip, tongue body, velum, larynx(voicebox)

Control of constrictors(articulators): Constriction location: where? 앞뒤

Constriction degree: how much? 상하

CL: Lips: 앞부터 bilabial, labiodental Tongue tip: 앞부터 dental, alveolar, palato-alveolar, 말리는 retroflex

Tongue body: 앞부터 palatal, velar

CD: Stops: p, t, k, b, d, g, m, n, ng Fricatives: z, s, f, h, v, th Approximants: r, l, w, j(y)

Vowels: vowels *모든 자음은 stops, fricatives, approximants 중 하나이다.

*모든 모음은 constrictor로 tongue body 만 사용한다.

Praat 음성 분석 구성: formant(can be multiple, Hz), pitch(Hz), duration(sec), intensity(dB)

Praat 사용 및 분석

Duration: 클릭 드래그로 특정 부분 선택, 윗부분에서 초를 읽기, 확대하여 자세히 보기

Praat 아래쪽: spectrogram, 이를 분석: spectral analysis Spectrogram의 x축은 time, y축은 frequency

색이 진한 것이 더 크게 나타나는 음. Low frequency: high energy High frequency: low energy

Sine Wave: frequency & magnitude Amplitude = magnitude

사인 그래프 x축: 시간, y축: 값 → x축: frequency(Hz), y축: amplitude → spectrum

모든 신호는 조금씩 다른 sine wave의 합으로 표현할 수 있다.

*Hz는 1초당 진동수를 의미

Simple (simplex tone) ←→complex tone

하나의 sine wave 다 합친 모습

사인 그래프들의 합 그래프는 가장 낮은 주파수(Hz)의 패턴과 거의 일치한다 = pitch = 1초에 성대가 떠는 수

→speech의 source는 sine wave들의 합이다.

→가장 주가 되는 sign wave는 amplitude가 높다.

→배를 이루며 나타나는데, 이를 harmonics라고 한다.

F0: lowest pure tone, Fundamental frequency, 가장 저주파로부터 산맥이 있음 → 첫번째 산맥: formant 1(F1)….

F1 = F0*2, F2 = F0*3, F3 = F0*4…….

여성: 첫 시작이 남자보다 높고, 더 듬성듬성하게 그려진다.

남성: 첫 시작이 여자보다 낮고, 더 촘촘하게 그려진다. →배음의 숫자는 남자가 더 크다.

Source: 성대: harmonics가 gradually decrease

Vocal tract로 소리가 filter: amplitude의 패턴이 깨진다.

Praat VowelEditor

Vowel space: F1과 F2가 모음을 결정

F1: 모음의 높낮이 F2: back & front

모든 언어에는 단어와 문법이 있음.

단어: 의미와 정보를 담는다 → 정보를 담는 그릇

컴퓨터 언어의 단어: 변수 → 숫자 or 글자

컴퓨터 언어의 문법 1: 변수에 정보를 assign하는 것

컴퓨터 언어의 문법 2: if로 정보를 conditioning 한다.

컴퓨터 언어의 문법 3: for를 이용, 여러 번 반복한다.

컴퓨터 언어의 문법 4: 함수로 packaging하는 것

왼쪽: variable, 오른쪽: 정보

function: 함수(입력) → ex) print(a)

셀을 선택 → b: 셀 아래에 만들기 a: 셀 위에 만들기 x:셀 삭제

문자를 변수에 지정할 때는 ‘ ‘ 혹은 “ “를 사용한다.

shift & enter: 실행 단축키

마지막에 변수 명을 하나만 치면 변수 값을 print 해준다.

줄을 바꾸기 싫으면 ;으로 대신할 수 있다. ,는 안됨.

변수 안에 정보를 한꺼번에 넣는 list를 작성할 때는 [ ]를 쓴다. *()도 가능 → list가 아닌 tuple이 됨, 보안 up

type(): 변수의 속성을 보여주는 함수

- list: 정보의 나열 list에서는 숫자와 문자를 동시 표기 가능, list 안에 list도 가능
- int: 정수
- float: 실수
- str: 문자
- dict: 사전 { } 사용

string과 list는 유사하다.

내부적인 정보: 대가로 사용, 대가로 안은 index

리스트의 partial information을 가져올 때는 대가로를 사용한다.

Ex) a = (1, 2)
b = (3, 4)
c = a[0] + b[0]

dict의 정보를 access 할 때는 앞부분을 index로 활용한다.

index의 카운팅: a b c d e f
0 1 2 3 4 5
-6 -5 -4 -3 -2 - 1

index 범위 지정: : 를 사용한다. → 이는 str, list 모두에서 적용된다.

a:b = a에서 b 직전까지 가져온다.

a: = a에서 끝까지 가져온다.

