<영어음성학 Summary>

<Praat>

- Sampling Frequency는 음질이 얼마나 좋게 하느냐를 결정한다. 음질을 좋게 하려면 수치를 높게 하고, 음질이 좋지 않더라도 파일 크기를 작게 하려면 수치를 낮게 한다.
- 파이썬과 같은 코딩을 통해 praat과 같은 프로그램을 만든다.
- x축은 시간을 의미. 등간격의 시간에 해당하는 수치가 있고 이들을 연결한 것이 소리. 소리는 숫자이다. 시간의 간격을 조절을 할 수 있는데 이에 해당하는 것이 Sampling Frequency이다. 예를 들어, 44100Hz라는 것은 1초를 44100으로 나눈 것을 의미한다.
- Edit에 cut를 누르면 원하는 부분을 자를 수 있다.
- 파란선은 pitch를 뜻하고, 여자이면 높은 pitch 값을, 남자이면 낮은 pitch 값을 갖는다. 노란선은 intensity를 뜻하고, 강도를 의미한다. Pitch가 높으면서 intensity가 낮을 수 있고, pitch가 낮으면서 intensity가 강할 수 있다.
- 아래에 있는 것을 소리의 스펙트럼이라고 한다.

<phonetics>

- English vowels들은 Monophthongs/ Dipthongs로 구분할 수 있다.
- phonology 음운론. Phonetics는 음성학. Phonology는 우리 속에 이루어지는 abstract한 과정. 음성학은 physical하다. 이 수업은 physical한 것에 초점을 맞추는 수업.
- Phonetics는 articulatory, acoustic, auditory phonetics로 나뉠 수 있다.
- Phonotation process를 담당하는 larynx는 voicebox라고도 불리는데, 여기서 성대 닫혀 있으면 voiced가 되고, 열려 있으면 voiceless가 된다.
- Oro-nasal process는 velum이 lowered 혹은 raised 되었는가에 따른다. Velum이 lowered 되면 m, n, ng, 모음과 같은 소리가 나고 나머지 음들은 velum이 raised되어 nasal tract이 막혀있게 된다.
- Articulatory process를 담당하는 부위는(constrictor) lips, tongue tip, tongue body가 있다. 각각의 constriction이 위치는 어디서 나는지(CL), 어느 정도로 나는지(CD)를 구분해야 한다. CL에 따라 음들을 lips, tongue body, tongue lips로 구분할 수 있고, CD에 따라 소리들을 constriction이 강하게 일어나는 순서대로 stops, fricatives, approximants, vowels 순으로 구분할 수 있다.

<Spectrum>

- Sine wave를 결정하는 것은 frequency와 magnitude. X축은 시간, y축은 수치 값(value)를 뜻한다. Sine wave를 x축은 frequency, y축은 amplitude로 나타낸 그래프로 변형할 수 있다.
- 모든 소리 signal들은 여러 간단한 sine wave의 합으로 나태낼 수 있다. 우리가 일상에서 듣는 모든 소리는 complex tone이다. Complex sound의 주기는 frequency가 가장 낮은 wave의 주기와 일치한다.
- Sine wave로 분해하는 과정을 analysis라고 하고, 합치는 과정을 synthesize(합성)이라고 한다.

<Source and Filter>

- Larynx에서 진동하는 것을 source라고 하고, 그 이후의 입 구조를 거치는 과정을 filter라고 한다. 똑 같은 soruce라고 하더라도 어떠한 filter 과정을 거치느냐에 따라 다른 소리가 난다.
- Source의 패턴을 보면 frequency가 배로 늘어날수록 일정하게 decreasing한다.
- 가장 첫번째 주파수를 f0 혹은 fundamental frequency라고 한다. F0에다가 X2, X3 배수하는 것을 harmonics라고 한다.
- Praat의 아래쪽에 볼 수 있는 것이 Spectrogram이라고 한다. x축은 시간, y축은 frequency. 까맣게 되어 있을수록 크기가 세다.
- 소리가 filtered를 거치게 되면 amplitude의 일정한 구조가 깨지고 지그재그처럼 된다. 하지만 배음의 구조는 여전히 유지한다.
- 주파수가 선호하는 곳을 formant, 산맥이라고 부르고 선호하지 않는 곳을 valley라고 부른다.
- Pure tone을 여러 개 만드는데, 주파수는 x2, x3 배수로 늘려가고 amplitude는 조금씩 낮춰가면서 만든 다음 synthesize하게 되면 complex tone이 만들어지게 된다. Combine to stereo는 합하기 전 병렬한 상태이고, mono는 하나로 합친 상태이다
- 가장 첫번째 formant를 f1, 두번째 formant를 f2라고 하고, spectrogram에 나타난 f1, f2의 위치를 확인할 수 있다. 그리고 각각의 영어 모음을 발음했을 때 f1과 f2의 위치가 각각의 모음에 따라 다르다. F1과 f2의 수치를 각각 y축과 x에 위치 시켜놓는다면 이는 바로 모음을 발음할 때 우리 혀의 위치를 결정하게 된다. F1은 혀의 높낮이, f2는 혀의 앞뒤 위치를 결정한다.

