NUMPY

기말고사 음성학 정리
데이터

1. 소리데이터
2. 영상데이터
3. 텍스트데이터
4. 숫자데이터-반드시 숫자의 열로 표현이 되어야 함. 이런 숫자의 열을 가리켜 벡터라고 한다.

이미지는 행렬이다.
확대했을 때 보이는 사각형-숫자 하나. (흰색10 그 사이(1~9) 검은색0)
직사각형의 형태로 숫자들을 배열(가로-행, 세로-열) 이것이 행렬.
숫자들을 한 줄로 펼치면 vector. 모든 데이터는 vector의 형태로 되어야 함. 행렬은 vector가 아님. 한 줄 한 줄 따서 나열해서 vector화 시켜서 다루어야한다.
즉, 이미지는 행렬이고 이럴 벡터화시켜야한다.
컬러와 흑백. 컬러는 rgb.

동영상
동영상은 시간별로 이미지가 계속 있는 것. 흑백은 2차원. 컬러 이미지는 3차원. 여기에 시간까지 있으면 4차원.

소리
소리의 waveform을 확대한 것을 자세히 살펴보면 하나하나가 값들을 가지고 있음. 이것들이 모두 숫자값으로 되어있고 소리도 vector화된다.

텍스트
50000개의 단어가 있다고 할 때 첫 번째 단어를 벡터화 시킬 때 10000---해서 50000개를 만듬. 마찬가지로 50000번째 단어는 000----1해서 50000개 표시. 이렇듯 텍스트도 벡터화 된다.

라이브러리를 어떻게 쓰느냐에 따라 파이썬 잘하느냐 마느냐
numpy라는 라이브러리가 있음.
list 중에서도 list안에 숫자가 들어갈 때,

Numpy라는 library 안에 패키지가 있고 또 그 안에 패키지가 있고...
numpy.A.D. numpy가 젤 상위 개념, 그 안에 패키지들을 .을 써서 표현.
import numpy
numpy.A.D.f
from numpy import A : numpy에 있는 A를 불러오자. 그러면 A.D.f 가능.

from numpy import A.D
import를 크게 할 수도 있고 from을 해서 import를 할 수도 있더라.
즉, Numpy 같은 library를 import 하는 방법
i) import numpy(numpy라는 큰 라이브러리를 import)
ii) from numpy import A(numpy 속의 package를 import)
(예시) numpy의 zeros 함수를 쓸 때,

i) 을 했으면 numpy.zeros([a,b])
ii) 를 했으면 zeros([a,b])가 가능 함.

중요한 건 .은 포함관계로 되어 있따는 것.

numpy가 필요한 이유
numpy는 list와 아주 비슷한데 수학적으로 계산할 수도 있고 앞으로 쓰게 될 모든 데이터는 list 말고 numpy처리를 해서 써야함.

```
import numpy as np
import matplotlib.pyplot as plt
```

기본적으로 library들을 import한 모습
위에서 numpy as np라는 것은 import한 numpy를 줄여서 np로 쓰겠다는 의미임. 마찬가지로 matplotlib의 pyplot이라는 패키지를 plt로 줄여서 import하겠다는 의미임.
위의 import pyplot을 from matplotlib import pyplot as plt 로 바꿔서 표현할 수도 있음.

함수

1. np.empty

np.empty([2,3], dtype='int')
하면

```
array([[ 538976288,  543581540, 1852731235],
       [1601463141, 1818585203, 1702045804]])
```

np.empty함수는 입력이 list로 들어가고, 위의 열을 해석하자면, data type이 'int'인 2,3 행렬을 만들어라라는 의미임. 또한, 행렬 안의 숫자들은 data type에 따른 random한 숫자가 나옴. 그래서 실행할 때마다 다른 값이 나옴.

2. np.zeros

np.zeros([2,3])
하면

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

상당히 중요한 함수인데, [a,b]의 matrix를 모두 0이 채워진 행렬로 만듦. 아 여기서 내가 해본건데 0뒤에 .이 찍혀있는건 아마도 dtype이 float이기 때문. 그래서 위 empty에서와 같이 dtype='int'로 해주면 0이 사라짐 ㅎㅎ. 밑에 array 함수도 마찬가지.

3.np.ones

np.ones([2,3])

하면

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

zeros함수와 거의 유사한 함수로 zeros는 0으로 채워진 array를 만들어준다면 ones는 1로 채워진 array를 만들어 줌.

4. np.array

np.array([[1,2,3], [2,3,4]])

하면

```
array([[1, 2, 3],
       [2, 3, 4]])
```

zeros, ones와 비슷한 함수인데 [1,2,3]은 list로서 계산이 불가능. list를 계산 가능한 array로 만들어 주는 함수가 array 함수임. 즉, zeros함수는 안에 몇 행 몇 열의 행렬인지를 list로 입력하여 [a,b]의 array를 출력시키는 함수이고, array함수는 긴 리스트를 집어넣어서 그 것을 list로 변형시키는 것임.

+array함수 쓸 때는 대괄호가 두 개씩 들어감!!

np.ones([2,3], dtype='float64')

아까 위에서 data type 설정에 관한 이야기를 하였는데 dtype을 float64로 하면 정교한 데이터를 만들 수 있는 대신에 용량이 큼.

