스트링

```
s1 = ['a' 'b' 'c' 'd' 'e']
s1 = 'abcde'
```

- whos 변수 : 변수의 name, size, bytes, class, attributes가 나온다.

```
>> whos s1
  Name        Size                Bytes  Class     Attributes
  s1          1x5                    10  char
>> s(1)
ans = 'a'
```

변수 및 벡터

```
>> a = [10 20 30 40]
a == 10     20     30     40

>> b = [2 2 2 2]
b = 2      2      2      2

>> a.*b
ans ==   20     40     60     80

>> a = [1 : 10 ] // 1부터 시작해서 10까지
a == 1      2      3      4      5      6      7      8      9          10

a=[1: 5: 10] // 1부터 시작해서 5씩 증가해서 10까지
a == 1  6
```

- 영행렬 만들기

```
zeros(3)
      ans =
           0      0      0
           0      0      0
           0      0      0
```

- 기본행렬 만들기

```
 eye(2)
        ans =
        1      0
        0      1
```

셀 만들기

```
A = [1 2 ; 3 4]
A =
     1     2
     3     4

>> c = {1}
c =
  1×1 cell 배열
    {[1]}

>> whos c
  Name          Size                Bytes  Class     Attributes

  c             1x1                   112  cell
>> c = {'abcde'}
c =

  1×1 cell 배열
    {'abcde'}

>> whos c
  Name          Size                Bytes  Class     Attributes
  c             1x2                   226  cell
>> c{3} = {1 2 ; 3 4}

c =

  1×3 cell 배열

    {[1]}    {'abcde'}    {2×2 cell}

>> s_list={s1 ; s2; s2}

s_list =

  3×1 cell 배열

    {'youngsoo'}
    {'chulsoo' }
    {'chulsoo' }

>> whos s_list
  Name            Size                Bytes  Class     Attributes
  s_list          3x1                   356  cell
```

스트럭

```
>> car(1).company = 'company a'

car =

  다음 필드를 포함한 struct:

    company: 'company a'

>> car(1).color = 'white'

car =

  다음 필드를 포함한 struct:

    company: 'company a'
      color: 'white'

>> car(1).year = 2019
car =

  다음 필드를 포함한 struct:

    company: 'company a'
      color: 'white'
       year: 2019
>> car(1).type ='sedan'

car =

  다음 필드를 포함한 struct:

    company: 'company a'
      color: 'white'
       year: 2019
       type: 'sedan'
```

size, length

length(a) : a의 길이를 구하여라
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
ans = 10

size(a) : a의 size를 구하여라 가로줄의 길이, 세로줄의 길이
ans = 1 10

a = [1 2 3 ; 4 5 6]
length(a)
ans = 3

size (a)
ans = 2 3

plot

- y= ax +b 그래프 그리기
x = [0 1 2 3 4 5 6 7 8 9 10]
y = [0 2 4 6 8 10 12 14 16 18 20]
plot(x, y) => y = 2x 그래프

x = [0 : 1: 10] // 0부터 1씩 증가해서 10까지
y = 10 * x // x에 10씩 곱한 벡터
plot(x,y) => y = 10x 그래프

xlabel('x-axis') // x축 제목
ylabel('y-axis') // y축 제목

- 여러 개 그래프 그리기
x1 = [0 : 1 : 10]
y1 = 2 * x

x2 = x
y2 = 2*x –5
plot(x1, y1, x2, y2) // y1 = x1 , y2 = x2 그래프를 그린다.

- 그래프에 주석 달기
legend ('f(x1) = y1(x1)' , 'f(x2) = y2(x2)')

- 그래프에 라인 추가
grid on

- 2차원 그래프 생성
x = -10 : 0.1 : 10;
y = x.^2 ;
plot(x, y, 'r') // y=x^2 그래프를 'red' 색으로 그려라

- sin, cos 그래프 생성
x = 0 : 0.1 : 10;
y_sine = sin (x) ;
y_cosine = cos (x) ;
plot (x, y_sine, x, y_cosine) // sin(x) , cos(x) (0<=x<=10) 생성

x = 0: 0.1 : 2*pi;
y = sin (x) ;
plot (x, y) // sin(x) (0<=x<=2pi) 생성

Comparison 비교

2 == 3 (2와 3은 같습니까?)
ans = 0 // 0(False)

2 == 2
ans = 1 // True

1 ~= 1 (1과 1은 같지 않습니까?)
ans = 0 // False

1 ~= 2 (1과 2는 같지 않습니까?)
ans = 1 // True

1 > 2 (부등식)
ans = 0 // False

Find

- find() : 벡터의 위치를 알려준다.

