핵심 수치해석

고려대학교 과학계산연구실

2011-09-06

머리말

본 핵심 수치해석은 수치해석에 전형적으로 나와 있는 내용에 수치해석의 핵심부분을 추가하였습니다. 초판이라 많은 오타와 오류가 있을 것이라 생각합니다. 오타나 오류를 알려주시면 다음 개정판에는 수정 보안할 것 입니다.

차 례

1 장

MATLAB 기초

MATLAB(www.mathworks.com)은 미국의 Math Works에서 만들어진 프로그램으로, 1984년도에 소개된 이후 오늘날 전 세계 50만 이상이 사용하고 있다. MATLAB은 MATrix+LABoratory의 약어로서 행렬을 기본으로 최적화되어진 프로그램으로 알고리즘 개발, 데이터 수치분석이나 시각화를 위한 컴퓨터 언어이다. 주로 공학계통에서 사용되던 MATLAB은 편리한 사용방법으로 최근 모델링을 위한 프로그래밍 언어로 인기를 얻고 있으며 이 책에서도 모든 코드를 MATLAB언어로 구현하였다.

제 1 절 기본 명령어

1.1 간단한 계산 명령어

- a에 1 대입.

```
>> a = 1   Enter
a = 1
```

명령어를 입력한 후 엔터를 누르면 MATLAB이 명령어를 실행한다. 명령어가 실행되면 입력한 명령어 다음 줄부터 실행된 결과물이 표시된다.

- b라는 이름으로 1행 3열 행렬(1×3) 생성.

```
>> b=[1 2 3]
b = 1     2     3
```

- b라는 이름으로 3행 1열 행렬(3×1) 생성.

```
>> b=[1;2;3]
b = 1
    2
    3
```

- b 행렬을 c에 대입.

```
>> c=b
c = 1
    2
    3
```

- b의 전치(transpose)행렬을 c 행렬로 대입.

```
>> c=b'
c = 1     2     3
```

- A라는 이름으로 3행 3열 행렬(3×3) 생성.

```
>> A=[1 2 3;4 5 6;7 8 9]
A = 1     2     3
    4     5     6
    7     8     9
```

- A의 전치행렬 A^T.

```
>> A'
ans = 1     4     7
      2     5     8
      3     6     9
```

- 화면(command window)에 출력하지 않기 위해선 세미콜론(;)을 붙인다.

```
>> a=1;b=2,c=3;
b = 2
```

- d라는 이름으로 2행 3열 행렬(2×3) 생성.

```
>> d=[1 2 3;4 5 6]
d = 1     2     3
    4     5     6
```

- d의 1행 3열의 원소값 출력.

```
>> d(1,3)
ans = 3
```

- 2×3의 계산값 출력.

```
>> 2*3
ans = 6
```

- 마지막으로 계산된 ans값 출력. Default로 ans에 마지막 결과 값이 저장된다.

```
>> ans
ans = 6
```

- 마지막으로 계산된 ans값에 6을 더하는 연산.

```
>> ans+6
ans = 12
```

- 행렬 b를 먼저 정의한 후(세미콜론(;)을 붙였으므로 출력은 되지 않는다.) b의 각 원소를 제곱 또는 세제곱을 한다. 행렬의 각 원소를 제곱하기 위해서는 .을 ^연산자 앞에 붙여야 한다.

```
>> b=[1 2 3];
>> b.^2
ans = 1     4     9
>> b.^3
ans = 1     8    27
```

- a행렬의 각 원소와 b행렬의 각 원소를 곱한다. 즉, $[1\times 7 \;\; 2\times 8 \;\; 3\times 9]$을 실행하여, [7 16 27]의 결과를 얻게 된다.

```
>> a=[7 8 9];
>> a.*b
ans = 7    16    27
```

- a행렬과 b행렬의 전치행렬의 원소를 각각 곱하여 더한다. 즉, a벡터와 b벡터의 내적(inner product)과 같다.

```
>> a*b'
ans = 50
```

- 각 원소 값들이 0인 3행 3열 행렬(3×3) 생성.

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

- 각 원소 값들이 0인 1행 3열 행렬(1×3) 생성.

```
>> zeros(1,3)
ans = 0     0     0
```

- A라는 이름으로 각 원소 값들이 0인 4행 5열 행렬(4×5) 생성.

```
>> A=zeros(4,5)
A = 0     0     0     0     0
    0     0     0     0     0
    0     0     0     0     0
    0     0     0     0     0
```

- A와 같은 크기의 영(zero)행렬을 생성.

```
>> zeros(size(A))
ans = 0     0     0     0     0
      0     0     0     0     0
      0     0     0     0     0
      0     0     0     0     0
```

- 크기가 3인 단위행렬 생성.

```
>> eye(3)
ans = 1     0     0
      0     1     0
      0     0     1
```

- 각 원소 값들이 1인 3행 3열 행렬(3×3) 생성.

```
>> ones(3)
ans = 1     1     1
      1     1     1
      1     1     1
```

- A라는 이름으로 각 원소 값들이 1인 4행 5열 행렬(4×5) 생성.

```
>> A=ones(4,5)
A = 1     1     1     1     1
    1     1     1     1     1
    1     1     1     1     1
    1     1     1     1     1
```

- 크기는 행렬 A와 같고 모든 원소가 1인 행렬을 생성.

```
>> ones(size(A))
ans = 1     1     1     1     1
      1     1     1     1     1
      1     1     1     1     1
      1     1     1     1     1
```

- MATLAB은 대소문자를 구분한다.

```
>> a=2, A = 3;
>> a
a = 2
```

- A행렬의 각 열의 원소들의 합.

```
>> A=[1 2 3;4 5 6;7 8 9];
>> sum(A)
ans = 12    15    18
```

- A행렬의 각 행의 원소들의 합. 즉, A의 전치행렬의 각 열의 원소들의 합.

```
>> sum(A')
ans = 6    15    24
```

- sum(A)의 각 원소들의 합.

```
>> sum(sum(A))
ans = 45
```

- A행렬의 각 열의 최댓값.

```
>> max(A)
ans = 7    8    9
```

- A행렬의 각 열의 최솟값.

```
>> min(A)
ans = 1    2    3
```

- max(A)의 최댓값.

```
>> max(max(A))
ans = 9
```

- max(A)의 최솟값.

```
>> min(max(A))
ans = 7
```

- A행렬과 B행렬의 정의.

```
>> A=[1 2;3 4]
A = 1     2
    3     4
>> B=[5 6;7 8]
B = 5     6
    7     8
```

- A행렬과 B행렬의 각 원소들의 합.

```
>> A+B
ans = 6     8
     10    12
```

- A행렬과 B행렬의 각 원소들의 차.

```
>> A-B
ans = -4    -4
      -4    -4
```

- A행렬의 각 원소에 2씩 더한 값.

```
>> A+2
ans = 3     4
      5     6
```

- size로 벡터 또는 행렬의 크기를 알 수 있다.

```
>> size(A)
ans = 2     2
```

- length로 벡터의 길이를 알 수 있다.

```
>> C=[1 2 3];
>> length(C)
ans = 3
```

- length는 행렬의 행과 열 중에 큰 값을 출력한다.

```
>> C=[1 2 3; 4 5 6];
>> length(C)
ans = 3
```

다음은 각각의 행렬에 대한 사칙연산이다.

