

*Phonetics는 물리적으로 어떻게 소리가 나는지와 관련된 것이다. 소리가 어떻게 인지되는지에 관심을 두는 phonology와는 차이가 있다. 영어에는 44개의 다른 소리가 존재한다.

Articulatory phonetics(mouth), Acoustic Phonetics(through air), Auditory Phonetics(to ear)

*Articulation(조음)

vocal tract(nose, pharynx, larynx, ear로 구성)

upper vocal tract[lip, teeth, alveolar ridge, hard palate, softpalate(velum), uvula, pharynx wall로 구성]

lower vocal tract[lip, tip, blade, front/center/back tongue, root, epiglottis로 구성]

한국어는 음절이 반복되고, 영어에서는 stress가 반복되기 때문에 혀의 위치가 중요

폐 -> 성대(수문과 비슷한 역할); 떨림이 느껴진다는 것은 성대가 떨력인다는 것

여자 250번 정도, 남자는 100번 정도

1) articulatory process [lips, tongue body, tongue tip]

constriction location : where exactly -(앞뒤)

constriction degree how much exactly(상하)

2) phonation process[larynx]

voiced(진동을 느낄 수 있음) - 모든 모음과 일부 유성 자음

voiceless(진동x) - p, h, k, s, f

3) oro-nasal process[velum]

velum이 내려가면서 소리가 지나가는 통로가 열림

모든 모음과 비음(m,n,ng)을 제외한 자음도 velum이 열리면서 소리남

*constriction location : where exactly -(앞뒤)

lips(2개의 소리: 앞 - [ㅂ][ㅍ] / 뒤 - [ㄴ][ㄹ])

tongue body(2개의 소리 - 앞 : [ʃ],[y] [ear], [yarn] / 뒤 : 그)

tongue tip(4개의 소리 - [th(윗니)],..)

constrictor 관점에서는 같지만, 로케이션 관점에서는 다른 소리

*constriction degree how much exactly(상하): 조금 막을 건지, 많이 막을 건지, upper part를 치는 정도에 따라

stops: [p], [b], [t], [d], [k], [g]

fricatives 마찰음 [f], [v], [θ], [ð], [s], [z], [ʃ], [ʒ], [h]

approximants [r], [l], [w], [j]

vowels 모음의 정의는 막힘이 없는 것 -> upper part를 별로 안 칩

tongue body, tongue tip, lip, velum, larynx의 위치로 어떤 소리가 나는지 알 수 있다.

*praat: 소리의 duration, intensity(loudness), pitch를 확인 가능

빨간 띠: formant(F1, F2에 따라 모음 구별 가능)

praat에 나타나는 큰 파동의 횟수는 larynx의 떨림 횟수와 일치

헤르츠: 1초에 성대가 떨리는 횟수

2주차

Pratt

intensity(db), pitch(hz), Formants(hz) 분석 가능

HZ(= frequency의 단위) : 1초 당 반복되는 구간이 몇 번 생겼는지

여기서, 반복되는 구간은 결국 sine wave이며, Larynx의 떨림과 일치한다.

그리고 이 세상의 모든 signal은 sine wave로 이루어졌다.

사람의 목소리도 마찬가지다. 목에다 장치를 설치해, 그 소리를 녹음하면, sine wave가 나타

나지만, 마이크를 통해 사람의 말소리를 녹음하면 sine wave의 합(=complex sound)이 나오

즉 입모양에 따라 합쳐지는 sine wave의 종류가 달라진다.

sine wave는 frequency와 magnitude 두 가지 특성에 의해 구분된다.

합쳐진 sine wave는 complex tone이라 하는데, F0과 반복되는 주기가 일치한다. 그리고 각 sinewave들의 frequency는 f0의 배수이다.

voice source(목에서 직접 측정한 소리): 스펙트럼이 점진적으로 감소하는 모양

filtered by voice tract(마이크로 측정한 소리): 점진적으로 감소하는 모양이 유지되지 않음

하지만, 둘 다 f0 배수의 frequency를 가진다는 공통점이 있다.

소리를 표현하는 방법

1)스펙트로그램

x축: 시간, y축: frequency

*위쪽으로 갈수록 열어짐

*frequency와 amplitude는 반비례

2)스펙트럼

x축: frequency, y축: amplitude

*주파수 별로 어떤 성분이 있는지 확인 가능하다.

