디지털통신시스템설계

Digital Communication System Capstone Design

ICE4009-001

3 주차 실습과제

정보통신공학과

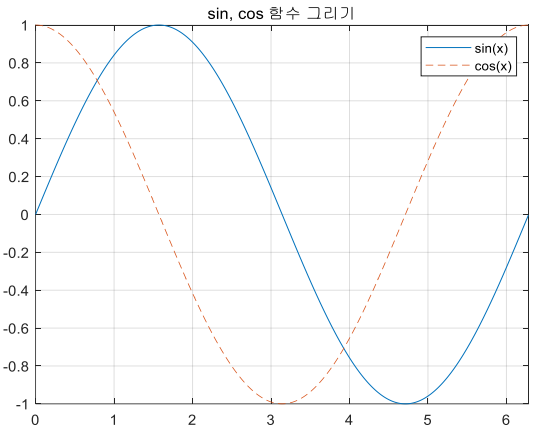
12191765

박승재

Introduction

```
x = 0:0.01:2 * pi;
y1 = sin(x);
y2 = cos(x);

box on;
grid on;
hold on;
plot(x, y1);
plot(x, y2, '--');
axis([0 2 * pi -1 1]);

title('sin, cos 함수 그리기');
legend('sin(x)', 'cos(x)');
```

Matlab 에서는 콜론(:)을 이용해서 벡터를 만들 수 있다. `x = 0:0.01:2 * pi;`는 0 부터 2pi 까지 0.01 간격을 가진 벡터를 생성하는 코드이다. x 에는 0, 0.01, 0.02, 0.03, ... 6.27, 6.28 이 들어가게 된다. sin 과 cos 함수는 인자로 벡터를 받아 각각의 값들에 연산을 취할 수 있다. `y1 = sin(x);`를 하게 되면 y1 에는 sin(0), sin(0.01), sin(0.02), sin(0.03), ... sin(6.27), sin(6.28) 값이 들어가게 된다.



`box on;`은 상자의 윤곽선을 표시하는 명령이다. `grid on;`은 좌표축의 격자 선을 표시하는 명령이다. `hold on;`은 여러 그래프를 하나의 화면에 표시할 수 있도록, 그래프를 겹쳐 그리는 명령이다.

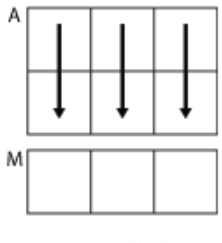
`plot` 은 X 값에 대한 Y 데이터의 2 차원 선 그래프를 그린다. x, y 이후에는 선을 꾸미는 옵션을 줄 수 있다. `'--'`는 선을 점선으로 그리라는 명령이다. `axis` 는 그래프를 보여줄 범위를 지정하는 명령이다. [x_min x_max y_min y_max] 순서로 값을 넣어주면 된다. `title` 은 그래프의 제목을 달아주는 명령이고, `legend` 는 왼쪽와 같이 그래프에 범례를 넣는 명령이다. 범례의 기본 위치는 우측 상단이다.

```
A = [-8.5 0 5.5 -2.6 4.8;
     1.9 7.3 -3.3 0.4 -6;
     -0.5 -8.1 -1.5 3.1 1;
     -8.9 2.5 0.9 1.6 -0.5;
     3.6 0.6 -0.7 -2.6 4.6];

A3_max = max(A(3, :)) % 3.1
```

```
[A5_min, A5_min_idx] = min(A(:, 5)) % 2

A_row_mean = mean(A, 2) % -0.16 0.06 -1.2 -0.88 1.1

A_sum = sum(A) % -12.4 2.3 0.9 -0.1 3.9

A_sum_all = sum(A_sum) % -5.4

A_abs_max = max(abs(A), [], 'all') % 8.9

A_round = round(A)
% -9   0   6  -3   5
%  2   7  -3   0  -6
% -1  -8  -2   3   1
% -9   3   1   2  -1
%  4   1  -1  -3   5
```