<Python>

- 변수와 정보

- 모든 컴퓨터 언어는 단어와 문법으로 이루어져있다.
- 정보를 담는 단어에 해당하는 것이 변수(variable)이다. 변수라는 그릇에다가 정보를 담는(assign)다. 숫자, 문자 등을 담을 수 있다.
- 모든 컴퓨터 문법은 if로 시작하는 Conditioning(~하면 ~해라)이 필요하다. If conditioning
- 여러 번 반복을 실행하는 for문법이 필요하다. Ex) 10번 반복해라
- 어떤 것을 입력하면 내가 원하는 것이 나오는, 내부적으로는 복잡한 함수. 입력과 출력을 패키징하는 것을 함수라고 한다. Ex) 자동차라는 함수. 나의 움직임은 입력이 되고 차가 움직이는 것은 출력.
- = 표시는 절대 같다는 의미가 아니라 오른쪽에 있는 정보를 왼쪽 variable에다가 담는다는 의미.
- Variable에 들어가는 정보의 종류는 숫자와 문자.
- 왼쪽을 클릭하고 a를 클릭하면 윗칸에다 셀 하나가 추가되고, b를 클릭하면 밑칸에다 셀 하나가 추가된다. X를 누르면 셀이 삭제된다.
- 쉬프트 + 엔터는 실행(running)의 단축키
- 여러 셀에 적을 필요 없이 세미콜론';'을 붙여 한 셀에다가 붙여 써도 된다. 혹은 한 셀에다가 엔터를 치는 것도 가능하다.
- 그런데 결과를 보여달라고 할 때는 마지막 줄에 적은 결과만 보여준다. 예를 들어
  a = 1; b = 2
  a
  a+b 라고 했으면 a+b 값만 보여준다.
- variable에다가 하나의 정보를 assign한 다음에, 그 밑에다가 똑같은 variable에 다른 정보를 넣으면 그 정보로 replace 된다. Ex) a = 1을 assign하고 다음 cell에다가 a = 2라고 assign을 하면 a에는 2가 들어가게 된다. 그 후에 a = 1을 다시 run하게 되면 a에는 1이 들어가게 된다.
- 영어 알파벳을 그냥 쓰게 되면 그것은 무조건 변수를 의미하게 된다. 따라서 b = love 를 입력하면 오류가 난다. 만약 love = 2라고 입력하면 love라는 variable에 2를 넣는다는 것을 의미한다. 문자는 반드시 ' 혹은 " / single quote, double quote 안에 입력해야 한다.
- variable에다가 variable을 넣는 것도 가능하다. Ex) love = 2; b = love; print(b) 하면 2가 나

온다.

- A = 1, type(a)라고 하면 int라는 결과가, print(type(a)) 라고 하면 <class 'int'>라는 결과가 나온다.
- #을 붙이면 뒤에 무엇을 쓰더라도 실행이 되지 않기 때문에 노트를 남길 수 있다. 혹은 셀의 type을 code에서 markdown으로 변경하면 그 셀을 주석처럼 사용할 수 있다.

- List, dictionary, tuple
  - a라는 변수에 여러 정보를 담으려면 대괄호 [ ]안에다가 다 넣으면 된다. list 안에는 문자, 숫자, 리스트가 들어갈 수 있다. 각각의 사이사이에는 콤마로 구분한다.
  - 대괄호[ ] list대신 괄호( )를 써도 되는데 이것의 타입은 tuple이다. Tuple은 list는 동일하지만 tuple이 보안에 더 강하다
  - 중괄호{ }는 dictionary이다. 여러 개의 정보는 콤마로 구분, 표제어와 설명으로 구성, 표제어와 설명 사이에는 :(페어, 콜론)로 구분, ex) a = {'a': 'apple', 'b': 'banana'}

- 함수
  - Print도 누가 만들어 놓은 함수이다. 좋고 유용한 함수들을 이미 만들어 놓은 것이 아나콘다, 파이썬이다. 함수를 쓰는 법은 괄호에다가 변수들을 입력 해주면 된다. Print(), print(a) 라고 해도 되지만 그냥 셀의 마지막에 a라고 하면(꼭 마지막에 해야 됨) 변수 a를 print out해준다.
  - Type()은 괄호 안에 들어가는 variable이 어떠한 유형인가를 밝히는 함수. 안에 들어가는 변수가 리스트인지, 숫자인지 문자인지 변수의 타입을 알려준다. Int(숫자), list(리스트), str(스트링, 문자), Float(실수. Ex1.2), dict(딕셔너리)
  - Float는 variable을 float로 바꾸어 주는 함수.
    ex) a = 1.2; a = int(a); print(a)
    a는 float(소수점 숫자)였는데 int함수 하니까 소수점이 날라가서 1이 된다.
    Ex) a = 1; a = float(a); print(type(a)); <class 'float'>.
    Variable a는 int였는데 이것을 float로 바꾸어주고 print하니까 float가 나온다.
  - Int는 variable을 int로 바꾸어주는 함수.
    ex) a = 1; a = float(a); print(a)
    a는 int(소수점 없는 숫자)였는데 float함수 하니까 소수점이 붙어서 1.0이 된다.
    Ex) a = 1.2; a = int(a); print(type(a)).
    Variable a는 float였는데 이것을 int로 바꾸어주고 print하니까 int가 나온다.

- A=list(a) variable a를 리스트로 만들어라
  a = '123'; a=list(a); a
  ['1', '2', '3'], 리스트 안에 포함된 것이 문자인 것에 주목.

