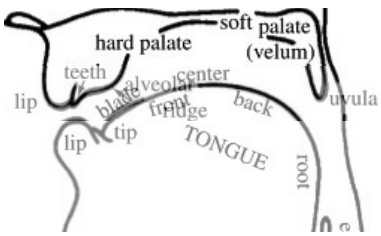
[2019-2 영어음성학]

이름(학번): 민지우(2016130575)

Week1 Summary

Spelling in English is inconsistent with its pronunciation, which call for the study of **phonetics**, all about the speech sound. We look at the actual sound of a word, and use phonetic symbols to describe it. There are about 44 distinguishable sounds that form words: **phonemes**, much diversified than 26 letters in the English language. Each phoneme represents vowels(monophthongs, diphthongs) or consonants. The study area of phonetics can be divided into three categories; articulatory, acoustics, and auditory, each accounting for production, transmission, and hearing of speech sound.

✱ Articulation



Making a speech sound is related to physical movements of human organs. Depending on subtle changes in the vocal tract (pharynx, epiglottis, uvula, soft palate(velum), hard palate, alveolar ridge, teeth, tongue body, tongue tip, and lip) while air flows, different sounds are articulated. We can divide five speech organs which control constrictions: lip, tongue tip, tongue body, larynx(vocal cords), and velum.

(1) Articulatory process in lips/ tongue tip/ tongue body

- Constriction Location(CL)
  - lips(bilabial/labiodental): p, b, m, w, f, v
  - tongue tip(dental/alveolar/palate-alveolar/retroflex): θ, ð, t, d, s, z, n, l, ʃ, ʒ, ʧ, ʤ, r
  - tongue body(palatal/velar): j, k, g, ŋ(+ change in oral cavity makes different vowels)
- Constriction Degree(CD)
  - upper part(nearly closed)> stop(p, b, t, d, k, g) - fricative(f, v, θ, ð, s, z, ʃ, ʒ, h) - approximant(w, l, r, j) - vowels <lower part(opened)

(2) Phonation process in larynx(voice box)

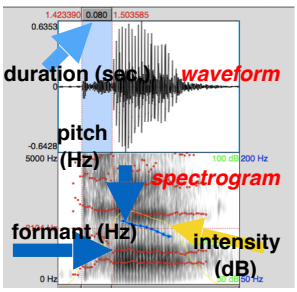
- Vocal cords(=vocal folds) vibrate in a varied degree when air flows from the lung.
- Voiced(vocal cords closed): b, d, g, v, ð, z, ʒ, ʤ m, n, ŋ, l, r, w, j
- Voiceless(vocal cords opened): p, t, k, f, θ, s, ʃ, h, ʧ

(3) Oro-nasal process in velum

- When the velum is lowered, the nasal tract is opened, so we can breathe through nose and produce nasal sounds m, n, ŋ.
  - Velum: m, n, ŋ
- *When the velum is raised, the nasal tract is closed, and we can produce every vowel and every consonant except for nasal sounds m, n, ŋ.*

We can predict any phonemes with description of constricting environment, for example, “t,” is a sound produced when velum is raised(not nasal sounds), larynx is opened(voiceless), constriction location is at tongue tip(alveolar) and constriction degree is upper part(stop).

✱ Acoustics



With Praat, we can examine the duration, intensity(loudness), pitch(65-200Hz for male speech, and 145-275Hz for female speech), spectrogram, formant(ed dots signify different vowels) of sounds. **Pitch** is defined as the number of occurrences of repeating event per second(Hz). We can see the repeated event as the vocal cords vibrate repeatedly. For example, 236.5Hz means there was 236 times of vibration in vocal cords per second, single event took 0.004228 seconds to be concluded. A pure tone of certain frequency shows sine wave and every natural sound can be represented with simplified sine wave of same frequency.

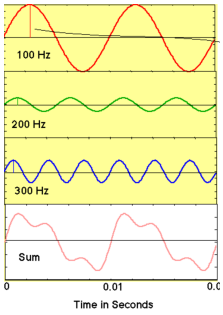
[2019-2 영어음성학]

이름(학번): 민지우(2016130575)

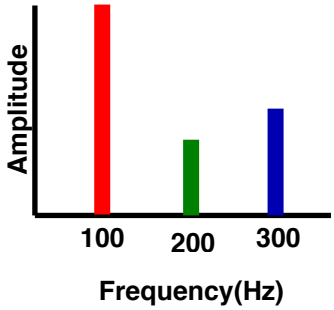
Articulation	CL	CD	Velum	Larynx
/p/	bilabial	stop	raised	open
/d/	alveolar	stop	raised	closed
/z/	alveolar	fricative	raised	closed
/n/	alveolar	stop	lowered	closed

Week2 Summary

Every signal(including sounds) can be expressed with a synthesis of various sine waves. We call sounds of a sine wave as a simplex(pure) tone and a sum of sine waves as a complex tone. A sine wave is shaped by frequency and amplitude of a sound over time. A lower frequency means fewer vibrations in vocal cords, which produces sparse graph and vice versa. 100Hz signifies 100 times of vibrations per second and, if doubled, it becomes 200Hz. Amplitude; air pressure, is not an intrinsic value of a sound source but varies over a wide range depending on the distance, closer the bigger, thus, producing bigger graph and vice versa.



In the graph left-side where sound waveforms are depicted, x-axis means time in seconds and y-axis means value. In spectrum right-side(spectral slice of spectrogram), x-axis corresponds to frequency(Hz) and y-axis corresponds to amplitude over time. Spectrogram is a temporal concatenation of spectrum, its x-axis being time and y-axis being frequency. Light-Dark gradation represents the amplitude of each frequency, lower in frequency being darker in the case of a pure tone spectrogram.



