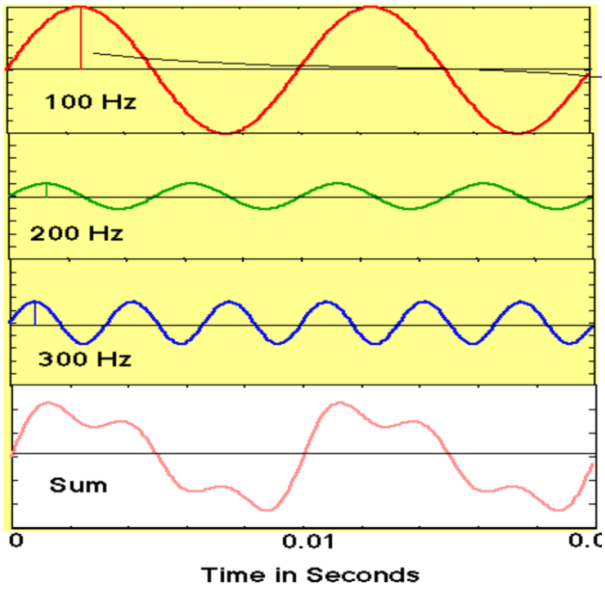
1. 소리는 숫자임. 등간격에 따라 할당된 숫자로 소리를 표현함. 그 점을 이어 그래프로 표현한 것. >>> 첫 수업, Praat을 이용하여 '안녕하세요'를 녹음한 것을 통해 알 수 있음
2. Consonants (자음)
    A. 소리랑 철자랑은 다름!
    B. 유의해야 할 표기는 j =/= 져 / j = 여
3. Vowels (모음)
    A. monophthong (단모음) / diphthong (이중모음): 6개가 있음
    B. 모든 모음은 voiced임
4. Phonetics
    A. A study on speech: speech는 사람과 관련된 발화임. 동물 소리에는 speech라 하지 않음.
    B. Articulatory (mouth), 조음 / Acoustic (through air): 물리적 영역, 음향 / Auditory (ear), 청각
5. Articulation
    A. Vocal tract: Nasal tract / oral tract
    B. Upper VT: lip ~ teeth ~ alveolar ridge ~ hard palate(구개) ~ soft palate(velum) ~ uvula ~ pharynx ~ larynx
    C. Lower VT: tongue structure (tip, blade 등) ~ epi(덮개)glottis
    D. Ex. 코로 숨 쉴 때 velum은 lower → '아' 소리 내면 올라감
    E. 사람은 기도(식도x)를 이용해서 소리가 남.
    F. Larynx: Voice box. Voiced/Voiceless: vibration 차이
    G. lips / tongue tip / tongue body: 메이저 조음에 관련된 것
    H. Constriction Location (where) / Constriction Degree (how much)
    I. CD: Upper part ~ Lower part: Stops > fricatives > approximants > vowels
    J. English 발음은 Constrictors, CD, CL에 따라 달라짐
        i. Ex. Velum raised, glottis opened, tongue tip, alveolar, stop = [t]
    K. cf. 모든 모음은 constrictor로 tongue body만 씀
6. Phonemes
        1. individual sounds that form words ex. /s aɪ k oʊ/
7. Praat
    A. Duration(sec), Intensity(dB), Pitch(Hz), Formant(Hz)
    B. 모든 시그널(신호)은 다르게 생긴 여러 사인 웨이브의 합으로 표현됨
    C. 이 세상에 존재하는 모든 신호는 sine wave = f(frequency, magnitude)
    D. Magnitude=amplitude



E. 사인웨이브(SW) 1은 크기는 크지만 freq. 작음 → slow. 1초에 저 모양이 100번 들어감

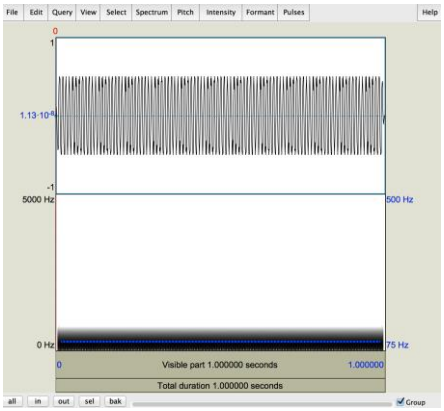
F. SW2는 2배 빠름.

G. Sum SW도 1초에 100번 반복됨.

H. x축은 시간, y축은 voltage(=value값)

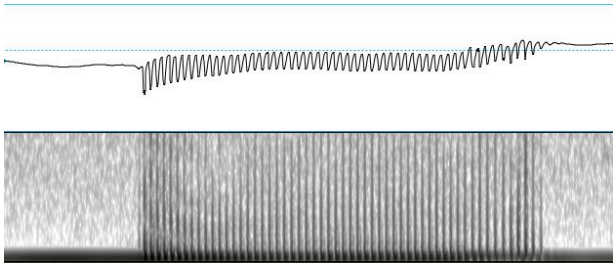
I. 합쳐서 Sum SW로 만드는 것 = synthesis / 그걸 어떤 것들로 이루어졌는데 스펙트럼으로 표현하는 것= analysis

J. 막대그래프로 만들면 스펙트럼: x축은 frequency, y축은 amplitude



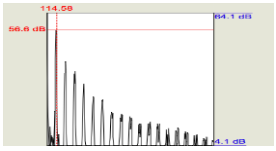
K. 검게 칠해진 것: spectrogram. 스펙트럼을 time으로 visualize한 것임. 스펙트럼 자체는 시간 개념이 없기 때문.

L. Human voice는 여러 사인 웨이브가 합해져 있음



M. Source의 filter를 어떻게 만드느냐에 따라 '아' 소리도 만들고 '이' 소리도 만듦.

N. Source만 가지고도 spectral analysis 가능
O. F0 = fundamental frequency
P. 스펙트럼의 x축: time, y축: frequency
Q. 진한 것이 amp가 큰 것임
R. 이건 gradually decrease harmonics. 저주파에서만 에너지가 강함.