:a = 처음부터 a전까지 가져온다.

: = 전부 가져온다.

len 함수 = length의 줄임, 정보의 개수를 알려주는 함수이다. len(a)

.upper = str를 대문자로 바꾸어 놓는다. a.upper()

*NLP(Natural Language Processing)

.find = 띄어쓰기 포함 내용이 몇 번째에서 시작하는지 알려준다. a.find('b') = a 변수에서 b가 몇번째에 등장?

.rindex= 왼쪽부터 카운트하여 가장 마지막에 나타나는 곳을 알려준다. a.rindex('b') = a에서 b가 마지막 언제?

.strip: a copy of s without leading or trailing whitespace : 여백을 없애줌 a = a.strip

.split : 긴 str를 delimiter를 활용, list로 잘라주는 함수 c = a.split('delimiter')

.join : 나눈 리스트를 다시 str로 합치는 함수 a = 'delimiter'.join(c)

.replace: 내용 하나를 다른 내용으로 바꾼다. a.replace('b', 'c') = 모든 b를 c로 바꾼다.

노트를 남기기 위해서는 앞에 #을 붙인다. 혹은 줄 하나를 markdown으로 지정하면 더 편하다.

markdown일 때 앞에 #을 붙이면 글씨를 크게 할 수 있다.

for loop

list에 있는 것들을 하나씩 일괄 프린트할 때: **for** i **in** a: → in 뒤에 있는 것을 i로 받아서 아래를 하라.

print(i)

in 뒤에 range를 사용할 수 있다. range는 index를 만들어 준다.

0에서 시작한다. ex) range(4): 0, 1, 2, 3

a[i]: a의 i번째가 된다. → **for** i **in** range(4)

len(a)를 사용하면 편하게 a 길이 지정이 가능하다. → **for** i **in** range(len(a))

enumerate 함수 → 번호를 추가로 매겨주는 함수이다.

for i, s in enumerate(a):

i 에는 번호, s에는 그 값이 들어간다.

어떤 특정 format으로 두 리스트를 출력하고 싶을 때, 입력값을 변수에 받아온 후,

" " 안에 format을 지정, 변수 부분을 { }로 지정하고, 변수 각각에 무엇이 들어갈지 .format으로 지정해준다.

"{variable 1}: {variable 2}".format(variable 1, variable 2)

zip을 사용할 때, a와 b의 length가 같아야 한다.

if 조건문

if 뒤 조건이 맞을 때 : 아래의 명령을 실행하라.

if a == b: a가 b라면. ==: 변수 지정이 아닌 진짜 =의 의미가 된다 .

if a != b: a가 b가 아니라면

if a >= b: a가 b보다 크거나 같다면.

else 조건문 if 조건문의 조건 외의 것이 나왔을 때 어떤 값을 실행할지 설정한다. → else:

range 안 숫자 2개를 넣을 시 ex) range(1, 3) → 1, 2의 값이 된다. ex2) range(3,5)→ 3, 4

string과 list는 유사하다.

n[:] = n

harmonics는 f0부터 모두

vocal tract는 harmonics의 위치는 변화시키지 않고, 그저 amplitude만 변화시킨다.

모음은 constriction location을 따로 정의하지 않는다.

amplitude 단위는 알 필요가 없다.

amplitude는 pressure → 목소리 톤에서 결정된다.

행렬

소리, 영상, 텍스트, 숫자 data 등은 숫자의 열로 표현이 되어야 함

숫자의 열: 벡터

흑백의 그림을 명도에 맞추어 숫자열로 표현: 직사각형의 형태로 숫자가 배열됨 : 행렬

숫자를 펼쳐서 놓음(한줄 한줄 길게 늘어놓음). → 벡터라이즈 한다.

흑백: 직사각형 1장 : 2차원

컬러: 직사각형 3장(RGB) : 3차원

동영상: 직사각형 3장이 시간별로 다르게 구성됨 : 4차원(시간 추가)

소리: wave form의 요소 하나하나가 값을 가짐, 그 값들을 숫자 값으로 표현 가능.

텍스트: ex) 5만개의 단어가 담긴 사전의 첫번째 단어: 10000000....000(수가 총 5만개)

두번째 단어: 01000000...-0000(수가 총 5만개)

NumPy

library를 import

import library as Variablename

library: 쓸 만한 함수들을 모아 놓은 창고

한 library 안에도 library가 있을 수 있다.

예를 들어, numpy 안에 A, B, C / A안에 D함수가 있을 때, D함수는 numpy.A.D로 나타낸다.

D 안에 함수 f가 있을 때, numpy.A.D.f가 된다.