Human voice source is a sound at larynx, which can be measured by ElectroGlottoGraph(EGG). When we focus on the sound pitch(magnitude and frequency, source can be sorted as a complex tone, consisting of harmonics. A complex tone is a sum of pure tones at integer multiples of the lowest pure tone(fundamental frequency = F0 = pitch). Source-filter theory represents speech production process where sound sources of distinct spectrum are filtered by the resonant properties of the vocal tract. While source shows gradually decreasing graph in the frequency spectrum, filtered by vocal tract(audio) shows zigzagging(carved) graph with peaks and valleys. Amplitude Peaks in the frequency spectrum of the sound are analyzed as formants, the spectral shaping by acoustic resonances of the vocal tract. The formant with the lowest frequency is called F1, the second F2. In a graph where x-axis is F2 and y-axis is F1, corresponding to the geometry of mouth, we can locate every monophthong as spots and diphthong as lines, as x-axis determines front/back position and y-axis determines height of vowels.

Week3 Summary

모든 언어는 사용자 간 소통을 가능케 하는 단어와 문법으로 구성된다. 단어가 사용자 의도에 따라 정보를 담는 그릇 역할을 한다면, 문법은 단어를 이어 상대에게 전달하는 역할을 한다. 프로그래밍 언어에도 정보를 담는 그릇인 변수와 사람과 기계가 소통하기 위한 문법이 존재하며, 크게 variable assignment(변수 할당), if conditioning(if문), for conditioning(for문), function(입력-출력 패키지 함수)로 표현할 수 있다.

MacOS 터미널에서 jupyter notebook을 실행하면, 웹 브라우저에서 컴퓨터 디렉토리를 전시하고, 파이썬 프로그램을 사용할 수 있다. 변수에 넣을 수 있는 정보의 종류에는 숫자와 문자열이 있다. 숫자는 그대로 사용할 수 있지만, 문자는 인용구로 사용해야 한다. 인용구가 아닌 문자는 변수명으로 인식된다. 하나의 cell에 여러 코드를 쓰려면 enter을 하거나 semicolumn(;)을 사용해야 한다. 리스트는 square bracket[] 안에 comma(,)를 이용하여 정보를 연결하며, 모든 자료형 정보를 가질 수 있다. 튜플은 () 안에 comma(,)를 이용하여 정보를 연결하며, 모든 자료형 정보를 가질 수 있다. 딕셔너리는 {} 안에 comma(,)를 이용하여, 표제어: 설명 형식의 정보를 연결하며, 표제어는 고정값을 갖는 자료형(숫자, 문자열, 튜플) 정보를 이 변수 정보를 가질 수 있다.

- 변수 할당: 변수명 = 정보
- 리스트 인덱싱: 리스트 변수명[인덱스#]

[2019-2 영어음성학]

이름(학번): 민지우(2016130575)

- print함수: print(입력)
- type함수: type(입력)
  - int: 정수 / float: 실수 / str: 문자열 / list: 리스트 / tuple: 튜플 / dict: 딕셔너리
- cell: 입력창
  - cell select A: create a new cell above the selected cell
  - cell select B: create a new cell below the selected cell
  - cell select X: delete the selected cell
  - cell select Run or Shift Enter: run cell by cell
- examples

b = 'love' : 변수명 b에 문자열 'love'를 할당한다.
love = 2 : 변수명 love에 숫자 2를 할당한다.
b = love : 변수명 b에 변수 love를 할당하는데, 변수 love는 숫자 2를 값으로 갖는다.

Week4 Summary

- 변수에 정보를 할당할 때, 변수를 사용하여 할당된 정보를 불러들여 처리할 수 있다.
  - 함수(입력값) 출력 결과는 입력값의 함수처리 값이다. (ex) float(숫자): 실수화, int(숫자):정수화, str(숫자, 문자열, 튜플, 리스트, 딕셔너리 cluster 문장 부호까지 문자열 요소로 다룸):문자열화, list([숫자, 문자열 cluster]또는 '문자열' slice):리스트화, tuple([숫자, 문자열 cluster] 또는 '문자열' slice):튜플화, dict([[]]): 딕셔너리화)
  - 요소 불러오기: 변수명[인덱스]
    - 인덱스는 정보 왼쪽은 0부터, 오른쪽은 -1을 기준 위치로 삼기 때문에, 첫번째 요소는 변수명[0]이고, 마지막 요소는 변수명[-1]이다.
  - 여러 요소 불러오기: 변수명[인덱스 시작:인덱스 끝+1]
  - len(변수명)함수는 정보의 길이를 구한다.
  - 변수명.upper()함수는 변수 안에있는 모든 문자를 대문자로 변환한다.
  - 문자열에서 + '문자열'은 문자열을 결합, 확장하고, *#은 요소를 #번 반복하여 문자열을 extend한다.
  - 리스트에서 + [리스트]는 리스트 extend(확장)이고,*# 은 요소를 #번 반복하여 리스트 extend한다.
  - 변수명.find('찾는 요소')함수는 찾는 요소의 위치를 출력한다.
  - 변수명.rindex('찾는 요소')함수는 찾는 요소를 오른쪽부터 찾아서 위치를 출력한다.
  - 변수명.strip('제거할 요소)함수는 문자열 앞뒤에서 '제거할 요소'를 제거한 결과를 출력한다.
  - 변수명.split('분리할 기준 요소')함수는 문자열에서 '분리할 기준 요소'를 기준으로 문자열을 분리하여 리스트화한다.
  - '합칠 기준 요소'.join(iterable)함수는 문자열구조에 합칠 요소를 기준으로 iterable 요소를 합쳐서 문자열화한다.
  - 변수명.replace('기존 요소값', '바꿀 요소값')함수는 문자열에서 '기존 요소값'을 '바꿀 요소값'으로 일괄 변경한다.
  - markdown은 기본 cell에서 #입력과 같이 실행 무효화 효과가 있다.

syntax

- **for문과 if문**
  - **for i in ___ :**
    - in 뒤 ___에 있는 요소값을 하나씩 불러서 i에 할당하고, : 뒤의 문장을 실행한다.