find([100 200 300])
ans = 1 2 3

find([0 100 200 300])
ans = 2 3 4

find ([100 200 0 300])
ans = 1 2 4

x = [100 200 0 300]
index = find(x)
index == 1 2 4

x (index)
ans == 100 200 300

find(x == 100) // x == 100과 같은 자리를 찾아라
ans == 1

```
find(x == 300) // x == 300과 같은 자리를 찾아라
ans == 4

find(x ~= 0) // x ~=0인 자리를 찾아라
ans == 1   2   4
```

For Loop 반복문

- for문 형식
- for I = start # : increment: end #

```
        statement
end
```

- 일반 for문

```
for a = 1 : 1 : 10
        a
end // a가 1부터 1씩 증가해서 10까지 반복.
```

- 이중 for 문

```
for I = 1 : 1 : 9
        for j = 1 : 1 : 9
                disp(i*j) // I * j를 display
        end
end // i가 1부터 1씩 증가해서 9까지 반복, j가 1부터 1씩 증가해서 9까지 반복
```

If Statement 조건문

- if문 형식

```
if expression
        statement
end
else if expression
        statement
end
else
        statement
end

if find(strcmp(name,list))
        disp('Invited Guest')
else
        disp('NOT Invited Guest')
end
```

Logical Operation

- 논리연산

```
true & true // true and true
ans == 1

true | true // true or true
ans == 1

true & false // true and false
ans == 0

true | false // true or false
ans == 1

false & false // false and false
ans == 0

2 >1 & 3 >1 // (2 > 1 == True) & (3 > 1 == True)
ans == 1

2 < 1 & 3 > 1 // (2 < 1 == False) & (3 > 1 == True)
ans == 0
```

While Loop

- while 문 형식

```
while expression
        statement
end // expression == True인 동안 statement 진행
// 사용 할 때 : 정확히 반복문이 언제 끝나는지 모르는 경우

balance = 10000;
objective_balance = 50000;
interest_rate = 0.1;
counter = 1;
while balance < objective_balance
        balance = balance + balance * interest_rate
        counter = counter + 1
        pause(3)
end // balance < objective_balance인 동안 수행
```

Switch문

- switch 문 형식

```
switch variable
        case expression 1
                statement
        case expression 2
                statement
        otherwise
                statement
end // variable가 case expression과 같다면 해당 statement를 실행한다.
```

- 예시 1

```
num1 = 1;
num2 = 2;
operation = 'add';
switch operation
        case 'add' // operation ='add'인경우
                num1 + num2 // 덧셈
        case 'subtract' // operation ='subtract'인경우
                num1 – num2 // 뺄셈
        case 'multiply'
                num1 * num2
        case 'divide'
                num1 / num2
end // operation의 case 값에 맞게 코드를 실행한다.
```

- 예시 2

```
num1 = 1;
num2 = 2;
operation = 'add2';
switch operation
        case {'add', 'add2'} // operation ='add', 'add2'인 경우
                num1 + num2 // 덧셈
        case 'subtract' // operation ='subtract'인경우
                num1 – num2 // 뺄셈
        case 'multiply'
                num1 * num2
        case 'divide'
                num1 / num2
```

텍스트 프린트

- disp => display

```
disp('string') // 문자열 한 개를 받아서 출력
```

- fprintf => file print formatted

```
fprintf(string) // string변수 출력
fprintf('string %d %d %d', var1, var2, var3) //  string var1 var2 var3 출력
```

- sprintf => string print formatted

텍스트 아웃풋

- fid = fopen('file.txt','w'); // file.txt 쓰기 권한으로 실행
- fprintf(fid,'I have 3 apples₩n'); //file.txt 에 'I have 3 apples₩n' 입력
- fclose(fid); //file을 닫아줌.

행렬과 선형대수

prime(x) : x의 소수를 나타내어라
A = [2 3 5 ;
7 11 13;
17 19 23]
- A(end, :) : A의 마지막 행의 전체 열
ans == 17 19 23

- A(2:3, 2:3) : A의 2~3번째행의, 2~3번째열
ans =
11 13
19 23

- A(3:-1:1,end) // 3행을 선택해라 -1칸씩 띄어서 선택해라 1행까지, end(데이터의 끝부분을 선택)
ans =
23
13
5

- A(:,:) // A행렬의 모든 열과 행을 보여주어라.
- A(:) // A행의 모든 인덱스를 열형태로 보여주어라.
- size(A) // A행의 행 열을 출력하라.
ans == 3 3