목소리가 정해지는 방법 = f0과 이에 대응되는 합

f0은 pitch와 일치

filter: 스펙트럼에 나타나는 mountain과 valley의 모양을 결정지음

입모양에 따라 비슷한 모양을 띠

f1은 formant와 일치

f1은 혀의 높낮이와 일치

f2는 혀의 front / back과 일치

따라서 일정한 주파수와 amplitude 수치에 따라 특정 모음과 자음의 소리를 구성할 수 있고 pure 톤들의 합으로 사람 목소리를 나타내는 것이 가능하다.

3주차

코딩이란?

자동화 - 계속 같은 것이 반복되기 때문에

언어 - 단어를 어떻게 결합하는지(단어와 문법)

단어 - 의미/정보를 담는 그릇

어떤 단어를 이용해서 어떻게 합칠 것인가

컴퓨터 언어에서 단어의 역할을 하는 것은 변수(variable), 변수에 담기는 것은 문자와 숫자가 전부

1. Variable assignment 변수에다 정보를 지정하는 것
2. if Conditioning 자동화, 기계화 - 조건문 문법 if
3. 자동화에 중요한 것은 여러 번 반복하는 것 for
4. 함수: 어떤 것을 입력했을 때 원하는 값이 나오는 것이 함수

1,2,3,4가 함수 안에 들어가는 것

Ex> 모든 사람이 함수다 : 먹고 싸는 거, 기름으로 자동차가 달리는 것

재사용과 반복을 위해 코딩을 한다.

= : 같다는 뜻이 아니라, 오른쪽에 있는 정보를 왼쪽에 있는 변수로 assign한다.

함수(입력)

각각의 셀이 따로 실행 가능

Variable은 정보를 override 함

한 변수에 여러 개를 정보를 넣고 싶을 때 - list[], tuple(), dictionary={'key':'value'} 표제어와 설명의 쌍

차이 - 튜플이 더 보안에 강하다(바꾸기 힘들다)

Type() : 변수가 어떤 특성을 가지고 있는지 알려줌

4주차

하나의 variable에 여러 정보를 담으면, 최신 정보로 업데이트 됨

어떤 variable 속에 정보가 들어 있을 때,

retrieve 정보를 가지고 온다.

변수 간의 덧셈, data access

a=[0번째, 1번째,...]

리스트에서 일부분의 정보를 가져올 때, 대괄호를 이용함

타입종류()어떤 변수의 type을 다른 타입으로 바꿔주는 함수

어떤 변수의 내부 정보를 가져오고 싶으면, 대괄호를 사용하여, 인덱스를 적으면 됨

dict의 access

pair set에서의 key 부분을 인덱스로 사용

string과 list는 정보의 접근 차원에서 비슷

```
In [47]: s='abcdef'
```

```
In [48]: print(s[0],s[5],s[-1],s[-6])
```

a f f a

왼쪽으로 가면 음수

print(s[1:3]) - 1에서부터 3까지 가져와라

print(s[1:]) - 첫 번째부터 끝까지

print(s[:3]) - 처음부터 2번째까지

리스트도 같은 방식

len() 함수 - 정보의 길이를 알려줌

s.upper() - 대문자로 변화

for i in a:

in 뒤에 있는 것을 하나씩 돌려서 I가 그 하나하나를 받고 어떤 일을 해라

if conditionint

function 입력이 들어가고 출력이 나오는 것들(이미 있는 것도 있고, 내가 만드는 것도 있음)

ipynb 확장자: 코드와 필기를 한번에

```
In [18]: a=['red', 'green', 'blue', 'purple']
b=[0.2, 0.3, 0.1, 0.4]

# enumerate도 함수명 - 번호 매기는 함수, 두개의 변수가 들어갈 수 있음
for i,s in enumerate(a):
    #i에다가는 번호(index 값), s에다가는 red 이런 애들
    print("{}:{}".format(s, b[i]*100))

red:20.0%
green:30.0%
blue:10.0%
purple:40.0%
```

```
In [19]: a=['red', 'green', 'blue', 'purple']
b=[0.2, 0.3, 0.1, 0.4]

# enumerate도 함수명 - 번호 매기는 함수, 두개의 변수가 들어갈 수 있음
for i,s in enumerate(b):
    #i에다가는 번호(index 값), s에다가는 red 이런 애들
    print("{}:{}".format(a[i], s*100))

red:20.0%
green:30.0%
blue:10.0%
purple:40.0%
```

```
In [21]: a=['red', 'green', 'blue', 'purple']
b=[0.2, 0.3, 0.1, 0.4]

# enumerate도 함수명 - 번호 매기는 함수, 두개의 변수가 들어갈 수 있음
for s,i in zip(a, b):
    #a와 b는 반드시 길이가 같은 리스트가 되어야 함, zip - 둘을 합친다는 뜻, 덕셔너리처럼 바뀌는 것
    #s에는 a의 값, i에는 b의 값이 들어가는 것, index 값은 들어가지 않음
    print("{}:{}".format(s, i*100))

red:20.0%
green:30.0%
blue:10.0%
purple:40.0%
```

```
In [28]: a = 3
#(크거나 같으면)
if a >= 0:
    print('yy')
else:
    print('nn')
```