다른 방법으로는, from numpy import A.D 라고 쓸 수 있다.

numpy: 리스트에 숫자가 들어갈 때 리스트들 간의 연산을 가능하게 해준다.

import numpy

numpy.array : 계산할 수 있는 행렬/벡터로 리스트를 변화시켜주는 것 →

variable = numpy.array(listname)

import numpy as np → np는 줄여 쓰는 이름

Variable = np.array([[list1] , [list2]]) → 2차원으로 만들어 줌

Variable.shape = (행, 열) 혹은 차원에 따라 요소가 늘어남.

matplotlib.pyplot → plotting을 할 때 사용하는 library, pyplot은 matplotlib의 하위 library 이다.

import matplotlib.pyplot as plt = from matplotlib import pyplot as plt

np.empty([행, 열], dtype='type')→ ex([2,3]) → 2행 3열

empty: numpy 안의 함수, 아무 수로나 행열을 만듬.

dtype = 행열을 채울 값의 속성을 지정해준다.

np.zeros([행, 열])

zeros: numpy 안의 함수, 0으로 이루어진 행열을 만듬.

np.ones([행, 열]): 1로 이루어진 행열을 만듬.

float64: 숫자가 64비트가 될 때까지 표시됨, 정확도 up, 데이터 up, 계산 시간 down

np.arange(a) 0~a-1까지 나열

np.arange(a, b) → a ~ b-1까지 나열

np.arange(a, b, c) → a~ b-1까지를 c간격으로. 나열

np.linspace(a, b, c) → a~b 까지를 c개, 같은 간격(space)으로 나눔.

linspace = linear space의 준말.

Variable.ndim → variable의 차원을 알려줌.

Variable.dtype → variable의 타입을 알려줌.

variable.astype(np.typename) → variable을 typename 타입으로 바꿔준다.

np.zeros_like(variable) → variable을 0으로 바꿔준다.

= variable * 0 으로도 가능.

Variable = np.random.normal(mean, standard deviation, data number)

np 안의 random 안의 normal

normal → normal distribution(정규분포)를 만들어준다.

plt.hist(Variable, bins = a)

plt = matplotlib.pyplot

.hist → 히스토그램을 만들어준다.

bins → 히스토그램 기둥 수

y값은 무조건 정수 값이 나와야 한다.

값들을 다 합하면 100개가 나온다.

plt.show

Variable2 = Variable1.reshape(a, b, c)

shape(a, b, c)의 element의 수는 a*b*c이다.

reshape → shape을 바꾼다. 그러나 element의 개수를 바꾸어서는 안된다. ex) abc → bca

-1: 값을 알아서 채우고 싶을 때 -1을 적는다 ex) bca → -1ca

np.allclose(variable1, variable2) → 두개의 행렬이 똑같은지 확인해줌, 맞으면 true 출력

a = np.random.randint(a, b [행, 열]) → a~b-1사이의 랜덤값으로

b = np.random.random([행, 열]) → 아예 랜덤으로!

np.saves("variable", a, b) → 파일 저장

del a, b → 없애기

npzfiles = np.load("variable.npz") → 로드하기

npzfiles.files[로드된 파일 중 하나] → 값 보여주기

sum을 쓰는 두 가지 방법 : variable.sum(), np.sum(variable)

variable.sum()을 쓸 수 있는 이유는 numpy안의 함수에 해당되기 때문.

Sound

1초간 무한대로 담을 수 없기 때문에 얼마나 빽빽하게 값을 담을 수 있는지

pure tone: 사인 코사인 wave → sinusoidal(사인, 코사인처럼 생긴) → phasor 1

phasor: sinusoidal을 만들어 내는 것

sin(radian값(세타로 표현))

degrees: 0도 180 360 720

radians: 0 pi 2pi 4pi

0~100pi까지 반복되면 sin, cos wave가 50번 반복된다

cos 0 = 1 cos 1/2 = 0 cos 1 = -1 cos 3/2 = 0 cos2 = 1

sin 0 = 0 sin 1/2 = 1 sin 1= 0 sin 3/2 = -1 sin 2 = 0

오일러 공식 = e의 세타*i 제곱 = cos(세타)+sin(세타)i → phasor 2

e = 2.71... 의 자연상수

세타=0 → 값 =1

세타=pi/2 → 값 = i

세타=pi → 값 = -1

세타=3/2pi → 값 = -i

세타=2pi → 값 = 1

i = imaginary의 약자, real의 반대말, 즉 허수(제곱해서 -1이 되는 값)

모든 수를 포함하는 수의 집합: 복소수

복소수: a + bi 로 정의되는 수

복소수를 plot하는 방법: x축을 a, y축을 b로 놓고 복소수 식에 대입되는 값을 표시한다.