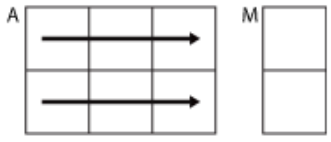
Matlab 은 [ ]와 세미콜론(;)을 이용해 행렬을 정의한다. [ ] 안의 숫자가 행렬 값이고 ;는 서로 다른 행이라는 것을 의미한다. A(3, :)는 행렬의 3 번째 행을 벡터로 뽑아내는 명령이다. A(3,5)와 같이 3 행 5 열의 숫자만 뽑아 낼 수 있다. 행렬의 5 번째 열을 뽑아내려면 A(:, 5)와 같이 첫 번째 인자에 :를 쓰면 된다. mean 함수의 두 번째 인자는 차원이다. 2 차원의 의미는 행렬의 열을 의미한다.

- mean(A,1)은 A의 각 열에서 요소들의 평균값을 구하고, 1×n 행 벡터를 반환합니다.



mean(A,1)

- mean(A,2)는 A의 각 행에서 요소들의 평균값을 구하고, m×1 열 벡터를 반환합니다.



mean(A,2)

행렬의 모든 수 중 가장 큰 수를 찾으려면 max 를 이용하면 된다. max 의 두 번째 인자와 세 번째 인자는 각각 size 와 dimension 을 의미한다. size 는 max 에서 필요없는 값이지만, 다른 함수들과 코딩 스타일을 맞추기 위해 만들어둔 인자같다. max 에서는 쓰지 않으므로 []을 넣어주면 된다. 세 번째 인자는 모든 차원에 대해서 가장 큰 수를 구하므로 'all'을 넣어준다. 만약 각 행에서 가장 큰 수를 찾으려면 max(A,[],2) 형태로 쓰면 된다.

```
function avg = mValue(X)
    total = sum(X(:));
    len = length(X(:));
    avg = total / len;
end
```

```
X = [1:10]; % 1~10
avg = mValue(X) % 5.5
```

Matlab 은 M-파일을 통해 함수를 만들고 사용할 수 있다. 파일 맨 위에 function 을 통해 함수를 선언한다. 함수의 이름은 파일의 이름과 같아야 한다. avg 는 함수가 반환할 값이고, mValue 는 함수의 이름이다. (X)와 같이 괄호 안에 함수의 인자를 작성하면 된다. 파일의 이름은 mValue.m 으로 만든다.

mValue 는 인자 X 의 값의 평균을 구하는 함수이다. test_mValue.m 은 mValue 를 호출해서 이용하는 파일이다. 1 부터 10 까지의 값을 가진 X 를 생성하고 mValue 에 넣어서 결과를 출력하면 5.5 가 나오는 것으로 mValue 가 의도대로 구현된 것을 확인할 수 있다.

```
A = randi(100, [1, 100])

A1 = A;
for i = 1:length(A1)
    [val, idx] = min(A1);
    B(i) = val;
    A1(idx) = 1000;
end
B

A1 = A;
i = 1;
while i <= length(A1)
    [val, idx] = max(A1);
    C(i) = val;
    A1(idx) = 0;
    i = i + 1;
end
C

D = zeros([1, 3]);
for i = 1:length(A)
    if 1 <= A(i) && A(i) <= 33
        D(1) = D(1) + 1;
    elseif 34 <= A(i) && A(i) <= 66
        D(2) = D(2) + 1;
    elseif 67 <= A(i) && A(i) <= 100
        D(3) = D(3) + 1;
```

```
    end
end
D
```

randi 는 주어진 범위로부터 정수인 난수를 n 개 생성하는 함수이다. randi(100, [1, 100])는 1~100 범위를 갖는 난수 100 개가 만들어진다.

X = randi(imax)는 1과 imax 사이의 정수형 의사 난수 스칼라를 반환합니다.

X = randi(imax,sz)는 배열을 반환합니다. 여기서 크기 벡터 sz가 size(X)를 정의합니다. 예를 들어, randi(10,[3 4])는 1과 10 사이의 정수형 의사 난수로 구성된 3×4 배열을 반환합니다.