MATLAB 명령어	연산 의미
`A*B`	$AB = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
`A/B` or `A*inv(B)`	$AB^{-1} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} -4 & 3 \\ 3.5 & 2.5 \end{pmatrix} = \begin{pmatrix} 3 & -2 \\ 2 & -1 \end{pmatrix}$
`A\B` or `inv(A)*B`	$A^{-1}B = \begin{pmatrix} -2 & 1 \\ 1.5 & -0.5 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} -3 & -4 \\ 4 & 5 \end{pmatrix}$
`A^2` or `A*A`	$A^2 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$

위에서 행렬과 행렬사이에 사용한 * 연산자는 행렬곱 연산자이다. 따라서 곱하는 두 행렬의 크기가 $(n,m) \times (m,k) = (n,k)$이어야 한다. 또한 / 연산자는 역행렬을 곱하는 것으로 연산하고자 하는 두 행렬의 크기가 같으며 정방행렬이어야 한다. 이제 위에서 본 연산자 앞에 .을 찍으면 어떤 결과가 나오는지 살펴보자.

MATLAB 명령어	연산 의미
`A.*B`	$\begin{pmatrix} 1 \cdot 5 & 2 \cdot 6 \\ 3 \cdot 7 & 4 \cdot 8 \end{pmatrix} = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}$
`A.^B`	$\begin{pmatrix} 1^5 & 2^6 \\ 3^7 & 4^8 \end{pmatrix} = \begin{pmatrix} 1 & 64 \\ 2187 & 65536 \end{pmatrix}$

연산자 앞에 .이 붙게 되면 행렬의 같은 위치에 있는 각각의 원소끼리 연산을 수행하라는 의미이다. 원소끼리의 연산이므로 반드시 두 행렬의 크기는 정확하게 일치해야 한다.

이제 행렬의 일부나 전체를 불러오는 표현을 배워보자. 아래와 같이 간단하게 A를 입력한다.

```
>> A = [1 1 1 1; 2 2 2 2; 3 3 3 3; 4 4 4 4]
A = 1     1     1     1
    2     2     2     2
    3     3     3     3
    4     4     4     4
>> A(2, 1:4)
ans = 2     2     2     2
>> A(2,:)
ans = 2     2     2     2
```

위의 실행 결과에서 볼 수 있듯이 두 개의 명령문은 동일한 표현이다. 모두 A 행렬의 2행의 1열부터 끝까지의 원소를 나열하라는 것으로, 여기서 :은 행렬에서 전체를 나타내는 표현이다.

MATLAB에서 abs(a)을 사용하면 a의 절대값을 나타낸다.

```
>> abs(-3)
ans = 3
```

MATLAB에서 round(a)을 사용하면 a에 가장 가까운 정수를 나타낸다.

```
>> round(3.2), round(3.6)
ans = 3 ans = 4
```

MATLAB에서 real(a)와 imag(a)를 사용하면 각각 a의 실수부와 허수부를 나타낸다.

```
>> real(3+2i), imag(3+2i)
ans = 3 ans = 2
```

MATLAB의 나머지 연산인 mod에 대하여 알아보자. 예를 들어,

$$10 = (-4) \times (-3) - 2$$

의 계산을 MATLAB에서 mod를 사용하면, 다음의 결과를 출력하게 된다.

```
>> mod(10, -4)
ans = -2
```

항목	명령어	기능
명령어 나열	,	두 개 이상의 명령어를 한 줄에 표현할 때 콤마(,)를 이용하여 구분한다.

줄넘기기	...	명령어가 너무 길어 다음 줄로 넘어가고 싶을 때 사용한다(Command Window에서도 사용가능).
		(예)
		`plot(x,y,'--rs',...`
		`'LineWidth',2,...`
		`'MarkerEdgeColor','k',...`
		`'MarkerSize',10)`

출력여부	;	Command Window상에서 명령어를 실행하면 화면에 결과가 출력되지만, 세미콜론(;)을 입력하고 실행하면 출력되지 않고 workspace에만 저장된다. (예) `>> a=1;b=2;c=3;`
주석	%	명령어의 앞에 %를 입력하면 주석으로 처리된다.
화면정리	clc	clc 명령어를 입력하고 엔터를 치면 Command Window에 표시되었던 모든 내용들이 지워진다.
화면정리	clf	clf 명령어를 입력하고 엔터를 치면 Figure에 나타난 모든 그림이 지워진다.
변수삭제	clear	변수 및 배열에 할당된 값들 모두 삭제. • 모든 변수를 삭제 : `clear all` • 특정 변수만을 삭제 : `clear 특정변수`

항목	명령어	기능
변수나열	whos	사용 중인 변수들의 size를 보여준다. >> whos Name Size Bytes Class Attributes a 1x3 24 double ans 1x3 24 double b 1x3 24 double c 1x1 8 double d 2x3 48 double >> whos a Name Size Bytes Class Attributes a 1x3 24 double

1.2 M-file 만들기

MATLAB을 이용하여 원하는 기능을 수행하는 방법은 크게 두 가지로 구분된다.

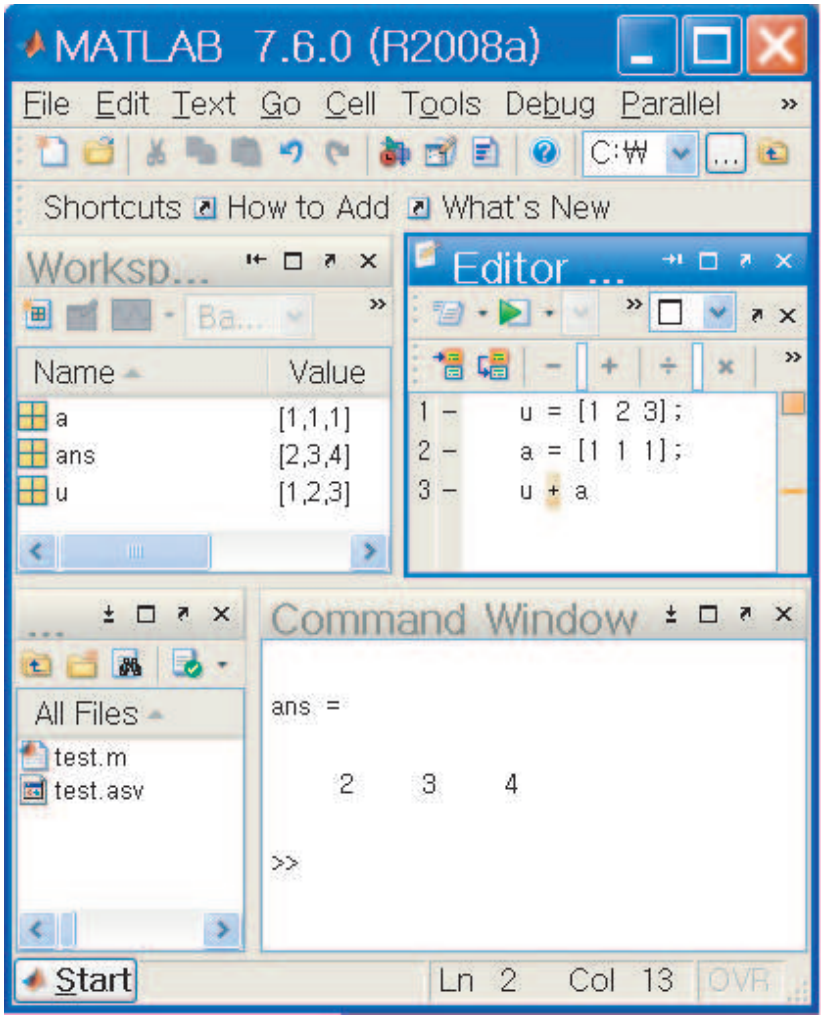
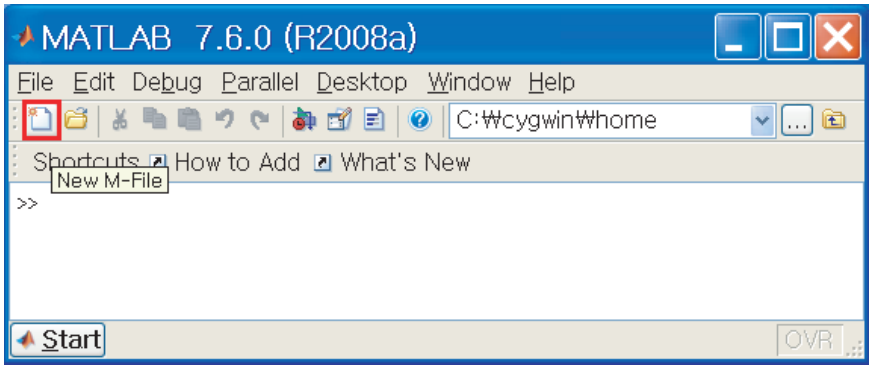