- List에서 정보 access하기
  - Ex) a = [1, 2] ; b = [3, 4] ; c = a[0]+b[0] ; c 결과는 4가 나온다. 왜냐하면 a[0]이라는 것은 variable a에서 첫 번째 정보를 가져오라는 것을 의미. 이것을 list에 access한다고 한다. a 중에서 어떠한 정보만 가져오라는 의미.
  - a = '123'; print(type(a)); print(a[1])
    a = [123, 124] ; print(type(a)); print(a[1])
    a = 123; print(type(a)); print(a[1])
    가장 밑에 것은 에러가 나는데 왜냐하면 int에서 몇 번째 것을 가져오는 것은 불가능
  - a = '123'; a = list(a); print(type(a)); print(a); print(a[2])
    str이었던 variable a를 list로 만들기
  - a = [1,'2', [3, '4']]; print(type(a)); print(a[0]); print(a[1]); print(a[2])
    list안에 list가 있는 형태. 첫 번째 정보는 숫자. 두 번째는 문자, 세 번째는 리스트에 access한다.
  - a ={"a": "apple", "b": "orange", "c": 2014} print(type(a)) print(a["a"])
    dict의 정보에 access할 때는 페어":"에서 앞부분을 index로 사용한다. a["a"]
    a ={1: "apple", 2: "orange", 3: 2014} print(type(a)) print(a[1])
    표제어에 "a"와 같이 str말고도, 1, 2, 3과 같은 int가 들어가도 무관하다. 표제어가 1이기 때문에 똑같이 apple이 뜬다.
    a = {'a': [1, 2, 3], 'b': 'orange', 'c': 2014}
    type(a['a'])는 list이고 type(a['b'])는 str이고 type(a['c'])는 int이다
  - a=[(1,2,3), (3,8,0)] a[0] type((a[0]))
    a의 0번째 것은 (1,2,3)이 나올 것이고, a의 0번째 것의 type은 tuple이 된다.
  - s = 'abcdef' print(s[0], s[5], s[-1], s[-6]) print(s[1:3], s[1:], s[:3], s[:])
    <class 'str'>, a f f a, bc bcdef abc abcdef
  - n = [100, 200, 300] print(n[0], n[2], n[-1], n[-3]) print(n[1:2], n[1:], n[:2], n[:])
    100 300 300 100 [200] [200,300] [100, 200] [100, 200, 300]

- len(s) length 6
  len함수는 str에서도 적용된다. Ex) a = 'abcdef'; len(a)는 6이다
- s[1]+s[3]+s[4:]*10
  bdefefefefefefefefef
- s.upper()
  'ABCDEF'
- a = s.upper()
  a
  a에는 'ABCDEF'가 들어가게 된다.

- String
  - s = ' this is a house built this year.₩n'
  - result = s.find('house') result 11
  - result = s.find('this') result 1
    this가 두 개 있는데 이 함수는 첫 번째 this를 찾아줌
  - result = s.rindex('this') 23
  - s = s.strip() s 'this is a house built this year'
    스페이스 같은 잡스러운 것을 없애고 순수한 text만 남겨주는 기능
  - tokens = s.split(' ') tokens
    일단 s에 들어가 있는 문장은 바로 바로 위의 strip을 거친 문장이다. 스페이스를 기준으로 split을 해주는 기능.
    ['this', 'is', 'a', 'house', 'built', 'this', 'year']
  - s = ' '.join(tokens) s
    'this is a house built this year.'
    cf) s = ','.join(tokens) s
    'this,is,a,house,built,this,year'
  - s = s.replace('this', 'that')
    'that is a house built that year.'
    모든 this를 that으로 바꾸는 것.

- for loop, if conditioning

  - a = [1, 2, 3, 4]
    for i in a:
        print(i)
    a에 있는 것을 루프를 하나씩 돌려서 i에 넣어라

  - a = [1, 2, 3, 4]
    for i in range(4):
        print(a[i])
    range함수는 0부터 뒤에 있는 숫자 직전까지 루프를 만들어 준다.

  - a = [1, 2, 3, 4]
    for i in range(len(a)):
        print(a[i])
    len(a)는 4가 되니까 range(4)가 되고 이는 0부터 3까지 루프가 돌게 된다.

  - a = ['red', 'green', 'blue', 'purple']
    for i in a:
        print(i)
    각각의 str이 루프로 print out된다.

  - a = ["red", "green", "blue", "purple"]
    b = [0.2, 0.3, 0.1, 0.4]
    for i, s in enumerate(a):
        print("{}: {}%".format(s, b[i]*100))
    enumerate함수는 i에는 0, 1, 2, 3가 순차적으로 루프에 들어가고, s에는 red, green, blue, purple이 순차적으로 루프에 들어가게 된다.

  - a = ["red", "green", "blue", "purple"]
    b = [0.2, 0.3, 0.1, 0.4]
    for p, q in zip(a, b):
        print("{}: {}%".format(p, q*100))
    p, q에 a, b에 담긴 것을 순차적으로 루프에 넣는다.

  - a = 1
    if a == 0:
        print(a)
    else:
        print(a+1)

- for i in range(1, 3):

    for j in range(3, 5):

        print(i*j)

  i에 1부터 2까지 넣고, j에 3부터 4까지 넣는다.i의 첫번째 루프에 1이 들어갈 때 j의 첫번째 두번째 루프가 3, 4에 해당하고 i의 두번째 루프에 2가 들어갈 때 j의 첫번째 두번째 루프가 3,4에 해당하니 값은 3,4,6,8이 나온다.

