영어음성학 정리 2017131518 언어학과 이윤재

영어의 자모는 크게 voiced 와 voiceless로 구분된다. 이 둘은 성대의 떨림을 기준으로 한다.
성대가 떨리면 voiced, 그렇지 않으면 voiceless이다.
자음은 b, d, g, j, l, m, n, ng, r, th, v, w, y, z가 voiced consonants이고,
Ch, f, k, p, s, sh, t, th는 voiceless consonants이다.
모음은 모두 voiceless이다.
모음은 크게 두종류로 나뉜다. 단모음(monophthongs)와 복모음(diphthongs)가 있다.

Phonetics는 '말(speech)'를 연구하는 학문이다.
Phonetics는 다시 세 종류로 나뉘는데, Articulatory phonetics, Acoustic phonetics, Auditory phonetics가 그 예시이다.
Articulatory phonetics는 우리의 구강구조에서 어떻게 말을 생산하는 가에 대해 연구한다.
Acoustic phonetics는 어떻게 말이 전달되는지 파동의 개념으로 본다.
Auditory phonetics는 어떻게 말이 우리의 귀로 전달되어 인식하는 가를 배운다.

조음(Articulation)
말소리가 이동하는 통로: 이 - 귀(ear), 비 - 코(nose), 인 - 인두(pharynx), 후 - 후두(larynx)

Vocal tract(upper):
입술(lip): v,p와 같은 소리의 조음 장소이다.
치아(teeth): th와 같은 치음 소리의 조음 장소이다.
Alveolar ridge: 치아 바로 뒤 잇몸부분이다. d와 같은 치경음의 조음장소이다.
경구개(hard palate): 입천장 중에 앞부분으로, 상대적으로 딱딱한 부분이다.
연구개(soft palate): 입천장 중 뒷부분으로, 상대적으로 부드러운 부분이다. Velum이라 부르기도 한다.
목젖(uvula): 목젖이다.
인두벽(pharynx wall): 인두 부분을 말한다.

Vocal tract(lower):
아랫부분은 크게 3군데로 나뉜다. 입술, 혀, 후두개(epiglottis)
그중에 혀는 혀끝(tip), 혓날(blade), 앞, 중간, 뒤, 혀뿌리(root)가 있다.
epiglottis는 후두에 있는 후두개를 말한다.
Phonation process in larynx
폐에서 나오는 공기는 larynx에서 phonation process 거친다. 이때 성대가 매우 빠른 속도로 진동하여 소리를 만든다. 보통 소리의 높낮이와 진동수는 비례한다. 후두에서 진동이 있느냐 없느냐에 따라 유성음 무성음으로 나뉜다. 영어의 모든 모음은 유성음이다.
Oro-nasal process in velum
m, n, ng와 같이 흔히 비음이라 말해지는 것들은 코로 공기가 감을 통해서 이루어진다.
이때, velum이 주역할을 한다. Velum이 열리면 비음이 되고, 다른 음성일때는 닫혀 있다.

Control of constrictors
constriction이 어디에 이루어지냐에 따라 영어 자모음이 달라진다.

입술로 이루어지는 소리들, p와 같은 소리들이 대표적이다.
Tongue body가 palatal과 velar 어디에 있느냐에 따라 소리가 달라진다.
velar는 k, palatal의 경우 g가 대표적이다.
Tongue tip의 경우 dental부터 치아 뒷부분인 alveolar, 더 뒷부분인 palato-alveolar부분(sh), 그리고 그 부분을 지나가면서 발성하는 retroflex음이 있다.
한편 세기에 따라 달라지기도 한다.
완전히 막고 세게하면 stop, 조금 틈을 열면 fricative, 모음보다 조금 더 열면 approximants, 그 다음은 모음이다.
이러한 소리들은 조음 기관의 조합으로 이루어진다. 이러한 소리들로 단어가 구성된다. 이때 구성되는 하나하나의 소리를 phoneme이라 한다.

Praat에서는 intensity를 알 수 있고, pitch를 통해 음의 높낮이를, 모음구분할 수 있는 formant 또한 확인할 수 있다.

음성을 나타내는 방법은 크게 두가지이다. Waveform과 spectrogram이 그것이다. waveform의 경우 x축과 y축 각각 time과 value를 나타낸다. Sine wave로 나타내진다. 폭이 넓을수록 value가 낮아지고 이는 낮은 소리를 나타낸다. 즉, sine wave의 폭이 소리의 높낮이와 비례한다.