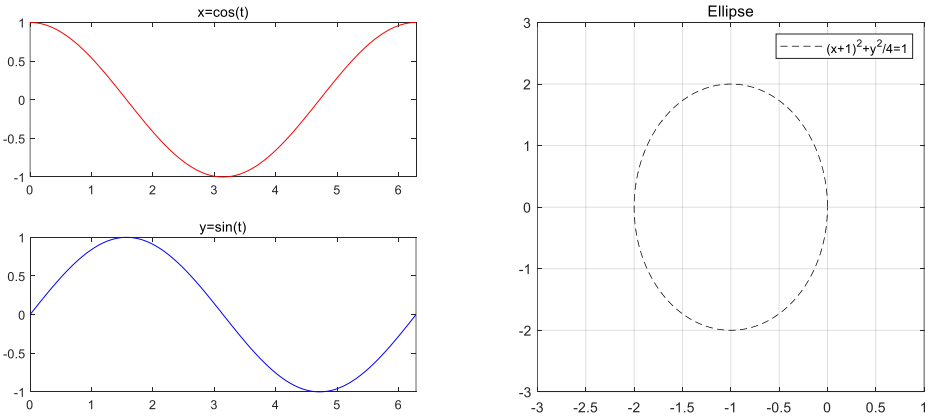
B 는 A 를 오름차순으로 정렬한 벡터이다. for 은 반복할 변수와 변수의 범위를 적어주면 된다. for 의 끝은 end 로 닫아준다. for i = 1:100 는 i 가 1 부터 100 까지 커지면서 반복하는 반복문이다. 오름차순 정렬은 가장 작은 값을 찾아서 B 에 집어넣고, 원래 값은 100 이상인 1000 으로 교체하는 방식으로 구현했다. min 은 최소 값과 최소 값의 위치를 동시에 반환할 수 있다.

C 는 A 를 내림차순으로 정렬한 벡터이다. 오름차순과 원리는 동일하지만 for 대신 while 을 이용해 구현했다. while 은 while 안의 값이 false 가 될 때까지 계속 반복한다. i 를 증가시키는 문법은 없으므로 i = i + 1;를 통해 직접 i 를 1 씩 증가시켰다.

마지막은 for 문과 if 문을 이용하여 행렬 A 의 원소 중 1~33 사이의 값, 34~66 사이의 값, 67~100 사이의 값을 갖는 원소의 수를 구하는 코드이다. for 을 통해 A 의 모든 값을 순회하도록 했고, if 문으로 A(i)가 어떤 범위에 속하는지 검사했다. elseif 를 통해 true 인 조건을 찾는다면 다른 조건은 확인하지 않고 다음 반복으로 넘어가도록 구현했다. 범위를 찾으면 D 를 1 씩 증가시켜서 원소 개수를 카운팅한다.

Problem

1. sin, cos 함수를 이용하여 타원 그리기



2. 다음의 행렬을 생성하고 아래의 예제를 풀어라

$$A = \begin{bmatrix} -8.5 & 0 & 5.5 & -2.6 & 4.8 \\ 1.9 & 7.3 & -3.3 & 0.4 & -6 \\ -0.5 & -8.1 & -1.5 & 3.1 & 1 \\ -8.9 & 2.5 & 0.9 & 1.6 & -0.5 \\ 3.6 & 0.6 & -0.7 & -2.6 & 4.6 \end{bmatrix}$$

① 행렬 A 의 각 행의 절대값의 합으로 이루어진 행렬 B(5x1)를 구하여라 (help abs, sum)

② 관계연산자를 이용하여, 행렬 A 의 원소가 0 보다 크거나 같을 때는 1, 0 보다 작을 때는 0 을 반환하는 행렬 C 를 생성

③ 행렬 A 의 원소 중 양수 값을 갖는 벡터 D 를 구하여라(help find)

④ 행렬 A 의 원소를 반올림하였을 때, 이전보다 커지거나 같으면 1, 작아지면 -1, 같으면 0 을 반환하는 행렬 E 를 생성하여라 (help round)