5. np.arange

np.arange(5)

하면

```
array([0, 1, 2, 3, 4])
```

arange함수는 index의 개수에 맞는 array를 만들어주는 함수임.

이렇게 index의 개수를 입력하는 방법도 있지만, range를 지정해주는 방법도 있음

np.arange(0,10,2, dtype='float64')

하면

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

위의 입력값들을 해석하면 0부터 10까지(10은 포함하지 않음 물론)중에서 '2'만큼의 증가분을 두고 dtype을 'float64'로 하여 만든 array를 만들라는 뜻임. 물론 저기서 증가분 빼고 입력할 수도 있고 니 맘대로임 ㅋㅋ.

6. np.linspace

np.linspace(0,10,6, dtype='int')

하면

```
array([ 0,  2,  4,  6,  8, 10])
```
이 출력되는데 입력값들을 해석하자면 0부터 10까지 중에서 arange함수와는 달리 '10을 포함'하고 총 6개로 분할하여 array를 만들라는 의미임. 참고로 np.linspace(0,11,6, dtype='int')를 내가 해봤는데 array([0, 2, 4, 6, 8, 11]) 이런 식으로 출력값이 나오더라 ㅇㅇ.

Q. 차원의 문제!!

ex) X = np.array([[1,2],[4,5],[8,9]])는 몇 차원일까?

->2차원. 직사각형의 matrix는 2차원. 직육면체 같은 것은 3차원. 3차원도 표기가 가능함. []개수를 보고 차원을 알 수 있음.

2차원짜리 2개를 가지고 3차원을 만들 수 있음.

ex) X = np.array([[[1,2],[4,5],[8,9]],[[1,2],[4,5],[8,9]]])

X

```
하면 array([[[1, 2],
            [4, 5],
            [8, 9]],

           [[1, 2],
            [4, 5],
            [8, 9]]])
```

차원을 알려주는 함수

1 .ndim

예를 들어) X = np.array([[[1,2],[4,5],[8,9]],[[1,2],[4,5],[8,9]]])

X

```
하면 array([[[1, 2],
            [4, 5],
            [8, 9]],

           [[1, 2],
            [4, 5],
            [8, 9]]])
```

가 있다고 할 때

X.ndim

하면 3이 나옴.

2 .shape

위와 같이 X라는 array가 정의되어 있다고 할 때

X.Shape를 하면 (2,3,2)가 나오는데 여기서 뒤의 두 숫자는 몇X몇의 행렬인지를 알려주는 숫자이고 첫 번째의 숫자는 몇 개의 행렬이 들어가는지를 알려주는 함수임.

이외 함수들/ 물론 X는 위에서 정의한 array라고 가정하고

1 .dtype

X.dtype

하면

```
dtype('int32')
```

data type을 알려주는 함수임.

2 .astype
X.astype(np.float64)
하면
```
array([[[1., 2.],
        [4., 5.],
        [8., 9.]],


        [4., 5.],
        [8., 9.]]])
```
이 나오는데 type을 바꿔주는 함수임.

3. np. zeros_like
np.zeros_like(X)
하면
```
array([[[0, 0],
        [0, 0],
        [0, 0]],

       [[0, 0],
        [0, 0],
        [0, 0]]])
```
이 나옴.
형태를 유지한 채 모든 숫자들을 0으로 바꿔주는 함수이다.
여기서, np.zeros_like(X)와 같이 복잡한 함수를 이용하여 안의 값들을 0으로 바꾸는 방법도 있지만, 간단하게 **X*0**을 해서 바꾸는 방법도 있다.

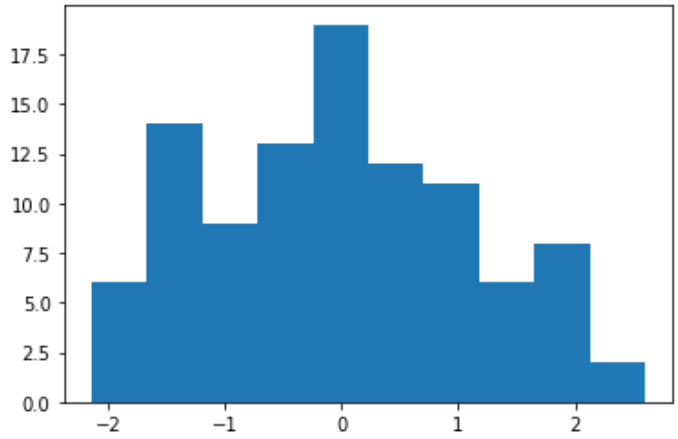
4. np.random.normal
np안의 random 안의 normal 이라는 함수임. 마찬가지로 위에서부터 강조해왔던 개념을 상기해보면 이를 from numpy import random 하면 random.normal이 가능함.

normal 함수는 normal distribution 즉, 정규 분포. 종 모양의 거꾸로된 shape의 data를 만들어주는 함수이다.
data = np.random.normal(0,1, 100)
위의 입력을 설명하자면 맨 왼쪽 0이 평균, 1이 뚱뚱한 정도, 100은 data의 개수를 의미한다.
data.ndim
을 하면 1이 나오는데 1차원임을 알려주고
data.shape
을 하면 (100,)가 나오는데 data의 개수가 100개이기 때문이다.