다른 표기 방법으로 spectrum은 각각 frequency와 amplitude를 가진다. spectrum을 보면 진한 부분이 있고 그렇지 않은 부분이 있다. 이는 spectrum에서 amplitude가 높은 부분이 진하게 보인다.

인간의 목소리 근원을 source라 하며, harmonics를 기반으로 한다. harmonics란 praat에서 Pure tone으로 볼 수 있는데, 일정한 주파수가 있고 그 주파수 대역에 있는 특성, 성분들을 나타낸다. Wave form의 경우 일정한 모습을 보이며, spectrogram으로 볼때는 frequency의 pitch 부분 당 amplitude가 gradually decreasing 하는 모습을 볼 수 있다. 여기서 가장 낮은 frequency를 fundamental frequency, F0라 부른다.

이러한 harmonics는 우리의 vocal tract를 지나며 달라진다. 구강구조로 인해 공기가 저항받아 성대의 떨림으로 일어나던 일정한 harmonics가 바뀌는 것이다. 이를 vocal tract에 의해 filtered 된다고 한다. Vocal tract를 지나가면서 filtered 된 voice는 음성적으로 peak 혹은 mountain을 가진다. harmonics와 다르게 일정하지 않다. 여기서 mountain 부분은 스펙트럼에서 진하게 보이고, 이는 소리가 구분되는 formant를 나타내기도 한다. 여기서 첫번째 mountain을 F1이라 부른다.

반복되는 부분을 pulse, 그것이 이어진다 해서 pulse train이라 부른다. 반복주기가 가장 낮은 주파수의 웨이브와 일치한다.

5주차

프로그래밍 언어와 언어:
프로그래밍 언어와 언어의 공통점은 단어와 문법이 있다는 것이다. 우리가 언어를 사용할때, 혹은 언어를 공부할때 흔히 단어와 문법을 가지고 공부하고 글과 말을 구성한다. 인간의 언어처럼, 프로그래밍 언어도 마찬가지다. 프로그래밍 언어도 단어와 문법을 기반으로 구성된다. 우리 언어가 단어와 문법을 어떻게 결합하느냐에 따라 의미가 달라지듯이, 프로그래밍 언어도 그렇다. 여기서 언어의 단어에 해당하는 것이 프로그래밍 언어의 variable, 변수이다. 단어가 뜻, 즉 정보를 담는 그릇이듯이, 변수도 정보를 담는 그릇이다.

우리가 프로그래밍을 공부하면서 알아야할 것들이 있다.
1. 변수에 정보를 넣는 것
2. 조건에 대한 것 -> if문
3. 반복 -> for문
4. 변수와 명령을 packaging하는 함수

우리가 프로그래밍하면 생각나는 것이 바로 자동화, 기계화이다. 인간이 하기 귀찮은 것들을 대신 하거나 반복되는 일들을 자동화하는 것이 바로 프로그래밍의 역할이다. 이 점을 생각했을때, 어떨때 어떻게 하라, 즉 조건에 대해 생각할 수 있다. 이것을 구현하는 것이 if문이다. 그리고 반복하는 것은 for문이다. 하지만 조건과 반복, 그리고 변수가 하나밖에 없을리가 없고, 또 여러개가 필요하기도 하다. 이러한 것들을 묶어 편리하게 사용할 수 있게 하는 것이 함수이다. 입력을 하면 함수를 거쳐 출력형태가 나오는 것이 기본 형태이다.

우리가 변수를 설정할때 오른쪽에 있는 정보를 왼쪽 변수에 assign한다. 따옴표를 통해 문자열이라는 자료형을 표현하고, print 함수를 통해 출력을 이끌어낸다. 대괄호를 통해 리스트라는 자료형을 표현하고, 이는 여러 자료형이 리스트 하나에 있다. 이와 비슷한 것이 튜플인데, 리스트와 달리 정보의 변경이 힘들다는 특징이 있다. '{}'로 딕셔너리라는 자료형을 나타낸다. 키값과 밸류값을 통해 연결되어 있다. type함수를 통해 자료형이 무엇인지 알 수 있다.