- size(A,1) // A의 행의 개수를 출력하여라.
ans == 3

- size(A,2) // A의 열의 개수를 출력하여라.
ans == 3

- zeros(x) // x행 x열을 0으로 모두 채워라.
- zeros(x,y) // x행 y열을 0으로 모두 채워라.
- A([x:y],[x:y]) // A의 x행부터 y행까지, x열부터 y열까지에 포함되는 행렬을 나타내어라.
- A(2,[2:4]) // A의 2행과 A의 2열부터 4열까지에 포함되는 행렬을 나타내어라.
- magic(x) // x행 x열을 가진 행렬을 나타내어라.
- A(x, :) = [] // A의 X행을 제외한 나머지를 나타내어라.
- primes(x) // x이하의 모든 소수들을 출력하여라.
A = primes(23)
A =
2 3 5 7 11 13 17 19 23

rand와 randn사용 불규칙 행렬생성
- rand(x) // x행 x열의 랜덤값이 들어간 행렬 생성
rand(3)
ans =
0.0357 0.6787 0.3922
0.8491 0.7577 0.6555
0.9340 0.7431 0.1712

특정행렬 반복 행렬 만들기
B = [1 2 ; 3 4]
- repmat(B,2) // B가 2*2번 반복되는 행렬 생성
ans =
1 2 1 2
3 4 3 4
1 2 1 2
3 4 3 4

블록 대각행렬을 만들기.
- blkdiag(eye(2)) // eye(2)인 블록대각행렬 생성
ans =
1 0
0 1

- blkdiag(eye(2),ones(2)) // eye(2), ones(2)가 포함된 블록대각행렬 생성.
=> [1 0 0 0 ; 0 1 0 0 ; 0 0 1 1 ; 0 0 1 1]
ans =
1 0 0 0
0 1 0 0
0 0 1 1
0 0 1 1

- diag(x) x리스트를 대각행렬로 표현
x = [1 2 3]
- diag(x)
ans =
1 0 0
0 2 0
0 0 3

행렬의 생성과 원소 – 특수행렬의 생성
- Toeplitz, Hankel, Gallery 행렬을 만들어라.
toeplitz([1 0 -1 -2], [1 2 4 8])
ans =

1 2 4 8
0 1 2 4
-1 0 1 2
-2 -1 0 1

```
hankel([3 1 2 0 ], [0 -1 -2 -3])
ans =
     3     1     2     0
     1     2     0    -1
     2     0    -1    -2
     0    -1    -2    -3

gallery(3)
ans =
  -149   -50  -154
   537   180   546
   -27    -9   -25
```

배열 연산 A*x = B를 만족하는 x를 구하여라

- x = A^(-1)*B

```math
A = [1 2 ; 3 4]
B = [1 1 ; 1 1]
x = inv(A)*B
```

배열 연산(차원이 다른 두 행렬의 kronecker 곱)

```
A = [1 10; -10 100]
A =
     1    10
   -10   100
B = [1 2 3; 4 5 6; 7 8 9]

B =
     1     2     3
     4     5     6
     7     8     9

kron(A,B)

ans =
     1     2     3    10    20    30
     4     5     6    40    50    60
     7     8     9    70    80    90
   -10   -20   -30   100   200   300
   -40   -50   -60   400   500   600
   -70   -80   -90   700   800   900
```

행렬내외부 원소에 대한 연산

- sqrt(A) : A의 내부 원소에 대해 루트를 씌운다
- sqrtm(A) : A의 행렬에 대해 루트를 씌운다.

성긴 행렬(Sparse Matrices)
- 0이 아닌 원소들의 수를 계산
- sparse() : 0이 아닌 원소들을 0을 표현하지 않고 표현하기 위한 함수
- nnz(A) : A의 0이 아닌 원소들의 개수

```
A = sparse([1 2 2 4 4], [3 1 4 2 4], 1:5)
# sparse() 형태 0을 제외한 숫자가 들어가 있는 행렬을 표시.
A =
   (2,1)        2 // 위치, 값
   (4,2)        4
   (1,3)        1
   (2,4)        3
   (4,4)        5

B = full(A)
# A의 full 행렬 보여주기
B =
     0     0     1     0
     2     0     0     3
     0     0     0     0
     0     4     0     5
```

선형 방정식 시스템
- 선형방정식의 해를 구하여라.

```
x1 + 2*x3 = 3;
3*x1 + 4*x2 +x3 =1;
-2*x1 + x2+ 3*x3 =2;
A = [1 0 2; 3 4 1; -2 1 3] // 식
B = [3 1 2] // 값
x=A₩B
x =
    0.5758 // x1
   -0.4848 // x2
    1.2121 // x3
```