그림 1.1: MATLAB창

첫 번째는 Command Window에 직접 명령어를 입력하는 방법이고, 두 번째는 스크립트(Script)파일을 이용하는 것이다. MATLAB에서 사용하는 파일을 보통 M-file이라고 부르며 파일의 확장자는 .m으로 저장된다. 이러한 M-file은 스크

립트에 있는 명령어들이 차례대로 Command Window에서 실행될 수 있도록 한다. 실제로 M-file을 만들어 보자.



New M-file 버튼을 누르면 m-file이 만들어진다.

주의할 점은 파일이름을 숫자로 시작하면 안되고, 이름 중간에 공백이나 하이픈(-)이 있으면 안된다. 또한 MATLAB에 이미 정의되어 있는 상수이름과 함수이름을 사용하면 안된다. 이를 확인하기 위해서는 .m을 생략하고 파일이름을 Command Window에 입력하고 엔터키를 누른다음 [??? Undefined function or variable]라는 에러 메세지를 확인한다.

```
edit1.m (O)   1edit.m (X)   edit 1.m (X)
```

메뉴에서 File → New → M-File을 클릭해도 좋다. 이제 만들어진 Editor에 원하는 명령어를 순서대로 쓰고, Save and Run을 하기 위해 단축키인 F5를 누르자.

```
a = 1; b = 2; c = a+b
```

위의 문장을 Editor에 적어 넣고, 단축키 F5를 누르자. 파일이름은 원하는 이름으로 쓰고 저장하면 원하는 결과를 얻을 수 있게 된다.

1.3 for ∼ end 문

'for'문은 'end'문과 짝을 이루어 사용된다. 'for'문과 같은 행에 있는 변수의 값을 초기값부터 증분의 크기만큼 누적시키면서 최종값에 도달될 때까지 'for'문과

'end'문 사이 문장의 명령을 수행한다. 증분이 '1'인 경우에는 '증분:'을 생략해도 무방하다.

for 변수명=초기값:(증분:)최종값

문장

end

[예제] for ∼ end 프로그램

```
for x=0:0.5:1
  a=2^x
end
for k=5:-2:1
  b=k
end
```

[프로그램 실행결과]

```
a = 1
a = 1.4142
a = 2
b = 5
b = 3
b = 1
```

1.4 if ∼ else ∼ end 문

여러 가지 조건에 따라 각각 다른 명령을 실행하고자 할 때, 'if ∼ else ∼ end'문을 사용한다. 아래 보기처럼 **조건 1**이 참이면 **문장 1**이 수행되고, **조건 1**이 모두 거짓이면, 'if' 문을 빠져 나와 **문장 2**가 수행된다.

```
if 조건 1
  문장 1
else
  문장 2
end
```

[예제] if ∼ else ∼ end 문 프로그램

```
a=3; if a<1
    b=a+1
else
    c=a+2
end
```

[프로그램 실행결과]

```
c = 5
```

1.5 while ∼ end 문

'while'문은 'end'문과 짝을 이루어 사용된다. 'while'문과 같은 행에 있는 조건이 참이면 'while'문과 'end'문 사이에 있는 문장의 명령을 반복적으로 수행한다.

```
while 조건
   문장
end
```

[예제] while ∼ end 문 프로그램

```
a=1;
while a<4
    a=a+1
end
```

[프로그램 실행결과]

```
a =  2
a =  3
a =  4
```

1.6 linspace 문

'linspace'는 a와 b사이의 간격이 동일한 n개의 성분을 갖는 벡터를 만드는 데 사용한다. 다음과 같이 이용하며 이에 대한 예제를 살펴보자.

linspace(a,b,n)

linspace(시작점,끝점,점의 총수)

[예제] **linspace 문 프로그램**

```
x = linspace(0,5,6)
y = linspace(-1,1,5)
```

[프로그램 실행결과]

```
x =   0    1    2     3    4    5
y =  -1  -0.5   0    0.5   1
```

1.7 MATLAB에서 미리 정의해 둔 변수

- i, j : 허수(imaginary unit, $\sqrt{-1}$) 복소수의 허수 단위를 나타낸다. for문을 쓸 때 index를 i, j로 사용하는 경우가 있는데 허수부가 있는지 고려해서 사용해야 한다.
  [예제] 허수 문 프로그램

```
A = 1+2i
```

  [프로그램 실행결과]

```
A = 1.0000 + 2.0000i
```

- NaN : 숫자가 아님
  NaN은 Not a Number의 약자로서 부정 혹은 불능값을 나타낸다.
  [예제] NaN 문 프로그램

```
A = 0 / 0
B = Inf / Inf
```

  [프로그램 실행결과]

```
A = NaN
B = NaN
```

- Inf : ∞ 무한대(Infinity)
  [예제] 무한대 문 프로그램

```
A = Inf*20000000000
B = Inf/10000000000
C = Inf - Inf
D = Inf / Inf
```

[프로그램 실행결과]

```
A = Inf
B = Inf
C = NaN
D = NaN
```

- pi : π, 원주율
  [예제] π 문 프로그램

```
A = pi
B = sin(pi)
C = cos(pi)
```

[프로그램 실행결과]

```
A = 3.1416
B = 1.2246e-016
C = -1
```

1.8 MATLAB 프로그래밍할 때 유용한 팁

- 함수 정의
  inline을 이용하면 간편하게 함수를 정의할 수 있다.

```
f=inline('x^5+3*x-1','x');
f(2)
```

[프로그램 실행결과]

```
ans = 37
```

eval을 이용하여도 위와 동일한 결과를 얻을 수 있다.

```
f='x^5+3*x-1'; x=2;
eval(f)
```

[프로그램 실행결과]

```
ans = 37
```

- 주어진 조건을 만족하는 index 찾기
  find를 이용하면 주어진 조건을 만족하는 index를 손쉽게 찾을 수 있다.

```
x=[1 2 3 5 8]; y=[0 2 1 4 9];
find(x<=y)
```

[프로그램 실행결과]

```
    ans = 2     5
```

- 파일 입출력

```
fp = fopen('test.m','w'); %test.m란 파일을 쓰기용으로 생성
fprintf(fp, '%d %d\n', 1, 2);        %파일에 1 2 쓰기
fprintf(fp, '%f %f\n', 3.5, 4.5);    %파일에 3.5 4.5 쓰기
fclose(fp); %파일 close
```

- 파일 불러오기
  load라는 함수로 파일에 있는 데이터를 가져올 수 있다. 단, 파일에 문자가 들어있으면 안된다.

```
 a = load('test.m')
```

[프로그램 실행결과]

```
a = 1.0000    2.0000
    3.5000    4.5000
```

제 2 절 MATLAB으로 그래프 그리기

2.1 MATLAB을 이용하여 그래프를 그리기 위한 기본 명령어

다음은 MATLAB을 이용하여 그래프를 그리기 위한 가장 기본적인 명령어이다.

- **clf**
  현재 figure 창에 실행된 모든 그림 제거.