S. Human voice source consists of harmonics!
T. Vocal tract의 shape에 의해서 소리가 만들어지는데 이걸 filtered된다고 말함
U. Peak / Mountain
V. 어디에 산맥이 나타나는가는 입모양에 따라 달라짐. 특정 높이의 소리는 특정한 입모양과 맞춰질 수밖에 없음. 그치만 같이 '아' 소리 내는 김씨, 이씨, 박씨의 산맥 패턴은 같음
W. 첫 번째 산맥 = f1 (=/= f0)
X. Praat을 통해 synthesize, combine source하는 법 배움

<Starting Python from Oct. 01st, 2019>

A. Variable
   A. 모든 language는 '단어'와 '문법'(어떻게 combine 하느냐)이 있음. '단어'는 그릇. '사과'라는 정보를 담으면 >> 사과가 됨. -→ 컴퓨터 언어에서 이것은 Variable(변수). 숫자, 문자 등을 담을 수 있음.
   B. 문법: 1) 변수라는 그릇에 어떤 정보를 assign 하는 것 2) conditioning 하는 것=if절 3) 여러 번 반복하게 하는 것=for절 (loop) 4) 함수 (어떤 입력을 넣으면 어떤 출력이 나오는 것).
   C. Directory 가려면 anaconda prompt >> jupyter notebook >> 원하는 directory를 browse >> New >> Python3
   D. = 표시는 오른쪽에 있는 정보를 왼쪽에 있는 variable로 assign한다는 뜻
   E. Print 함수는 ( )에 변수를 넣으면 그 정보를 print out 하는 것 ex. Print(a) 에서 입력은 a
   F. Variable은 unique 하기 때문에 처음에 a=1하고 다음 셀에 a=2 쓰면 a=2로 overwrite됨. 근데 a=1셀을 클릭하고 run하면 그 아래 a=2 셀 있어도 print(a)=1 나옴
   G. 문자는 그냥 쓰면 무조건 변수취급임. 그래서 정보값으로 하고 싶으면 ' ' 해야 함
   H. Shift+enter = 실행
   I. Love = 2 / b = love / print(b) = 2 : love라는 변수에 2값을 넣고, 다시 b라는 변수에 love가 가진 정보값을 도출하게 b=love라는 식을 세우면, print(b)하면 love의 정보값인 2가 나옴. 그리고 엔터 대신에 a=1; b=2; c=3도 가능
   J. 한 셀에 a=1 엔터 b=2 엔터 c=3 하고 같은 셀에 엔터 c 이렇게 쓰면 마지막 c는 print(c)랑 같은 뜻임
   K. [ ] 이거는 리스트를 표현하는 것

L. Type( )하면 각 변수가 무슨 타입인지 알려줌 ex. a=1 → type(a): int / a=1.2 → type(a)=float 실수 / a=[1,2,3] → type(a)=list / a='love' → type(a)=str 스트링 / a=[1,2,3,'love'] → type(a)=list 즉, 리스트는 숫자만일 필요는 없음 / [ ] 이거를 ( ) 얘로 바꿔도 됨. Type은 tuple임. Tuple과 list은 완전 똑같은데, 다만 보안에 더 강함.

M. Dictionary ex. a = ('a': 'apple', 'b': 'banana') 여기엔 2개가 들어있는 것, 표제어 : 설명 쌍으로 되어있음. → type(a) = dict

N. Variable 이름 적고 대괄호 쓰고 안에 index 쓰면 그 값을 가져옴. a=[1,2,3]. Print(a[1])=2

O. Dict의 정보를 access 할 때는, 페어에서 앞부분을 인덱스로 씀. 그니까 0,1,2 안 씀. Ex. a={"a":"apple", "b"="orange"}; print(a["a"]) = apple

P. S='abcdef' ; print(s[1:3]) = bc(1번부터 3번 직전까지)

Q. #을 붙이면 그 뒤가 실행이 안 됨. 쓰고 싶은 노트 쓰면 됨. / Code대신에 markdown으로 바꾸면 노트로 쓸 수 있음

<10.29 시험 리뷰>

A. 영어의 nasal consonant는 발화할 때 velum이 lowered 되어서 nasal tract으로 air flow가 있지만, oral tract는 막혀서 air flow가 없다. (True)

B. 일반적으로 영어 모음 /a/를 발화할 때, nasal tract의 air flow는 차단된다. (T)

C. 다음 중 발화시 air pressure와 가장 관계가 있는 것은? (Intensity)

D. 영어의 자음 중 코를 막고 숨을 쉴 때와 가장 유사한 articulation 상태인 음소는? (h)

E. 영어 /h/는 다음 중 어떤 articulator에서 constriction을 가지는가? (lips/tongue tip/tongue body/**none**)

F. 영어 음소 중에서 tongue body애서 constriction을 가지며 constriction degree는 fricative, constriction location은 velar인 자음의 개수는? (0개)

G. 영어 자음 /v/의 articulator의 constriction location을 bilabial로 바꾸고 constriction degree로 approximant로 바꾸면 어떤 음소가 되는지 쓰시오. (w)

H. /s/, /l/은 major articulators (lips, tongue tips, tongue body) 중에서 공통적으로 **tongue tip**에서 조음된다.

I. 어떤 모음의 pitch가 128Hz이다. F1은 128Hz보다 반드시 크다. (T)

J. 어떤 화자의 /j/ 모음을 120Hz의 pitch로 발화했다. 같은 화자가 똑같은 pitch로 /a/ 모음을 발화했다. /i/와 /a/ 모음에 대해, 0Hz~5000Hz 사이에 몇 개의 harmonics가 존재하는지 구하고, 각각의 개수를 a와 b라고 할 때, b-a는? (0)

K. a=[[1],[2,3].[4,5,6]]

```
n=0
for bin a:
    for d in b:
        n+=1
print(n)
```

답: 6

L. 12-13번 총 몇 개의 unique한 함수가 있는가? (3개)

M. 몇 번 실행 되는가? (6번)

```
n=[1,2,3,4]
for i in range(len(n)):
    print(i)
```

N. 14-16번

```
a=[1,2,[3,4]]
```

위의 코드를 실행하면 결과가 4이다.

```
b=[1,'a',{'a':[3,6], 'b':[d]}, 'b']
print(b[-2]['b'][-1])
```

O. C=[1,'a',{'a':'abc', 'b':def'}]

```
Print(c[-1]['b'][-2])
```

답은 2

P. 비슷한 문제

<중간고사 이후>

A. 숫자열을 가리켜 벡터라고 함.

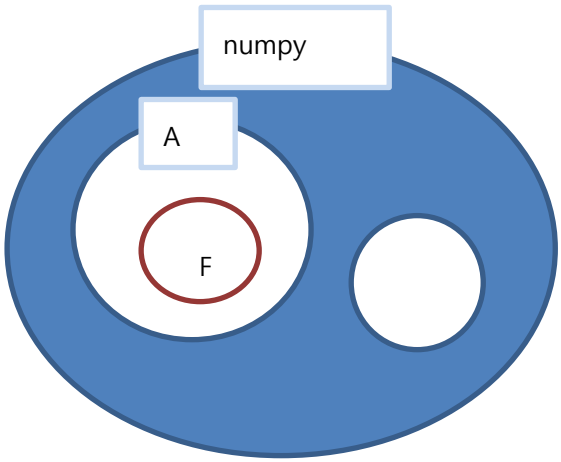
B. 사진을 픽셀로 나누면 숫자 값을 넣을 수 있음. 직사각형의 형태로 행과 열: 행렬