연립 방정식의 풀이

1. 가감법
- 미지수를 없애는 풀이 방법

2. 계수만을 이용한 풀이
- 연립일차방정식을 첨가행렬로 나타내어 풀이

- 풀이 예시

```
X + 2Y = 3
-X -3Y = -1
```

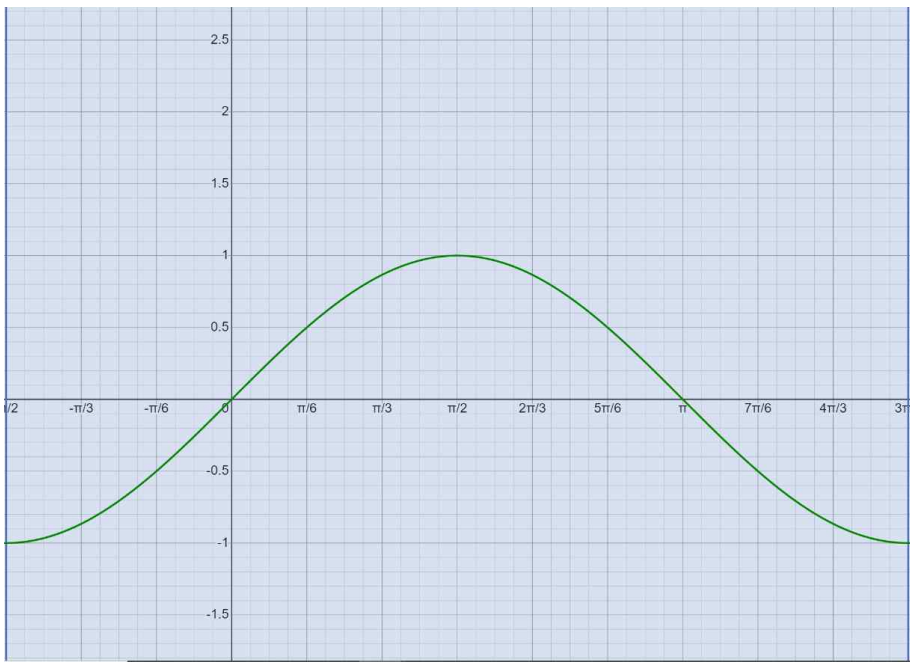
```
syms x y
S1 = solve('4*x+y=6','2*x+y=4')
S1 = [S1.x, S1.y]

S1 = [1, 2]

syms x y
S1 = solve(4*x+y==6, 2*x+y==4)
S1 = [S1.x, S1.y]

S1 = [1, 2]

A = [1 2 3; -1 –3 –1]; // X + 2Y =3 / -X –3Y = -1
rref(A)`
1 0 7
0 1 -2
X = 7 / Y = -2
```

연속 신호와 이산 신호

1. 연속(시간) 신호 x(t)
- 시간에 대해 끊어지지 않는, 즉 모든 시간에 대해 정의되는 신호
- ex) 전압, 음성, 심전도

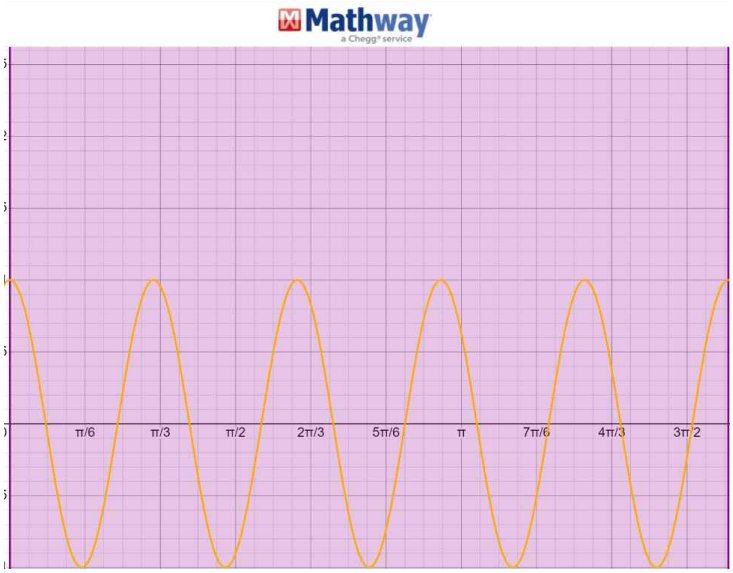
2. 이산(시간) 신호 x[n]
- 띄엄띄엄 특정한 시간에서만 정의되는 신호
- ex) 상품의 분기별 판매량, 매일의 주식 가격, 일병 최고온도.