다음으로 이렇게 만든 정규 분포를 plotting하는 과정이다.
```
data = np.random.normal(0,1, 100)
print(data)
plt.hist(data, bins=10)
```
플로팅은 위에서 import한 matplotlib.pyplot를 사용, 그 안의 hist라는 함수. bin이 x축이고 이걸 바구니 정도로 생각하면 됨. x축의 range에 들어가는 값들이 몇 개인지가 y축. 당연히 y축 값들은 소수값 말고 정수값이 나올 수밖에 없음. 그리고 당연히 100개의 data를 plotting했으므로 이 값들

다 합하면 100개가 나옴.

```
<a list of 10 Patch objects>)
```



위의 그림이 plotting의 결과임. 원래 100개의 data가 나열된 1차원의 array print된 값 나오는데 생략하겠음.

5 .reshape

예를 들어

X = np.ones([2, 3, 4])

를 하면(위에 나왔던 것과 달리 이번엔 행렬의 개수까지 지정해 준 형태다)

```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

가 나오는데,

Y=X.reshape(-1,3,2)

Y

를 하면

```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

이런식으로 나옴. 즉, reshape는 array의 shape을 바꿔주는 함수임.

근데 여기서 reshape 옆 입력 값들은 곱했을 때의 총 element가 ones에서 만든 array의 총

element의 개수와 같아야 함. 즉 24가 되어야한다는 것. 여기서 -1이라는 것은 곱셈하기 귀찮을 때 너거 알아서 만들어라와 같은 의미임.

CF)Reshape할 때 숫자는 맨위부터 오른쪽으로 차례차례 입력하더라.

6. np.allclose(,)

np.allclose(X.reshape(-1,3,2),Y)

하면

True가 나오는데

두 개의 array를 비교해서 완전히 똑 같은 지 묻는 함수임. 즉, 위의 입력을 해석하면, X를 reshape하여 저렇게 만들면 Y와 완전히 똑같은가? 의 뜻임.

7. np.random.randint, np.random.random, np,savez

```
a = np.random.randint(0, 10, [2, 3])
b = np.random.random([2, 3])
np.savez("test", a, b)
```

위의 random.randint(0,10,[2,3])을 해석하면 0, 10 사이에서 선택하고 [2,3] array를 만들어라는 의미고 b = np.random.random([2, 3])는 말 그대로 [2,3]의 random한 값을 지닌 array를 만들라는 의미이다.

```
a= array([[7, 5, 1],    b= array([[0.78393234, 0.49033937, 0.56846429],
          [6, 9, 4]])              [0.47881012, 0.00964813, 0.50176119]])
```

np.savez("test", a, b)는 np package의 함수인데 실제 file로 만들어주는 함수임. a라는 b라는 variable을 file로 만들어주는 것. f탐색기 들어가서 보면 npz file 생성 된 것 볼 수 있음.

!ls -al test*를 통해 file로 저장된 것을 보여줌.

8. del

variable을 없애고 싶을 때 del a, b와 같은 형태로 사용함.

9. np.load 등등

test.npz라고 variable들을 담아서 file들을 저장을 한 후에

```
npzfiles = np.load("test.npz")
```

- np.load라고 한 뒤 file 이름을 적고

npzfiles.files ->저장한 file 불러올 때.

그리고 저 이름들을 npzfiles['arr_0']에 넣으면 각각 a,b가 나옴.

```
npzfiles = np.load("test.npz")
npzfiles.files
```

하면 `['arr_0', 'arr_1']`이 나옴.

```
npzfiles['arr_0']를 하게 되면
array([[7, 5, 1],
       [6, 9, 4]])
```

와 같이 a라고 정의했던 variable이 나옴.

arr_1 하면 b값이 나올 것임.

csv는 coma separated values. 엑셀에서도 불러지는 format. 제일 많이 쓰게 될 format.
data = np.loadtxt("regression.csv", delimiter=",", skiprows=1, dtype={'names':("X", "Y"), 'formats':('f', 'f')})
data
loadtxt해서 불러오고 delimiter ‘,’로 분리 skiprows=1 첫 번째 row 데이터 안 쓸거다(X, Y라고 적힌 것은 그냥 제목이니까). dtype에 첫 번째 것은 X, 두번 째 것은 Y라고 하자. format은 둘 다 float으로 하겠다. 이렇게 하면 data가 file로부터 data라는 변수에 들어옴.
역으로 np.savetxt("regression_saved.csv", data, delimiter=",")는 데이터를 불러올 때가 아니라 저장할 때 씀.

@위의 내용 복습 (Inspection)

```
arr = np.random.random([5,2,3])
print(type(arr))
print(len(arr))
print(arr.shape)
print(arr.ndim)
print(arr.size)
print(arr.dtype)
```

하면 결과

```
<class 'numpy.ndarray'>
5
(5, 2, 3)
3
30
float64
```

가 나옴. 여기서 len은 행렬의 개수를 의미하는 걸로 추정되고 size는 총 element의 개수. 나머지는 알지?