⑤ 행렬 A 의 모든 원소들을 오름차순으로 나열하는 벡터를 생성하고, 이를 행렬 F(5x5)로 변환하여라 (help sort, reshape)

⑥ 행렬 A 의 제곱과 행렬 A 의 원소별 제곱을 구하여라

⑦ 행렬 A 를 고유값 분해하고 고유값, 고유 벡터, 랭크를 구하여라 (help eig, rank)

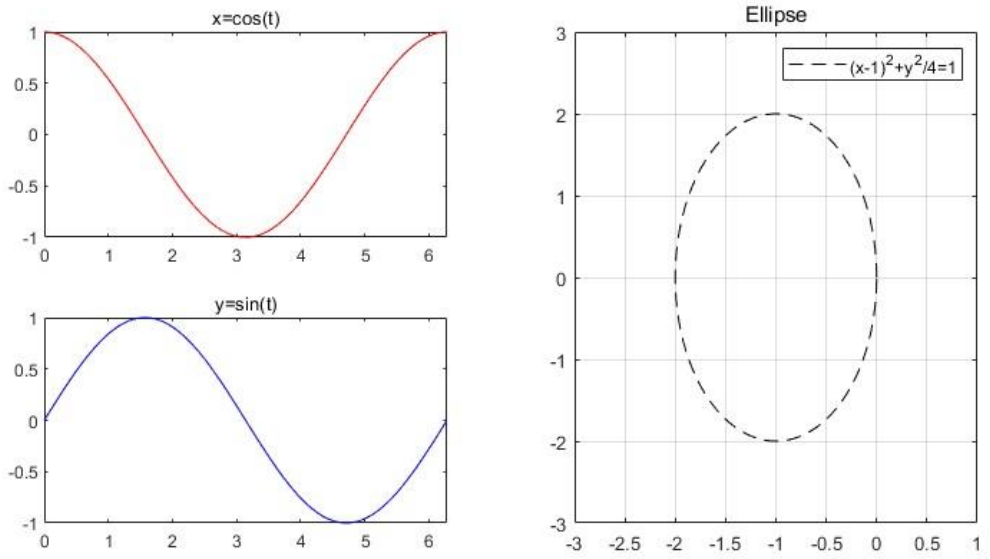
3. 0~1 사이의 임의의 실수 1000 개를 갖는 행렬의 원소의 구간별 빈도수를 값으로 구하여라 (help rand)

4. countNum 함수를 이용하여 다음의 난수 행렬의 pdf 를 그리고, 이론 값(이론 식 또는 matlab 함수 사용)과 비교하여라

Result

```
t = 0:0.01:2 * pi;

x = cos(t);
y = sin(t);

f = figure();
f.Position(3:4) = [840 420];

subplot(2, 2, 1);
```

```
plot(t, x, 'r');
axis([0 2 * pi -1 1]);
subtitle('x=cos(t)');

subplot(2, 2, 3);
plot(t, y, 'b');
axis([0 2 * pi -1 1]);
subtitle('y=sin(t)');

subplot(2, 2, [2, 4]);
plot(x - 1, 2 * y, 'k--');
grid on;
xticks(-3:0.5:1)
axis([-3 1 -3 3]);
subtitle('Ellipse');
legend('(x-1)^2+y^2/4=1')
```



`f.Position` 의 3:4 는 figure 창 크기를 의미하는 값이다. 가로 840 세로 420 로 설정해 사진과 비슷한 크기의 figure 창을 만들도록 했다. subplot 을 이용해 하나의 figure 에 여러 그래프를 넣을 수 있게 했다. 타원 그래프는 `subplot(2, 2, [2, 4])`를 통해 2 칸을 이용해서 그래프를 그리도록 했다. 타원 그래프는 grid 를 표시하도록 했고, 그리드의 x 축 간격을 0.5 로 맞춰주었다.