6주차

numpy를 사용해서 array를 구성할 수 있다.

```
import numpy as np
a = [(1,2,3),(3,8,0)]
b = np.array(a)
type(b)
```

index를 사용해서 자료형 속 일부를 나타낼 수 있다. A = [1,2]라 할때, A[0]이라 하면 1이 나온다. Variable[index]의 형식으로 쓰인다. D = {1:'key'}라 할때, 1이 key, 'key'가 value 이다. 문자열도 해당된다. s = 'abced'라 할때, s[1]은 'b'이다.

:을 통하여 범위를 지정할 수 있다. s[1:3]은 'bc'이다.

find는 해당 문자열을 찾아 시작점의 인덱스를 알려주는 함수 이다.
‘my house’.find(‘house’)라 하면 ‘house’의 시작 인덱스인 3을 아웃풋으로 내놓는다.

rindex는 오른쪽 부터 해당 문자열을 찾아 해당 문자열의 시작 인덱스를 알려준다.
‘home, my home’.rindex(‘home’) -> 9

range함수를 사용하여 일정한 범위를 지정할 수 있다.
list(range(4)) 라 하면 [0,1,2,3]이 나온다.
len() function은 해당 자료형의 길이나 개수를 나타낸다.
일정 범위의 숫자를 출력하는 함수는 for 문을 사용하면 된다.

```
a = list(range(1,8))
for w in range(len(a)):
    print(a[w])
```

enumerate함수는 인덱스와 값을 연결시킨다. a = ['red', 'green', 'blue', ‘purple']가 있으면 0 - ‘red’와 같은 방식으로 구성된다. 이것을 응용하면 다음과 같이 출력할 수 있다.

```
for i, s in enumerate(a):
    print('{}: {}%'.format(s,b[i]*100))
```

format이라는 함수를 통해 일정한 형식으로 출력하게끔 할 수 있다.

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

zip이라는 함수를 사용하면 두 리스트를 묶을 수 있다. [1,2,3] , [‘a’,’b’,’c’]를 묶어서 리스트로 만들면 [(1,’a’) …]형식으로 도출된다.

파이썬에서 같다는 ==이다. 같지 않다는 !=이다. for문안에 또 쓸 수 있으며, 조건문도 마찬가지이다. for문이 두개가 있다면 우선 하위 반복문에서 먼저 반복이 이루어진 뒤에 다시 상위 반복문으로 넘어간다.

10/28 ~ 11/03

Raw data의 처리 과정에 대하여:
우리가 흔히 데이터라고 하는 것은 전처리가 있어야 한다. 사진이나 동영상, 문장들, 소리와 같은 것들을 컴퓨터가 인식하고 처리하게끔 하려면 raw data를 다듬어주어야 한다. 어떻게 다듬냐 하면, 숫자로 처리하는 것이다.
흑백 사진이 있다고 하자. 흰색을 10 검정을 0 그 사이의 회색의 경우 흰색에 가까울수록 숫자가 커지는 0과 10 사이의 값이라 하자. 각각 구역을 설정해서 0과 10 사이의 값으로 모든 구역을 바꾼다고 하자. 이러한 경우 하나의 직사각형 모양의 행렬이 생긴다. 행렬은 말 그대로 행과 열에 숫자가 있는 구조를 말한다. 이러한 행렬을 한 줄로 처리를 하면 그 데이터는 벡터가 되며 컴퓨터가 데이터를 처리하려면 한줄 길게 느러뜨려서 벡터화 하여야 한다. 이와 같은 원리로 동영상은 시간별로 사진이 달라지는 것이라 생각하여 시간마다 달라지는 값들을 측정하면 될 것이다. 컬러의 경우 흑과 백이라는 2차원 구조에 색깔이라는 1차원이 더해져 3차원의 이미지가 될 것이다. 여기에 아까 동영상과 같이 시간이라는 것을 같이 한다면 4차원의 구조가 될 것이다.
소리의 경우는 어떠한가? 우리가 전 수업시간에도 살펴본 것과 같이 소리는 웨이브로 구성된다. 그 웨이브에 높낮이에 일정한 값을 매긴다면 discrete한 값들을 볼 수 있다. 이러한 discrete한 값을 바탕으로 소리의 wave도 벡터화 시킨다. 텍스트도 마찬가지다. 단어와 구조에 따라 일정한 값을 주어 마찬가지의 방법으로 벡터화 시킨다.