C. 모든 데이터는 벡터의 형태로 되어야 함

D. 사진 흑백은 행렬데이터가 1장, 컬러는 3장(RGB)로 이루어짐

E. Array는 계산 가능한 것으로 바꿔 줌

<Numpy>



- Import numpy as np #넘파이를 줄여쓰기 위해
- Import matplotlib.pyplot as plt 얘는
  from matplotlib import pyplot as plt 얘로 바꿔서 쓸 수 있음
- Per sec의 개념이 들어가면 항상 Hz 단위를 써야 함. Sampling rate(1초에 몇 개 조각으로 나누는가)도 그러함.

In [9]:
```
np.empty([2,3], dtype='int')

#[2,3] 2x3 2행 3열의 가로로 긴 것
```

Out[9]:
```
array([[72683148,        0,        0],
       [       0,        1,        0]])
```

In [10]:
```
np.zeros([2,3])

#0으로 채워진 행렬 리스트가 만들어짐
```

Out[10]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [11]:
```
#(0, 0, 0), (0, 0, 0) 얘를 2by3의 리스트를 만들고 싶으면
#쟤 자체는 계산이 안 되니까 쓸모가 없음, 그래서 계산 가능한 형태로 만드는 것

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

Out[11]:
```
array([[0, 0, 0],
       [0, 0, 0]])
```

In [12]:
```
np.ones([2,3])

#ones, zeros의 default type은 float이니까 1. 이렇게 나올 것
```

Out[12]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [13]:
```
np.ones([2,3], dtype='int')

#이러면 소수점 사라짐!
```

Out[13]:
```
array([[1, 1, 1],
       [1, 1, 1]])
```

In [19]:
```
np.arange(5)
```

In [25]:
```
np.linspace(0,10,6)

#얘는 arange랑 다르게 0, 10 포함, 또은 0,10 포함하여 6개로 똑같이 나눠준다
```

Out[25]:
```
array([ 0.,  2.,  4.,  6.,  8., 10.])
```

In [26]:
```
np.linspace(0,10,7)
```

Out[26]:
```
array([ 0.        ,  1.66666667,  3.33333333,  5.        ,  6.66666667,
        8.33333333, 10.        ])
```

In [31]:
```
x=np.array([[1,2],[3,4],[5,6]])
x
```

Out[31]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [33]:
```
#3차원도 표기가 가능함, 2차원은 대괄호가 위처럼 양옆이 2개씩 붙어있음, 3차원은 3개씩,
#2차원 행렬이 2개 나오면서 3차원이 됨
x=np.array([[[1,2],[3,4],[5,6]],[[7,8],[9,10],[10,11]]])
x
```

Out[33]:
```
array([[[ 1,  2],
        [ 3,  4],
        [ 5,  6]],

       [[ 7,  8],
        [ 9, 10],
        [10, 11]]])
```

In [34]:
```
x.ndim

#3차원이라는 것을 알려줌
```

Out[34]:
```
3
```

In [37]:
```
x.astype(np.float64)

#원래는 int32 data type인데 이걸 바꾸고 싶으면 astype함수를 씀
```

Out[37]:
```
array([[[ 1.,  2.],
        [ 3.,  4.],
        [ 5.,  6.]],

       [[ 7.,  8.],
        [ 9., 10.],
        [10., 11.]]])
```

In [38]:
```
np.zeros_like(x)

#형태를 유지하고 내용물을 0으로 바꾸고 싶을 때: zeros_like함수

#근데 이거는 x*0 이렇게 해도 됨
```

Out[38]:
```
array([[[0, 0],
        [0, 0],
        [0, 0]],

       [[0, 0],
        [0, 0],
        [0, 0]]])
```

In [46]:
```
#numpy 안의 random 패키지 안의 normal이라는 함수
#from numpy import random 이런 식으로도 쓸 수 있음?

data=np.random.normal(0,1,100)
#정규분포 모양의 데이터를 만들어주는 것임!!shape자체를 만들어주는 게 아니라
#normal(mean,standard deviation,data size)
```

In [52]:
```
data=np.random.normal(0,1,100)
print(data)
plt.hist(data, bins=10)
plt.show()
```

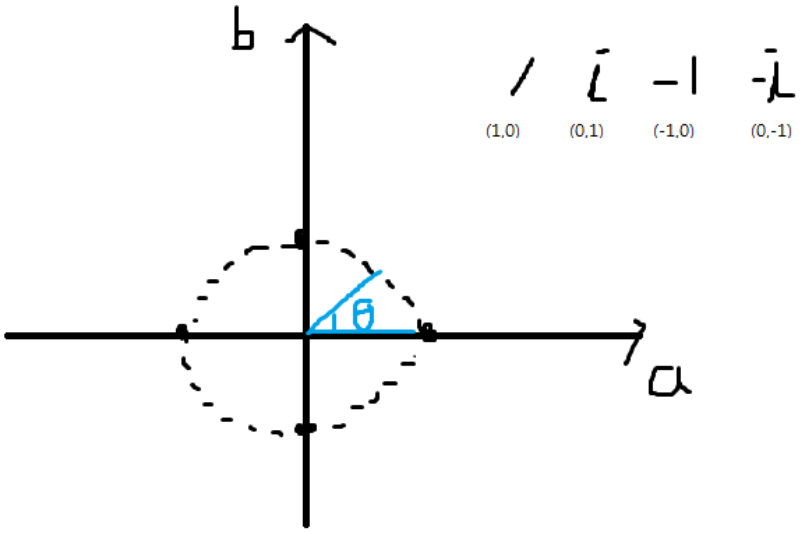
3.Numpy I/O

```
In [59]: a=np.random.randint(0,10,[2,3])
         b=np.random.random([2,3])
         np.savez('test',a,b)

         #test라고 적으면 파일로 저장됨

In [58]: !ls -al test*
         #위의 파일이 만들어졌는지 확인하는것

         'ls'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는
         배치 파일이 아닙니다.