- **for i in range(#) :**
  - **range(#)**
    - 0부터 #까지 인덱스를 만든다
    - # = len(___)이면 ___의 길이만큼 인덱스를 만든다.
- **enumerate(___) **a변수 b변수가 길이가 같아야 한다**
  - ___에 추가로 번호를 매긴다.
  - for i,s in enumerate(a):
    - i: 인덱스 번호값, s:요솟값
- **'{^^}: {**}%'.format(^^값,**값)**
  - == ^^값: **값%
- **zip(___,___)**
  - 앞 변수와 뒷 변수가 페어로 묶인다.
  - for i, s in zip(a, b):
    - i a 첫 요소~, s b 첫 요소~
- **nested for loop**
  - range(#)은 0부터 #까지 인덱스가 만들어진다.
  - range(#1, #2)는 #1~(#2-1)까지 인덱스가 만들어진다.
  - 바깥 루프는 바깥 range만큼 돌고, 안 루프는 바깥 range * 안 range만큼 돈다.(곱셈)

Week6 Summary

데이터를 다루기 위해서는 데이터를 벡터화해야하는데, 이는 영상, 소리, 텍스트 등 숫자회되지 않은 데이터를 숫자의 나열인 벡터 형태로 변환하는 것이다.

NumPy Basics

Creation

- 라이브러리 불러오기
  - 리스트에 숫자가 들어갈 때, 요소 간 수학적 처리를 하기 위해서는 numpy 라이브러리가 필요하다.

```
#import numpy as variable
import numpy as np
a = [1, 3, 5]
b = [2, 4, 6]
#np.array(variable)
A = np.array(a)
B = np.array(b)
a + b
>> [1, 3, 5, 2, 4, 6]
A + B
>> array([ 3,  7, 11])

#np.array( [ [  ],[  ] ] ) : [   ] 이 한 행을 의미한다.
X = np.array([[1, 2, 3], [4, 5, 6]])
>>array([[1, 2, 3],
        [4, 5, 6]])
```

[2019-2 영어음성학]

이름(학번): 민지우(2016130575)

#variable.shape

X.shape

>>(2, 3) ***#[variable 갯수] * [요소 갯수] matrix : 차원 수는 숫자 갯수***

x.shape ***#2개가 있다 2*3 matrix가***

>> (2, 2, 3)

#np.empty((배열 크기), dtype='') : 배열을 생성만 하고 특정한 값으로 초기화 하지 않는다. *empty* 명령으로 생성된 배열에는 기존 메모리에 저장되어 있던 값으므로, 배열의 원소 값을 미리 알 수 없다.

np.empty([2, 3], dtype='int')

>>array([[0, 0, 0],
[0, 0, 0]])

#np.zeros(ones)((배열 크기), dtype='') : 크기가 정해져 있고, 모든 값이 *0*인 배열을 생성 *np.zeros(*숫자*)*는 한 행을 출력하고 *np.zeros((*숫자*1,* 숫자*2))*는 숫자*1**숫자*2 matrix*를 출력한다. 리스트를 출력하기 때문에 인덱싱하여 요솟값을 입력하는 게 가능하다. *0*이 아닌 *1*로 초기화된 배열을 생성하려면 *ones* 명령을 사용한다.

np.zeros((2, 3))

>>array([[0., 0., 0.],
[0., 0., 0.]])

d = np.zeros(4)

d[0] = 1 ; d[1] =100; d[2]=-100; d[3]=-1; d

>>array([1., 100., -100., -1.])

+ ***#np.zeros(ones)_like(variable, dtype='') :*** 크기를 명시하지 않고, 다른 배열과 같은 크기의 배열을 생성
+ ***#x*0과 같은 결과***

#np.arange(시작, 끝(포함하지 않음), 간격, dtype='') : NumPy버전의 range 명령으로, 특정한 규칙에 따라 증가하는 수열을 만든다.

np.arange(0, 10, 2, dtype='float64') ***#0 .. n-1***

>>array([0., 2., 4., 6., 8.])

#np.linspace(시작, 끝, 갯수, dtype='') : 선형 구간을 지정한 구간의 수만큼 분할한다.

np.linspace(0, 10, 6, dtype=float)

>>array([0., 2., 4., 6., 8., 10.])

#variable.dtype: variable의 타입을 알려준다.

>>dtype('')

#variable.astype(np.바꿀dtype): variable의 dtype을 바꿔준다.

X = np.array([[1,2,3],[4,5,6]])

>>array([[1, 2, 3],
[4, 5, 6]])

X.astype(np.float64)

>>array([[1., 2., 3.],
[4., 5., 6.]])

x = np.array([[[1, 2, 3], [1, 2, 3]], [[3, 4, 5], [3, 4, 5]]])***#3차원 데이터 만들기 np.array([[[ ], [ ]], [[ ], [ ]]])***

#variable.ndim: variable이 n차원인지 알려준다.

x.ndim

>> 3

import matplotlib.pyplot as plt **또는** ***from matplotlib import pyplot as plt***

#np.random : 무작위 표본 추출

#np.random.normal(loc, scale, size) : 정규분포

np.random.normal(size=5)

>>array([-0.80393769, -0.27551346, 1.00166459, 1.45758192, -0.05037177])

np.random.normal(size=(2, 2, 2))

>>array([[[0.63333144, -0.26495004],
[-0.18745881, 0.47088579]],

[[1.0518922 , 0.41355533],
[2.00158582, 0.15248479]]])

np.random.normal(0, 1, 10)

array([1.58356498, 0.39114895, -0.12747642, 0.10523307, -0.00723903, -0.62562718, 1.32489554, -1.08353975, -1.24026282, 0.66835564])

data = np.random.normal(0, 1, 100) ***#normal distribution을 갖는 random한 100개의 데이터***

#plt.hist(variable, bins=#) : variable에 대해 막대기 갯수가 #개인 히스토그램을 만든다. 얼마나 세세하게 보여줄 것인가에 따라 bin 갯수를 조절한다.