yy

```
In [31]: for i in range(1,3):
        for j in range(3,5):
            if j >=4:
                print(i+j)
            #!!!인덴테이션!!!
            #계층적인 개념을 잘 알기!!
```

```
File "<tokenize>", line 4
    print(i+j)
    ^
```

11/3 - 5주차 컴퓨터가 처리하는 모든 정보들은 데이터화되어야지만, 가치 있는 정보들이 된다. 영상, 이미지 - 각 픽셀에 할당된 값으로 표현 소리 - 수치로 표현 가능 벡터, 행렬로 표현해야지 수학적인 처리가 가능하다. 그것을 가능하게 해 주는 것이 numpy이다. import numpy as np : numpy라는 라이브러리를 np라는 이름으로 가져와라 1. array() : 계산이 가능한 벡터 형태로 바꿔주는 함수(적은 메모리로 데이터를 빠르게 처리) <리스트와 다른 점> 모든 원소가 같은 자료형이다 / 원소의 개수를 바꿀 수 없다. 리스트의 원소들은 계산이 불가능하지만, 배열의 원소는 가능하다.

2. 2차원 배열인 행렬을 만들 수 있다. X = np.array([[1,2,3], [4,5,6]]) print(X) : array([1,2,3], [4,5,6]) / X.shape : (2,3)

3. numpy 속 subpackage를 불러오고 싶을 때 import numpy.A.D.F = from numpy import A, D import matplotlib.pyplot as plt = from matplotlib import pyplot as plt

4. 다차원 배열 생성 : np.empty/ones/zeros(만들고자 하는 배열을 튜플로 표현, 변수 종류)

5. np. arrange : 특정한 규칙에 따라 증가하는 수열을 만들 np.arrange(시작, 끝, 증가하는 수) 시작과 끝을 포함하지 않음

6. np. linspace(시작, 끝, 원소의 개수, dtype = ~) 시작 ~ 끝까지의 수들 중 n개만큼의 원소의 벡터를 만드는 데, 숫자들 사이의 거리가 같다.

7. 하나의 행렬에 대해서 속성을 알아낼 수 있는 다양한 함수가 존재한다. X = np.array([4,5,6], [1,2,3]) X.ndim #배열의 차원 / X.shape #행렬의 형태 / X.dtype #행렬 속 데이터 정보를 알려줌 X.astype(np.float64) #행렬 속 데이터 정보를 바꿔줌 / np.zeros_like(X) = X*0

8. 정규 분포 data = np.random.normal(표준편차, 평균, 데이터 수) #정규분포를 만들어 주는 함수 plt.hist(data, bins = 10) #위의 데이터로 10개 범위의 히스토그램을 만들어라 = 100개 데이터를 10개의 범위로 표현 plt.show() #그 히스토그램을 보여줘

9. 3차원의 행렬 X = np.ones([차원의 수, 가로줄의 수(행의 수), 세로줄의 수(열의 수)])

#차원의 개념 : 줄은 일차원 / 직사각형은 2차원 / 3차원은 그 직사각형이 두개인 것 Y = X.reshape(-1, 바꾸고 싶은 만큼의 가로의 수, 바꾸고 싶은 세로의 수) # Y의 원소의 수는 X의 원소의 수와 반드시 일치해야 한다. 즉 차원 * 행 * 열의 값이 같아야 한다. 이 계산에 맞게 알아서 계산해주는 것이 바로 -1