사진과 동일한 크기와 범위의 그래프를 보여주기 위해 `axis` 함수를 이용했다. `plot` 의 마지막 인자는 그래프 선을 꾸미는 옵션으로 이 부분을 이용해 그래프의 색을 설정할 수 있다. `plot(t, x, 'r')`에서 r 은 red 를 의미하고, 'b'는 blue, 'k'는 black 을 의미한다. `'k--'`는 검은색 점선을 이용해 그래프를 그리라는 명령이다.

```
A = [-8.5 0 5.5 -2.6 4.8;
     1.9 7.3 -3.3 0.4 -6;
     -0.5 -8.1 -1.5 3.1 1;
     -8.9 2.5 0.9 1.6 -0.5;
```

```
     3.6 0.6 -0.7 -2.6 4.6]

A_abs = abs(A);
B = sum(A_abs, 2)

C = A >= 0

D = A(find(A > 0))

E = (round(A) > A) - (round(A) < A)

F1 = sort(A(:))
F2 = reshape(F1, [5, 5])

G1 = A ^ 2
G2 = A .^ 2

[A_V, A_D] = eig(A)
k = rank(A)
```

```
A =

   -8.5000         0    5.5000   -2.6000    4.8000
    1.9000    7.3000   -3.3000    0.4000   -6.0000
   -0.5000   -8.1000   -1.5000    3.1000    1.0000
   -8.9000    2.5000    0.9000    1.6000   -0.5000
    3.6000    0.6000   -0.7000   -2.6000    4.6000

B =

   21.4000
   18.9000
   14.2000
   14.4000
   12.1000

C =

  5×5 logical 배열

   0   1   1   0   1
   1   1   0   1   0
   0   0   0   1   1
   0   1   1   1   0
   1   1   0   0   1

D =

    1.9000
    3.6000
    7.3000
    2.5000
    0.6000
    5.5000
    0.9000
    0.4000
    3.1000
    1.6000
    4.8000
    1.0000
    4.6000
```

abs 에 행렬을 넣으면 행렬의 각 요소의 절댓값을 취한 행렬이 나온다. sum 의 두 번째 인자는 dimension 으로, 2 를 넣으면 각 행의 합을 구한다.

행렬에 관계연산자를 사용하면 조건을 만족하는 요소만 1 로 표시되고, 그렇지 않은 요소는 0 으로 표시하는 행렬이 나온다. A >= 0 를 실행하면 0 이상인 요소만 1 로 표시되는 행렬이 나오게 된다.

find 는 0 이 아닌 요소의 위치(index)를 반환하는 함수이다. A > 0 를 넣으면 0 보다 큰 요소의 위치만 1 로 나오게 되고, 이것을 find 에 넣으면 0 보다 큰 요소의 위치가 나오게 된다. A(i)와 같이 행렬도 인덱스로 접근할 수 있는데, i 자리에 벡터를 넣게 되면 A 에서 해당 위치의 값을 가져와서 새로운 벡터로 반환하게 된다. 따라서 A(find(A > 0))를 통해 A 의 양수 값만 가져올 수 있다.

```
E =

    -1     0     1    -1     1
     1    -1     1    -1     0
    -1     1    -1    -1     0
    -1     1     1     1    -1
     1     1    -1    -1     1

F1 =

   -8.9000
   -8.5000
   -8.1000
   -6.0000
   -3.3000
   -2.6000
   -2.6000
   -1.5000
   -0.7000
   -0.5000
   -0.5000
         0
    0.4000
    0.6000
    0.9000
    1.0000
    1.6000
    1.9000
    2.5000
    3.1000
    3.6000
    4.6000
    4.8000
    5.5000
    7.3000

F2 =

   -8.9000   -2.6000   -0.5000    1.0000    3.6000
   -8.5000   -2.6000         0    1.6000    4.6000
   -8.1000   -1.5000    0.4000    1.9000    4.8000
   -6.0000   -0.7000    0.6000    2.5000    5.5000
   -3.3000   -0.5000    0.9000    3.1000    7.3000
```

round(A) > A 는 반올림했을 때 값이 커지는 숫자의 위치만 1 로 표시한다. round(A) < A 는 그 반대다. round(A) > A 결과에서 round(A) < A 를 빼면 반올림하였을 때 이전보다 커지거나 같으면 1, 작아지면 -1, 같으면 0 을 반환하는 행렬이 만들어진다.