In [60]: del a,b        #variable이 assign 안 된 상태로 없어짐
         %who           #지금 available variable

         data     np     plt     x     y

In [61]: npzfiles=np.load('test.npz')     #이러면 불러올 수 있을 것에 npzfiles는 variable이름이니까 맘대로 바꿔도 됨
         npzfiles.files
Out[61]: ['arr_0', 'arr_1']

In [62]: npzfiles['arr_0']
Out[62]: array([[7, 5, 4],
                [9, 8, 9]])

In [ ]: data=np.loadtxt('regression.csv', delimiter=',',skiprows=1, dtype={'type':('x':'y'), 'formats'})

        #skiprows 1번째 줄 뺀다

        #csv는 comma separated value 콤마로 분류되어 있는 것.
        #regression file을 깃헙에서 가져와서 해볼 것!!!
        #그 파일을 다운 받아서
```

- Phasor: 우리가 아는 것은 사인, 코사인 function. Sin(radian)임. Function input값에 넣는 것은 degree가 아니라 radian임.
  +)0도~2파이(360도). 이 도는 degree라고 함. 각도를 파이로 표현하는 것을 radian이라고 함. Radian은 theta θ 로 표현. Ex. θ=3/2π면 cos값은 0.
  Θ= 0, 1/2π, π, 3/2π, 2π(=0)
  +)사인, 코사인 함수.

+)오일러공식: $\cos(\theta) + \sin(\theta)I = e^{\theta i}$
I는 imaginary. 실수의 반대말, 허수. $\sqrt{-1}$
$F(\theta) = e^{\theta i}$ = 숫자(왜냐면 e도 2.71, i도 루트 마이너스, 전부 수로 표현 가능)
복소수: a+bi 복소수로 우리가 생각하는 모든 수를 표현할 수 있음.
즉, 위의 식 $F(\theta) = e^{\theta i}$도 a+bi로 표현할 수 있음.
Θ= 0, 1/2π, π, 3/2π, 2π ➔ 1, i, -1, -i

- Projection



실수만 보고 싶으면(cos함수) a축에서 왔다 갔다, 허수만 보고(sin함수) 싶으면 b축에서 왔다갔다.

- 소리라는 실체는 각도뿐만 아니라 시간의 개념이 반드시! 들어가 있어야 함.

In [27]:
```
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
```

In [28]:
```
#parameter setting
amp=1
sr=10000
dur=0.5
freq=200.0
```

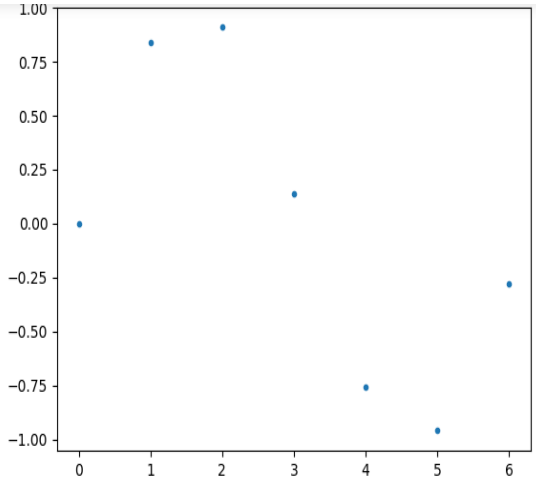
In [29]:
```
theta = np.arange(0,2*np.pi)   #단위는 radian
theta
```

Out[29]:
```
array([0., 1., 2., 3., 4., 5., 6.])
```

In [30]:
```
s=np.sin(theta)
s
```

Out[30]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ])
```

In [18]:
```
fig=plt.figure()
ax=fig.add_subplot(111)      #figure는 화면 전체,
ax.plot(theta, s, '.')       #7개의 theta 값이 있으니 corresponding 값도 7개
                             #s는 sin함수
```



Out[18]: [<matplotlib.lines.Line2D at 0x93012c8>]

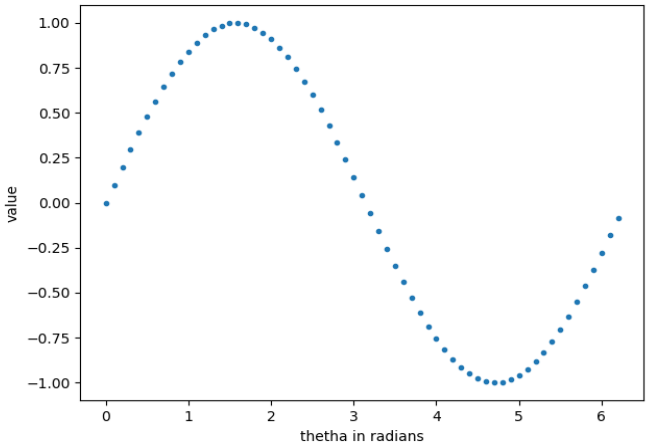
#221= 2행 2열, 1,2,3,4 subplot. 왼쪽 위1, 오른쪽 위2, 왼쪽 아래3, 오른쪽 아래4

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, '.')

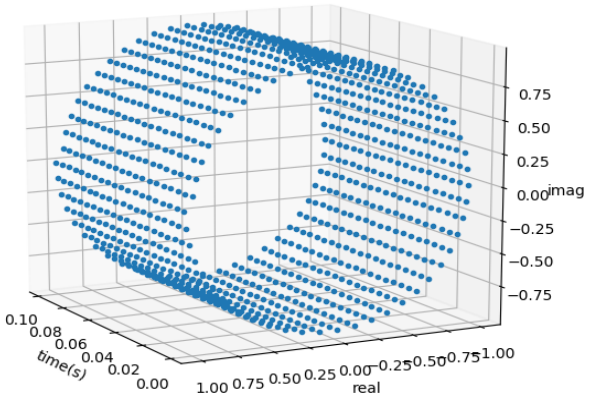
In [34]:
```
#더 빽빽하게 만들려면
theta = np.arange(0,2*np.pi, 0.1)   #위에거는 np.arnage(0,2*np.pi) --> 1 단위로 쪼갬, 바꾼 식은 0.1씩 쪼갬
theta
```