- **figure(n)**
  새로운 figure 창을 생성하고 이름을 figure(n)으로 지정.

- **hold on**
  현재 그래프에 다음 생성될 그래프를 추가.

- **hold off**
  hold on 기능을 해제.

2.2 2차원 그래프

이 절에서는 여러 가지 MATLAB 명령어를 이용하여 2차원 그래프를 그리는 방법에 대하여 알아보고자 한다.

2.2.1 plot을 이용한 그래프 그리기

`plot`은 가장 기본적인 그래프를 그리는 명령어로, 실행결과를 2차원 그래프로 나타내고자 할 때 사용된다. 'plot'문을 사용하기 위해서 2개의 변수가 필요하며 각 변수는 같은 크기의 1차원 배열(벡터)이어야 한다.

- `plot(X,Y)`
  x축은 X, y축은 Y를 값으로 갖는 2차원 그래프를 보여준다.

- `plot(Y)`
  x축의 값을 주지 않으면 default로 x축은 index값을, y축은 Y를 값으로 하는 2차원 그래프를 보여준다

- `plot(X,Y,S)`
  S는 선의 종류, 심볼(symbol) 또는 색을 나타낼 수 있는 옵션값이다.(자세한 내용은 표 1.1 참고.)

- `plot (X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...)`
  여러 개의 벡터를 한 번에 같이 나타낼 수 있다.

색상		모양		라인	
b	Blue	.	Point	-	Solid
g	Green	o	Circle	:	Dotted
r	Red	x	x mark	-.	Dashdot
c	Cyan	+	Plus	--	Dashed
m	Magenta	*	Star	(none)	No line
y	Yellow	s	Square		
k	Black	d	Diamond		
w	white	v	Triangle(down)		
		^	Triangle(up)		

표 1.1: plot 명령어의 옵션

예를 들어, `plot(x,sin(x),'k--',x,cos(x),'ko')`를 실행하면, 다음 결과를 얻게 된다. 이는 y축의 값을 $\sin(x)$, $\cos(x)$로 하는 두 개의 그래프를 나타내며, 첫 번째 $\sin(x)$는 검은색 점선으로, $\cos(x)$는 검은색 원으로 표현된다 (그림 1.2 참고).

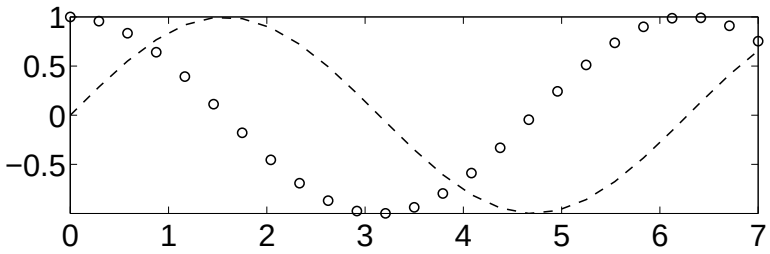


그림 1.2: plot문 옵션을 이용한 프로그램 실행결과

다음은 부가적인 명령어이다.

- `title`
  그래프의 제목 넣기

- `xlabel`
  x축에 이름 넣기
- `ylabel`
  y축에 이름 넣기
- `legend`
  여러 개의 곡선이 있을 때 각각의 곡선의 제목 넣기
- `grid`
  생성된 그래프에 격자 넣기
- `text`
  지정한 위치에 원하는 글 넣기
- `axis([xmin xmax ymin ymax])`
  그래프의 x축과 y축의 크기를 조절하기

그림을 그릴 때, MATLAB은 자동으로 좌표축을 설정한다. 생성된 그래프의 전체적인 크기 비율은 각각의 좌표축에 대한 aspect ratio에 따라 다르게 나온다. 예를 들어 반지름이 2인 원을 만들어 보자.

```
t=linspace(0,2*pi,100); x=2*cos(t); y=2*sin(t);
plot(x,y)
```

그림 1.3으로부터 알 수 있듯이 원의 모양을 갖는다고 볼 수 없다. 원의 모양보다는 오히려 타원의 모양을 갖고 있다. 그러므로 원래 원하는 원의 모양을 갖고 싶으면 aspect ratio를 바꾸어 주어야 한다. Aspect ratio를 "square"로 하면, Figure window의 모양을 직사각형 모양의 좌표계가 아닌 정사각형의 모양의 좌표계로 바꾼다.

```
t=linspace(0,2*pi,100); x=2*cos(t); y=2*sin(t);
plot(x,y)
axis square
```

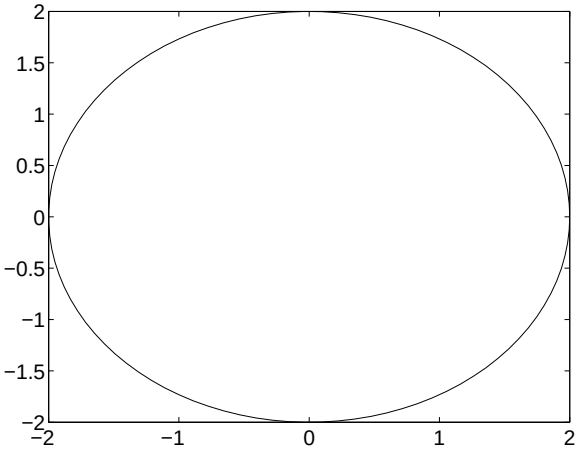


그림 1.3: 좌표계가 default인 경우

그림 1.4는 그림 1.3보다 훨씬 더 원하는 원의 모양을 갖고 있다. 그러나 이것은 어디까지나 Figure window를 정사각형 모양으로 바꾸어 준 결과로 나타난 것이다. 결국 각 좌표축의 units는 무시한 결과이다.

만일 실제 사물의 크기 비율(aspect ratio)과 같게 하려면, 각각의 좌표축에 대한 units는 동등한 크기를 가져야 한다. 이때 이용되는 aspect ratio로는 "equal"과 "image"가 있다. 해당 그래프에 대한 데이터의 범위까지만 display하는 경우가 "image"에 해당한다.

```
t=linspace(0,2*pi,100); x=2*cos(t); y=2*sin(t);
plot(x,y)
axis equal
```

```
t=linspace(0,2*pi,100); x=2*cos(t); y=2*sin(t);
plot(x,y)
axis image
```

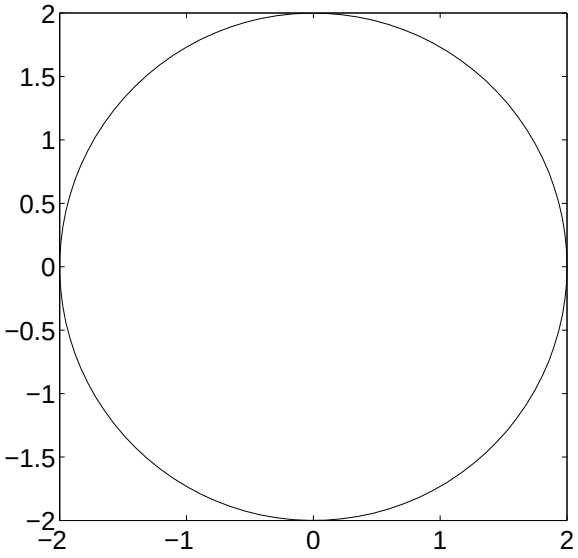


그림 1.4: 좌표계가 square인 경우

2.2.2 기타 그래프 그리기

극좌표 형식의 그래프

다음은 극좌표 형식의 함수 $r = f(\theta)$ 그래프를 그리는 명령어이다.