파이썬에는 일정한 함수들을 모아둔 library가 있다. 보통 외부에서 가져오는 형태들이다. 수업시간에 다룰 것 중 하나는 NumPy이다. 계산과 형식화에 도움이 된다. 보통 리스트 자료형은 수학적 계산에 다소 불편한 점이 있다. 이러한 리스트를 수학적인 데이터를 쓸 수 있도록 만든 library이다. 보통 이러한 library를 파이썬에서 쓰기 위해선 import를 쓴다 import numpy 하여 라이브러리를 불른다. 이러한 라이브러리에는 그 밑에 있는 sublibrary들이 있다. 이러한 것들을 호출하려면 '.'을 사용한다. matplotlib이라는 함수에서 그 밑에 있는 pyplot만을 사용하려면 import matplotlib.pyplot 이라 하면 된다. 다른 방법으로는 from [library] import [sublibrary] 형태로도 가능하다. 위의 라이브러리들은 이름이 다소 긴 편이다. 뭔가 귀찮음을 벗어나기 위해 'as' 를 사용할 수 있다. Import numpy as np 라 하면 이후에 np만 쳐도 numpy를 사용할 수 있다.

numpy에는 다양한 함수들이 있다.
np.empty([2,3], dtype='int')를 예로 들면, [ ] 속은 가로x세로를 나타낸다. dtype의 경우 데이터 타입을 나타낸다. 이러한 정보를 바탕으로 랜덤한 수가 있는 array를 만든다.

```
array([[8, 0, 0],
       [0, 0, 0]])
```

array는 데이터를 넣고 뺄 수 있는 형태를 말한다.
np.zeros()는 0이 있는 array를 같은 원리로 만든다.
np.arange(0,10,2, dtype='float64')의 경우 0부터 10까지 2 간격의 벡터를 만들고,
np.linspace()의 경우 일정한 값부터 일정한 값까지 일정한 값만큼의 수 만큼 사이를 만든다.
np.array([[1,2,3],[4,5,6]])과 같이 두 리스트를 엮어서 하나의 array로 만들 수 있고,
astype(np.float64)를 통해 내부 자료형도 바꿀 수 있다.
여기서 뒤에 붙는 숫자는 개수를 뜻한다.
Np.zeros_like()를 통해 행렬의 형태는 같지만 그 안은 0으로 이루어진 어레이를 만들 수 있고,

np.random.normal(0,1,100)과 같은 함수를 통해 정규분포사엥서 일정한 수 사이의 일정한 값의 array를 만들 수 있다.
Import matplotlib.pyplot as plt를 통해 pyplot을 부르고,
여기에 plt.hist(data,bins=10)을 통해 10개 간격의 히스토그램을,
plt.show()를 통해 히스토그램을 볼 수 있다.
zeros 외에도 ones도 있고,
행렬을 다른 형태로 다시 구성할 수 있는 [array].reshape()도 있다.
어레이를 비교할 수 있는 np.allclose()도 있다.
np.random.randint()를 통해 일정한 int값 사이의 랜덤값으로 구성되는 어레이를,
np.random.random()으로 랜덤한 값으로 구성되는 어레이를 만들 수 있다.
np.savez()를 통하여 파일로 저장될 수 있고,
Np.load()를 통하여 부를 수 있고,
흔히 쓰는 type, len 함수도 사용 가능하며
Shape, ndim, size, dtype을 통해 행렬의 모양, 차원, 크기, 데이터 타입을 알 수 있다.

이러한 값들은 곱하거나 더하거나 뺄셈도 가능하다.
==이나 >와 같은 비교연산자도 가능하며 어레이 내의 숫자들에 일일이 True나 False값이 출력된다.

Sum,min,max,mean,median,std도 모두 가능하다. axis=~을 통하여 어느 축을 기준으로 할 것인지도 가능하다.