Out[34]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
```

In [35]:
```
s=np.sin(theta)

fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(theta,s,'.')
ax.set_xlabel('thetha in radians')
ax.set_ylabel('value')
```



In [45]:
```
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')
```



<11.12&11.14>

In [14]:
```
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
```

In [15]:
```
#parameter setting
amp=1                       #진폭
sr=10000
dur=0.5
freq=440.0
```

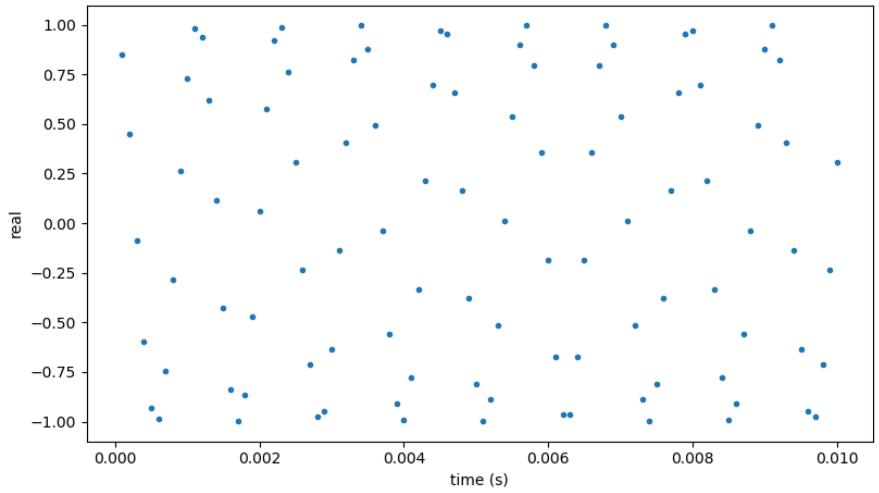
In [16]:
```
#t=np.arange(1,sr*dur)  애는 시간 개념을 먼저 넣어준 것. 근데 여기는 time tic을 준 거고, '몇 초'로 만들려면 sr로 나눔
t=np.arange(1,sr*dur+1)/sr     #마지막것 포함해줬고 더하기 일 함으로서, sr로 나눠주면서 초 개념 생성
```

In [21]:
```
#generate signal by cosine-phasor
s=amp*np.cos(theta)
#sin,cos 둘다 소리 똑같음. 귀가 phase(각도)에는 sensitive하지 않다. 우리는 phase shift 인식하지 못함.
#우리는 frequency에 민감
```

In [22]:
```
theta=t*2*np.pi*freq     #freq는 몇 바퀴인지
```

In [20]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(t[0:100],s[0:100],'.')      #0~1000 안 정하면 너무 빽빽해서 태극문양이 안 보임
ax.set_xlabel('time (s)')
ax.set_ylabel('real')

#2차원 벡터로 표현. 여기있는 점들의 개수는 1000개.
```

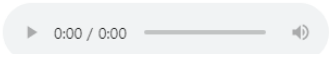
```
<IPython.core.display.Javascript object>
```



Out[20]: Text(0, 0.5, 'real')

In [23]:
```
ipd.Audio(s,rate=sr)
```

Out[23]:

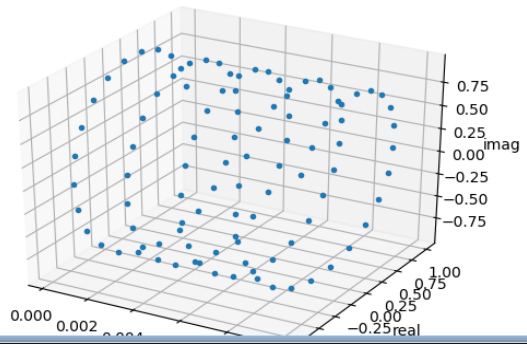



```
c=amp*np.exp(theta*1j)          #np.exp = e라고 생각하면 됨. 오일러식에 나오는, 1j는 i임. 허수 뜻하는.
c
```

Out[24]:
```
array([0.96202767+2.72951936e-01j, 0.85099448+5.25174630e-01j,
       0.67533281+7.37513117e-01j, ..., 0.85099448-5.25174630e-01j,
       0.96202767-2.72951936e-01j, 1.        +3.13806691e-14j])
```

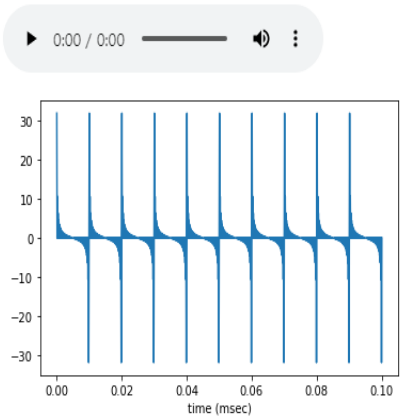
In [26]:
```
fig=plt.figure()
ax=fig.add_subplot(111, projection='3d')
ax.plot(t[0:100],c.real[0:100],c.imag[0:100], '.')
ax.set_xlabel('time(s)')
ax.set_ylabel('real')
ax.set_zlabel('imag')
```

<IPython.core.display.Javascript object>



In [13]:
```
F0=100
Fend = int(sr/2)    #nyquist frequnecy
s=np.zeros(len(t))
#time은 위에서 만들었으니 이미 만들었다고 가정.
for freq in range(F0, Fend+1, F0):
    theta = t*2*np.pi*freq
    tmp=amp*np.sin(theta)
    s=s+tmp
    #제일 처음 s가 무엇인가 정의를 해줘야 함. s*tmp니까 저 s가 뭔지 정의되어야 저 수식의 첫 s가 나옴.
    #그게 위에 있는 s=np.zeros(lent(t)) 얘의 의미
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)

#얘는 spectrum이 아님.(녹음)
```

Out[13]:



In [29]:
```
def hz2w(F,ar):
    NyFreq = sr/2
    w = F/NyFreq *np.pi
    return w                    #출력

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
```

In [30]:
```
RG=0
BWG=100
a,b=resonance(sr,RG,BWG)        #Bandwidth 산맥이 얼마나 뚱뚱하냐, 뾰족하냐
s=lfilter(b,a,s,axis=0)
ipd.Audio(s,rate=sr)

#decreasing한 효과가 있음.
```

Out[30]:

In [31]:
```
RG=500
BWG=60
a,b=resonance(sr,RG,BWG)        #Bandwidth 산맥이 얼마나 뚱뚱하냐, 뾰족하냐
s=lfilter(b,a,s,axis=0)
ipd.Audio(s,rate=sr)
```