예제
다음의 신호를 그리고, 신호의 길이를 구하라.
(a) x(t) = sin(0.5*pi*t), (-1<=t<=3) / x(t) = 0, (그 외)



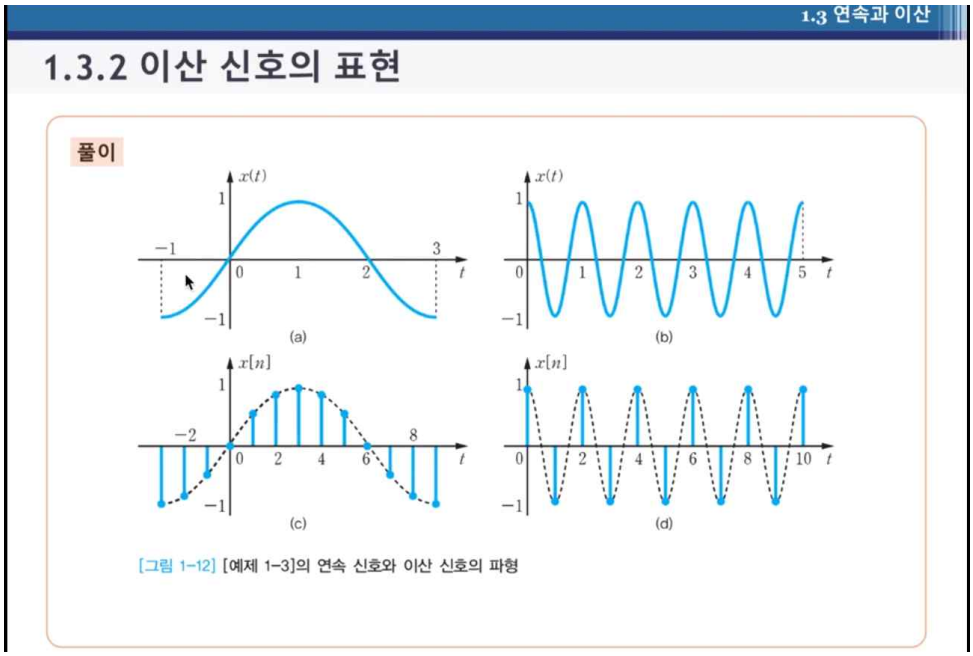
길이 = 4

(b) x(t) = cos(2*pi*t) (0<=t<=5) / x(t) = 0, (그 외)



길이 : 5

(c) (a)의 x(t)를 간격 1/3초로 샘플링한 이산 신호 x[n]
(d) (b)의 x(t)를 간격 0.5[초]로 샘플링한 이산 신호 x[n]



3. 신호의 길이
- 연속 신호 : 신호 값이 존재하는 시간 구간의 길이 L로 정의
x(t)가 t1<= t < t2 에서 정의 -> L = t2 - t1

- 이산 신호 : 신호를 이루는 샘플의 개수 N으로 정의
x[n]이 N1 <= n <= N2 에서 정의 -> N = N2 - N1 +1

4. 유한 구간 신호와 무한 구간 신호
- 유한 구간 신호 : 시간적으로 유한한 길이를 갖는 신호
- 무한 구간 신호 : 시간적으로 무한한 길이를 갖는 신호

5. 이산 신호의 표현
- 값을 발생 순서대로 늘어놓은 단순한 수열
- 이산 신호의 시간 값 n은 항상 정수

6. 연속 신호의 샘플링에 의한 이산 신호
- 샘플링 : 연속 신호에 대해 특정한 시간 간격으로 값을 취함.
- 샘플링의 결과로 이산 신호를 얻게됨.