1 = (1, 0)

i = (0, 1)

-1 = (-1, 0)

-i = (0, -1) → 이런 값들은 모두 벡터라고 할 수 있다.

이를 사분면에 표시하면 원과 같은 모양이 나오게 된다(반시계 방향).

그 각도를 세타값이라고 할 수 있다.

projection flashlight: 특정 방향에서 관측하는 것을 의미

a값에만 집중하면 가로로 움직이는 모습을 볼 수 있다(아래, 위에서 관측).

b값에만 집중하면 세로로 움직이는 모습을 볼 수 있다(양 옆에서 관측).

공통적으로 radian 각도값이 입력값,

pure tone에 넣는 입력값: radian, frequency(1초에 몇 번 움직이는가)

sin, cos에는 시간의 개념이 들어가 있지 않음.

from matplotlib import pyplot as plt = import matplotlib.pyplot as plt

matplotlib: plotting 할 때 사용

pyplot: sublibrary of matplotlib

from mpl_toolkits.mplot3d import Axes 3D → 3D 만드는 곳에 사용.

amp = range

ex amp = 1 → range: -1.0~1.0

amp = 2 → range: -2.0~2.0

amp = amplitude (진폭)

amplitude = complex phasor의 반지름

sr = a

sr = sampling rate = 1초에 몇 개의 숫자로 : 얼마나 고해상도로

dur

duration

freq =

frequency, wave의 반복 정도 : 1초 동안 왔다갔다 하는 정도

*sr는 1초당 점이 몇 개 있는지

*freq는 1초당 wave가 몇 번 왔다갔다 하는 지

generate time

t 0.001 0.002 0.003 → 이렇게 하면 끝이 없다.

1초 동안 10000 개의 숫자(sr)를 쓰려면 0.0001로 놓아야 한다.

t = np.arange(1, sr* dur+1)/sr → sampling rate 와 duration을 사용한다.

t = np.arange(1, sr*+1) → 1~sr까지의 수가 만들어진다. (예시에서는 10000개

여기서 duration이 1/2초이기 때문에,

t = np.arange(1, sr*dur+1) → 5000까지만 만들어진다.

그것을 다시 sr값으로 나눠주면 5000개가 다시 10000개의 숫자로 나누어진다.

time이 1초일 때, time tick은 sr과 일치한다.

따라서 sr에 duration을 곱하여 시간을 맞추고, 그걸 다시 sr로 나눠 개수를 맞춘다.

arange는 마지막 것을 제외시키므로 duration에는 1을 더한다.

phase = 세타값

time과의 연동으로 phase값으로 변환시킨다.

theta = t * 2*np.pi * freq

np.pi = pi

0~1초의 경우, π~2π 가 된다. 그러나 이 경우 한 바퀴만 돌게 된다.

그래서 freqency를 곱하여 몇바퀴를 돌지 정의한다.

t = 시간

2np.pi = 2pi를 돌 것이다.

freq = 몇 바퀴를

그 후 s = np.sin(theta) 로 대입한다. → time이 연동된 theta

np.sin( ) = sin( )

np.sin 앞에 진폭값(amp)을 넣어줄 수 있다.

➔ s = amp*np.sin(theta)

타임(t) 벡터의 사이즈와 theta의 벡터의 사이즈는 같다.

fig = plt.figure( )

figure를 지정

figure는 화면 전체를 말함.

ax = fig.add_subplot( )

subplot → 화면 분할

abc(행, 열, 번호)

ex) 221 → 2행 2열의 화면 중 1번째

ax.plot(a, b, c) 함수를 사용하여 plot을 함. 이 경우에는 a = time, b = sin의 결과값, c는 없어서 그냥 ‘.’

ax.set_xlabel( )

ax.set_ylabel( )로 각각 x축, y축의 이름을 지정한다.

c = np.exp(theta*1j) : 오일러 함수 식이다.

np.exp = 자연상수 e

다 상수이고, theta값만 받는 함수가 된다.

c = complex phasor

똑같이 amp를 곱해줄 수 있다

➔ c = amp*np.exp(theta*1j)

2d: ax.plot(2개의 숫자가 들어간다.)

3d: ax.plot(3개의 숫자가 들어간다.)

c.real, c.imag로 복소수 c를 실수와 허수 부분으로 나눈다.

3d그래프를 어떻게 보느냐에 따라 sin, cos등의 모양을 확인할 수 있다.

ipd.Audio(s, rate=sr) → 소리 재생하는 함수

s = sin 최종값

sr = sampling rate 정의

Generate Pulse Train

*sampling rate가 100hz일 때, 1초에 100개가 표현 가능

100개의 숫자로 1hz의 frequency를 표현 가능? → 가능.