Broadcasting

```
1)a = np.arange(1, 25).reshape(4, 6)
a
```

하면

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24]])
```

이 나오는데

먼저,

ex) np.arange(1,5) 하면 1,2,3,4 나옴.

ex) np.arange(9,5,-1)하면 9,8,7,6 나오는데 역순으로 쓰고 싶을 때 쓰는 방법임. 큰 숫자부터 2개 쓰고 뒤에 -1 쓰는 식. cf)(1,5,-1)하면 값 이상하게 나옴 마찬가지로 (9,5)도 이상하게 나오더라.

각설하고 1~24까지의 만들어진 1차원 arange를 reshape을 통해 (4X6)의 행렬로 만들 결과이다.

2)덧셈

a=

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18],
```

```
       [19, 20, 21, 22, 23, 24]])
```
이고
b=
```
array([0, 1, 2, 3, 4, 5])
```
이라고 할 때
a+b=
```
array([[ 1,  3,  5,  7,  9, 11],
       [ 7,  9, 11, 13, 15, 17],
       [13, 15, 17, 19, 21, 23],
       [19, 21, 23, 25, 27, 29]])
```
이 나옴.

여기서, 만약 b가 똑 같은 shape이였다면 +가 가능함. 그런데, 만일 b=np.arange(1,13).reshape(2,6)와 같은 함수에서 나온 2,6의 matrix라면 덧셈이 안되더라. 내가 해봤음 ㅇㅇ.

3)a==b
각각의 정보가 같은 지 틀린 지.
```
array([[False, False, False, False, False, False],
       [False, False, False, False, False, False],
       [False, False, False, False, False, False],
       [False, False, False, False, False, False]])
```
만일 모든 element의 정보가 다른 [4,6]의 matrix a,b를 비교하면 위와 같은 결과 값이 나옴.

4)a>b
위와 비슷한 것임 대소를 묻는.

그런데 여기서 주의할 점은 3,4를 정의할 때 있어서 **두 array a,b는 dimension, shape이 완전히 똑같아야만 함!!**

Sound

**오일러공식과 관련된 내용은 워드 필기 참조.

기본적인 import

```
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
```

Phasor

```
# 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
```

위는 parameter settings이다.

parameter setting을 분석해보면 amp가 1이라는 것은 진폭을 1로 하겠다는 의미임. dur은 소리가 얼마나 지속되는 지를 말해주는 것임. sr rate는 frequency와 마찬가지로 단위로 hz를 사용하는데 소리의 해상도를 결정하는 요인임.
하지만, 위의 parameter settings에는 time이 없기 때문에 time을 만들어주어야 함.

Sine Phasor

1)time 만들기

먼저, sampling rate은 1초를 몇 개의 time tick으로 쪼개주는 지를 보여주는 지표임. duration은 지속 시간. 그러므로 sampling rate과 duration만 있으면 time을 만들 수 있음.
0.0001이 첫 번째 time tick이고 0.5000이 duration인 마지막 time tick임.
이렇게 time tick이 있고,
t=np.arange(1.sr+1)하면 1초 동안의 time의 index임. 그러나 이건 duration이 1일 때의 얘기고 현재는 duration이 0.5이므로 time tick이 절반밖에 안 됨. 따라서 sr에 dur을 곱해야 time의 index를 구할 수 있음.
t=np.arange(1,sr*dur+1)하면 time의 index임. 그러나 이렇게 구한 값은 어디까지나 time의 index일 뿐 실제의 time은 아니다. 실제의 타임은 이 index를 sr로 나눈 값이다.
결론적으로 t=np.arange(1,sr*dur+1)/sr 인 셈이다.
위의 parameter로 만든 t값은

```
array([1.000e-04, 2.000e-04, 3.000e-04, ..., 4.998e-01, 4.999e-01,
       5.000e-01])
```

인데 여기서 1.000e-04라는 말은 1에 10의 -4승을 의미하는 것임.

2)theta 만들기

time을 만들었으면 이를 theta와 연동시켜 phase로 바꾸어야 함.
결론부터 말하자면 theta = t * 2*np.pi * freq이다.
여기서 np.pi라는 것은 numpy속에 정의 된 pi로 그냥 상수값이다.
ex)time이 1이라 가정했을 때 time에 2pi를 곱한 값은 2pi로 총 한 바퀴를 돌았다는 의미가 된다. 그리고 frequency는 초당 회전수, 즉 초당 돌아야 할 바퀴 수이므로 곱해준다. 그러므로 이러한 원리로 위와 같은 식이 나온 것이다.
여기서 **Q.** time vector의 사이즈와 theta vector의 사이즈는 같다?(o) 왜냐하면 theta vector는 time vector에 사칙 연산을 가하여 만든 vector이기 때문!

3)sin 함수 만들기

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

위의 time과 연동하여 만든 theta가 있기 때문에 우리는 phasor을 만들 수 있다.