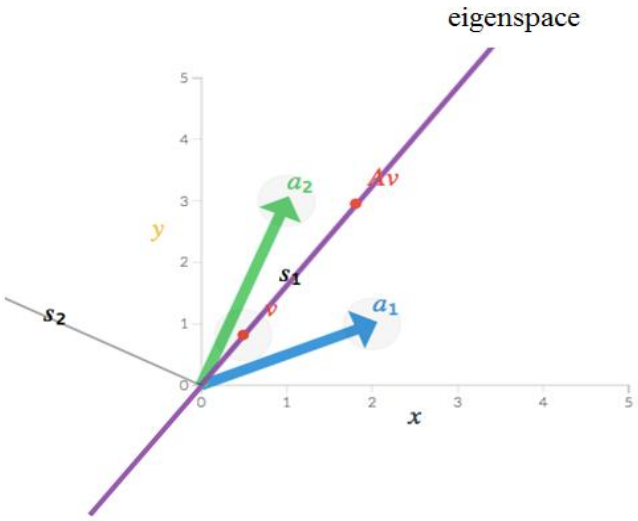
Out[31]:

Linear Algebra

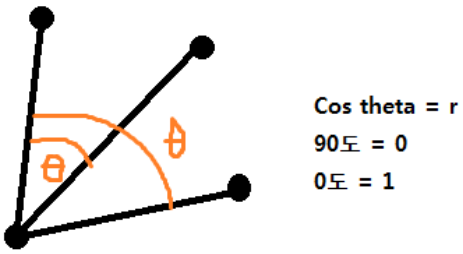
- <데이터> ((((기계: 인공지능)))) <데이터>
  벡터 함수(행렬) 벡터
  Ex. 음성 인식, 음성 합성, 기계 번역 등
- $\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$ M by n 행렬
- To become a vector space: linear combinations(C*V + D*W) still stay in the space
- 무수히 많은 여러 벡터들이 만들어내는 공간을 'vector space'라고 함
- N차원의 공간은 n개의 컴포넌트로 이루어진 벡터로 채워진 것
- 칼럼 벡터들의 리니어 컴비네이션으로 만들어 내는 공간 (= 칼럼 벡터가 스패닝 해서 만들어내는 공간). 원점만 있으면 0차원
- col1 & 2 NOT on a line -> independent
- Whole space: 벡터 자체가 갖고 있는 스페이스.
  Column space: 스패닝 했을 때
- col1 & 2 on a line -> dependent: 라인만 확장되고 스페이스 전부 채울 수 없음
- $A = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 4 & 1 & 5 \end{bmatrix}$ 칼럼3 = (칼럼1*1)+(칼럼2*1). Independent는 1,2뿐
- Null space: whole이 3차인데, column space가 1차. 남은 2차가 null space
- Spanning해서 만들어진 space의 수직이 되는 orthogonal한 것을 null space라고 부름
- Linear Transformation: Ax = b $\begin{bmatrix} 0.9 & -0.4 \\ 0.4 & 0.9 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.3 \end{bmatrix}$
- Detransformation: inverse matrix 역행렬: $A^{-1}b = x$

  $\begin{bmatrix} 0.9 & -0.4 \\ 0.4 & 0.9 \end{bmatrix}^{-1}\begin{bmatrix} 0.5 \\ 1.3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
- 하지만 invert 할 수 없는 것도 있음
- Eigenvector
  given matrix의 아이겐벡터는 매트릭스의 transformation 하고 난 후의 것과 하기 전의 점이 일직선상에 놓임
  Av=b; A transforms v to b

- Among all v, some v is parallel to Av: Eigenvector



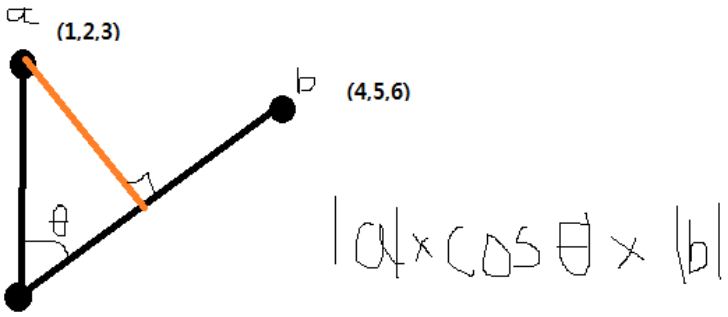
- 어떤 입력이 들어오든 출력에 영향을 미치지 않는: null space
- Null space에 대한 수학적 / 기하적 / 응용적 해석
- http://setosa.io/ev/eigenvectors-and-eigenvalues/
- 벡터는 방향임. 어떤 값만 존재한다는 느낌보다는(그래도 되지만) 방향이라고 생각해야 함. eigenvector space는 그 라인 전체를 말함(보라색 선)
- 상관관계(Correlation): '같이 간다'. -1 <= r <= 1. 상관관계는 0일 때 가장 낮음. 완전한 선상에 있을 때 -1 or 1이 나옴(기울기는 무관). 완벽하게 동그라미 모양일 때 r=0.



- Inner product

  [1 2 3]

  [4 5 6] 두 벡터가 있을 때 1*4 + 2*5 + 3*6 안 쪽으로 곱해서 다 더하기



Correlation이 많이 있으면 inner product가 적게(?) 나옴

|a| = $\sqrt{x^2 + y^2 + z^2}$

a•b / |a|*|b| = cos θ

- A*v (transformed된 결과) = 상수*v

  만족하는 v가 eigenvector, 만족하는 람다(상수)가 eigenvalue

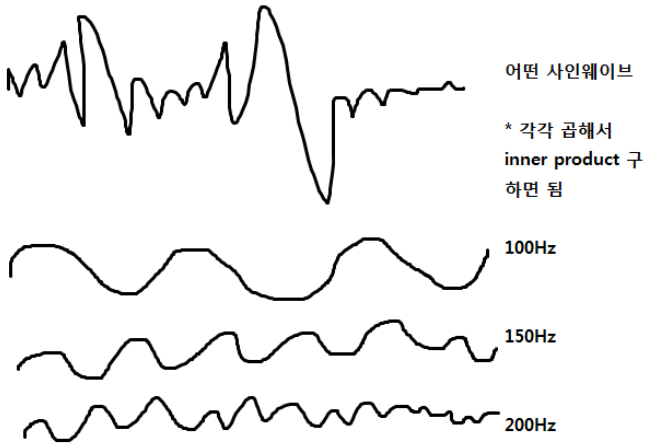
- a•b = 1by1의 결과가 나옴 (inner product)
- cos similarity(=cos θ): a와 b가 얼마나 비슷한지를 수치적으로 이야기해줄 수 있는 방식
- 두 사인 코사인 그래프는 90도 차이인데, 그 둘의 inner product가 0이 됨.