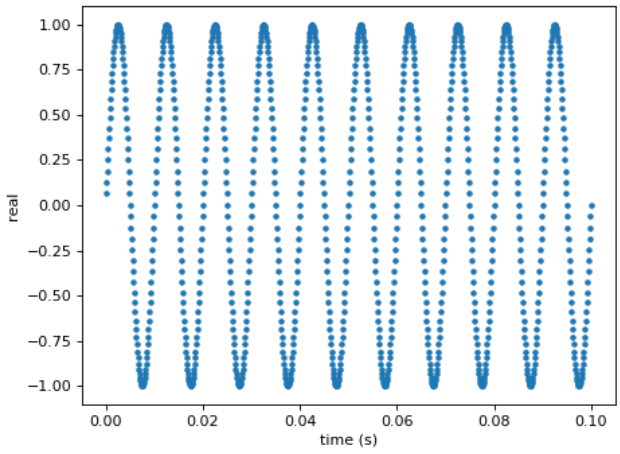
Analyzing Sound by Sinusoid: Base

우리가 익히 알고 있듯이, 음성은 사인파로 분석될 수 있다. 이러한 측면에서 사인을 어떻게 사용하여 음성을 분석할 것인가가 중요하다. 여기서 오일러의 공식이 등장한다. 오일러의 공식은 복소수를 길이와 각도의 관점에서, 회전하는 선분을 복소수의 관점에서 연산을 할 수 있도록 돕는다. $e^{i\Theta} = cos(\Theta) + isin(\Theta)$ 으로 표현이 가능하다.
복소수는 a+bi 로 정의 가능하다. $f(\Theta) = e^{\Theta i}$에서 Θ에 0, $\pi/2$, π, $3\pi/2$, 2π 를 대입하면 1, i, −1, −i 값이 산출된다. 이를 $(\Theta, f(\Theta))$의 복소수 평면에 삽입하면 (1,0),(0,1),(−1,0),(0,1)이다.
theta 값과 대응하는 것인데, 커지면 커질수록 원의 형태로 값이 이동한다.
이러한 개념을 사용하는 것이 Phasor이다. 사인 곡선의 한주기를 다룰 수 있게 해준다.

우선 필요한 모듈을 삽입하자
```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.axes_grid1 import make_axes_locatable
import IPython.display as ipd
import numpy as np
%matplotlib notebook
from scipy.signal import lfilter
```

음성을 분석하기 위한 예시로 다음과 같은 parameter들이 있다고 하자
Amplitude(이하 amp) = 1, sampling rate(이하 sr) = 10000 (여기서 sampling rate는 아날로그 신호를 데이터화 할때 샘플링을 진행하는데, 이 샘플링이 1초 동안 몇개의 부분으로 나누느냐를 말한다.) duration(이하 dur) = 0.5 frequency(이하 freq) = 100
Time(이하 t) = np.arange(1, sr*dur+1)/sr이라 하자. 1/10000초부터 0.5초의 시간 범위를 말하며, dur+1을 하는 것은 arange가 마지막 숫자를 고려하지 않기 때문에 더해준 것이다.
Theta = t * 2*np.pi * freq (파이썬에서 파이 값은 np.pi로 구현된다)
이러한 요소를 바탕으로 다음과 같이 signal을 생성한다. S = np.sin(theta)

이를 시각화 하려면 다음과 같은 과정이 필요하다
fig = plt.figure() #도표 만들기
ax = fig.add_subplot(111) #1by1 에서 1번째 subplot 1,1,1이라고 표현 가능
ax.plot(t[0:1000], s[0:1000], '.') #범위 설정 및 형식 설정(점 표시)
ax.set_xlabel('time (s)') #x축 이름 정하기
ax.set_ylabel('real') #y축 이름 정하기



다음과 같이 결과가 나온다.

이렇게 2차원의 형식으로 표현할 수 있지만, 3차원으로도 가능하다

3차원으로 표현하기 위해서는 시간과 우리가 봤던 실수부에 허수부를 포함시켜야 한다. 이러한 과젱에는 c = np.exp(theta*1j)을 사용한다. theta에 허수부인 1j를 곱하고, 그것의 expectation을 구함으로써 표현한다. 이를 시각화하려면 다음 과정이 필요하다.

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d') #projection을 3차원화 된것으로 표현
ax.plot(t[0:1000], c.real[0:1000], c.imag[0:1000], '.') # ~.real은 실수부, imag는 허수부
ax.set_xlabel('time (s)')
ax.set_ylabel('real')
ax.set_zlabel('imag') #imaginary number, 즉 허수부의 약자이다.

이러한 과정을 거치면 시간, 실수부, 허수부의 삼차원 그래프가 생성된다.
이러한 소리를 ipd.Audio(s, rate=sr)를 통해 표현할 수 있다.