- for i in range(1, 3):

    for j in range(3, 5):

        if j >=4:

            print(i*j)

  j가 4보다 크거나 같으면 print out 한다. 값은 4, 8이 나온다.

- Range(4)는 range(0, 4)와 동일하고 range(1, 3)은 1부터 2까지 루프에 넣어라는 말과 동일하다.

<Numpy>

- Numpy 안에는 여러 가지 함수들이 담겨 있다. 그 함수들을 불러 오는 방법

numpy.A.D.f : numpy 안에 있는 A 안에 있는 D 안에 있는 함수 f를 불러와라

from numpy import A : 이것을 입력하면 그 이후로는 numpy를 입력하지 않고 A부터 입력하면 된다.

import numpy as np : 앞으로 np로 줄여 쓰겠다

import matplotlib.pyplot as plt : matplotlib 안에 있는 pyplot을 plt로 줄여 쓰겠다

from matplotlib import pyplot as plt 라고 해도 앞으로 plt만 써도 된다

- np.empty([2,3], dtype='int') : ()가 있는 것으로 보아 empty는 함수. 함수 중에서도 np 안에 어느 카테고리에도 포함되지 않은 함수. 2행 3열의 행렬을 만든다. 안에 들어가는 숫자의 data type은 int에 해당하고 랜덤하게 정해진다.

np.zeros([2,3]) : zeros라는 함수. 0으로 다 채워진 2행 3열 matrix를 만든다.

2행 3열 matrix를 직접 입력하여 만들 수도 있다 : [[0,0,0], [0,0,0]]. 하지만 이 list는 아무 쓸모가 없다.

np.array([[0,0,0], [0,0,0]]) : 위에 직접 만든 list 행렬을 array로 만들어준다. 그러면 zeros에서 만든 행렬과 똑같은 것이 만들어진다.

np.ones([2,3]) : 1로 채워진 2행 3열 array 행렬을 만든다.

zeros, ones 함수를 쓰면 숫자들 뒤에 .이 붙어 있는데 이것은 이 함수들의 default가 float이기 때문이다. np.ones([2,3], dtype = int)라고 입력하며 숫자들 뒤에 .이 없어진다. Float64를 int 대신 입력하면 .이 생긴다. float에도 종류가 있다. 소수점 몇 번째까지 표시할 것인가.

np.arange(0,10,2, dtype='float64') : 0부터 10직전까지. Increment는 2로 증가. Datatype은 float64로.

np.linspace(0,10,6, dtype=float) : linearspace의 준말. 0부터 10까지(10포함). 그 처음부터 끝까지 구간을 6개로 나누어준다. 1번째에서 2번째, 2번째에서 3번째(계속 반복해서)로 가는 구간 차이가 동일하다.

X = np.array([[1,2,3],[4,5,6]]) / X : list를 numpy의 형태의 데이터인 array로 만들어 준 다음 X에 대입한다. 그냥 list는 계산이 안되지만 array는 계산이 가능하다.

[[1,2,3],[4,5,6]]은 대괄호가 2개이고 직사각형 이차원 행렬에 해당한다. [[[1,2,3],[4,5,6]], [[1,2,3],[4,5,6]]]은 대괄호가 3개이고, 2차원 행렬이 두 개인 직육면체 삼차원 행렬에 해당한다.

X.ndim을 입력하면 X가 몇 차원인지 표시된다.

X에다 [[[1,2,3],[4,5,6]], [[1,2,3],[4,5,6]]]를 대입한 상황에서 X.shape를 입력하면 (2, 2, 3)가 나오는데 이는 '가장 큰 괄호 안에 2개가 있고 그 다음 괄호 안에 2개가 있고 그 다음 괄호 안에 숫자 3개가 들어 있다'를 의미한다.

X.dtype : X 안에 들어 있는 숫자의 datatype이 알고 싶다면 X.dtype을 입력하면 된다.

X.astype(np.float64) : X 안에 들어 있는 숫자의 datatype을 괄호 안에 것으로 바꾸어라.

np.zeros_like(X) : X 안에 들어 있는 숫자를 모두 0으로 바꾸어라. X는 계산이 가능하기 때문에 X*0을 입력하더라도 같은 결과를 얻는다.

- data = np.random.normal(0,1, 100) : np라는 라이브러리 안에 들어 있는 random이라는 subpackage에 들어 있는 normal이라는 함수를 사용한다. normal이라는 함수는 정규분포를 만들어 주는 함수. 정규분포를 위해 필요한 것은 평균/표준편차. 0은 평균이고 1은 표준편차. 100은 데이터 개수. Data는 random하게 가져와라. Data.ndim 값은 1 왜냐하면 괄호가 하나 있으니까. Data.shape은 (100,,)가 나온다.

print(data)

plt.hist(data, bins=10) : bins는 x축 값에 들어갈 수치가 10개라는 것을 의미. plt안에 있는 hist라는 함수를 사용한다. 나타나는 그림이 히스토그램. y값에 들어가는 것은 x값에 해당하는 데이터들의 개수. 따라서 y값은 항상 자연수.

plt.show()

<Manipulation>

- X = np.ones([2, 3, 4]) : 3차원에 3행 4열인 행렬을 만들어라. 들어가는 숫자는 모두 1.