  Phase shift에 민감하게 작용함. → 이것을 덜 민감하게 만들기 위해서 sin, cos가 아닌 complex phasor를 사용할 것임

- Dot product의 값은 각도가 0에 가까울수록 커지고, 90에 가까울수록 작아진다
- 두 벡터가 얼마나 가까운지 cos similarity
- 웨이브 속에 어떤 frequency 성분이 많은지 아는 것이 중요함: spectral analysis

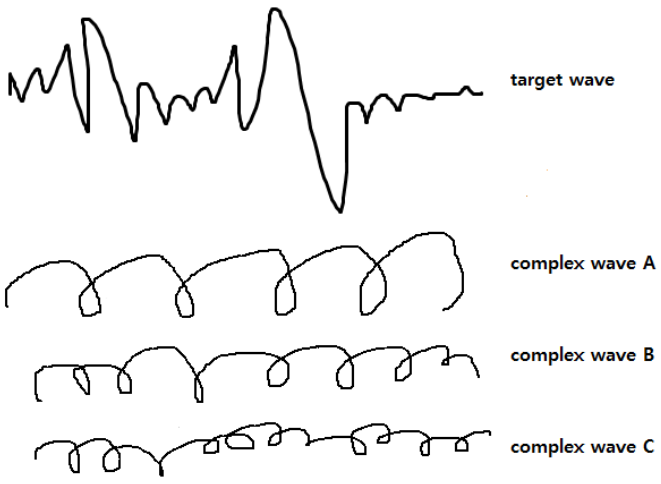
  한 시점에서의 analysis: spectrum

- 사인 웨이브

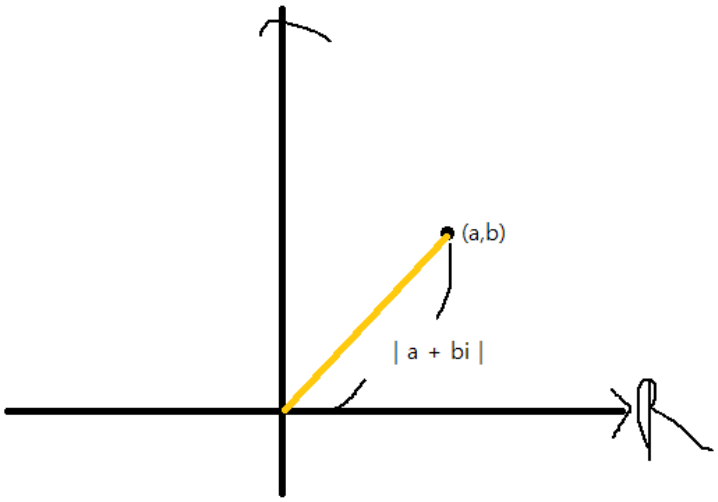


- 주의해야 할 점: target wave에 대해서 여러 웨이브를 만들어서 probe를 하는데 target이 shift하면서 phase에 따라 sensitive하게 바뀜

  ➔ phase에 대해서 sensitive하지 않은 complex를 쓰기로 함. 그래서 probe한 숫자가 허수 complex number가 될 수 있음



Plotting이 불가능함 ➔ |a+bi| 절댓값을 씌워서 실수로 만듦: 원점에서부터 벡터 값 까지의 거리

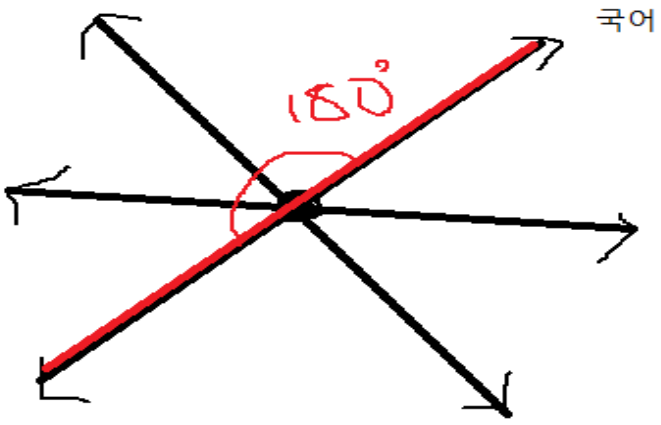


- Cos(θ) =r

  R=1이면 일직선 상에 있는 것

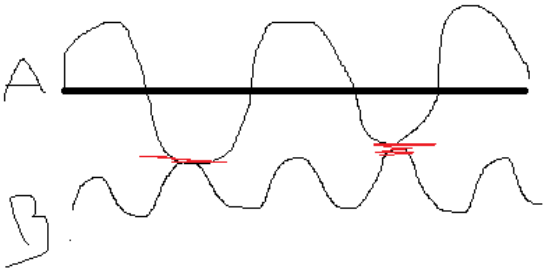
  영어, 국어, 수학, 사회문화, …

  영어와 국어의 r=1이면 (일직선상, 양의 기울기) 두 각이 180도임.

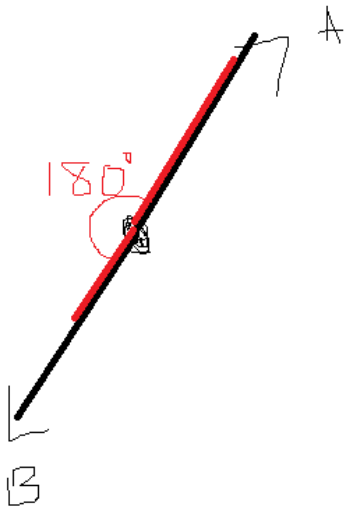


영어

- 이런 경우도 있음



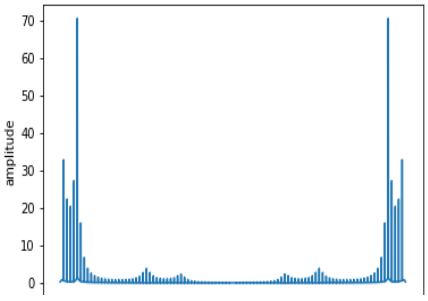
-1인 것과 1인 부분이 만나도 여전히 180 도임. 길이는 진폭의 크기도 관련 있음