11/12,14

- !pip install [module name]으로 module 설치 가능
- amplitude를 곱해주느냐에 따라 진폭이 달라짐.
- 즉, c = np.exp(theta*1j)에서 amp의 곱에 따라 진폭이 달라짐
- Complex phasor amplitude와 string의 반지름은 같음
- 이렇게 진폭이 커지면 우리가 듣기에 더 크게 들림.
- Sampling rate를 100Hz라 할때, frequency는 50Hz만 표현 가능
- 이렇게 표현할 수 있는 최대 주파수를 Nyquist frequency라 함.
- Nyquist frequency = (sampling rate)/2
- 그래서 CD에서 쓰이는 sampling rate가 44100Hz임. 가청주파수가 20000정도 인데, 44100 이면 그의 절반인 22050으로 가청주파수 범위를 커버할 수 있음.
-

```
# generate samples, note conversion to float32 array
F0 = 100; Fend = int(sr/2); s = np.zeros(len(t));
for freq in range(F0, Fend+1, F0):
# F0~Fend+1, F0만큼의 incremental, +1은 제일 마지막거 포함용
    theta = t * 2*np.pi * freq
    tmp = amp * np.sin(theta)
    s = s + tmp
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(t[0:1000], s[0:1000]);
ax.set_xlabel('time (msec)')
```

위의 코드를 바탕으로 pulse train을 만들 수 있다.
여기서 pulse train인 이유는 패턴이 그렇기 때문이다.

- frequency를 배수로 하면 다 같은 음이다. 옥타브라고 생각하면 된다.
- 그래서 같은 배수를 쌓아서 합치면 그것이 산맥을 이루고, 이러한 산맥은 formant가 됨
- sin과 cos은 pi/2정도의 차이가 있는 정도이다.
- RG를 frequency, BWG를 bandwidth라 하면, 여기서 산맥의 뚱뚱함과 뾰족함을 정하는 것은 BWG이다.
- RG와 BWG를 달리하여 쌓으면 formant가 형성된다

```
RG = 3500  # RG is the frequency of the Glottal Resonator
BWG = 200  # BWG is the bandwidth of the Glottal Resonator
a, b=resonance(sr,RG,BWG)
s = lfilter(b, a, s, axis=0)
s = lfilter(np.array([1, -1]), np.array([1]), s) #입술 만들기
ipd.Audio(s, rate=sr)
```

이렇게 해서 소리를 쌓고 마지막에 입술 역할을 하는 것을 형성하면 사람의 소리와 비슷한 소리를 낼 수 있다.

우리가 흔히 인공지능이라 할때, 인풋 데이터와 그 중간에 기계라고 하는 것, 그리고 기계를 거쳐 나오는 아웃풋 데이터가 있다. 이 데이터들의 형태는 벡터라고 부르는 숫자의 형태로 구현되며, 이러한 벡터로 구성된 행렬이 기계, 즉 인공지능의 기초이다.

$\begin{bmatrix}1\\5\end{bmatrix}$ 이러한 vector를 column vector라 하자. 이러한 벡터는 평면 상에서 원점에서 (1,5)까지의 과정을 기하적으로 표현할 수 있다.

이러한 벡터에 대하여 일정한 숫자를 곱하거나 더할 수 있다. 이를 linear combination이라 한다. $c \cdot v + d \cdot w$ 에서 c, d는 scalar, v,w를 vector라 하여 정의 할 수 있다.
scalar에 따라 linear combination의 값이 달라지는데, 이러한 combination 수를 무한대로 확장하면 모든 점이 성립한다. 이러한 가능한 linear combination의 결과값이 있을 수 있는 공간에 대하여 column space라 한다.
여기서 n component의 모든 벡터를 담고 있는 space를 whole space라고 하고, 모든 linear combination으로 구성되는 subspace를 column space라 한다. C(A)라 표현한다. Whole space와 column space는 차원이 다를 수 있다.
$A = \begin{bmatrix}2 & 1\\1 & 3\end{bmatrix}$ 이라는 행렬이 있다고 하자. 이 행렬은 같은 선상에 있지 않아 모든 면에 대하여 공간을 채울 수 있다. 이러한 관점에서 (2,1) (−1,3)은 서로 independent하다
$B = \begin{bmatrix}1 & 2\\2 & 4\end{bmatrix}$ 의 경우에는 한 선에 있다. (1,2), (2,4) 이들은 서로 dependent하다.
$A = \begin{bmatrix}1 & 3\\2 & 3\\4 & 1\end{bmatrix}$ $A^T = \begin{bmatrix}1 & 2 & 4\\3 & 3 & 1\end{bmatrix}$ 역행렬을 이렇게 표현할 수 있다.
Row vector
Null space는 영점을 향해가는 vector들을 말하는데 다음과 같이 표현할 수 있다.
$Ax = \begin{bmatrix}1 & 2\\2 & 4\\3 & 6\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}$ 에서 Ax = 0 을 만드는 모든 vector들을 말한다.
반대로 [3 6]과 같은 vector를 row vector라 할때, 우리는 column space처럼 row space를 말할 수 있다. 여기서 row space에서는 null space, column space에서는 left null space라 한다.