Y = X.reshape(-1, 3, 2) : X의 형태를 reshape하는 함수. 그런데 reshape을 하더라도 들어가는 데이터의 숫자는 같아야 한다. X의 데이터 개수는 2x3x4=24. 그렇다면 -1대신에 4가 들어가야 하는데, -1은 안에 무엇인 들어가야 할지 모를 때 적는다. -1 대신 4를 적어도 무관하다.

np.allclose(X.reshape(-1, 3, 2), Y) : X를 reshape한 것과 Y가 동일한가.

밑에 assert는 생략

<Numpy I/O>

- a = np.random.randint(0, 10, [2, 3]) : 0부터 10직전까지 숫자들을 random한 방식으로 골라서 2행 3열의 matrix를 만들어라. 실행할 때마다 들어가는 숫자가 달라짐.

b = np.random.random([2, 3]) : random한 숫자들을 가지고 2행 3열 matrix를 만들어라.

np.savez("test", a, b) : a와 b라는 variable을 실제 file로 저장을 해준다.

!ls -al test* : 저장이 되었는지 확인해 준다. 실제로 탐색기에 검색해보면 test.npz라는 파일이 저장되어 있음

del a, b : variable a, b를 삭제한다

who : available한 variable을 보여준다.

npzfiles = np.load("test.npz") / npzfiles.files : test.npz파일에 담겨 있는 variable a, b을 npzfiles라는 새로운 variable에 담는다. 그러면 npzfiles에는 ['arr_0', 'arr_1']라는 결과가 나오는데 각각이 a, b를 나타낸다.

npzfiles['arr_0'] : variable a에 담겨 있는 것을 보여준다.

data = np.loadtxt("regression.csv", delimiter=",", skiprows=1, dtype={'names':("X", "Y"), 'formats':('f', 'f')}) : regression.csv라는 textfile을 data라는 variable에다가 넣겠다. csv file이라는 것은 comma separated value. Delimiter는 분리하는 것을 ','로 하라는 것. Skiprows라는 것은 첫번째 줄에 담겨 있는 X, Y는 생략해라. Dtype은 안에 들어가 있는 data의 type은 X, Y라고 하자. Format은 두 개다 float로 하겠다.

np.savetxt("regression_saved.csv", data, delimiter=",") : data라는 variable을 앞에 있는 이름처럼 저장을 하겠다.

!ls -al regression_saved.csv : 만들어진 것을 확인.

<Inspecting>

arr = np.random.random([5,2,3]) : 3차원의 행렬을 random하게 만들어서 arr이라는 variable에다 넣어라

print(type(arr)) : <class 'numpy.ndarray'>

print(len(arr)) : 5

print(arr.shape) : (5, 2, 3)

print(arr.ndim) : 3

print(arr.size) : 30

print(arr.dtype) : float64

<Arithmetic>

a = np.arange(1, 5) : 1에서부터 5 직전까지 array를 만들어라

b = np.arange(9, 5, -1) : 9에서부터 5직전까지 하나씩 감소하는 array를 만들어라

print(a - b) : [-8 -6 -4 -2]

print(a * b) : [9 16 21 24]

나머지 생략

<Comparison>

a = np.arange(1, 10).reshape(3,3) : 1부터 9까지 1차원 행렬을 3X3 2차원 행렬로 만들어라

b = np.arange(9, 0, -1).reshape(3,3) : 9부터 1까지 1차원 행렬을 3X3 2차원 행렬로 만들어라

a == b : equal 기호가 두개 이면 "같다"를 뜻한다. 행렬 안의 data들이 같냐 다르냐를 뜻함. A와 b의 dimension과 shape이 같아야 실행할 수 있는 기능임.

array([[False, False, False],

[False, True, False],

[False, False, False]])

a > b : a가 b보다 크냐. 마찬가지로 a와 b의 dimension과 shape이 같아야 실행할 수 있음.

array([[False, False, False],

[False, False, True],

[True, True, True]])

a.sum() : 행렬a에 들어 있는 데이터를 모두 더해라.

Np.sum(a) : 위에 것을 쓰는 다른 방식. A는 이미 numpy가 만들어낸 산물이기 때문에 굳이 np를 붙이지 않아도 되지만 이렇게 써도 된다

a.sum(axis=0) : 첫 번째 차원에서 열을 다 합쳐라

a.sum(axis=1) : 두 번째 차원에서 행을 다 합쳐라

<Broadcasting>

a = np.arange(1, 25).reshape(4, 6) : 1부터 25직전까지 숫자로 array를 만들고 그거를 2차원의 4X6의 행렬로 reshape을 해라.