10. np.allclose(X.reshape(-1, 3, 2), Y) 두 개의 배열이 서로 호환 가능하면, True를 출력

11. np.random.randint(시작, 끝, [행, 열]) 시작을 포함하여 끝-1까지의 수 중, 균일 분포의 정수 난수(random number) np.random.random(받고 싶은 자료의 크기) ex> np.random.random(5) -> 1*5의 랜덤 함수/ np.random.random((2,2)) -> 2*2의 함수

12. np.save("test",a,b) -> 실제 파일에 다음의 변수를 저장 / who : 지금 available한 변수 np.load() : .npy 파일을 배열로 불러오기! / npzfiles.files : 현재 available한 파일 / npzfiles['이름'] -> 해당 배열

16. arr = np.random.random([5,2,3]) -> 5차원의 2개의 행 3개의 열 type(arr) : numpy.ndarray / len(arr): 5 / arr.shape: (5,2,3) / arr.ndim: 3 arr.size: 30 #원소의 개수 arr.dtype : float64 #원소의 데이터 타입

11/10

phaser : 짜인 코사인을 만들어 내는 것

0~360도[degree] = 0 ~ * 파이[radiant]

sin 함수와 코사인 함수 모두 패이저(sin(radiant 값) / cos(radiant 값- input, 세타 값))

<오일러 공식 - 패이저>

e: 2.71 / loge -> 자연로그 / 오일러 공식의 I 허수

수 전체를 포괄하는 것 -> 복소수 = a + bi

복소수를 표현하는 방법 - complex plate (a, b)로 표현

벡터 : 숫자 열 -> 복소수 형면에서의 좌표도 벡터라고 할 수 있음

“프로젝션”(그래프를 위에서 본다면 코사인과 유사하고, 옆에서 본다면 사인과 유사하다.)

radiant 값은 시간의 개념이 포함되어 있지 않지만 소리의 실체는 시간의 개념을 포함하기 때문에 radiant 값과 함께 시간의 개념이 필요:

<오일러 공식 외우기>

theta = t* 2*np.pi*fre

--> 1초에 2pi만큼 도는 게 한 바퀴 거기에다가 frequency를 곱하여 그 구간이 얼마나 반복 되는지 정하기!!

몇 초동안의 흐름이 있는지가 관심

ax.plot(t[0:1000], s[0:1000], ',')

점들의 개수는 몇 개인가 - 1000개

엑스 축 와이축의 개수가 같지 않으면, 실행이 되지 않는다.

오일러 공식의 패이저를 이용할 수도 있음

np.exp -> 숫자

lj -> I

c = np.exp(theta*lj)

짜인하고 같은 개념 모든 게 다 고정되어 있고 세타만 바뀌기 때문

c의 하나한의 값들은 a + bi의 형태임

-01은 십분의 일 (소수점의 위치와 관련된 수치)

fig = plt.figure()

ax = fig.add_subplot(111)

ax.plot(t[0:100], c.real[0:1000], c.imag[0:1000], ',')

ax.set_xlabel('time(s)') / ax.set_ylabel('real') / ax.set_zlabel('imag') / s =

np.sin(theta) / ax.plot(theta, s, '*')

c라는 이름으로 받은 부분 -> 복소수 부분 a + bi

x축은 real 값, y축은 c에세의 a값만 c.real, c.imag : 허수값만

projection

오일러 공식 중 cos에 해당하는 것 real num, sin에 해당하는 것 imaginary

real만 본다는 것의 의미(real의 y 값) -> 코사인 함수와 똑같음

ipd.Audio(s, rate = sr) / ipd.Audio(c.img = sr) / ipd.Audio(c.real = sr)

Phasor는 시간 영역에서 주파수 영역으로 계산하기 위한 도구이다.

우리가 잘 알고 있는 phasor에는 사인 함수와 코사인 함수가 있다.

이들은 모두 input으로 radian 값을 받으며 일정 주기로 반복된다. 0, $\pi/2$, π , 2π

여기서 더 심화한 페이지가 오일러 공식이다.

이는 $a + bi$ 의 형태로 표현되며, 모든 수를 표현할 수 있다. (a,b) 의 벡터값으로 좌표 평면에도 표현이 가능하다.

하지만 소리를 만들려면 이런 각도값 이외에도 시간이 필요하다. 따라서

1. 시간을 설정해야 한다.