우리는 grid를 바꿀 수 있는데, 이를 linear transformation이라 한다. Ax = b로 표현할 수 있다.
$\begin{bmatrix}0/9 & -0.4\\0.4 & 0.9\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}0.5\\1.3\end{bmatrix}$ 이라 할때, (1,1)의 기준 점이 (0.5,1.3)으로 변환된다고 보면 된다.
여기서 우리는 다시 돌릴 수 있는데, detransformation 이라한다. $A^{-1}b = x$ 로 표현할 수 있다.

여기서, 위에서 살펴본 dependent의 사례라 생각할때, invertible하지 않아서 어디로 다시 돌아가야 할지 모른다.

그렇다면 null space는 어떻게 응용이 가능한가. 흔히 AI라고 할때 강아지 사진을 학습시켜 구분하도록 하는 예시가 흔하다. 이러한 구분에 null space가 사용한다. Null space는 0으로 수렴하는 벡터들이다. 이렇게 0으로 수렴한다면 항상 같은 값이 나온다. 강아지 사진에서 어느 일정한 값들이 null space와 평행하게 이동한다면 항상 같은 값을 갖는다. 즉, 출력에 영향을 미치지 않는 값들이 null space이기 때문에 그렇다. $Ax = b$라는 식에서도 봐도 그렇다.

Eigenvector란 어떠한 matrix A를 transform 했을때, 그 결과가 상수배가 되는 0이 아닌 vector를 말한다. $Av = \lambda v$로 표현할 수 있다. 여기서 람다는 상수이다. 여기서 람다는 eigenvalue라 하며, 우측 v는 eigenvector라 할 수 있다. 즉, 변형된 벡터가 상수배인 것을 말하다. 기하학적인 정의로는 변환 전과 변환 후가 평행한 vector라고 말할 수 있다.
이러한 eigenvector는 그 평행한 값에 대하여 모든 점이 가능하며, 이러한 공간에 대하여 위에서 살펴본 다른 space처럼 eigenspace라 한다.

상관관계에 대하여 다음과 같이 바라볼 수 있다. 국어 점수와 영어점수가 있다고 하자. 그리고 각 30개의 표본이 있다고 하면 이를 행렬로 변환한다면 30차원이 생긴다. 이에 대하여 기하학적으로 어떠한 일정한 선을 그리고, 그 각각의 선을 가지고 이를 그렸다고 생각해보자. 이 선에 대하여 어느 정도의 각이 생긴다. 이러한 각에 대하여 분석하는 것에 대하여 cosine similarity로 분석할 수 있다. 이러한 각을 코사인 값으로 구하여 −1과 1사이의 값으로 보아 상관관계를 구할 수 있다.

Inner product에 대한 기하학적 해석은 다음과 같다. 어떤 직선과 직선 사이에 각 θ에 대하여, 한 직선의 점에서 내린 수선의 발까지의 길이와 어떠한 점 까지의 길이와 같다. 즉, 한 벡터와 다른 벡터에서 수선의 발을 내린 길이의 곱이며, 수선의 발을 내린 길이는 각의 코사인 값에 다른 벡터의 값을 곱한 것과 같다.

이러한 inner product를 응용하여 주파수를 분석할 수 있다. 어떠한 주파수의 합들에 대하여 있다고 하자. 그 주파수에서 점점이 끊어 그 값마다 벡터를 설정하고 행렬을 만들고 다른 행렬의 inner product로 구성이 가능하다. 주파수가 중첩이 많이 되어있다고 하면, 같은 값이 많이 있다면 그 inner product한 값은 높을 수 밖에 없다. 이러한 원리를 바탕으로 스펙트로그램의 진하기를 구성할 수 있다.

하지만 이러한 inner product는 단순 phasor를 사용시 문제가 생긴다. 똑같은 주파수 그래프를 가져왔다고 가정해보자. Inner product를 했을때 원래는 그 inner product의 값이 더 높아져야 하는 것이 맞다. 하지만, 같은 그래프를 약간 오른쪽으로 위치하였다고 가정할때, 두 주파수의 inner product는 0의 값을 지니게 된다. 여기서 이러한 차이는 theta가 90도일 때 그러하다.