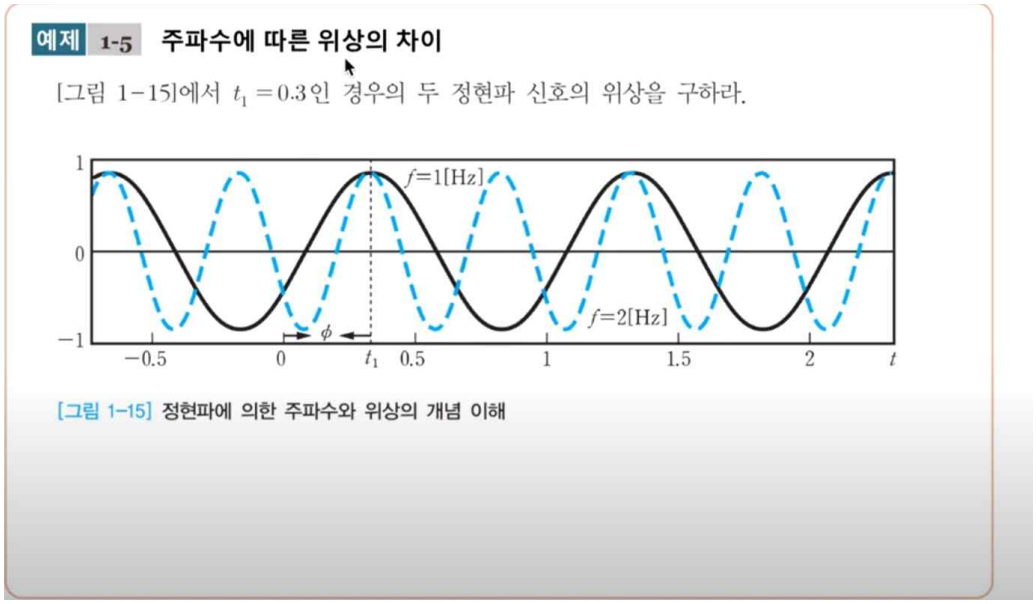
7. 연속 시스템과 이산 시스템
- 연속 시스템 : 연속 신호 입력에 대해 연속 신호 출력을 내는 시스템
- 이산 시스템 : 이산 신호 입력에 대해 이산 신호 출력을 내는 시스템

8. 정현파
- 등속 회전 운동체의 위치를 시간에 대해 그린 파형을 갖는 신호
- 진폭(A), 위상(Φ), 주파수(w/f)의 3가지 요소에 의해 완전하게 정의됨
- x(t) = Acos(wt+Φ)

9. 정현파의 진폭, 위상, 주기, 주파수의 정의
- 진폭 : 정현파가 진동하면서 변할 수 있는 값의 범위
- 위상 : 각으로 표시된 정현파의 출발 위치
- 주기 : 정현파가 같은 파형을 반복하는 (최소) 시간 간격
- 각주파수 : 정현파가 1초에 이동할 수 있는 radian 각

10. 주파수의 물리적 의미
- 주파수는 신호의 시간적인 변화 속도와 관련됨
- 주파수가 높을수록 신호 파형이 시간적으로 더 빨리 변함
- 파형이 더 빠른 변화를 보이는 신호에는 더 높은 주파수 성분이 포함됨.

예제 t =0.3인 경우의 두 정현파 신호의 위상을 구하라.



f1 = 1[Hz] , f2 = 2[Hz]
주파수는 주기의 역수
f1의 주기 T1은 1의 역수인 1이다. f2의 주기 T2는 2의 역수인 1/2이다.
w1 = 2*pi/1 / w2 = 2*pi/(1/2)
x(t) = Acos(wt + Φ)
f1, f2의 A = 1 //진폭 => 진동의 중심에서 최대변위의 크기
1 = 1*cos(2*pi/1*0.3 + Φ), 1 = 1*cos(2*pi/(1/2)*0.3 + Φ)
1 = 1*cos(0.6*pi + Φ), 1 = 1*cos(1.2*pi + Φ)
cos(x)가 1인 x의 값의 경우는 x = 0
Φ(1) == -0.6pi, Φ(2) == -1.2pi

함수 만들기

- 일차방정식 함수 만들기

```
function y = linear_equation(x, slope, y_intercept) // x, x계수, y절편
y = slope.*x + y_intercept; // f(x) = slope .*x + y_intercept
linear_equation([1 2 3], 2, 0) // x벡터, x 계수, y절편
ans == 2 4 6
```

- 이차방정식 함수 만들기

```
function y = quadratic_equation(x, a, b, c) // x, x^2 계수, x계수, y절편
y = a*x.^2 +b*x +c; // f(x) = a*x.^2 +b*x +c
quadratic_equation([1 2 3], 2, 1, 0)
ans == 3 10 21 // 2*1^2 + 1*1, 2*2^2+ 2*1, 2*3^2 + 3*1
```

MatLab PATH 지정

- which mean : mean함수의 위치를 확인하라.
- which plot : plot함수의 위치를 확인하라.
- Path에 함수를 지정해서 사용 가능.
- MatLab의 Set Path에 custom 함수 지정 가능.
- addpath('경로명') // path 지정.