plt.hist(data, bins=10)

#plt.show() : 방금 만든 히스토램만 출력한다.

plt.show()

✱Manipulation

#variable.reshape(배열 크기) : variable 배열 내부 데이터는 보존한 채 형태(배열 크기)만 바꾼다.
사용하는 원소의 갯수가 정해져 있기 때문에 배열 크기에 -1을 넣으면 자동으로 계산해서 크기를 맞춰준다.

a = np.array([[[1, 2, 3], [4, 5, 6,]], [[6,7,8], [8, 9, 10]]])

a.reshape(3, -1)

>>array([[1, 2, 3, 4],
[5, 6, 6, 7],
[8, 8, 9, 10]])

a.reshape((2,6))

>>array([[1, 2, 3, 4, 5, 6],
[6, 7, 8, 8, 9, 10]])

np.allclose(a.reshape(-1,2), a.reshape(6,2))

>>True

✱NumPy I/O

#numpy.random.random(/rand)(size=None): 0부터 1사이의 균일 분포의 난수를 생성한다.

np.random.random((2,3))

>>array([[0.49784128, 0.96587066, 0.13008896],
[0.44370748, 0.65424579, 0.19724159]])

#numpy.random.randint(low, high=None, size=None): 균일 분포의 정수 난수를 생성한다.

np.random.randint(0,10, (2, 3))

>>array([[9, 5, 3],
[1, 7, 1]])

#numpy 배열 외부 파일로 저장하고, 불러오기

#np.save(): 1개의 배열을 NumPy format의 바이너리 파일로 저장하기

#np.savez(): 여러 개의 배열을 1개의 압축되지 않은 *.npz 포맷 파일로 저장하기

np.savez('testt', np.random.normal(0, 10, 100), np.random.randint(0, 10, (3, 4, 2)))

#np.load(): np.save() 또는 np.savez()로 저장된 *.npy파일 또는 *.npz파일을 배열로 불러오기

npzfiles = np.load('testt.npz')

np.load('testt.npz').files

>>['arr_0', 'arr_1']

np.load('testt.npz')['arr_0']

>>열을 출력한다.

#ls 찾는값*: 디렉토리에서 값을 찾는다.

#ls-t 찾는값*: 디렉토리에서 값을 찾아 시간 순으로 나열한다.

#ls-s 찾는값*: 디렉토리에서 값을 찾아 크기 순으로 나열한다.

#ls-a 찾는값*: 디렉토리에서 값을 찾아 .(현재 디렉토리), ..(상위 디렉토리)까지 출력한다.

#ls-al 찾는값*: 디렉토리에서 값을 찾아 .(현재 디렉토리), ..(상위 디렉토리)까지, 좀 더 자세한 정보를 출력한다.

#del : 변수를 지운다

#%who: 변수를 확인한다.

#np.loadtxt('text.txt', delimiter = ',', skiprows = 1, dtype = 'int'): 텍스트 파일 불러오기

dtype = 입력 포맷, delimiter = 구분 기호, skiprows = 특정 행 skip, usecols = 특정 컬럼만

np.loadtxt("regression.csv", delimiter=",", skiprows=1, dtype={'names':("red", "blue"), 'formats':('float', 'float')})

>>array([(3.3 , 1.7), (4.4 , 2.76), (5.5 , 2.09), (6.71 , 3.19),
(6.93 , 1.694), (4.168, 1.573), (9.779, 3.366), (6.182, 2.596),
(7.59 , 2.53), (2.167, 1.221), (7.042, 2.827), (10.791, 3.465),
(5.313, 1.65), (7.997, 2.904), (5.654, 2.42), (9.27 , 2.94),
(3.1 , 1.3)], dtype=[('red', '<f8'), ('blue', '<f8')])

#np.savetxt('text.txt', 저장할 변수, delimiter = ','): 텍스트 파일 저장하기

np.savetxt("regression_saved.csv", data, delimiter=",")

!ls -al regression_saved.csv

>>-rw-r--r--@ 1 minjiwoo staff 253 11 1 16:46 regression_saved.csv

* Inspecting

arr = np.random.random([5,2,3])

print(type(arr))

>><class 'numpy.ndarray'>

print(len(arr))

>>5
print(arr.shape)
>>(5,2,3)
print(arr.ndim)
>>3
print(arr.size) #5*2*3
>>30
print(arr.dtype)
>>float64

* Operations

a = np.arange(1, 5); a
>>array([1, 2, 3, 4])
b = np.arange(9, 5, -1)
>>array([9, 8, 7, 6]
a - b
>>array([-8, -6, -4, -2])
#1차원을 2차원으로(arange(1,10) 요소가 총 9개니까 그 안에서 reshape 가능하다)
a = np.arange(1, 10).reshape(3,3)
b = np.arange(9, 0, -1).reshape(3,3)
#각각 원소에 대해서 비교한다. matrix 배열이 다르면 error message
a == b
>>array([[False, False, False],
[False, True, False],
[False, False, False]])
a > b
>>array([[False, False, False],
[False, False, True],
[True, True, True]])
#variable.sum(), np.sum(variable)
a.sum(), np.sum(a)
>> (45, 45)
#variable.sum(axis=0), np.sum(variable, axis=0) : 첫 번째 차원의 관점에서 실행해라 세로축 압축 (열)
a.sum(), np.sum(a)
>>(array([12, 15, 18]), array([12, 15, 18]))
#variable.sum(axis=1), np.sum(variable, axis=1) : 두 번째 차원의 관점에서 실행해라 가로축 압축 (행)
>>(array([6, 15, 24]), array([6, 15, 24]))