2hz → 가능

10000hz → 불가능

➔ sampling rate이 충분해야 일정 주파수를 표현할 수 있다.

➔ 주어진 숫자의 개수, 즉 sr의 반만큼의 주파수만 표현 가능

➔ sr/2 만큼의 frequency까지만 표현 가능하다.

➔ ex) sr = 10 이면 5hz의 frequency까지 표현 가능.

⇨ Nyquist frequency: 특정 개수의 sr로 표현할 수 있는 최대 frequency, 즉 sr/2

➔ cd 음질이 44100, 즉 Nyquist frequency가 22050hz, 사람이 평균적으로 들을 수 있는 최대 frequency가 20000hz, 즉 22050hz 정도로 설정하면 적절한 음질이다.

그러나 그 이상의 고주파는 표현할 수 없을 수 있다.

20000hz 이상의 음은 초음파라고 한다.

➔ 예전 유선 전화기의 sr이 8000, Nyquist frequency가 4000 → 사람 목소리가 헷갈릴 수 있음. 요즘 전화기들은 16000 sr, Nyquist 8000hz까지 가능

F0 (가장 낮은 frequency)를 설정한다.

Fend(마지막 frequency) = int(sr/2)

Nyquist Frequency

int = 반올림 처리

첫 s값을 정해주어야 함.

s = np.zeros(len(t)) → 가장 초기의 s값 = 0

타임 벡터의 개수만큼 0을 만들어준다.

sin과 cos는 pi/2 차이가 남.

sin과 cos의 각도(phase) 차이는 소리 차이와는 관련이 없음.

인간은 phase의 차이를 인식하지 못한다.

함수 만들기

def 이름(입력1, 입력2):

    return 출력

decreasing을 하면 소리가 부드러워진다.

<u>선형대수</u>

데이터 → 기계 → 데이터

벡터 → 함수 → 벡터

음성 → 텍스트 ➔ 음성 인식

텍스트 → 음성 ➔ 음성 합성

언어 1 → 언어 2 ➔ 번역

선형대수

5 3 0 1 X -1 0 2 = -3 6 23

0 1 3

3 -5 7

2 3 4

a. -1*5 + 0*3 + 3*0 + 2+1

b. 0*5 + 1*3 + -5*0 + 3*1

c. 2*5 + 3*3 + 7*0 + 4*1

벡터로 된 음성, 텍스트 등등을 다시 다른 형태로 변환하는 과정. → 선형대수

행렬로 하는 보든 것: 선형대수

Linear algebra

Matrices

행렬의 크기(차원) → dimension

a_{11} ..... a_{1n}

.

.

a_{m1} .....a_{mn} → m by n 행렬, 혹은 m x n 행렬

a sequence of numbers

100차원에 벡터가 있더라도 점 자체는 하나이다. 그저 차원이 늘어났을 뿐이다.

vector multiplication

0.5 X 6

8

0.5 X 6

-2

4

vector addition

3 + 2 = 5

4 1 5

6 + -1 = 5

-2 0

4 2

vector spaces: 무수하게 많은 벡터들이 만드는 공간

requirements become a vector space.

linear combinations still stay in the space.

Linear combinations

c*v + d*w

c, d = scalars

v, w = vectors

나누기, 지수 등이 들어가면 linear combination이 아니다.

vector space: n차원의 모든 공간을 차지해야 vector space 이다.

R^n space → consists of all vectors with n components

1 차원: 선

2 차원: 면

3 차원 : 공간 ....

column space

A = 2 -1 → column vector: 2개[(2, 1), (-1, 3)], 두 점을 가지고 linear combination

1 3 linear combination을 무한 번 하게 되면 모든 면을 채우게 된다.

이 공간을 column space라고 한다.

column space는 원래의 차원 이상의 공간을 차지할 수 없다.

원점과 두 column vector를 확장시켜서 면을 덮는 것을 spanning이라고 한다.

이 또한 column space이다.

두 column vector는 같은 선상에 있지 않음 → independent

원점에서 그은 선을 기준

A = 2 -1 → (2, 1), (-1, -0.5)가 같은 선상에 있음 → dependant

1 -0.5 이 벡터는 linear combination을 무한 번 해도 같은 선안에만 있게 된다.

이 경우 column space는 1차원에 불과하다.

dimension (whole space) = n rows (열의 수)/n columns(행의 수)

dimension (column space) = n of independent columns/n of independent rows

A = 1 3 -2 → whole space: R^3

2 3 0 → column space: R^3

4 1 5

A = 1 1 2 → whole space: R^3

2 1 3 → column space: P P (Plane) : 2차원 → (1, 2, 4), (1, 1, 1)을 더하면 (2, 3, 5)

4 1 5

A = 1 2 4 → whole space: R^3

2 4 8 → column space: L

4 8 16

orthogonal(?)