a + 100 : 행렬 a의 각 데이터에 100 씩을 더해라

b = np.arange(6) : 0부터 6직전까지 숫자로 구성된 array를 만들어라

a+b = a와 b를 더해라

<Phasor>

- sin, cos, 오일러 공식과 같은 것을 phasor라고 한다.
- 오일러 공식: $e^{\theta i}=\cos\theta+i*\sin\theta$ e는 파이처럼 무리수. 제곱해서 -1이 되는 허수 i. $e^{\theta i}$는 θ에 어떠한 수가 들어가든 결국 복소수에 포함되니까 e^{θ} x=a+bi로 표현될 수 있다. 입력값은 θ에는 반드시 radian 값이 들어가야 한다. Degree가 아니라.
  θ 에 0이 들어가면 1.
  θ 에 π/2가 들어가면 cos(π/2)+sin(π/2) i=i
  θ 에 π가 들어가면 cos(π)+sin(π) i=-1
  θ 에 3π/2가 들어가면 cos(3π/2)+sin(3π/2) i=-i
  θ 에 2π가 들어가면 cos(2π)+sin(2π) i=1
- 이게 계속 반복될 것임. 그런데 이 output들은 실수가 아니기 때문에 표현될 수 없다. 복소수를 plot하는 법을 배워야 한다.
- 그래서 나온 것이 complex plane(복소수 평면) x축에는 a, y축에는 b. 위의 값들을 복소수 평면에 옮겨 보자. 그러면 하나의 원이 그려져서 반복되는 것을 알 수 있다.
- Projection. x축을 projection한다는 것은 x 축 위에서 내려다 본다고 하는 것. 그러면 radian 값에 따라 수치가 x축으로 왼쪽 오른쪽으로만 이동한다. y축을 projection하면 y축에서만 위아래로 이동할 것이다. X축 projection으로 a만 보는 것은 a+bi에 따르면 실수 부분만 보고 허수 부분은 보지 않겠다는 것. 그리고 x축 projection은 cos그래프와 일치한다(1에서 시작해서 0, -1로 이동) y축 projection은 sin그래프와 일치한다(0에서 시작해서 1, 0, -1 순으로 이동)
- 그런데 위의 phasor에서는 frequency의 개념(1초에 몇 번 들어가느냐)이 들어가지 않아서 소리의 실체를 가질 수 없다. 소리에는 반드시 시간의 개념이 들어가야 한다. 결국 나중에 sin 안에 값을 넣는 과정이 있는데 여기에는 반드시 theta(radian) 값이 들어가야 한다. 근데 그냥 radian에는 시간 개념이 없다. 그래서 t를 먼저 설정하고 그 t와 theta를 연동해야 한다.

- from matplotlib import pyplot as plt
  (문제) 위의 것과 같은 역할을 할 수 있는 문장은? Import matplotlib.pyplot 똑같은 것임.
  from mpl_toolkits.mplot3d import Axes3D
  3D plot을 위해서.
  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

<시간 개념 도입>

- Amp = 1은 진폭이 위아래로 1, -1 값을 갖도록. Sampling rate : 1초를 몇 개의 수치로 표시하겠다. SR=10000이라면 1초를 1만개의 수치로 표시하겠다는 의미. 즉 SR은 해상도 역할을 한다고 볼 수 있다. Dur(duration) 몇 초 동안 소리가 유지될 것인가를 결정. 이 모든 것들을 parameter라고 부른다. 이렇게 parameter를 variable로 설정하는 이유는 왜냐하면 amp나 freq를 바꿀 때 이렇게 설정해놓으면 variable은 내버려 두고 구체적인 수치만 바꾸면 되니까.

- t = np.arange(1, sr * dur+1)/sr
  절차적으로 먼저 time을 만들어야 한다. 0.0001초 0.0002초 0.0003초 … 0.5000초(duration=0.5이니까) 이렇게 우리가 필요한 시간을 일일이 입력해서 만들 수도 있다. 이렇게 하여 1만개의 SR을 만들 수 있다.
  그런데 이렇게 손으로 쳐가면서 일일이 만들 수 없다. 그래서 위와 같은 방식을 이용한다. (1, sr * dur+1)에서 1부터 5천까지 만들어지고 그것을 sr로 나누면 위와 같은 시간을 만들어 낼 수 있다
  t 값을 실제로 입력해서 알아보면, 1.000e-04, 2.000e-04 … 5.000e-01 이렇게 나오는데 e-04라는 것은 10의 4승/1을 의미하는 것이다.

- theta = t * 2*np.pi * freq
  위에서 time을 만들었다. 그리고 이 time과 theta를 연동하자. Time 없이 theta를 만들면(아래 응용 과정) 소리의 실체를 갖추지 못하는 것. 2*np.pi는 2파이를 뜻한다. 위에서 만든 time에다가 2파이를 곱한다. 그래야 sin함수 안에 넣을 radian 값이 만들어진다. 그런데 우리는 한 바퀴만 만들자고 하는 것이 아니기 때문에 freq를 곱해주어야 한다.
  (문제) time의 벡터 개수와 theta의 벡터 개수는 같은가? 같다.

- s = np.sin(theta)
  사실 그대로 두면 amp를 1로 default 설정하는 것이다. 그러면 s값은 항상 -1에서 1사이의 값을 가질 것이다. s = amp*np.sin(theta)라고 하고 parameter에서 amp값을 변경해주면 진폭이

달라진다. 예를 들어 amp=2로 설정하면 s값은 -2에서 2사이의 값을 가질 것이다.