In [19]:
```
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))              #complex phasor를 이용한 라인
                                                            # (exponential) (e^wi)^[0 ... 100 벡터]
                                                            # 첫 번째, omega w = (2π * 0 [0,..,100]/100)i
                                                            # 두 번째, omega w = (2π * 1 [0,.100]/100)i
                                                            #                  = [0,..,2π]i: 결국 한 바퀴 도는 것, 한 바퀴를 100개의 샘플로
                                                            # 세 번째: (2pi * 2 [0,.100]/100)i = [0,..,4π]i
    amp.append(np.abs(np.dot(s,z)))                         #amp는 루프의 횟수만한 크기: len(amp)
                                                            #amp에는 허수가 없음, absolute(절대값)을 했기 때문
```

In [20]:
```
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')

#bar의 개수와 sample의 개수가 같음
#왼쪽과 오른쪽이 대칭: 5000까지 (half)만 의미가 있음, nyquist frequency
#해당 freq의 에너지가 얼마나 있는지
```

Out[20]:
```
Text(0,0.5,'amplitude')
```



define helper function

In [21]:
```
def frame_signal(signal, srate, win_size, win_step):
    frame_size = int(win_size * srate)
    frame_step = int(win_step * srate)

    # Get number of frames
    num_frames = int(np.ceil(np.abs(len(signal) - frame_size) / frame_step))
    # Pad signal
    pad_len = num_frames * frame_step + frame_size
    pad = np.zeros((pad_len - len(signal)))
    signal_padded = np.append(signal, pad)
    # Get within-frame sample indices
    idx1 = np.tile(np.arange(0, frame_size), (num_frames, 1))
    # Get vectors of frame_step increments
    idx2 = np.tile(np.arange(0, num_frames * frame_step, frame_step),
                   (frame_size, 1)).T
    # Get total indices divided by each frame
    indices = idx1 + idx2
    # Get frames divided by each frame based on indices
    frames = signal_padded[indices.astype(np.int32, copy=False)]
#     if cutoff is not None:
#         cutoff_bin = round(cutoff * nfft / (srate))  # hz -> bin
#         frames = frames[:, :cutoff_bin]
    return frames

def get_window(win_size, srate, win_type, win_samp=None):
    '''Get window samples for win_size'''
    if win_samp is None:
        win_samp = int(win_size * srate)  # sec -> sample

    if win_type == 'rect':
        return np.kaiser(win_samp, 0)
    elif win_type == 'hamming':
        return np.hamming(win_samp)
    elif win_type == 'hanning':
        return np.hanning(win_samp)
    elif win_type == 'kaiser':
        return np.kaiser(win_samp, 14)
    elif win_type == 'blackman':
        return np.blackman(win_samp)

def plot_spectrogram(S):
    fig, ax = plt.subplots(facecolor='white', figsize=(14, 6))
    im = ax.imshow(S.T, aspect='auto', origin='lower', cmap=plt.get_cmap('Greys'))
    divider = make_axes_locatable(ax)
    cax = divider.append_axes('right', size='5%', pad=0.05)
    cbar = fig.colorbar(im, cax=cax)
    return fig, ax

def preemphasis(s, pre_emp=0.97):
    # Emphasize high frequency range of the waveform by increasing power(squared amplitude).
    s = lfilter([1, -pre_emp], [1], s)
    return s
```

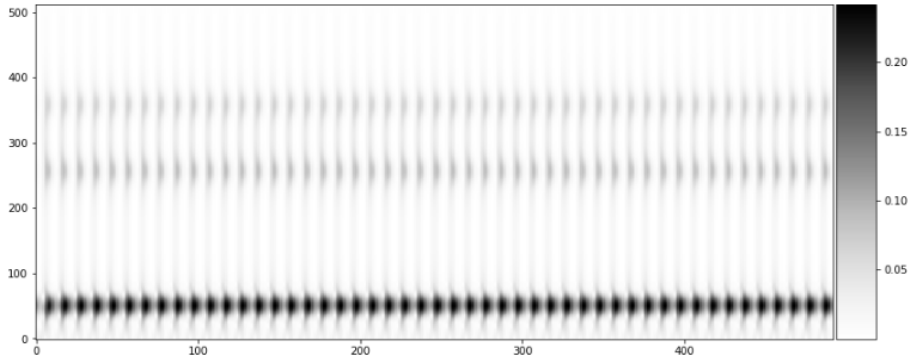
Preprocessing Signal

In [22]:
```
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)
```

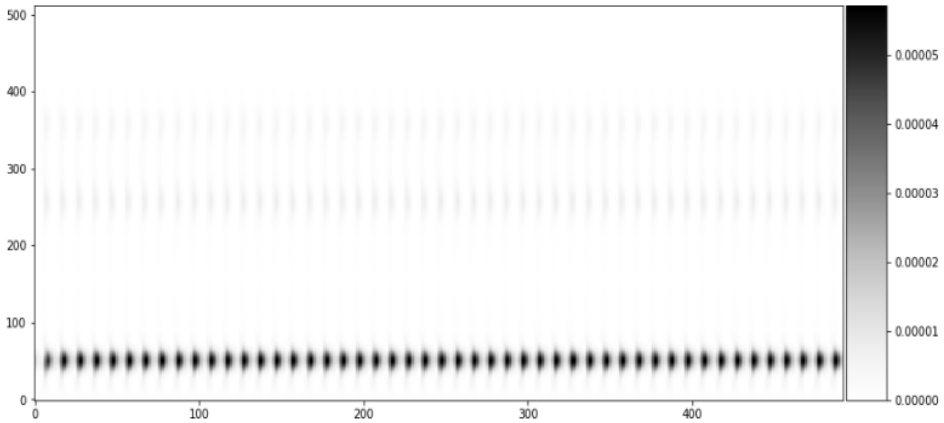
```
frames: (492, 80)
```

In [23]:
```
magspec = np.abs(np.fft.rfft(frames, n=nfft)) # frames x (nfft//2 + 1)
plot_spectrogram(magspec)
```

Out[23]:
```
(<Figure size 1008x432 with 2 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0xa750eb8>)
```



In [24]:
```
powspec = 1/nfft * (magspec**2)
plot_spectrogram(powspec);

#연한 부분은 1보다 작은 값이 나오고 진한 부분은 1보다 큰 값이 나옴
#이 값에 어떤 제곱값을 취함. 그 다음 로그 처리함. 그럼 reasonable한 숫자가 나옴. ex) 0.01 -> log10^-2 = -2
```



In [25]:
```
logspec = 10 * np.log10(magspec) # dB scale
plot_spectrogram(logspec);
```