A = 1 3 → whole space: R^3

2 3 → column vector: P

4 1

모든 행렬은 그것의 transpose가 있음 (역행렬?) → A^T = 1 2 4 → whole space: R^2

3 3 1 → column space: R^2

이 경우 column vector는 3개지만 차원은 2차원이다.

transpose와 원본의 column space는 바뀌지 않는다.

four spaces in a matrix

an m x n matrix has two whole spaces R^m and R^n → R^m = column space, R^n = row space

row의 관점에서도 위와 같은 space가 나옴

1 2 → (1, 2), (-1, 0), (3, 5)의 점 → 2차원의 공간에 생김 → whole row space = 2

-1 0 spanning: 2차원 → row space = 2

3 5 null space = 0

column space는 항상 row space와 일치함

independent: 서로 linear combination으로 만들어질 수 없는 것

n차원에서 independent 할 수 있는 점은 n개밖에 없다.

2개를 rank라고 함.

1 2 whole column space: R^3 column space: R^1 null space = R^2

2 4 whole row space: R^2 row space: R^1 null space = R^1

3 6

independent 한 점들이 spanning 된 모든 공간을 whole space에서 뺀 공간이 null space이다.

null space의 수학적 정의: 어떤 행렬에 무엇을 곱하든 모든 결과값이 0이 되도록 하는 행렬

$$\begin{matrix}1 & 2\\ 2 & 4\\ 3 & 6\end{matrix} \times \begin{matrix}x_1\\ x_2\end{matrix} = \begin{matrix}0\\ 0\\ 0\end{matrix}$$

column space에 나오는 null space는 left null space라고 부른다. – A에 x가 왼쪽에 붙기 때문

row space의 null space는 (right) null space라고 하는데, right는 생략 가능하다.

m x n

Linear transformation → 차원, 숫자를 곱하고 더해서 바꾸기 때문에 붙은 이름이다.

Ax = b

→ 대문자: 행렬, 소문자: 벡터

x가 입력 벡터, b가 출력 벡터

x가 행렬 A에 곱해져서 벡터 b 출력

x와 b의 차원은 같을 필요가 없다.

transformation matrix = A

a x b 행렬과 c x d 행렬이 곱해지려면 b = c여야 한다.

a x b 행렬과 c x d 행렬을 곱하면 a x d 행렬이 된다.

1. place x on original grid
2. transform x onto A grid
3. read transformed x on original grid (=b)

$$\begin{matrix}0.9 & -0.4\\ 0.4 & 0.9\end{matrix} \times \begin{matrix}1\\ 1\end{matrix} = \begin{matrix}0.5\\ 1.3\end{matrix}$$

transformation matrix * 입력 = 출력

1 0 → basis vector – (1, 0) , (0, 1)

0 1

detransformation

inverse matrix

$A^{-1}b = x$ → detransformation

두 점이 dependant 일 때는, 즉 A grid가 선일 때는 not invertible 하다.

determinant: 행렬식이 0이 될 때 역함수가 없다.

A grid 한 면의 면적이 determinant와 같다. 그것이 0이 된다는 것은 역함수가 없다는 뜻이다.

eigenvector : 고유 벡터 → unique

어떤 행렬의 eigenvector은 무엇이냐.

Av = b 일 때, v 와 b가 같은 일직선상에 있게 하는 v가 eigenvector이다.

- → 원점과 입력과 출력이 평행하게 하는 값
- → Av = 람다v 람다는 상수
- → v = eigenvector, 람다 = eigenvalue

벡터 : 방향

given 행렬의 eigenvector의 총집합 = 어느 방향 전체 = eigenvector space

2 x 2 행렬에서 eigenvector는 2개 있다. → 2 x 2를 다시 2 x 2로 만들었다고도 볼 수 있음

eigenvalue: eigenvector의 출력값의 증가율, eigenvalue의 수는 eigenvector의 수이다.

상관관계: 어떤 다른 것들끼리 같이 가는 느낌

상관관계 수치: -1<=r<=1

상관관계가 가장 낮은 것은 0, 음의 상관관계: 반비례, 양의 상관관계: 비례

-1, 1 : 완전히 한 선상에 있으면 -1, 또는 1이 된다. (기울기는 상관이 없다.)

0 : 점들이 완벽하게 원을 이루는 경우, 0이 된다.

85차원에서 1차원은 1사람을 나타냄.

원점과 두 점을 남기면 삼각형 모양이 된다. 이때, 점 A, B, C가 있을 때,

각도 AOB, 각도 AOC가 있다고 하자

cos 세타 = r (correlation 값), 정확히는 cosine similarity 하고 함.