* Broadcasting

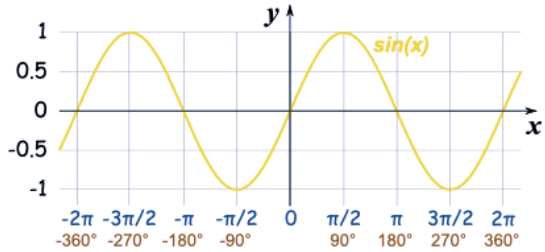
#한 차원만 맞으면 연산 가능하다 : 4*6과 4*1 가능

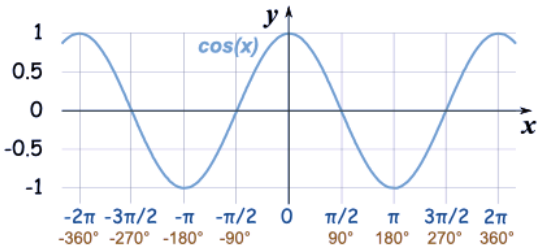
[2019-2 영어음성학]

이름(학번): 민지우(2016130575)

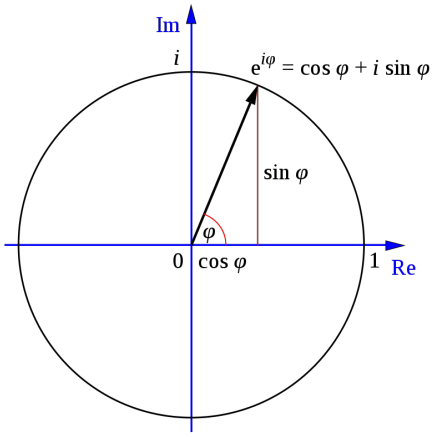
Week7 Summary

Sound





- Sine curve
  - a curve in rectangular coordinates of the equation $y=a\sin bx$ where a and b are consonants
  - it starts at 0, heads up to 1 by π/2 radians (90°) and then heads down to –1.
- Cosine curve
  - a curve whose equation in Cartesian coordinates is of the form $y = a \cos x$
  - it starts at 1 and heads down until π radians (180°) and then heads up again.
- Phasor
  - A line used to represent a complex electrical quantity as a vector
- Euler's formula



  - $f(\theta) = e^{i\theta} = a + bi = \cos(\theta) + \sin(\theta)i$
  - for any real number (θ), where i is the imaginary unit
  - θ = 0, π/2, π, 3π/2, 2π

Parameter Setting

```
amp = 1          # range [0.0, 1.0]
sr = 10000       # sampling rate, Hz
dur = 0.5        # in seconds
freq = 100.0     # sine frequency, Hz
```

1. Generate time

```
t = np.arange(1, sr+1)     #sampling rate만큼의 time tick/ 1초면 *1
t = np.arange(1, sr * dur+1)/sr     #duration만큼 1초면 *1, 0.5초면 *0.5: *duration, /sr
```

- t 0.0001 0.0002 ... 0.5000: samling rate이 1초에 10000번 들어가야 하는데, duration이 0.5니까 1초에 10000개 단위로 찍히나 1초까지 안 간다. Time tick을 0.5까지만 만든다. 1~ 5000, index는 있으나 실제 time은 아니니, sampling rate으로 나눠야 한다.

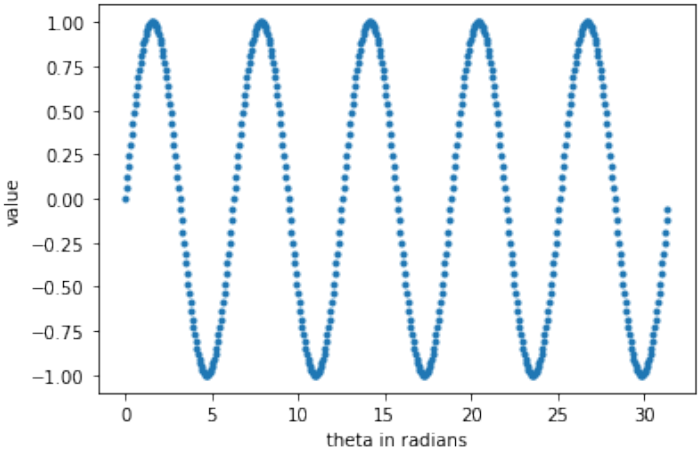
0부터 2*np.pi 하면 시간 개념 전혀 안 들어 있어서, t를 가지고 theta로 확장 시켜야 한다.

2. Generate phase

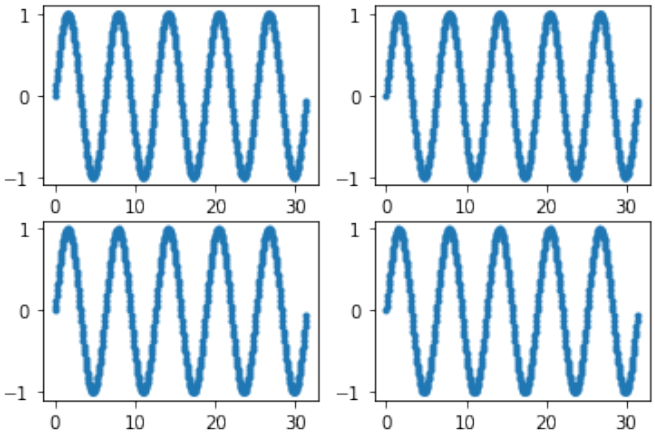
- theta = t * 2*np.pi * freq #np.pi np 안에 있는 pi를 불러온다. 2*np.pi 사인 한 바퀴 몇 바퀴? *freq
- 1초에 2pi ~ 한 바퀴 도니 ~ * frequency = 총 만들 순환 수