- **polar(theta,r)**
- **polar(theta,r,'linetype')**

다음 코드는 극좌표 형식의 다음 함수의 그래프를 그린 것이다.

$$r = \sin 2\theta \cos 2\theta, \quad 0 \le \theta \le 2\pi$$

```
t = 0:0.01:2*pi;
polar(t,sin(2*t).*cos(2*t),'k-')
```

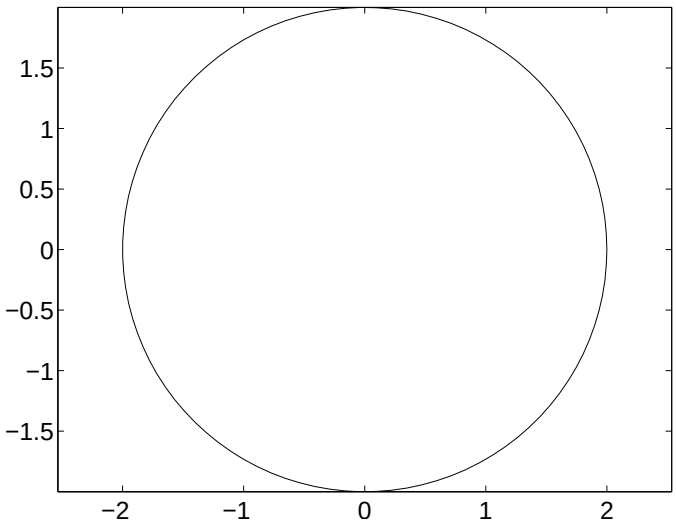


그림 1.5: 좌표계가 equal인 경우

막대 그래프

다음은 막대 그래프를 그리기 위한 명령어이다.

- **bar(x)**
  벡터 x를 막대 그래프로 그린다.

- **bar(x,y)**
  벡터 x에 지정된 벡터 y의 원소를 막대 그래프로 그린다.

다음은 함수 $y = e^{-x^2}$의 그래프를 bar 명령어를 이용하여 막대 그래프로 그린 것이다.

```
x = -2.9:0.2:2.9;
bar(x,exp(-x.*x),'k')
```

계단 그래프

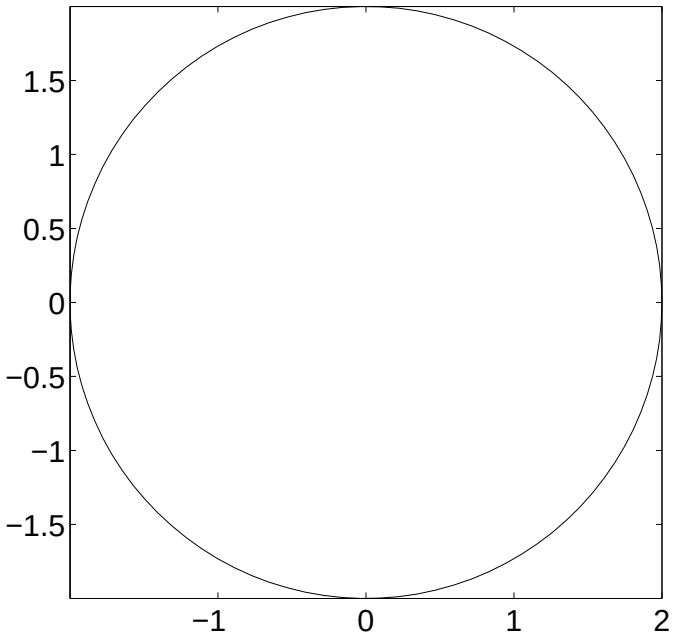


그림 1.6: 좌표계가 image인 경우

다음은 계단 그래프를 그리기 위한 명령어이다.

- **stairs(x)**
  벡터 x를 계단 그래프로 그린다.

- **stairs(x,y)**
  벡터 x에 지정된 벡터 y의 원소를 계단 그래프로 그리는 데, x의 원소는 오름차순이고, 일정한 간격을 가져야 한다.

다음 코드는 $y = \sin x$의 그래프를 stairs 명령어를 이용하여 계단 그래프로 그린 것이다.

```
x = -5:0.25:5;
stairs(x,sin(x),'k-')
```