4)plotting하기

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(t[0:1000], s[0:1000], '.')
ax.set_xlabel('time (s)')
ax.set_ylabel('real')
```

위에서 plt안의 figure이라는 function을 사용하였고

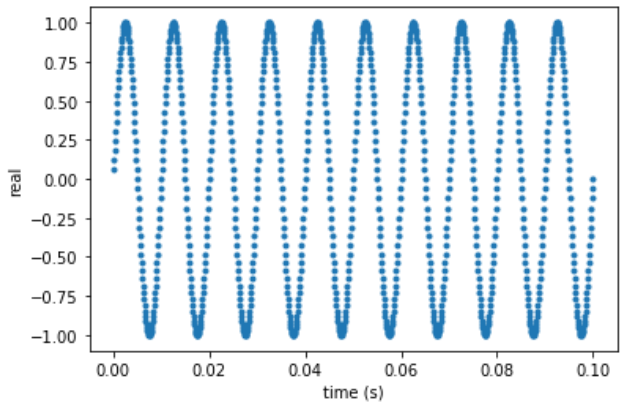
fig안의 add_subplot은 화면 분리의 의미인데 1,1,1에서 앞의 두 개는 1X1의 분할 중에서 뒤의 1 첫 번째의 의미임.
여기서 만일 2X2분할이었다면 221 222 223 224 이런 식으로 쓸 수 있음.

ax.plot(t[0:1000], s[0:1000], '.')의 의미는 0~1000개까지의 time index까지만 범위를 정해서 plotting 하겠다 그 의미임. 그리고 plotting할 때 '.'을 이용해서 하겠다. 이걸 -로 바꿀 수도 있음. 앞에 넣은 값이 x값, 뒤에 넣은 값이 y값. 이렇게 부분만 정해서 plotting하는 이유는 그냥 sin 그래프 모양을 알아보기 쉽게 편의상 하는 것임.

ax.set_xlabel('time (s)')
ax.set_ylabel('real')
의 의미는 각각 x축의 이름을 'time(s)', y축의 이름을 'real'로 하겠다는 의미임.

CF)임의로 sin 그래프 만들고 싶을 때 theta 설정할 때 theta값을 np.arange(0, 2pi)하면 한 사이클 구할 수 있어서 가능. 근데 이런 식으로 만들면 너무 그래프가 sparse하기 때문에 theta=np.arange(0,2*np.pi,0.1)와 같이 0~2pi사이의 index 간격을 설정해주어서 더 빽빽하게 만들 수 있음.
위의 행을 실행하면

```
Text(0, 0.5, 'real')
```



Q. Linear vs Unlinear
-x축의 값들이 equidistant (등간격)할 때, Linear한 그래프의 경우에는 y축의 값도 모두 equidistant 한 반면에, Unlinear한 그래프에서는 y축의 값들이 equidistant하지 않음.

Complex Phasor
1) Complex Phasor이란?
c = np.exp(theta*1j)
exp는 exponential 함수. 쉽게 생각해서 np.exp가 oyler function의 e라고 생각하면 된다. 그 함수의 입력값으로 theta*1j가 들어가는 것은 e의 지수로 저 입력값이 들어가는 것. 하나하나의 값들이 a+bi의 형태.

위의 c는 실행했을 때

```
array([0.99802673+6.27905195e-02j, 0.9921147 +1.25333234e-01j,
       0.98228725+1.87381315e-01j, ..., 0.9921147 -1.25333234e-01j,
       0.99802673-6.27905195e-02j, 1.        +1.96438672e-15j])
```

와 같은 array가 나오는데, e- 어쩌구로 표기함으로써 쓰는 형태와 정보량을 통일시킨다.

아무튼, c에는 복소수 i가 들어 있기 때문에 complex number(복소수) 형태의 vector가 들어 있다.

2)Complex Phasor Plot하기

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(t[0:1000], c.real[0:1000], c.imag[0:1000], '-')
ax.set_xlabel('time (s)')
ax.set_ylabel('real')
ax.set_zlabel('imag')
```

Complex Phasor을 plot 할 때는 projection을 '3d'(3차원)로 하기 때문에 ax.plot할 때 sin phasor와 다르게 입력값이 3개가 들어가야 한다. 즉, 하나의 점을 표현하기 위해 3차원의 vector값이 들어가는 것임. 또한 당연하게도 각각의 입력값(t, c.real, c.imag)의 개수는 동일해야 함.

c값은 대부분이 복소수값임(a+bi). 즉, exponential oyler function을 쓰면 두 개의 정보가 나오는 것임. 그래서 그 정보를 따로 받아옴.

c.real 은 실수부만, c.imag는 허수부만 받아 옴.

이렇게 x축은 time, y축은 실수부, z축은 허수부를 따로 받아서 plotting함.

cf)오일러 공식을 보면 $e^{\theta i} = cos(\theta) + sin(\theta)i$

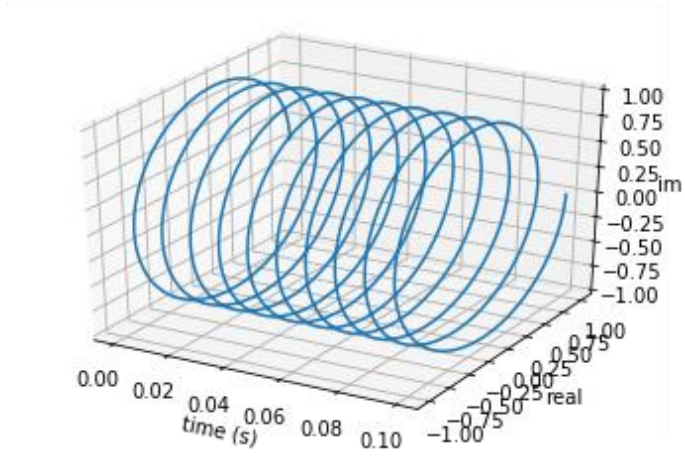
인데, 이를 볼 때 cos은 실수부, sin은 허수부와 관련이 있음. real만 본다면 위에서 보면 됨.