기본 신호들 만들기

- unit-impulse : 0에서 1로 튀어나오고 나머지 부분에서는 0인 신호

```
t = [ -10 : 0.01 : 10 ];
xt = ( t == 0);
stem(t, xt); // t가 0일 때만 True이므로 1(근사값)
```

- unit-step : 0 이상 1이고 나머지는 전부 0

```
t = [ -10 : 0.01 : 10 ];
xt = ( t >= 0);
plot(t, xt); // t>=0일 때만 True이고 1값을 가진다.
ylim([-0.5,2]);
```

- unit-pulse : -0.5, 0.5사이에서만 1을 가짐.

```
t = [ -10 : 0.01 : 10 ];
xt = ( -0.5 <=t ) & (t<=0.5);
plot(t, xt);
ylim([-0.5 2]);
```

- unit-ramp : 구간별 표현 식이 다르다. (t>=0 일 때 xt=t, t<0일 때 xt =0)이다.

```
t = [ -10 : 0.01 : 10 ];
t1 = (t >=0);
xt = t1 .*t;
plot(t, xt);
ylim([-0.5 2]);
```

2개의 신호를 만들어보자

```
t = [ -10 : 0.01 : 10 ];
t1 = (t >=0);
t2 = (t < 0);
xt1 = t + 1.;
xt2 = -t +1;
plot(t, xt1, t, xt2); grid;

xt1 = (t + 1).*t2;
xt2 = (-t +1).*t1;
plot(t, xt1, t, xt2); grid;
```

Sinusodial signal(정현파)

- 주기적 물리 현상을 수학적으로 나타내는데 유리함.

```
T = 3; // 주기
f = 1/T; // 진동수
t = [ 0 : 0.01 : 4*pi ];
xt = sin(2*pi*t*f); // f(t) = sin(2*pi*t*f)

plot(t, xt);
```

구간신호, 주기신호

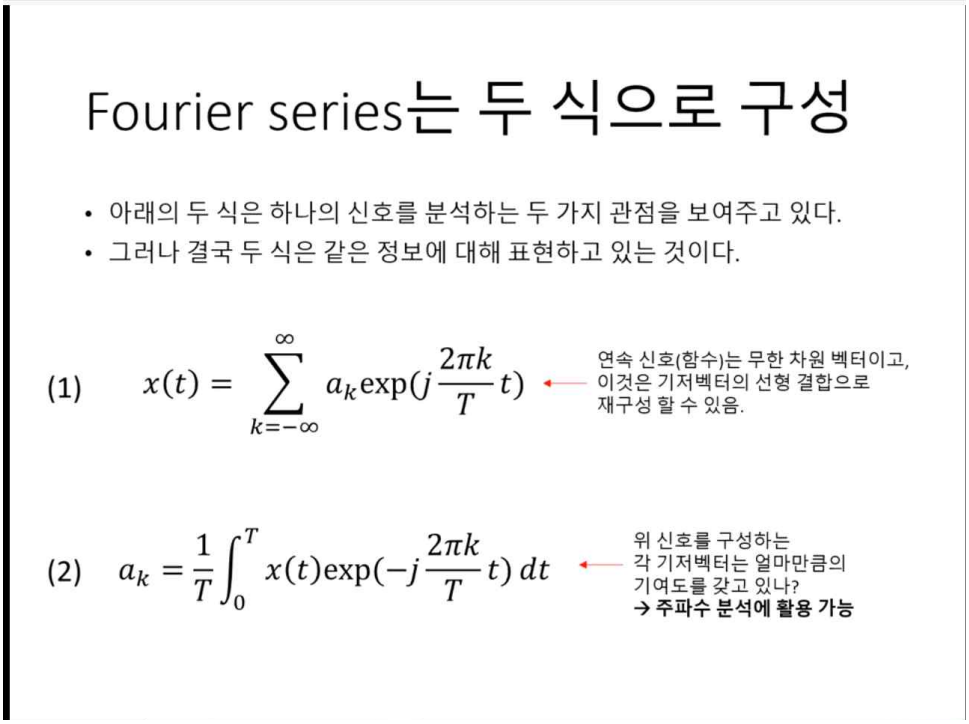
- 주기 함수 전에 구간별로 신호를 나타내는 식이 다른 신호

```
t = [ -10 : 0.01 : 10 ];

xt1 = 2*t - 1; // 1차식
xt2 = -1*t - 7;
plot(t,xt1); grid
```