시간을 만들 땐, sampling rate를 고려해야 하는데, sr은 1초에 얼마만큼의 소리 인자가 존재하느냐라는 것이다. sr이 10000이라고 한다면, 1초에 만개가 나올 수 있게 1초를 10000으로 잘라주는 것이 필요하다. `np.(1,sr)**여기 모르겠음!!`

2. 세타와 t를 연동시키기

`theta = t* 2*np.pi* frequency` - 한 번의 주기가 얼마만큼 돌 것인가...

: 세타는 결국 input 값이므로 얼마만큼의 시간과 얼마만큼의 각도값에 따라 sin 그래프가 그려지는 것을 보는 것이다.

3. `s = np. sin(theta)`

*여기서 amp를 곱해주므로써 진폭을 높일 수 있다. : `amp *np.sin(theta)`

이것을 plot할 수 있다.

오일러 공식의 경우, 하나의 값을 넣으면 a와 b 두 개의 값이 나온다. 이들에 대해 projection 할 수 있다.

generate pulse train - 실제 목소리 만들기가 가능

1. f0을 설정

2. sign wave를 누적시키면 된다.

그렇다면 f0을 얼마까지 누적시켜야 할까??그것은 sampling rate의 수치에 달려 있다.

우리가 표현할 수 있는 주파수는 최대한 sr의 절반이다.

스펙트럼 : 한 순간에 어떤 성분들이 있는지를 보여주는 것[즉, 한 점의 시간이다.]

스펙트로그램: 이것들을 시간 순으로 축 나열하는 것!

formant : 산맥에 따라 다른 소리가 남!!

1. 높낮이가 없는 그래프

2. gradually decreasing한 그래프

3. 산맥을 만들어주기

RG : 실제로 gradually decreasing하게 만들어 주는 값

Bandwidth: 그래프의 너비

데이터 → 행렬 → 데이터 (또는 통계숫자열이요) → 행렬은 일렬

행 = 행렬

데이터가 벡터로 표현되는 이유: 행렬 계산하려고

→ 인공지능의 기본 틀은, 입력 벡터를 출력 벡터로 변환시키는 것.

ex) 음성인식 (음성 → 텍스트), 음성합성 (텍스트 → 음성), 기계번역 (텍스트 → 텍스트), 안과고 (동의상태 → 수)

<행렬 계산>

$$\begin{array}{cccc} & & \times & \\ & & \swarrow & \\ \begin{array}{cccc} 5 & 3 & 0 & 1 \\ -1 & 0 & 3 & 2 \\ \text{"} & \text{"} & \text{"} & \text{"} \\ -5 & + & 0 & + & 0 & + & 2 & = & -3 \end{array} & \rightarrow & \begin{array}{ccc} \boxed{\begin{array}{c} -1 \\ 0 \\ 3 \\ 2 \end{array}} & \begin{array}{c} 0 \\ 1 \\ -5 \\ 3 \end{array} & \begin{array}{c} 2 \\ 3 \\ 7 \\ 4 \end{array} \\ & & & \rightarrow -3, 6, 23 \end{array}$$

$$\begin{array}{cccc} 0 & 1 & -5 & 3 \\ \text{"} & \text{"} & \text{"} & \text{"} \\ 0 & + & 3 & + & 0 & + & 3 & = & 6 \end{array}$$

$$\begin{array}{cccc} 2 & 3 & 7 & 4 \\ \text{"} & \text{"} & \text{"} & \text{"} \\ 10 & + & 9 & + & 0 & + & 4 & = & 23 \end{array}$$

기계학습은 결국

많은 데이터로부터

학습하며 특정 행렬을 얻는 것이다!

<선형대수>

행렬: 직사각형 형태, dimension (크기)를 표현 가능. 행 x 열, $m \times n$ 행렬

벡터: 일종의 행렬 - $[a_1, \dots, a_n]$ $\begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$ column vector : $1 \times n$ 행렬 (sequence of numbers)

ex) $\begin{bmatrix} 3 \\ 4 \end{bmatrix} \rightarrow \begin{array}{|c|} \hline 3 \\ \hline 4 \\ \hline \end{array} \rightarrow \begin{bmatrix} 3 \\ 4 \end{bmatrix}$; 차원이 늘어날 뿐, 표현할 수 있는 점만 개!

중첩 $0.5 \begin{bmatrix} 6 \\ 8 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \\ 4 \end{bmatrix}$

더하기 $\begin{bmatrix} 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$; 힘의 방향이 일치한다.

vector spaces : 무수하게 많은 벡터들이 만들어 내는 공간 (2차원 벡터 \leftrightarrow 2차원 평면에서 표현 가능)

: 무한하게 만들어낼 수 있음.