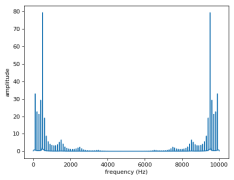
이러한 이유로 inner product를 사용한 분석에서 일반 sine phasor를 사용하지 않는다. 복소수 공간을 사용하는 complex phasor를 사용하여 inner product를 한다. 이렇게 complex phasor를 사용할 경우 이동에 대한 민감도가 떨어지기 때문에 보다 원활하게 값을 구할 수 있다.

영어와 국어의 상관관계에 대하여 다음과 같이 생각할 수 있다. 영어의 벡터가 이루는 직선과 국어의 벡터가 이루는 직선이 있다고 하자. 이 선이 같은 선상에 있다고 하자. 이 둘 간의 관계는 2가지로 표현될 수 있다. 만약 정반대라면, 둘 사이의 기하학적 각도는 180도이다. 이 각도의 코사인값은 -1이다. 반대의 경우는 0도이며, 코사인 값은 1이다. 이러한 측면에서, $cos\theta = r$의 식이 성립한다. 즉, 기하학적 각도가 코사인을 통해 상관관계로 이어질 수 있다는 것이다. 이러한 이유로, 만약 같은 곡선이 180도 이동한다고 가정했을때, 둘 간의 관계는 다음과 같다. 한 곡선의 값이 내려갈때, 이동한 곡선의 값이 상승한다.

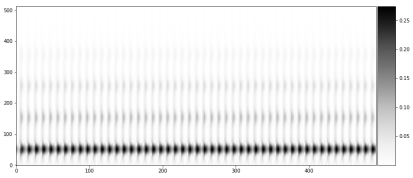
위와 같은 이유로 complex phasor를 사용한다. 기존의 값들은 위와 같이 이동함에 따라서 민감하게 반응한다. 이러한 측면에서 complex number, 복소수와 곱을 통해서 complex phasor를 생성한다. 여기서 단순 곱이기 때문에 벡터의 개수는 곱 이전과 같아야 한다. 이러한 complex phasor의 값은 근사값의 소수로 바꾸는 floating이 불가능 하다. 그래서 그 절대값을 구한다.

이를 코드로 구현하면 다음과 같다.

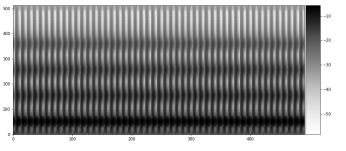
```
from scipy.io import wavfile
# sr, s = wavfile.read('a.wav')
nSamp = len(s) ### 샘플의 개수를 불러온 파일의 길이로 구한다.
dur = nSamp / sr
t = np.linspace(1/sr, dur, nSamp)
nFFT = nSamp ### 샘플의 개수는 푸리에변환의 개수와 같다.
amp = [];
for n in range(0,nFFT):
    omega = 2*np.pi*n/nFFT # angular velocity
     ### 푸리에변환 개수 분의 0과 샘플 개수의 사이의 수를 2pi
와 곱해줌으로써 해당하는 각속도를 구한다.
    z = np.exp(omega*1j)**(np.arange(0,nSamp))
     ### 위에서 구한 각속도와 복소수를 곱한 것의 평균과 그
의 지수로 0과 sample사이의 수의 array로 한다.
    amp.append(np.abs(np.dot(s,z))) ### 파일 읽은
것과 위에서 구한 값의 inner product한 것을 절댓값화
하고 amp에 넣는다.
```



이를 그래프로 표현하면 다음과 같다.



여기서 양쪽이 같은 것은 복제됐기 때문인데, 그러므로 왼쪽 것만 보면 된다. 왼쪽의 그래프를 살펴보면 쌓여져 있는 frequency가 보인다. 이를 변환시켜 표현하면 우리가 익히 아는 spectrogram이 생성된다.



하지만 표현하면 위와 같이 약해지기 때문에, 이를 좀 더 진하게 표시하려면 다음과 같이 하면된다.

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

먼저 작은 것은 작게 큰 것은 크게하기 위해 제곱을 통해 구현한다.

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

이런 로그 스케일 같은 경우 어느정도 보기 쉬운 적당한 숫자로 바꾸는 용도에 적합하다.