3. Generate signal by sine-phasor

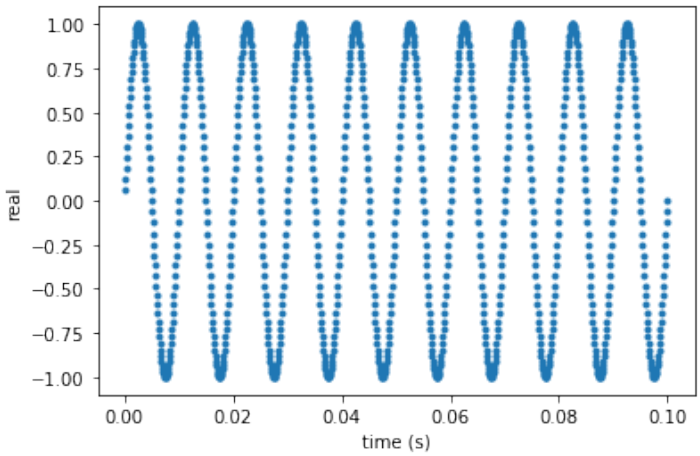
- **1. time 만들기 2. phase 연동(각도)**
- theta = np.arange(0, 2*np.pi*주기, 2*np.pi*주기/간격) #radian
- s = np.sin(theta)
  - time 벡터 크기와 theta 벡터 크기는 같다
- fig = plt.figure() # figure은 화면 전체
- ax = fig.add_subplot(111)
  # x축에서 equidistance 한데, y축에서 equidistance하지 않다. linear function이면 x, y equidistance 하나, curve이기 때문에, x equidistance 성격이 y에 적용되지 않는다.
- ax.plot(theta, s, '.')
- ax.set_xlabel('theta in radians')
- ax.set_ylabel('value')



- fig = plt.figure()
- ax = fig.add_subplot(221)
- ax.plot(theta, s, '.')
- ax = fig.add_subplot(222)
- ax.plot(theta, s, '.')
- ax = fig.add_subplot(223)

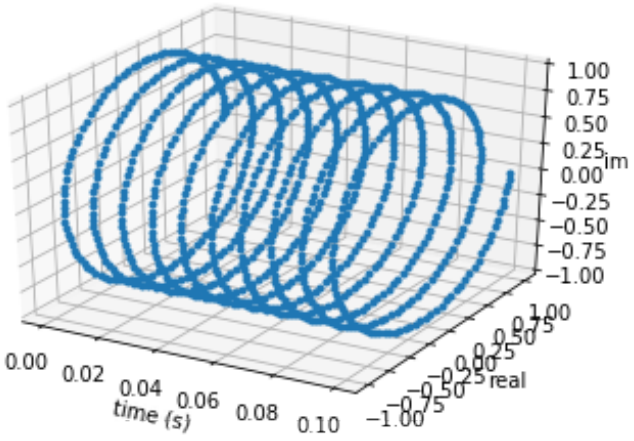


- ax.plot(theta, s, '.')
- ax = fig.add_subplot(224)
- ax.plot(theta, s, '.')
- fig = plt.figure()
- ax = fig.add_subplot(111)
- ax.plot(t[0:1000], s[0:1000], '.') # x축 y축 단위 같아야 한다. 각각 1000개씩
- ax.set_xlabel('time (s)')
- ax.set_ylabel('real')
- ipd.Audio(s, rate=sr)



4. Generate signal by complex-phasor

- c = np.exp(theta*1j) #np.exp가 exponential, e; 1j = i
- fig = plt.figure()
- ax = fig.add_subplot(111, projection = '3d')
- ax.plot(t[0:1000], c.real[0:1000], c.imag[0:1000], '.')
  # complex 대표하는 c라는 변수로 받음
  # 복소수는 a+bi니까 c값에 a, b 두개의 값이 나온다; a값이 real 이고 b가 imaginary
- ax.set_xlabel('time (s)')
- ax.set_ylabel('real')
- ax.set_zlabel('imag')



- ipd.Audio(c.imag, rate=sr)

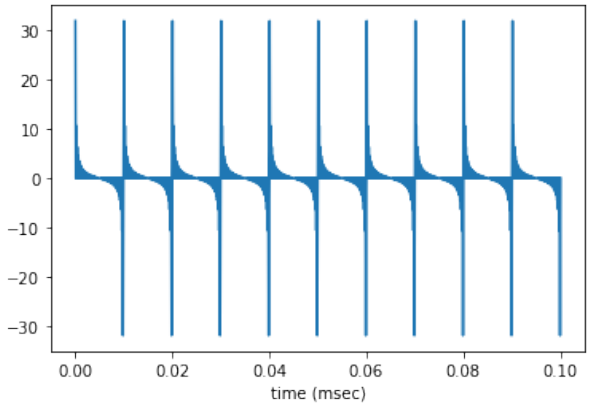
Week8 Summary

```
    s = np.sin(theta)*amp # 진폭 구현
    c = np.exp(theta*1j)
!pip install sounddevice
import sounddevice as sd
    sd.play(c.real, sr) # ipd.Audio(s, rate=sr) 와 같은 역할
```

 # sampling rate가 100Hz이면, frequency는 1, 2, ... 50까지 가능하지 100 이상은 불가능하다.
 # 주파수 표현은 주기를 돌아야 하니까, nyquist frequency = sampling rate/2 까지 가능하다.
 # CD 음질이 sampling rate = 44100Hz인 이유는 nyquist frequency = 22050Hz이고, 사람의 가청 주파수가 20000Hz이기 때문이다.