A(:)는 행렬의 모든 요소를 가져오는 문법이다. A 의 모든 요소를 가져와서 sort 함수에 넣으면 오름차순으로 정렬된 벡터가 나온다. 이것이 F1 이다. F1 을 reshape 함수를 통해 5x5 형태로 바꾸면 크기순으로 정렬된 행렬이 나온다. 이것이 F2 이다.

```
G1 =

  109.9200  -48.1700  -60.7000   22.5100  -11.9200
  -25.7900   77.4200   -4.1300    3.9900  -65.7800
  -34.3800  -38.6300   28.3200   -4.2300   47.7500
   63.9100   14.6600  -56.7600   30.7900  -59.9200
   10.5900    6.3100   13.3100  -27.4100   35.4400

G2 =

   72.2500         0   30.2500    6.7600   23.0400
    3.6100   53.2900   10.8900    0.1600   36.0000
    0.2500   65.6100    2.2500    9.6100    1.0000
   79.2100    6.2500    0.8100    2.5600    0.2500
   12.9600    0.3600    0.4900    6.7600   21.1600
```

연산자 앞에 .이 붙으면 각각의 요소에 연산자를 취한 결과가 나온다. A ^ 2 은 행렬을 제곱하는 것이고, A .^ 2 는 행렬의 각 원소를 제곱하는 것이다.

```
A_V =

  -0.7908 + 0.0000i   0.2288 + 0.0000i  -0.0626 - 0.0063i  -0.0626 + 0.0063i   0.3332 + 0.0000i
   0.1518 + 0.0000i  -0.7513 + 0.0000i   0.4732 + 0.0664i   0.4732 - 0.0664i   0.2304 + 0.0000i
   0.2232 + 0.0000i   0.4116 + 0.0000i  -0.4420 - 0.1451i  -0.4420 + 0.1451i   0.6266 + 0.0000i
  -0.5421 + 0.0000i  -0.4158 + 0.0000i   0.4045 - 0.2460i   0.4045 + 0.2460i   0.6520 + 0.0000i
   0.0885 + 0.0000i   0.2022 + 0.0000i   0.5721 + 0.0000i   0.5721 + 0.0000i   0.1348 + 0.0000i

A_D =

 -12.3720 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  10.3656 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   3.4051 + 1.3251i   0.0000 + 0.0000i   0.0000 + 0.0000i  k =
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   3.4051 - 1.3251i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i  -1.3039 + 0.0000i       5
```

eig 는 고유 값과 고유 벡터를 구한다. V 가 고유 벡터(eigen vector)이고 D 가 고유 값(eigen value)이다. rank 는 행렬의 랭크를 구하는 함수이다.

```
function num = countNum(in, gap)
    N = length(in);
    R = length(gap);
    step = gap(2) - gap(1);
    num = zeros(1, R);
    for i = 1:N
        idx = floor((in(i) - gap(1)) / step) + 1;
        if idx >= 1 && idx <= R
            num(idx) = num(idx) + 1;
        end
    end
end
```