두개의 벡터를 주고 cosine similarity를 구하라는 문제는 무조건 냄. → 기말

cos AOB = A와 B의 상관관계

cos AOC = A와 C의 상관관계

cos 90도 = 0

cos 0도 = 1

inner product의 cos세타, 즉 r은 무조건 0이 나올 수 밖에 없다.

➔ 각도가 클수록 상관이 없다. (0도 ~ 90도)

inner product(dot) : 두 벡터가 있을 때, 원점이 있고 a, b라는 product가 있을 때,

두 벡터가 있을 때, 각각 곱하고 더한 값

ex) A = (1, 2, 3), B = (4, 5, 6) ➔ 1*4+2*5+3*6 = 4+ 10 + 18 = 32 = inner product

|A| X cos 세타 X |B|를 곱한 값

길이를 구하는 법: A = (a, b, c)일 때, 원점에서 A의 길이는 루트($a^2+b^2+c^2$) 가 된다.

한번 더 inner product 예시

a = [1, 2, 3]

b = [2, 4, 7]

a x b =?

1. $a \times b^T$ = (1, 2, 3) X (2 = 31 → 1X1 = inner product

   4

   7)

2. $a^T \times b$ = (1 X (2, 4, 7) = 3X3 행렬 → outer product

   2

   3)

inner product → 무조건 1X1 행렬이 나옴.

이렇게 원점O, A[1, 2, 3], B[2, 4, 7] 세 점이 나오는데,

OA에서 OB로 수직선을 그릴 수 있음.

만나는 점을 c라고 했을 때, inner product는 OC의 길이 곱하기 OB의 길이가 된다.

이는 |a| X cos세타 X |b| 의 값도 같다.

cos 세타에서 각도는 어떻게 계산?(각 AOB)

a * b/|a|*|b| = cosAOB = inner product / 루트($1^2 + 2^2 + 3^2$) * 루트($2^2 + 4^2 + 7^2$)

여러 헤르츠의 sine wave가 있다고 가정하자.

각각의 사인 웨이브를 벡터의 사이즈가 똑같도록 100개의 백터값으로 표시한다고 하자.

→ inner product를 구하는 것이 가능하게 된다. → 하나의 숫자가 나올 것임.

100hz, 200hz, 10000hz 등등으로 같은 간격의 벡터로 만든 후, inner product를 만든다.

hz의 개수만큼 inner product가 만들어지게 됨. 성분이 많을수록 스펙트로그램에서 진해지고, 적을수록 옅어지게 됨. spectrogram에서는 100hz가 가장 아래에 있음.

숫자 100개짜리 벡터가 가장 단순한 웨이브 a라고 가정

거의 비슷한 b가 있고, 두배 빠른 c가 있다고 가정

a * b = a가 올라갈 때 b도 같이 올라가고, vise versa → correlation이 상당히 큼.

a * c = a와 c가 같이 가지 않고, 불일치가 큼 → correlation이 작음

axb > axc

복잡한 웨이브에서 inner product들을 각각 구했을 때 어떤 성분이 많은지를 확인할 수 있다.

➔ 복잡한 웨이브와 세부 성분들의 inner product를 구함.

A = 0, 1, 0, -1, 0, 1, 0, -1

B = 0, 1, 0, -1, 0, 1, 0, -1

A * B = 0+1+0+1+0+1+0+1 = 4

일치하지 않을 경우 값이 더 작게 나오게 된다.

그러나 이 방법에는 맹점이 하나 있음

A = 0, 1, 0, -1, 0, 1, 0, -1 0 → sine wave

B = 1, 0, -1, 0, 1, 0, -1, 0, 1 → cosine wave

➔ 서로 90도 차이

➔ inner product = 0+0+0+0+0+0+0+0+0=0

➔ 주파수가 같고 속도도 같으나 조금 이동하였기 때문에 제대로 반영하지 못함.

➔ 9차원 상에서의 a와 b 벡터의 90도와 웨이브 상의 90도가 같아져 버림 (민감도 높)

결론: phase에 너무 민감하게 작용하지 않도록 해야 함. → sine cosine phasor를 사용하기 힘듦.

phase로 sine cosine이 아닌, complex phasor를 쓰게 됨.

complex value 또한 같은 수의 벡터로 만들어서 inner product를 구하는 것이 가능함.

complex $e^{세타\ i}$

➔ vector이긴 하지만, complex number(복소수)가 나옴.

➔ 즉 inner product가 complex number(복소수)가 나옴.

➔ complex number는 plotting이 불가능함.