그렇게 했을 때 1부터 시작함. cos의 형태임. projection 즉 visualization이 왜 중요한 지를 보여줌.

projection을 해 보았을 때

sin과 cos의 기능을 모두 가진 complex한 phasor임을 알 수 있음.

```
Text(0.5, 0, 'imag')
```



3)Audio Play하기

오디오 play하려면 ipd를 import 하고 입력값은 signal(벡터값)과 원하는 sr을 넣어주면 됨. sin으로 만든 값을 써도 만들어지고, c.real c.imag 같은 형태로 해도 소리 만들 수 있음.

ipd.Audio(s, rate=sr) 하면 됨.

다른 방법으로는
#!pip install sounddevice
import sounddevice as sd
sd.play(c.real.sr)
이렇게 sound 를 play하는 것도 있음.

Generate Purse Train

s=np.sin(theta) ->-1부터1까지 가는 그래프이고 전체에 x2를 하면 -2부터2 이게 amplitude임. s=amp*np.sin(theta) 이렇게 sin값에 amp를 곱해주면 해주면 amplitude가 구현이 됨.

Q, sampling rate과 frequency의 차이-Nyquist Frequency

둘 다 unit은 hz로 같지만, sr은 1초동안 점들이 얼마나 많이 나오느냐의 개념이고 frequency는 shape이 몇 번 반복되느냐의 개념임. 이러한 이유로 아무리 점들을 아껴 써도 sr으로 maximum 표현 가능한 frequency는 sr의 절반임. 즉 Nyquist Frequency=sr/2=sr로 표현가능한 frequency의 maximum.

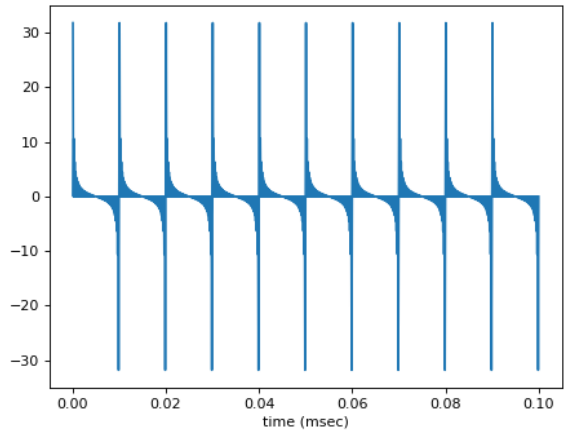
이런 Nyquist Frequency로 인해서 cd의 sr가 44100hz로 설정이 돼있는 것임. 인간의 가청 주파수가 약 20000Hz이므로 그것의 두 배 언저리 정도로 sr을 설정하여 소리를 경제적으로 표현할 수 있는 것이다.

Purse Train 만들기

```
# 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):
      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)')
ipd.Audio(s, rate=sr)
```

위를 해석하면 F0를 100으로 설정을 했고, 이 끝은 Nyquist Frequency에 의해 sr의 절반까지 올라갈 수 있음. 그리고 마지막을 Fend(=Nyquist Frequency)라는 variable로 설정함. 그리고 지저분하지 않게 int로 반올림함. 위의 식에서 time은 이미 만들어졌다고 가정 된 상태이고 따라서 freq 만 for loop에 따라서 바꾸면서 반복을 하면 됨. range(F0, Fend+1, F0(increment))를 하면 100~5000까지 100 간격으로 나오기 때문에 총 50번의 loop.
또, s=s+tmp에서 처음의 s값이 정의 되어 있지 않으면 error message가 나옴. 그래서 위에 s = np.zeros(len(t))로 s값을 정의. 이 의미는 time tick의 개수만큼 0이 들어간 array를 만드는 것.
여기에 loop가 돎에 따라 sin wave를 더하고 또 더하고.. 100hz부터 5000hz까지 다 더해진 s를 plot 함.
여러 sound를 harmonix해서 만든 소리로 매우 정교함. 이를 purse train이라고 명명한 이유:

```
<IPython.core.display.Javascript object>
```



다음이 plotting 된 purse train의 모습이다.

위의 그림은 waveform이지 spectrum이 아님. spectrum은 이 waveform에서 한 부분을 따서 만든 frequency-amplitude그래프. sine wave를 더해나가면 purse train의 형태가 나온다는 것까지 배웠음. x축은 time. y축은 value(energy). waveform 말고 spectrogram의 형태로 볼 수도 있더라(frequency 성분대별로 확인, spectrogram도 시간 amplitude 그래프임). 한 슬라이스만 잘라서 이 부분에 어떤 frequency 성분이 많은지 분석하는 것이 frequency(x)-amplitude(y)그래프인 spectrum. spectrum을 시간 순으로 나열해서 보여주는 그래프가 spectrogram. formant.

특히 f1. f2에 따라 모음의 종류 구분 하는 것. 그 과정을 파이썬으로 할 것임. source의 스펙트럼 보면 gradually decreasing.-첫 번째 산맥을 하나하나씩 만들 것임-두 번째 이후에도 산맥을 하나씩 더 만듦.