- t가 0보다 클 때는 1, 작을 때는 0을 따라가는 신호와 반대인 신호 표현

```
t = [ -10 : 0.01 : 10 ];
t1 = (t >=0);
t2 = (t < 0);

subplot(2, 1, 1); plot(t, t1); ylim([-1 2]);   // subplot( 가로줄, 세로줄, 인덱스), t>=0일 때 t1==1
subplot(2, 1, 2); plot(t, t2); ylim([-1 2]);   // subplot( 가로줄, 세로줄, 인덱스), t< 0일 때 t2==1
```

푸리에 급수



- 푸리에 시리즈의 의미
연속 신호(함수)는 무한 차원 벡터이고, 이것은 기저벡터의 선형 결합으로 재구성 가능하다.
1. 벡터의 직교
vector i, j가 직교한다면?
i와 j의 내적은 0이다.
x축의 단위 벡터와 Y축 단위 벡터로 2차원 벡터 공간의 모든 벡터를 표현할 수 있다.
벡터 세 개가 직교한다면? => 3차원 벡터 공간의 어떠한 벡터라도 X,Y,Z의 단위벡터로 표현 가능

2. 즉, N차원 벡터 공간을 표현하기 위해서는 N개의 직교하는 벡터가 필요.

3. 함수는 일반화된 벡터로 취급할 수 있다.

4. N차원 벡터는 N개의 숫자의 나열.(튜플)

5. 실수(혹은 복소수)함수는 실수(혹은 복소수)값을 무한 개 나열한 것.
함수를 벡터로 표현 => [f(a).(f(a+bx)...f(b))]

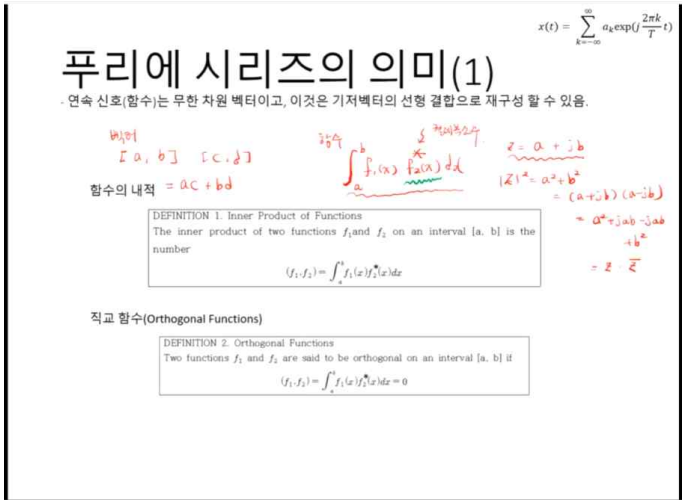
6. 즉, 실수(복소수) 함수는 무한 차원 벡터라고 생각가능.

7. 일반적으로 실수(혹은 복소수)함수는 무한차원 벡터로 생각할 수 있음.

8. 그러면 무한차원의 벡터 공간을 표현해야 함.

9. 함수의 직교
직교하는 함수를 무한 개 찾을 수 있다면?
같은 구간[a,b]의 직교 함수를 이용해서 어떤 실수(혹은 복소수)함수라도 그 구간 내의 함수를 표현 가능하다.

10. 함수의 내적



푸리에 변환

- 음향을 주파수로 분해

- 시간에 대한 기압의 그래프 = > 두 음을 주파수로 분해했다고 가정하면, 두음을 합한 값이 시간에 대한 기압의 그래프이다. = > 음이 늘어날 수록 복잡한 그래프 처럼 보이지만 사실은 모든 음을 더한 그래프이다.

- 합쳐진 주파수 그래프를 다시 순수한 주파수 그래프로 어떻게 변환할 것인가? // 푸리에 변환의 이유.

- 특정한 주파수의 신호를 다른 주파수의 신호와 다르게 다루는 수학적 기계를 만든다.

- 핵심 아이디어는 그래프를 하나의 원 주변에 감는다.

- 매 시점에서 벡터의 크기는 그 시점의 그래프의 높이와 같다.

- 그래프에서 높은 점은 원점에서 먼거리 낮은 점은 원점에서 낮은 거리

- 감는 주파수를 결정하는 것은 감는 그래프가 정한다.

연속시간 푸리에 변환 유도

- t-domain(시간 도메인)에서 보기 힘든 신호를 => f-domain(주파수 도메인)에서 분석할 수 있게 된다.

- 주기함수가 극한으로 가면 비주기함수로 볼 수 있다. => 일반적인 신호의 주파수 분석이 가능해진다.

- 일반적인 0~T 주기함수를 양옆으로 무한하게 크게 한다면 => 비주기 함수

- CTFS에서 유도된다.

- 극한으로 유도할 수 있다면 분석이 가능해진다.