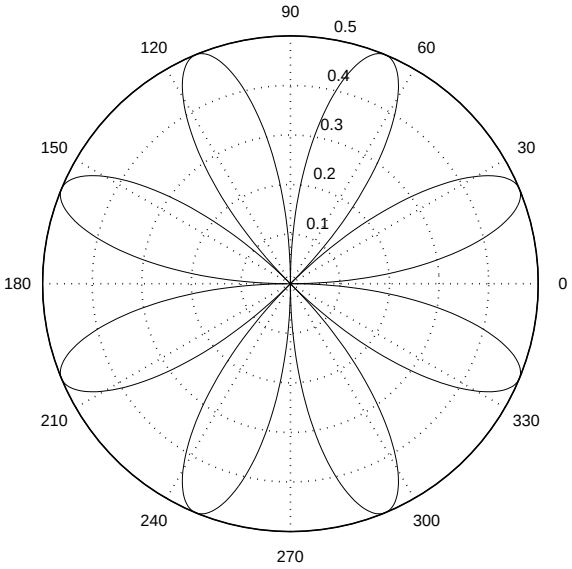


그림 1.7: $r = \sin 2\theta \cos 2\theta$, $0 \le \theta \le 2\pi$의 그래프

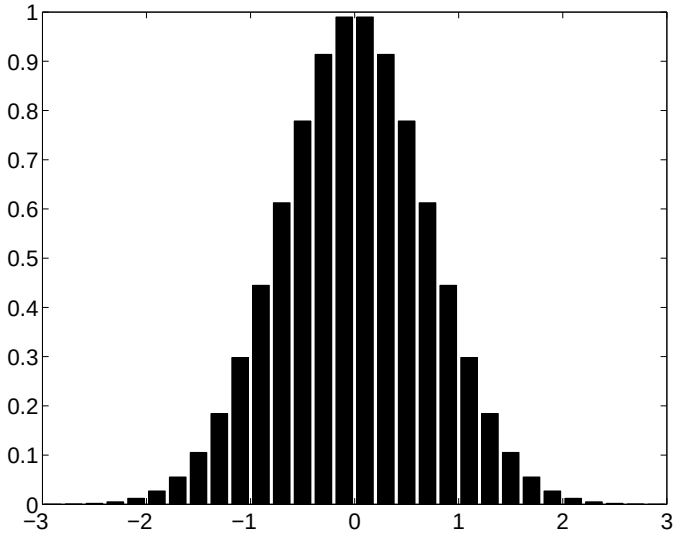


그림 1.8: $y = e^{-x^2}$의 막대 그래프

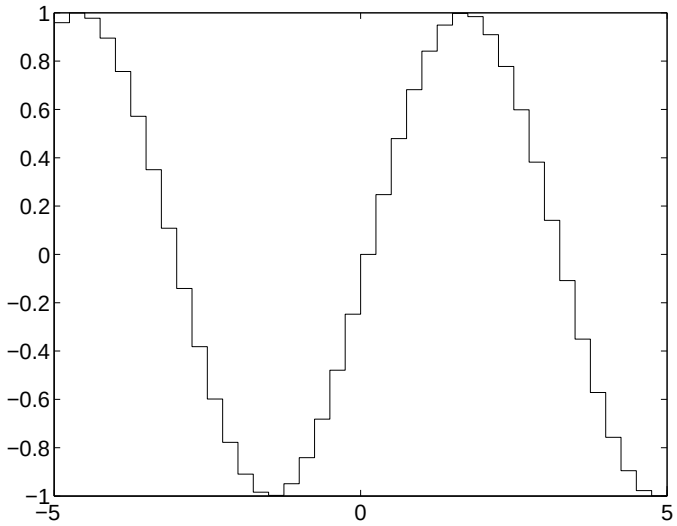


그림 1.9: $y = \sin x$의 계단 그래프

원 그래프

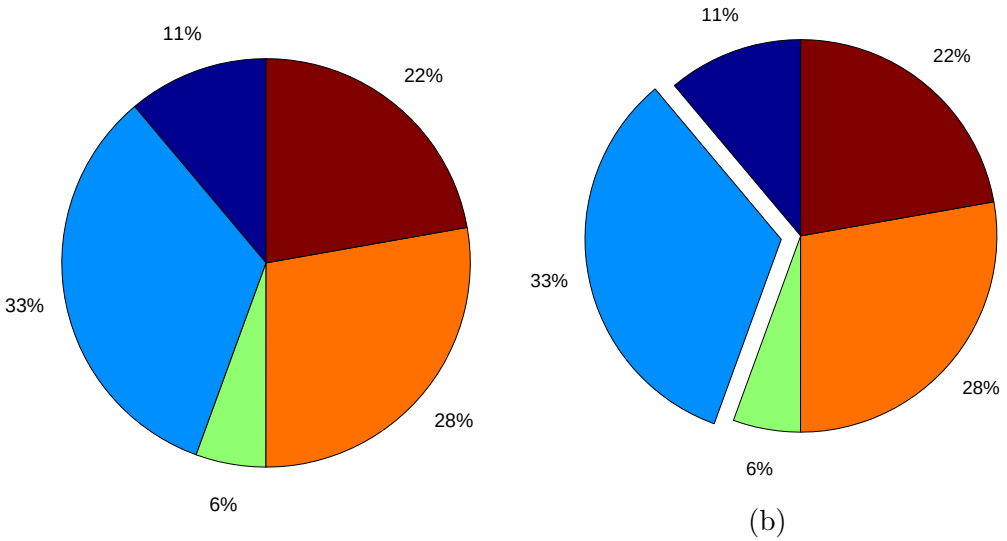
자료 x를 손쉽게 비교해서 볼 수 있는 그래프가 원 그래프이다. 다음은 원 그래프를 그리기 위한 명령어이다.

- **pie(x)**
  자료 x를 가지고 원 그래프를 그린다.

- **pie(x, x==max(x))**
  가장 큰 부채꼴을 원에서 추출할 때 사용한다.

다음 코드는 변량 '$1, 3, 0.5, 2.5, 2$'를 pie 명령어를 이용하여 원 그래프를 그린 것이다.

```
x = [1 3 0.5 2.5 2];
pie(x)
```



(b)

그림 1.10: (a) 변량 '$1, 3, 0.5, 2.5, 2$'의 원 그래프. (b) 가장 큰 부채꼴을 원에서 추출한 원 그래프.

2.3 3차원 그래프

2.3.1 Line 그래프

다음은 3차원의 그래프를 그리는 명령어이다.

- **plot3(x,y,z)**
  $\mathbf{f} = x(t)\mathbf{i} + y(t)\mathbf{j} + z(t)\mathbf{k}$의 그래프를 그린다.

다음 코드는 plot3 명령어를 이용하여 t가 $[0, 10]$에서 $\mathbf{f} = \cos 3t\mathbf{i} + \sin 3t\mathbf{j} + t^2\mathbf{k}$의 그래프를 그린 것이다.

```
t = 0:0.01:10; x = cos(3*t); y = sin(3*t); z = t.^2;
plot3(x,y,z,'k-')
```

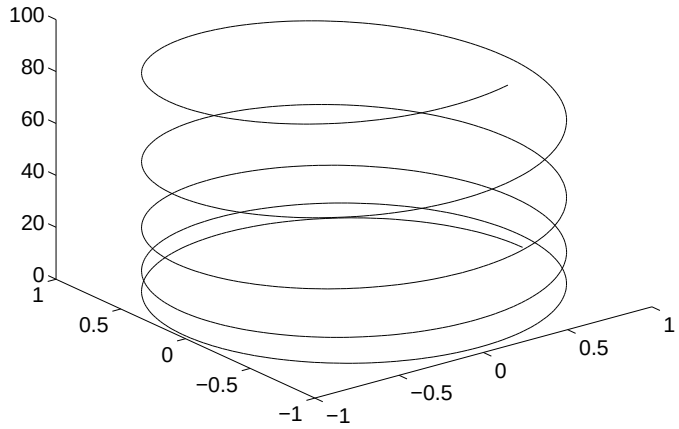


그림 1.11: $\mathbf{f} = \cos 3t\mathbf{i} + \sin 3t\mathbf{j} + t^2\mathbf{k}$의 그래프

2.3.2 2변수 함수의 그래프

$z = f(x, y)$와 같은 2변수 이상의 함수의 그래프는 plot 명령어를 이용할 수가 없다. 그래서 이러한 그래프를 그리기 위해서 mesh란 명령어를 이용한다.

특히, 2변수 그래프에서 좌표 (x, y)를 설정하기 위해서 meshgrid 명령어를 이용한다.

- **[X,Y]=meshgrid(xmin:xmax, ymin:ymax)**

아래는 2변수 함수 $z = xe^{-x^2-y^2}$를 x는 $[-2, 2]$, y는 $[-2, 2]$ 범위 안에서 그린 것이다.

```
[X,Y] = meshgrid(-2:0.1:2, -2:0.1:2);
Z = X.*exp(-X.^2 -Y.^2); mesh(Z)
```

Mesh를 이용하여 함수의 그래프를 그릴 때에는 함수의 데이터 크기에 주의하여야 한다. 아래는 함수 $z = xe^{-x^2-y^2}$를 x는 $[-2, 2]$, y는 $[-1, 1]$ 범위 안에서 그리는 MATLAB 코드이다.

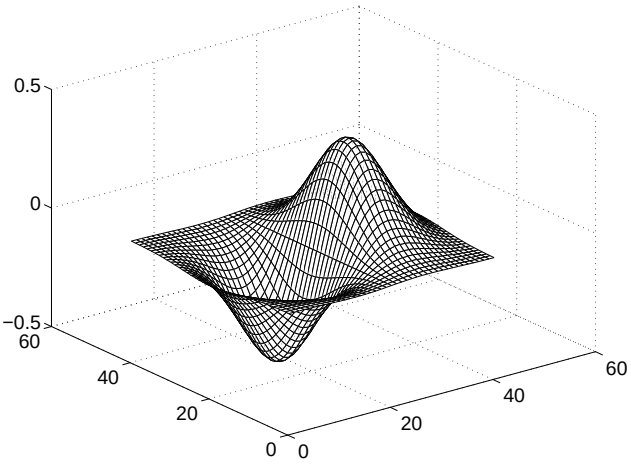


그림 1.12: mesh를 이용한 $z = xe^{-x^2-y^2}$의 그래프

```
x=linspace(-2,2,41); y=linspace(-1,1,21); [xx,yy]=meshgrid(x,y);
for i=1:41
 for j=1:21
  z(i,j)=x(i)*exp(-x(i)^2-y(j)^2);
 end
end
mesh(xx,yy,z)
```

이 MATLAB 코드를 실행할 경우 다음과 같은 오류가 생긴다.

```
Data dimensions must agree. Error in ==> mesh(xx,yy,z)
```

이는 데이터의 차원이 일치되지 않았음을 의미한다. 이 경우 whos를 이용하여 데이터 크기를 알아보면 다음과 같다. xx, yy와 z의 차원 값이 다른 것을 확인할 수 있다.

```
>> whos
  Name        Size              Bytes  Class     Attributes
```

```
ans        1x1              8  double
i          1x1              8  double
j          1x1              8  double
x          1x41           328  double
xx        21x41          6888  double
y          1x21           168  double
yy        21x41          6888  double
z         41x21          6888  double
```

따라서 MATLAB 코드가 제대로 실행되기 위해서는 `z'`을 해서 `z`를 transpose해줘야 한다.