Amplitude 구현한 phasor plotting

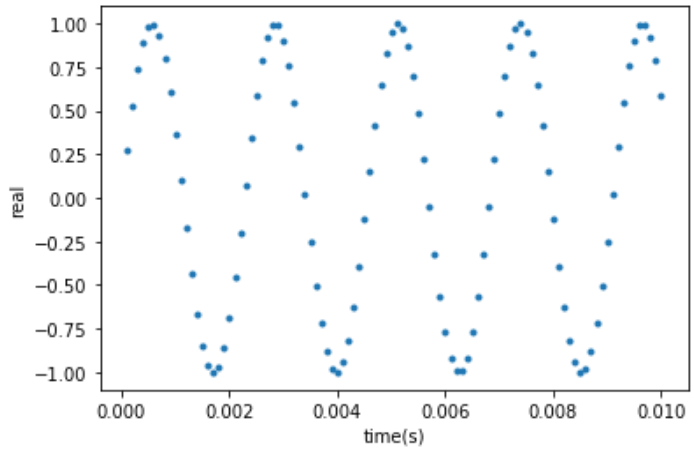
1) Sin-Cos Phasor

```
#parameter setting
amp = 1            # range [0.0, 1.0]   진폭
sr = 10000         # sampling rate, Hz 정보  촘촘히
dur = 0.5          # in seconds   정보  길게
freq = 440.0       # sine frequency, Hz 사인웨이브 반복 횟수
t = np.arange(1, sr * dur+1)/sr
theta = t * 2*np.pi * freq
s=amp*np.sin(theta)
```

Q. Sine vs Cosine?

phasor로 sin을 쓰던 cos을 쓰던 소리는 똑같이 들림. 그 이유는 sin과 cos이 frequency의 변화에는 민감하게 반응하지만 phase shift에는 민감하지 않기 때문임. 실제로 두 그래프를 조금씩만 이동하면 똑같아지기 때문에.

```
Text(0, 0.5, 'real')
```



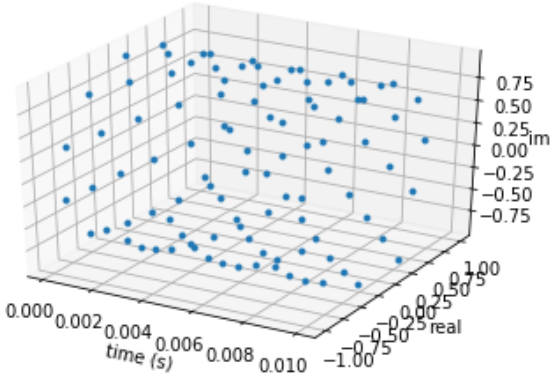

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(t[0:100], s[0:100], '.')
ax.set_xlabel('time(s)')
ax.set_ylabel('real')
```

를plotting한 결과임.

-Frequency 440Hz가 '라'음이고 옥타브를 뛰려면 주파수를 배수로 하면 된다. 1760 880 220 110 등.

2) Complex Phasor

```
c =amp*np.exp(theta*1j)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(t[0:100], c.real[0:100], c.imag[0:100], '.')
ax.set_xlabel('time (s)')
ax.set_ylabel('real')        #cos 값과 같음
ax.set_zlabel('imag')        #sin 값과 같음
ext(0.5, 0, 'imag')
```



이게 플로팅 결과임.

Formant 만들기

```
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
```

def를 적고 옆에 function names 및 ()에 입력값들. function을 만드는 과정임. 지금은 function을 구성하는 법은 몰라도 되고 사용하는 법만 알면 됨.
아래 function은 입력값이 3개, 출력값이 두 개(return이 두 개니깐)
cf)아래 함수는 함수를 만들면서 다른 함수(위에 쓰인 hz2w)를 불러옴. 그래서 위의 함수는 아래 함수의 내부적 정보임.
아무튼 쓰는 법: resonance 함수를 쓸 때, 첫 번째 입력으로 sampling rate을 넣고, F에 산맥의 위치, 즉 frequency를 넣고, BW-band width는 뚱뚱한 정도로 산맥의 shape을 결정. 100이면 꽤 뚱뚱한 정도임.

```
#1
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)
```

위의 식은 0의 위치에 100만큼 뚱뚱하고 gradually decreasing하는 shape??
식을 설명하자면 resonance 함수로 a,b값을 구하고 그것을 lfilter에 집어넣어주는데 lfilter 속의 s는 이전 단계의 signal 그 값을 받고 쓰고 또 쓰고.
rg(산맥의 위치)에 0을 넣었음. spectrum에서 gradually decreasing한 것을 만들어야 됨. 0을 중심으로 해서 0을 중심으로 큰 산을 만든다고 생각. 0이 봉우리. 0을 중심으로 아주 뚱뚱하고 완만한 산을 만들어라. gradually decreasing하는 shape.
위의 purse train은 귀가 찔리는 듯한 소리였는데 여기선 좀 더 부드러운 소리. 이것이 decreasing한 효과라고 생각하면 됨.

```
#2
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)
500 hz에 f1을 만들고 위보다 홀쭉함.

#3
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)
두 번째 산맥은 1500 hz에 만듦. 더 뚱뚱.