5. Generate pulse train

```
# generate samples, note conversion to float32 array
    F0 = 100; Fend = int(sr/2) #nyquist frequency
    s = np.zeros(len(t))
    for freq in range(F0, Fend+1, F0): # F0(100)부터 Fend(sr/2)까지 늘려가는데, F0(100)단계로
        theta = t*2*np.pi*freq
        tmp = amp * np.sin(theta)
        s = s + tmp
        print('tmp:', tmp)
        print('s: ', s)
  # 처음의 s값을 정의해줘야 한다. np.zeros(len(t)); sine wave를 계속 더하는 시작은 [0., 0.,…0.]
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(t[0:1000], s[0:1000]);
    ax.set_xlabel('time (msec)')
    ipd.Audio(s, rate=sr)
```




```
# sine wave 부드러웠던 곡선이 없어지고 선 하나가 남는다.
```

 # 고주파일수록 amplitude가 낮은 spectrum을 만들고, formant creation을 해준다.
 # 입술이 공명 역할을 해주는데, F1이 500이고, F2가 1500이면 입의 중간의 음을 낸다.

```
def hz2w(F, sr):
    NyFreq = sr/2;
    w = F/NyFreq *np.pi;
    return w
```

```
def resonance (srate, F, BW):
    a2 = np.exp(-hz2w(BW,srate))
    omega = F*2*np.pi/srate
    a1 = -2*np.sqrt(a2)*np.cos(omega)
```

```
    a = np.array([1, a1, a2])
    b = np.array([sum(a)])
    return a, b
```

```
RG = 0  # RG is the frequency of the Glottal Resonator
BWG = 100  # BWG is the bandwidth of the Glottal Resonator
a, b=resonance(sr,RG,BWG)
s = lfilter(b, a, s, axis=0)
ipd.Audio(s, rate=sr)
```

```
RG = 500  # RG is the frequency of the Glottal Resonator
BWG = 60  # BWG is the bandwidth of the Glottal Resonator
a, b=resonance(sr,RG,BWG)
s = lfilter(b, a, s, axis=0)
ipd.Audio(s, rate=sr)
```

```
RG = 1500  # 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)
ipd.Audio(s, rate=sr)
```

```
s = lfilter(np.array([1, -1]), np.array([1]), s)
ipd.Audio(s, rate=sr)
```

Week9 Summary

* Linear algebra

- 인공지능은 입력 벡터를 조작하여 출력 벡터를 산출하는 행렬의 곱으로, 많은 데이터를 통해 학습한 기계화된 과정을 말한다.
- 행렬의 차원은 m행 * n열이며, m*n 행렬이라고 표현한다.
- Vector = a sequence of numbers; 숫자 데이터 ~ 기하로 변환 가능
- Vector space requires all the linear combinations available in the space
    - c*v + d*w (v, w == 차원이 같은 두 개의 벡터/ c, d == 상수)
    - 여러 벡터가 만들어내는 공간(1~n차원 공간; 1~n개의 성분을 가진 벡터로 구성)
    - 1차원: line, 2차원: plane, 3차원: volume
- Column space requires all the linear combinations of m행
    - spanning: 각 column vector와 원점과 연결하여 만든 삼각형을 무한대로 확장하여 space를 채운다.
    - column space 차원은 whole space 차원을 넘어설 수 없다.
    - whole space == n행, column space == independent한 n행
    - col1과 col2의 관계가 linear == dependent ~ X spanning X whole space
- Null space
    - Column-wise whole space - Column space == left null space
    - Row-wise whole space - Row space == (right) null space
    - 어떠한 행렬을 영행렬로 만드는 모든 행렬

- Linear transformation(Ax = b)
    - 행렬 == 대문자, 벡터(1줄 행렬) == 소문자
    - A(인공지능)x(입력 벡터) = b(출력 벡터): A는 x 행렬의 차원을 바꿔줄 수 있다.

$$A=\underbrace{\begin{bmatrix}1 & 2\\ 2 & 4\\ 3 & 6\end{bmatrix}}_{\text{whole space (row)}}\ \text{whole space (column)}\quad A=\begin{bmatrix}1 & 3 & -2\\ 2 & 3 & 0\\ 4 & 1 & 5\end{bmatrix}\quad A=\begin{bmatrix}1 & 2 & -2\\ 2 & 4 & 0\\ 4 & 8 & 5\end{bmatrix}\quad A=\begin{bmatrix}1 & 1 & 2\\ 2 & 1 & 3\\ 4 & 1 & 5\end{bmatrix}$$

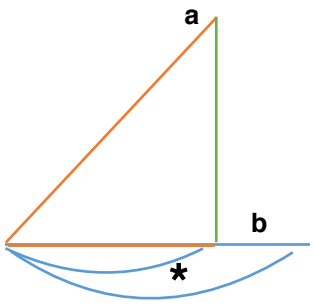
	(1)	(2)	(3)	(4)
column-wise whole space	R3	R3	R3	R3
column space	R1	R3	R2	R2
left null space	R2	None	R1	R1
row-wise whole space	R2	R3	R3	R3
row space	R2	R3	R3	R3
right null space	None	None	None	None

  - place x on original grid > transform x onto A grid > read b(transformed x on original grid)
  - 입력 row * 행렬 함수 column = 출력 row * column
- Detransformation = Inverse matrix($A^{-1}b = x$)
  - Ax = b
  - A역함수 * A * x = A 역함수 *b
  - $x = A^{-1}b$
  - transformation grid space == 0, dependent linear vector이므로 영행렬의 역함수가 없다.
- Eigenvector
  - Av = b
  - A transforms v to b

Week10 Summary

✱ Linear algebra

- Eigenvector
  - among all v, those being parallel to Av is eigenvector
  - 기본 matrix에 eigenvector(방향)이 2개가 있으면 eigenvalue도 2개가 있다.
  - 2개의 column vector를 또다른 2개의 eigenvector로 바꾼다.
- 85명의 국어 점수를 벡터화/ 영어 점수를 벡터화하면 85차원 each dimension for each individual
  - 원점, 국어 벡터, 영어 벡터를 점 찍으면 삼각형이 만들어진다.
  - cos(θ=각도값) = correlation coefficient
    - if cos(0) = 1 = r = 정비례 관계 : 두 벡터가 일직선 상 linear combinations
    - if cos(π/2=90) = 0 = 관계가 없다.
    - if cos(π=180) = -1 = 반비례 관계 : 두 벡터가 일직선 상 linear combination
- 차원에 상관없이 원점, (1, 2, 3) a vector, (4, 5, 6) b vector가 있을 때,
  - inner product: (1*4) + (2*5) + (3*6) = 32
  - a와 b 사이의 거리 == $|a|(\sqrt{1^2+2^2+3^2}) * \cos(\theta) * |b|(\sqrt{4^2+5^2+6^2})$
- b에 a가 project된 길이 * b = cos(θ=각도값) = correlation coefficient
- cosine similarity; a와 b가 얼마나 비슷한가를 수치적으로 이야기한다.