Mesh로 그린 그래프에 contour의 효과를 주고 싶을 때 meshc 명령어를 이용을 한다. 아래 코드는 위의 예제를 meshc를 이용하여 그린 것이다.

```
[x, y] = meshgrid(-2:0.1:2, -2:0.1:2);
z = x.*exp(-x.^2 -y.^2); meshc(x,y,z);
```

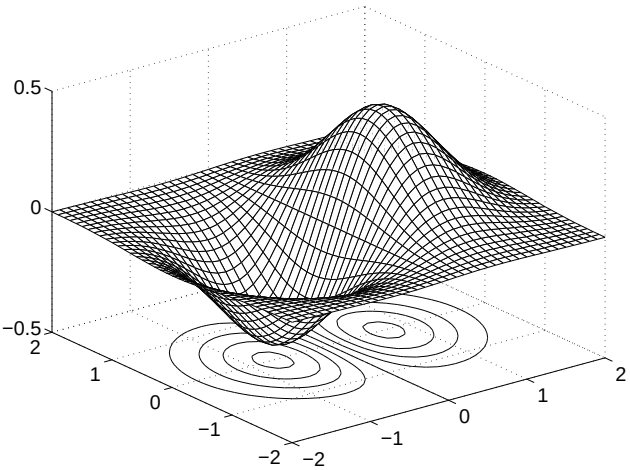


그림 1.13: meshc를 이용한 $z = xe^{-x^2-y^2}$의 그래프

2.3.3 Contour 그래프

Contour는 함수값을 높이로 나타내는 등고선의 형태로 나타내는 그래프이다. 종류에 따라 2차원 또는 3차원 그래프로 그릴 수 있다. 다음은 $z = f(x, y)$의 2차원 contour 그래프를 그리는 명령어이다.

- **contour(z)**
  행렬 z에 대한 값을 높이로 하는 2차원 contour 그래프를 그린다.

- **contour(z,n)**
  2차원 contour 그래프에서 등고선의 수를 n인 그래프로 나타낸다.

- **contour3(z)**
  행렬 z에 대한 값을 높이로 하는 3차원 contour 그래프를 그린다.

- **contour3(z,n)**
  3차원 contour 그래프에서 등고선의 수를 n인 그래프로 나타낸다.

다음은 함수 $z = xe^{-x^2-y^2}$의 x는 $[-2, 2]$, y는 $[-1, 1]$ 범위 안에서x와 y가 각각 $[-2, 2]$의 범위에서 20개의 등고선을 갖는 2차와 3차의 contour 그래프를 그려보자.

2차 Contour 그래프

```
[X,Y] = meshgrid(-2:0.1:2, -2:0.1:2);
Z = X.*exp(-X.^2 -Y.^2); contour(Z,20)
```

3차 Contour 그래프

```
[X,Y] = meshgrid(-2:0.1:2, -2:0.1:2);
Z = X.*exp(-X.^2 -Y.^2); contour3(Z,20)
```

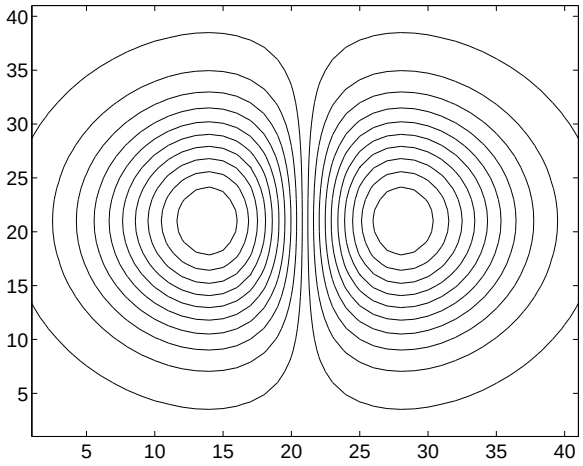


그림 1.14: 2D Contour 그래프

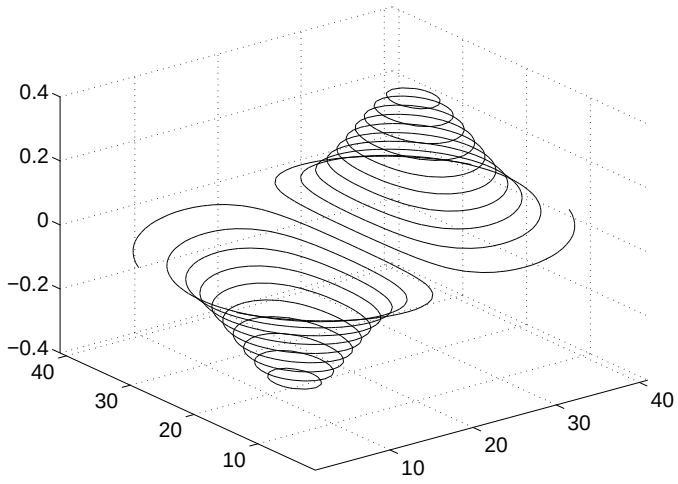


그림 1.15: 3D Contour 그래프

2.3.4 그래디언트(Gradient)와 벡터장(Vector Field)

이번에는 $z = f(x, y)$의 근사 그래디언트(Gradient)를 구하고 이를 이용하여 벡터장을 그려보는 방법을 배워보도록 하자. 이 방법은 미분방정식의 근사 방법 중 하나인 Direction Field를 그릴 때 유용하게 쓰인다. 다음은 그래디언트와 벡터장을 그리는 명령어이다.

- **[px,py]=gradient(z,dx,dy)**
  그래디언트 구하기 $px = dz/dx$, $py = dz/dy$
- **quiver(x,y,u,v)**
  점 (x, y)에서 (u, v) 성분을 갖는 벡터장을 그리는 명령어

다음은 함수 $z = xe^{-x^2-y^2}$를 x와 y가 각각 $[-2, 2]$의 범위 에서 등고선과 벡터장의 그래프를 그린 것이다.

```
[x, y] = meshgrid(-2:0.2:2); z = x.*exp(-x.^2 -y.^2);
[px,py] = gradient(z,0.2,0.2); contour(x,y,z)
hold on; quiver(x,y,px,py)
```

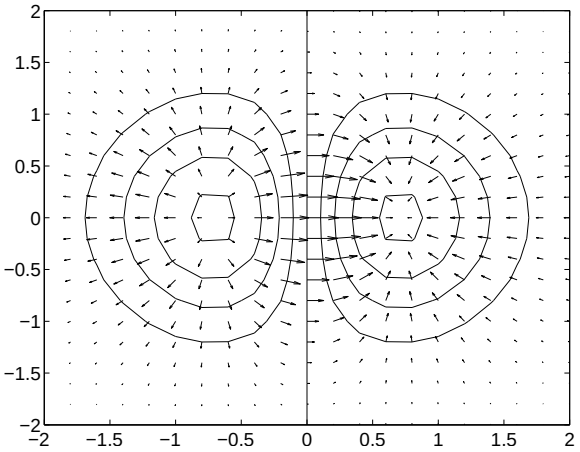


그림 1.16: $z = xe^{-x^2-y^2}$의 그래프

2.3.5 매개 변수 방정식의 그래프

$y = f(x)$와 같은 함수가 아닌 매개 변수 t로 된 매개 변수 방정식 $x = f(t)$, $y = g(t)$의 그래프를 그리는 방법에 대해서 알아보자. 보통의 매개 변수 방정식은 plot이라는 명령어를 이용하여 그래프를 그린다. 다음 코드는 매개 변수 방정식을 t가 $[0, 2\pi]$ 안에서 그래프를 그린 것이다.