- fig = plt.figure()
  figure는 화면 전체를 말한다.
  ax = fig.add_subplot(111)
  subplot은 그 화면 전체를 분리해서 바둑판을 만들 수 있다. 바둑판을 한 figure에 여러 개 만들 수 있다. Subplot 안에 있는 111이라는 것은 바둑판을 1행 1열의 개수(1개)로 만들고(2행 2열이면 subplot은 4개가 된다) 그 중에 첫 번째 것을 의미한다. 224라면 2행 2열의 개수(4개)로 바둑판을 만들고 그 중에 4번째 것을 의미한다.
  ax.plot(t[0:1000], s[0:1000], '.')
  t[0:1000], s[0:1000]는 처음부터 천 개의 수치값만 plotting하라는 의미. 그래서 당연히 안에 들어가는 수치값의 개수가 앞과 뒤에 것이 1000개로 같아야 한다. '.'는 plotting을 '.'으로 표시하라는 의미. '.'대신에 –를 넣으면 line으로 형성. T가 x축에 오고 s가 y축으로 입력된다. 물론 x축에 theta 값을 입력할 수도 있겠지만 한바퀴 돌면 2pi라는 것을 뻔히 알고 있는데다 우리가 관심 있는 것은 시간이라는 것을 감안하면 t를 입력하는 것이 맞다.
  ax.set_xlabel('time (s)')
  x축 이름을 time (s)이라고 하자
  ax.set_ylabel('real')
  y축 이름을 real라고 하자.

- 위의 과정을 응용하기(시간 개념 제외)
  theta = np.arange(0, 2*np.pi)
  s = np.sin(theta)
  fig = plt.figure()
  ax = fig.add_subplot(221)
  fig = plt.figure()
  ax = fig.add_subplot(222)
  fig = plt.figure()
  ax = fig.add_subplot(223)
  fig = plt.figure()
  ax = fig.add_subplot(224)
  theta = np.arange(0, 2*np.pi) increment가 1이라 너무 sperse하니까, theta = np.arange(0, 2*np.pi, 0.1)하면 increment가 0.1이 되어서 더 빽빽하게 된다.
  (문제) x축에서는 equidistance(간격이 동일하게 증가)하다. y축에서 equidistance 한가? 틀림. x축에서만 equidistance할 것임. X, y축 모두 equidistance하다는 것은 linear(직선)이라는 말이다. 하지만 위에는 sin그래프 이니까 y축은 equidistance하지 않음.

<Complex Phasor>

- 이제까지 sin함수를 이용했다면 지금부터는 오일러 공식을 이용한 함수를 사용.
  c = np.exp(theta*1j)
  exp도 함수임. Exp는 오일러 공식의 e라고 생각하면됨. 1j는 i를 뜻함.
  theta는 위에서 정의 했음
  실제로 c 값을 쳐보면 그 c의 값들은 복소수들임. 그러면 이 복소수들을 어떻게 plotting 할 것인가?
  c = amp*np.exp(theta*1j)처럼 앞에다 amp를 곱해주면 진폭을 설정할 수 있다. Complex phasor의 amp와 원통형의 반지름과 같다. Amp=2이면 원통형의 반지름=2 (위의 오일러 공식에 radian 값을 넣은 그림 생각해보면 이해 쉽게 됨)

- fig = plt.figure()
  ax = fig.add_subplot(111, projection='3d')
  1행1열에 바둑판을 만들고, projection을 삼차원으로.
  ax.plot(t[0:100], c.real[0:100], c.imag[0:100], '.')
  3개의 입력값이 들어갔다. 그렇다면 plotting되는 그림의 한 점은 3차원 벡터가 된다. 입력값들의 개수는 100개로 동일. C는 복소수(a+bi)이기 때문에 두 개의 정보 a, b가 있다. A는 real 값. B는 imaginary값.
  이전에 말했던 projection의 측면에서 보자. e θi=cosθ+i*sinθ. Real part는 cos과 일치할 것이고, imaginary part는 sin과 일치할 것이다. 그래서 plotting된 것을 적절하게 돌려서 real 값을 표시하는 축과 시간 축만 보이게 돌리면 cos그래프가 나온다. 반대로 imaginary 값을 표시하는 축과 시간을 표시하는 축만 보이게 돌리면 sin그래프가 나온다.
  ax.set_xlabel('time (s)')
  ax.set_ylabel('real')
  ax.set_zlabel('imag')

- ipd.Audio(s, rate=sr)
  sin으로 만든 s.
  s의 주파수는 440인데 이는 '라'(A)음에 해당한다. 그런데 주파수가 880, 1760, 220, 110이더라도 모두 '라'음이 발생한다. 배수를 하면 옥타브 차이만 나는 것이다. 그리고 sin대신 cos을 넣더라도 소리는 달라지지 않는다. 우리의 귀는 phasor shift에는 sensitive하지 않다. 우리의 귀는 frequency의 차이에만 sensitive하다.
  ipd.Audio(c.real, rate=sr)
  지금까지 만든 것은 pure tone.
  import IPython.display as ipd
  IPython이라는 라이브러리 안에 있는 display라는 sub 라이브러리. 그 서브라이브러리 안에 있는 Audio라는 함수.

- import sounddevice as sd
  sd.play(c.real, sr)
  sounddevice안에 있는 play라는 함수
  이것도 똑같이 소리를 들려줌.
  #!pip install sounddevice
  이것은 sounddevice라는 라이브러리가 안깔려 있는 컴퓨터에 깔게 해주는 명령어

<generate pulse train>

- Sampling rate가 100hz인데 frequency가 1hz이 될 수 있는가? 가능하다. 2hz도 가능한가? 가능하다. 그렇다면 frequency가 10000hz가 될 수 있는가? 불가능하다. 주어진 sampling rate에서 maximum frequency는 절반임. 만약 sr이 10이면 최대 frequency는 5이다. 최대 frequency를 Nyquist frequency라고 한다. Sr의 절반에 해당한다. CD음질은 44100hz(sr) 이것은 1초에 44100개의 숫자가 있다는 것. 그렇다면 nyquist는 22050hz. 이것은 굉장히 높은 소리까지 표현할 수 있다는 것을 의미. 사람은 2만hz보다 높은 소리는 듣지 못하므로 이 정도만 해도 된다. 그래서 시디가 44100에 해당하는 sr을 가지는 것이다. 2만 보다 훨씬 높은 것을 초음파라고 한다. 박쥐가 내는 소리가 만약 10만hz의 frequency를 가지면 44100hz(sr)를 가진 CD로는 그 소리를 담아낼 수 없다.