#4
RG = 2500 # 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)

#5
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)
ipd.Audio(s, rate=sr)

Fourier tranform

```
nFFT = nSamp
amp = [];
for n in range(0,nFFT):
    omega = 2*np.pi*n/nFFT # angular velocity
    z = np.exp(omega*1j)**(np.arange(0,nSamp))
    amp.append(np.abs(np.dot(s,z)))
```

위의 내용을 분석해보면

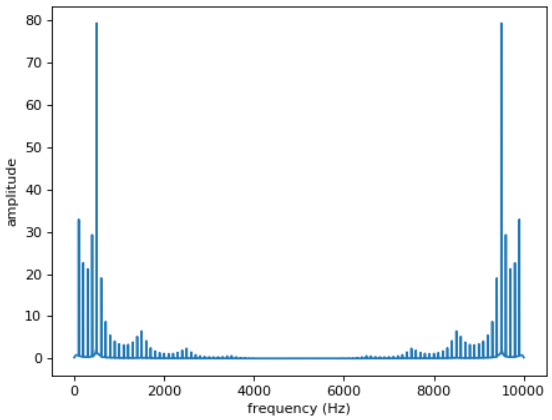
```
z = np.exp(omega*1j)**(np.arange(0,nSamp))
```

complex phasor을 만드는 줄임(e^{wi})의 지수로 [0~Samplingrate]까지 들어가는 것임.
**은 지수를 말함.

```
omega = 2*np.pi*n/nFFT
```
omega=2π x n/ 100인데
n이 100일 때
첫 번째 loop에서는 0
두 번째 loop에서는 2πx1 /100
이거 노트 참고!!

```
fig = plt.figure()
ax = fig.add_subplot(111)
freq = np.arange(1,nFFT+1)*sr/nFFT;
ax.plot(freq, amp)
ax.set_xlabel('frequency (Hz)')
ax.set_ylabel('amplitude')
```
위를 해석하면
```
ax.plot(freq, amp)
```
plot 할 때 x값과 y값을 만들어야하는데 y에 해당하는 값은 amp로 이미 만들어 놓음. amp라는 variable 이름을 쓴 것은 각 frequency 성분에 대한 에너지 값이니깐.
Q. 그렇다면 x에 해당하는 좌표는?
```
freq = np.arange(1,nFFT+1)*sr/nFFT;
```
1부터 nFFt+1 해서 개수를 정하고 sr을 곱하고 sr의 개수인 nFFT로 나누어줌.
[100---10000]까지 100개의 개수를 가진 x값이 나옴.

```
<IPython.core.display.Javascript object>
```



위를 plotting한 결과임. 여기서 나오는 하나의 막대기가 inner product 한 값임.
Q. 여기 있는 총 bar의 개수= nSamp의 개수와 같다.
Q.위 그래프의 의미: 왼쪽 오른쪽이 대칭인데, Nyquist Frequency 때문에 5000까지 의미 있는 frequency임. 그래서 대칭인 것은 중복된 부분으로 의미가 없는 부분임. 그리고 이 half의 의미는 한 순간에서의spectrum. peak들은 formant. 각 frequency 성분들의 inner product의 absolute 값이다.
2000hz라는 성분이 저만큼 있더라. 500hz라는 성분은 저만큼 크더라와 같은 의미.
이러한 것들이 지니는 의미: 우리가 처음에 원 s를 어떻게 만드냐 sine wave를 여러가지 harmonix로 만듦. 근데 이를 점점 작아지게 만들었음. ->500 1500 2500에 산을 만듦. 만약 우리가

2500에 산을 안 만들어놨으면 저기에 산으로 표시가 안 됐을 것임. 또 gradually decreasing하게 만든, resonance를 써서 만든 과정을 하지 않았다면 flat한 형태의 그래프가 나왔을 것임.
스펙트로그램은 이 스펙트럼들이 모여서 만들어 진 것.

```
max_freq = None  # cutoff freq
win_size = 0.008 # sec
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)
```

위를 분석하면,
wave가 있을 때 한 장짜리 스펙트럼이 나올텐데 어떻게 이것이 연결된 스펙트로그램의 형태로 만들까?
win_size
win_step 은 길게 wave가 있을 때 앞부분에서 조금만 잘라서 spectrum을 만듦. 이걸 조금 이동해서 또 만들고 또 만들고(영상 봤던 것처럼). 한 것이 spectrogram. 그 한 장 한 장씩 만들 때 얼만큼의 vector을 가지고 dot product를 하겠다 정하는 size가 win_size. win_step은 이동하는 간격.

그렇게 만든 스펙트로그램을 보면 첫 번째 formant만 진하게 나오고 이후의 formant는 연하게 나오는데 이것이 스펙트럼의 formant 높고 낮고가 반영된 결과임. 이걸 그냥 보면 색이 너무 희미해서 처리를 가함.

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

우리가 계산할 때 complex number로 dot product가 나오고 그것의 absolute를 취해서 위의 실수값이 나오고 그것이 plotting한 결과. 진한 것은 1보다 큰 수가 나오고, 연한부분은 1보다 작은 값들이 나옴. 이 값들을 제곱을 해주게 되면, 진한 값들은 훨씬 커지고, 1보다 작은 값은 훨씬 작아지는데 이것이 power spec. 위에서 **2는 제곱.
이 작업을 하는 이유는 log를 취하기 위함.

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

위의 것에 log10을 취하면 아주 작은 숫자는 적당히 크게, 아주 큰 숫자는 적당히 작게 떨어짐. 우리가 다룰 수 있는 범위 내로 바꾸어주도록 하는 것이 log를 취하는 이유. 기껏해야 -4, -5, 3,4

이런 식으로 나오는 값에 +10 정도 해주면 양의 정수값으로 딱 떨어지는 magic.

즉 위의 작업들을 함으로써 spectrogram을 더 visualize하도록 만듦.