✱ **Wave**

- wave에는 큰 곡선과 작은 곡선의 합이 있다

- sine wave를 등간격으로 100Hz에서 10000Hz까지 만들고(phasor) **inner product**한다.
    - 같은 성분에만 response를 준다 ~ 값이 크게 나온다.
    - frequency가 다르면 0이 더 많으니까 합한 값이 적다.
    - frequency가 같으면 1이 많으니까 합한 값이 크다.
    - 같은 frequency의 cosine wave는 sine wave를 π/2만큼 옮긴 값이니(phase shift), 곱한 값의 합은 0이 된다.
- 같은 성분에만 response를 줘야하는데 같은 성분인데 조금 위치가 달라졌다고 해서 다른 성분으로 판단하면 안된다.
    - inner product가 0이 되려면, a wave와 b wave의 각도가 90도가 되어야 한다.
    - vector space에서의 90도와 2차원에서의 90도는 같은데 왜냐; slice하면 다 평면이기 때문이지; 세 점을 연결하면 다 면일 뿐이다.
    - 한 wave 내에서 어떤 주파수 성분이 많은가를 알아내려고 한다.
    - phase에 민감하게 작용하지 않도록 하기 위해서, complex phasor를 쓴다.
- complex value도 하나의 값이니 n개로 만들어서 inner product한다.

Week 11 Summary

✱ Fourier transform

- 분석하고자 하는 target wave, s, 벡터가 몇 개인지, 샘플 갯수가 몇 개인지

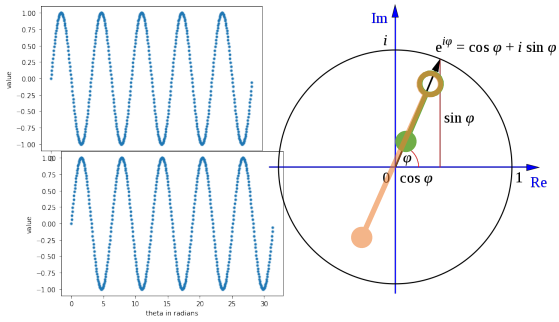
```
nFFT = nSamp                                  #샘플 갯수를 nFFT에 할당한다.
amp = [];
for n in range(0,nFFT):                       #샘플 갯수 만큼 for루프를 돈다
    omega = 2*np.pi*n/nFFT # angular velocity 1. omega = 2π * 0 / 100
                                              [0…2π]θi=1바퀴/ [  ]안 샘플 갯수 100
                                              2. omega = 2π*1/ 00=2바퀴/ [ ]안의 샘플 갯수 100
                                              3. omega = 2π * 2 / 100 = 4
    z = np.exp(omega*1j)**(np.arange(0,nSamp)) z는 complex wave를 바퀴 수 만큼 만든다.
    amp.append(np.abs(np.dot(s,z)))           루프의 갯수만큼 amp 크기가 정해진다.
                                              amp에 허수가 들어갈 수 없다. absolute를 해버렸다.

fig = plt.figure()
ax = fig.add_subplot(111)
freq = np.arange(1,nFFT+1)*sr/nFFT;           총 bar의 갯수는 샘플 갯수와 같다.
ax.plot(freq, amp)                            좌우 대칭이기 때문에 half만 의미가 있다.
ax.set_xlabel('frequency (Hz)')               spectroanalysis에 있어서 half = spectrum
ax.set_ylabel('amplitude')
```

- 각 frequency성분이 probing을 하면서 inner product에서 끄집어내는 absolute값이다.
- 2000Hz의 성분이 얼만큼 있는가
- resonance를 안 쓰면 flat ~ spectrum은 equalizer
- spectogram의 하나의 slice가 spectrum

[2019-2 영어음성학]

이름(학번): 민지우(2016130575)



-cos(θ=각도값) = r=1 이면, a 방향대로 b가 간다.

-한 점이 subject인 그래프/ 한 axis가 subject인 그래프

-일직선에 위치할수록 r값이 1에 가깝다. 직선에서 왼쪽이면 r이 마이너스, 오른쪽이면 플러스이다.

-데이터를 그래프로 그렸을 때, 일직선으로 표현되는데, 각도가 플러스면 θ=각도값이 0, 360도, 마이너스면 180도이다.

-sine wave를 180도 phase change했을 때, cosθ = -1

-amplitude(진폭)이 달라도 상관없다.

✱ Preprocessing signal

- complex number로 dot product가 나오는데, 진한 색은 1보다 큰 값이고, 연한 색은 0.1보다 작은 값

```
max_freq = None  #cutoff freq
win_size = 0.008 #sec                                    #0.008초씩 이동
win_step = 0.001  #sec
win_type = 'hanning' #options: 'rect', 'hamming', 'hanning', 'kaiser', 'blackman'
nfft = 1024
#Emphasize signal
s = preemphasis(s)
#Frame signal
frames = frame_signal(s, sr, win_size, win_step)
#Apply window function
frames *= get_window(win_size, sr, win_type)
print('frames:', frames.shape)
magspec = np.abs(np.fft.rfft(frames, n=nfft)) # frames x (nfft//2 + 1)
plot_spectrogram(magspec)
powspec = 1/nfft * (magspec**2)     # 제곱을 하면 진한 부분은 더 커지고, 연한 부분은 더 작아진다.
plot_spectrogram(powspec)
logspec = 10 * np.log10(magspec) # dB scale; base가 10이 되는 log를 치면 자잘한 값을 떨어뜨릴 수 있다.
plot_spectrogram(logspec)            # 우리가 다룰 수 있는 범위; 10^-4 = -4 처럼 간단해진다.
```