- F0 = 100; Fend = int(sr/2); s = np.zeros(len(t));
  F0는 첫 번째 sine wave의 주파수를 뜻한다. Fend(제일 마지막 sine wave)에는 sr의 절반 nyquist를 대입. 혹시 sr의 절반이 소수점이 나올 수 있으므로 int에 대입. Time은 위에서 만들었다고 전제. Len(t)를 하면 시간을 위에서 결국 5천개의 수치(0.00001~0.50000)로 표현했기 때문에 5000이 나온다. 거기에다가 np.zeros함수를 쓰면 array([0., 0., ..., 0., 0.]) 총 5천개의 0이 들어간 array가 만들어진다. 여기에다가 sine wave들의 값들이 루프를 돌면서 차곡차곡 더해지는 것임. 결국 여러 harmonics들을 합한 것이 됨. 이게 우리 성대에서 나는 소리. 우리가 프랏에서 합성해낸 wave form. (이것은 wave form이지, spectrum이 아님!)
  for freq in range(F0, Fend+1, F0):
  F0에서 Fend+1까지 increment는 F0만큼으로 증가한다. +1한 것은 마지막 것을 포함하기 위해서.
  계속 거의 0에 수렴하면서 가다가 긴 선이 하나 있는 형태를 띠기 때문에 pulse train이라고 한다.

- 위에서 harmonics를 합쳐 만든 wave form은 그런데 spectrum상에서 frequency마다 amplitude가 모두 동일하다. Human voice source는 spectrum 상에서 gradually decreasing 한다. 따라서 위에서 harmonics를 합쳐 만든 wave form을 spectrum 상에서 gradually decreasing하게 만들어 주고 formant를 이루는 산맥을 겹쳐주게 되면 진짜 사람 목소리가 된

다.

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

  이때까지 우리는 이미 만들어진 기능을 가져다 썼는데, 지금은 새로운 기능을 만들어내는 과정이다. hz2w은 function name. (F, sr)은 function의 입력으로 들어가는 것들. W는 출력되는 것.

  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

  resonance를 정의하면서 위에서 정의한 hz2w를 사용하고 있음. Resonance만 사용할 것이기 때문에 hz2w는 신경쓰지 않아도 됨

  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)

  BWG라는 것은 W가 width를 뜻하는 것. 결국 지금 하고자 하는 것이 frequency가 0인 위치에다가 넓이가 100정도 되는 산맥(크기가 크면 넓이가 넓은 산맥이고 크기가 작으면 홀쭉한 산맥이 된다)을 세우고 거기에 맞게 frequency를 gradually decreasing하게 잘라내고자 한다. Jupyter notebook에서 pulse train과 지금 gradually decreasing한 소리를 다운로드 받아서 praat에 open해서 spectrogram을 비교해보면 전자는 모든 주파수에서 강도가 센 반면에, 후자는 저주파에서 강도가 세고 주파수가 올라가면서 점점 강도가 약해진다

  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)

이 과정은 이제 frequency=500 지점에 넓이가 60정도로 홀쭉한 산맥을 만들어서 겹치는 filter 과정이다. 점점 사람의 소리와 비슷해진다.

다음 과정에서 1500, 2500, 3500에서 산맥을 만든다. 3500까지 만들고 praat을 확인해보면 spectrogram에서 forment가 점점 연하게 형성되어있는 것을 확인할 수 있다.

s = lfilter(np.array([1, -1]), np.array([1]), s)

ipd.Audio(s, rate=sr)

이것은 사람의 입술 역할을 하는 것이다. 나팔을 보면 입술 모양이 있어서 소리가 더 크게 나는 것을 알 수 있다. 입술이 있는 것과 없는 것의 소리를 비교해보면 확연히 소리가 커졌음을 알 수 있다.

<선형대수>

행렬의 곱셈하는 방법

인공지능이란 어떠한 정보가 들어가면 다른 정보가 나오는 기계.

데이터가 들어가고 다른 데이터가 나오는., 벡터가 다른 벡터로 나오는

벡터는 그 차원에서의 하나의 점

Linear combination은 벡터들 앞에 각각 scalars를 곱한 뒤에 더해주는 것

중간에 들어가 있는 인공지능, 기계는 행렬의 형태로 되어 있다.

행렬과 행렬의 곱. 인공 지능은 행렬의 곱이다.

입력 벡터를 중간에 있는 기계(행렬)에 의해 행렬의 곱이 이루어지고 그 결과 나온 결과 갑이 결과 벡터이다.

모든 인공지능은 선형대수와 연관이 있다.

행렬에서 column vector를 뽑아내고 그 칼럼 벡터들의 linear combination이 column space이다

Eigenvector는 transformation을 거치기 전의 vector와 원점을 이은 선 위에 있다.