→ 일부분은 하나의 완성된 vector space라 할 수 없음.

즉, 2차원, 3차원, ..., (R^1, R^2, \dots, R^n) 의 모든 공간이 vector space 이

———— 1차원 ———— 2차원 ———— 3차원

column space

$A = \begin{bmatrix} 2 & -1 \\ 1 & 3 \end{bmatrix}$; $(2, 1)$, $(-1, 3)$ 을 가지고 linear combination (사칙연산) 을 무한대로 하면,

모든 2차원 공간이 채워짐. \Rightarrow column vector에 의한 column space

Q) column vector는 column space보다 차원이 높을 수 있다. (x) ... 차원을 넘겨 서지는 않음.

→ 이 자체가 평면을 나타낸다. (= 원점 + 정렬 벡터 > 삼각형)

whole space: 벡터 자체가 갖는 dimension. - - - - - R^2 → 두 점이 같은 선상에 있을 때! (dependent)

column space: 평면을 확장했을 때, 4 개까지는 dimension. - L (같은 선상에 있다면, 모든 공간을 채울 수 없음)

12/ 1 (11/25 - 11/30)

Eigen analysis(고유 벡터)

기하적 관점 : 원점, 입출력이 평행한 상황에서 eigen vector가 형성된다. 이때 eigen vector는 방향이라고 이해하는 것이 수월하다.

또한 eigen vector 상에서는, 입력값이 출력값으로 나타나는 비율이 일정하다.

즉 $Av = (\text{상수})v$ 이는 기존에 배웠던 linear combination의 정의와도 일치한다.

실용적 관점

어떤 벡터가 있다고 하면, 그 벡터는 여러 가지 수치가 합해진 값이다. 하지만 eigen vector는 그 벡터의 고유성을 보여준다. 즉, 주어진 벡터는 컬럼 벡터의 수치들이 일정하지 않게 분포된 값이지만, eigen vector는 그 수치들이 어떤 경향성을 가지고 있는지 보여주는 것이다.

벡터를 통해서 두 수치 사이의 상관 관계를 구할 수도 있다.

보통 통계에서 어떤 자료들이 반비례/비례 관계를 보이면 그 상관성을 $-1 < r < 1$ 의 수치로 표현한다. 이는 자료들이 어떤 선을 이루고 있으면 $-1, 1$ 과 가깝고 전혀 그렇지 않을 때 0의 값을 갖는다. 신기하게도 이것을 벡터 차원으로 표현할 수 있다. 예를들어 80명 학생의 국어와 영어 점수에 대한 경향성을 구한다고 할 때, 영어 점수 전체는 80차원의 column vector로써 한 점으로 표현될 수 있다. 국어도 마찬가지다. 이 두 벡터 사이에 형성되는 각의 코사인 값이 위에서 이야기했던 r 값과 일치한다. 따라서 벡터 사이의 각이 두 수치 사이의 유사성을 설명해 줄 수 있는 것이다.

그렇다면 이 theta 값은 어떻게 구할까?

inner product, dot product 방법을 활용할 수 있다.

국어 벡터 a 값, 영어 벡터 b 값, 그리고 원점이 있다고 하자.

(b 에 a 를 프로젝트한 길이) * 원점에서 b 까지의 길이 = inner product 이다.

여기서 b 에 a 를 프로젝트 한 길이는 $a * \cos\theta$ 값과 일치한다.

따라서 inner product(a 행렬 * b 행렬)와 oa 의 길이 ob 의 길이를 통해 세타값을 구할 수 있다.

이렇게 얼마나 일치하는지를 알려주는 정보인 theta 값을 통해 spectrogram을 실제로 만들어낼 수 있다.

1. 페이지로 특정 주파수 만들기

2. 이 주파수가 만들고자 하는 spectrogram과 얼마만큼 일치하는지의 수치를 통해서 특정 주파수가 얼마만큼의 비중을 차지하고 있는지 파악할 수 있다.

3. 프랏한다.

하지만 여기에서도 문제가 생긴다. 같은 주파수를 갖고 있지만 시간이라는 차이로 그 상관성이 0이 되어 버리는 경우가 있다. (코사인과 사인의 관계)

따라서 조금 더 복잡한 페이지를 이용해야 한다.