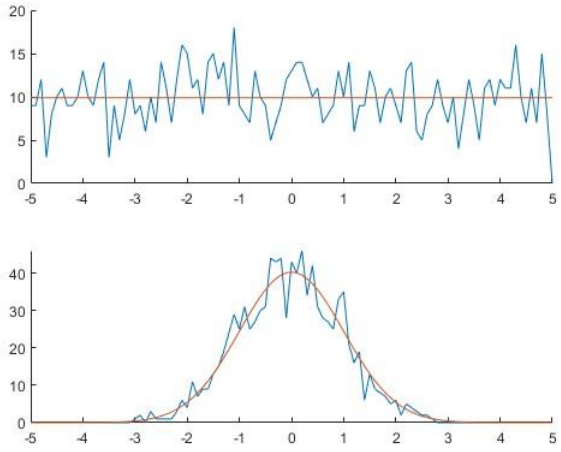
```
A = rand(1, 1000);
U = -5 + rand(1, 1000) * 10;
G = randn(1, 1000);
```

```
gap1 = 0:0.1:1;
numA = countNum(A, gap1)

gap2 = -5:0.1:5;
numU = countNum(U, gap2);
numG = countNum(G, gap2);

f = figure();
subplot(2, 1, 1);
hold on;
plot(gap2, numU);
plot(gap2, ones(1, 101) * 1000/100);
hold off;

subplot(2, 1, 2);
hold on;
plot(gap2, numG);
plot(gap2, 100 / sqrt(2 * pi) * exp(-1/2 * (gap2 .^ 2)));
```

countNum 는 행렬의 구간 별 빈도수를 구하는 함수이다. 입력으로 빈도수를 구할 행렬 in 과 구간의 시작값이 들어있는 벡터 gap 을 받는다. 출력으로는 각 구간의 빈도수 num 을 내보낸다. 구간의 크기는 gap(2) - gap(1)을 통해 계산하고 숫자를 구간의 크기로 나누는 방식으로 어떤 구간에 속하는지, 구간의 위치를 찾는다.

```
numA =

    99    86    92   105    94   100    98   102   112   112     0
```

countNum 을 통해 0 부터 1 까지의 난수의 빈도를 계산했다. 구간의 크기는 0.1 이기 때문에 gapA 는 1x11 크기의 행렬이다. 따라서 이상적인 numA 값은 10000/100=100 으로 채워진 값이 들어간 1x11 크기의 행렬이다. numA = [100 100 100 … 100 100 0]



이번에는 -5 부터 5 까지의 난수가 들어있는 U 의 PDF 를 그려보았다. numA 와 같은 방식으로, countNum 함수를 이용해서 개수를 세고 plot 을 통해 그림을 그렸다.

수치적으로 계산한 이상적인 빈도수도 같이 그렸다. 위 사진과 같이 10(주황)을 중심으로 빈도수가 분포(파랑)하는 것을 확인할 수 있었다. 마지막에 0 으로 그래프가 떨어지는 이유는 gap2 의 마지막 값이 5 이고, `-5 + rand(1, 1000) * 10` 코드로는 5 가 나오지 않으므로 빈도가 항상 0 이기 때문이다.

`randn` 으로 정규분포를 갖는 난수를 생성할 수 있다. G 도 U 와 같은 방식으로 PDF 를 그렸다. gap 은 U 에서 사용한 gap2 를 재사용했다. 따라서 위의 두 그래프의 x 축이 동일한 것을 확인할 수 있다. 정규분포 난수의 이상적인 빈도수는 PDF 를 따라간다. 수식으로 그린 주황색 곡선이 정규분포 곡선인데, 실제 측정한 값과 유사한 것을 확인할 수 있다.

Conclusion

실습을 통해 Matlab 의 전반적인 사용법과 함수를 응용해서 원하는 기능을 구현하는 법을 배웠다. 함수의 기능을 검색할 때 Matlab 의 F1 기능을 이용해 도움말을 확인하는 것이 큰 도움이 됐다.

그래프를 그릴 때 다양한 옵션을 활용해서 시각적으로 정돈된 형태의 그래프를 그리는 법과 Matlab 의 기본 함수를 이용해 행렬을 다루는 법을 배웠다. 특히, find 를 통해서 원하는 값만 뽑아오는 기능과 . 연산자를 이용해 각 요소에 연산자를 적용하는 문법은 앞으로도 도움이 될 것 같다.

마지막 과제에서 `hold on` 으로 측정값과 이상값을 동시에 그려서 비교하는 방식은 내가 값을 올바르게 측정했는지 시각적으로 확인할 수 있어서 유익했다. 앞으로도 내가 올바르게 과제를 수행했는지 확인하기 위해 자주 사용할 것 같다.