➔ 따라서 a + bi 에 절댓값을 씌운다 → |a + bi| → 실수가 된다.

complex number a + bi를 절댓값으로 씌우는 법

x축이 실수, y축이 허수인 그래프에 (a, b)를 나타내었을 때, (0, 0)과 (a, b)의 길이가 절댓값이다.

from scipy.io import wavfile → 외부에서 가져옴.

z = np.exp(omega*1j)**(np.arange(0,nSamp)) → complex phasor를 만드는 줄

exp(omega*1j) = e^{wi} omega = w, 1j = i

exp(omega*1j)**(np.arange(0,nSamp)) = $(e^{wi})^{[0,.....100]}$

w = 2pi * 0 / 100

[2pi * 0 / 100 [0...100]] i = [2pi * 0 / [0...1]] i

2pi * 0 [0……1] = 0

2pi * 1 [0……1] = [0……2pi] = 세타 → 샘플 개수 100개로 한바퀴를 돈 것임.

2pi * 2 [0……1] = [0……4pi] = 2세타 → 샘플 개수 100개로 두바퀴를 돈 것임.

……

2pi * n [0……1] = [0……2npi]

→ 이 경우에는 n이 100이라 100바퀴

amp.append(np.abs(np.dot(s,z)))

s를 기준으로 여러 z들과의 inner product를 구함.

np.dot = inner product 만들기

np.abs = 절댓값만들기

amp = [ ]; → amp.append

→ amp에 루프의 크기만큼 값을 회수

→ 회수한 후 amp의 길이는? 아마 len(amp)는 샘플값일 것이다.

→ amp에 허수가 들어있다? → 거짓. 없다.

fig = plt.figure()

ax = fig.add_subplot(111)

freq = np.arange(1,nFFT+1)*sr/nFFT;

[1…..100] *10000 / 100

= [100……..10000] …..

ax.plot(freq, amp) → (x축, y축)

ax.set_xlabel('frequency (Hz)')

ax.set_ylabel('amplitude')

➔ 결과 그래프는 대칭이 되어야 한다.

➔ 결과 그래프의 바의 개수 = 샘플 개수

➔ 따라서 50까지만 의미가 있다. → 절반까지만 의미가 있음. 절반까지만 주파수를 표현

pi/2 pi 2pi

각도 90도 = orthogonal

cos(seta) = 0

서로 orthogonal한 관계일 때 둘의 dot product는 0이 나오게 된다.

그래프를 주면서 찍었는데 완전히 라인이 되었다 = "r=1" =두 벡터의 각도가 0이 된다.

r = -1일 경우, 두 벡터는 180도를 이루게 된다.

벡터를 공간에 표현했을 때의 길이 = wave의 진폭

다시 복습

A

1 5 3

2 6 -1

2 x 3

row vector whole space = 3차원

row (1, 5, 3), (2, 6, -1) → 서로 independent, row vector space = 2차원

null space = 1차원 → 평면에 직각으로 만나는 선 하나가 null space임. 그러나 반드시 그 선 위에 있을 필요는 없다. 평면에 orthogonal하게 위치를 바꾼다면 출력에 영향을 미치지 않는다.

입력이 들어와도 출력(output)에 영향을 미치지 않는 값

수학적 해석 : Ax = 0이 되게 하는 값

Ax = b → 변화를 주어도 Ax = b 인 경우 → null space를 평행하게 이동시켰을 때의 값

기하적 해석: 평면에 orthogonal(수직으로 오고, 그 다음 거기서 평행할 때)하게 오는 값

실용적 해석: 입력을 해도 출력에 영향을 미치지 않는 공간

skilled action → 기교를 많이 부림.

살아가는 것 → null space를 늘리는 것

스펙트럼 한장한장이 쌓여서 스펙트로그램이 됨.

아래쪽이 저주파, 위쪽이 고주파

앞부분에서 조금 잘라서 스펙트럼을 만듬

계속 한장한장씩 만듬

win size = 자르기 간 간격

win step = 자르는 정도

```
magspec = np.abs(np.fft.rfft(frames, n=nfft))
plot_spectrogram(magspec)
```

진한 부분 = 1보다 큰 값

연한 부분 = 1보다 작은 값

조금 희미해짐.

```
powspec = 1/nfft * (magspec**2)
plot_spectrogram(powspec);
```

→ 제곱을 시키면, 진한 부분은 훨씬 커지고, 연한 부분은 훨씬 작아지게 됨.

```
logspec = 10 * np.log10(magspec)   #dB scale
plot_spectrogram(logspec);
```

→ 이후 log처리를 하면 다룰 수 있는 범위로 넣을 수 있음.

→ 고주파쪽도 더 clear하게 볼수 있게 됨.