$$x = 3\sin t, \qquad y = 3\cos t$$

```
t = linspace(0,2*pi); x = 3*sin(t); y = 3*cos(t);
plot(x,y,'k-'); grid on; axis image
xlabel('x'), ylabel('y')
```

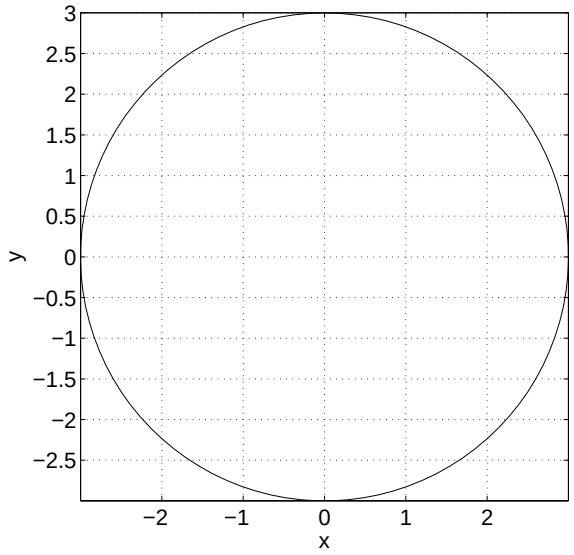


그림 1.17: $x = 3\sin t$, $y = 3\cos t$의 그래프

매개 변수 방정식이 두 개의 식이 아닌 세 개의 식으로 되어있을 때 그래프를 그리는 방법을 알아보자. 보통 식이 3개인 방정식은 아래와 같다.

$$x = f(t), \quad y = g(t), \quad z = h(t)$$

위와 같은 매개 변수 방정식을 그릴 때에는 plot3 명령어를 이용한다. 다음은 아래 매개 변수 방정식을 t의 범위를 $[0, 2\pi]$ 안에서 그래프를 그린 것이다.

$$x = \sin t, \quad y = \cos t, \quad z = t$$

```
t = linspace(0,2*pi); x = sin(t); y = cos(t); z = t;
plot3(x,y,z,'k-'); grid on;
xlabel('x'), ylabel('y'), zlabel('z')
```

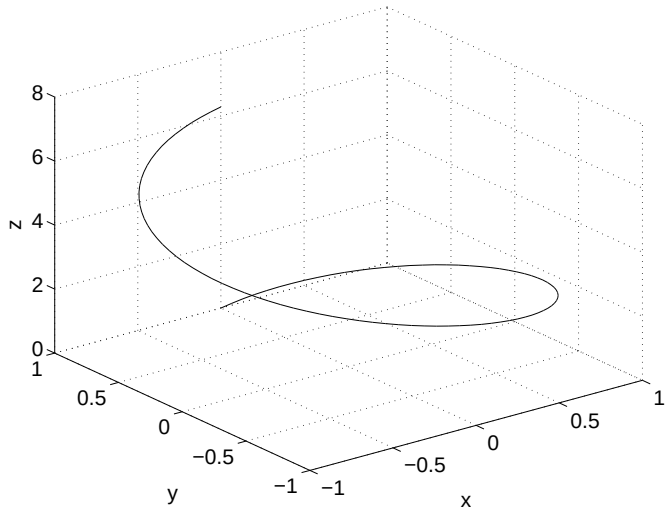


그림 1.18: $x = \sin t$, $y = \cos t$, $z = t$의 그래프

이제 변수가 한 개가 아닌 두 개일 경우의 그래프를 그려보자. 다음은 아래 매개 변수 방정식을 s와 t가 각각 $[-2, 2]$ 범위 안에서 그래프를 그린 것이다.

$$x = s, \quad y = t, \quad z = t^2/2 - s^2/3$$

```
s = linspace(-2,2,30); t = linspace(-2,2,30);
[ss tt] = meshgrid(s,t);
x = ss; y = tt; z = tt.^2/2 - ss.^2/3; mesh(x,y,z)
```

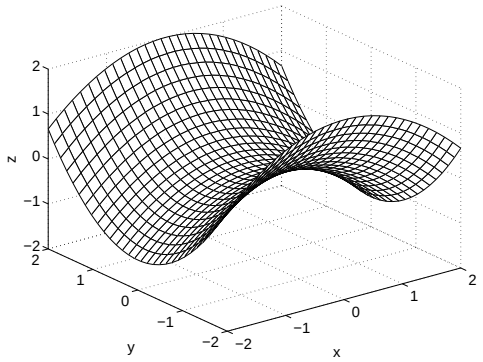


그림 1.19: $x = s$, $y = t$, $z = t^2/2 - s^2/3$의 그래프

중심이 $(0.5, 0.5, 0.5)$이고 반지름이 0.25인 구의 그래프를 그려보자.

patch는 꼭지점 좌표를 입력하고 꼭지점으로 이루어진 면의 정보를 입력해서 도형을 그리는 명령어이다. x, y, z 좌표를 patch형태로 변환하기 위해 isosurface를 사용한다.

```
>> p=patch(isosurface(x좌표,y좌표,z좌표,f데이터,기준숫자))
```

이는 '기준숫자'와 같은 값을 갖는 'f데이터'의 그래프를 그려준다. 예를 들어 $f(1,1,1) = 1$, $f(1,2,1) = 1$, $f(1,3,1) = 1$라는 데이터가 있고 기준숫자를 1로 잡는다면 $(1,1,1)$, $(1,2,1)$, $(1,3,1)$을 연결해주는 도형을 그려준다.

set 명령어를 이용하면 면과 꼭지점의 색깔을 정의할 수 있다.

```
>> set(p,'FaceColor','red','EdgeColor','none');
>> daspect([1 1 1])
```

다음은 명암을 주는 명령어이다.

```
>> camlight; lighting phong;
```

다음은 'isosurface'와 'patch'를 이용하여 전체적인 3차원 형상을 그린 것이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% show_isosur.m %%%%%%%%%%%%%%%%%%%
clear; n=64; x=linspace(0,1,n); y=x; z=x;
[xx,yy,zz]=meshgrid(x,y,z);
for k=1:n
 for j=1:n
  for i=1:n
   S(i,j,k)=0.5*(1.0+tanh(0.25-sqrt((x(i)-0.5)^2+(y(j)-0.5)^2+...
            (z(k)-0.5)^2)));
  end
 end
end
p=patch(isosurface(xx,yy,zz,S,0.5));
set(p,'FaceColor','magenta','EdgeColor','none'); daspect([1 1 1])
camlight;lighting phong; axis image;
axis([0.2 0.8 0.2 0.8 0.2 0.8])
```

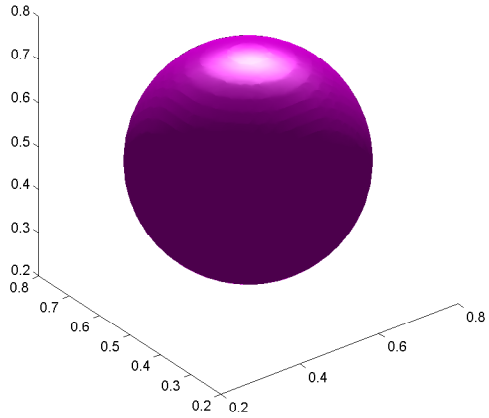


그림 1.20: 중심이 $(0.5, 0.5, 0.5)$이고 반지름이 0.25인 구의 그래프

참고 문헌

[1] Bell JB, Colella P, Glaz HM. A second-order projection method for the incompressible Navier-Stokes equations. *Journal of Computational Physics* 1989; **85**:257–283.

[2] Chorin AJ. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics* 1967; **2**:12–26.

[3] Li J, Renardy Y. Numerical study of flows of two immiscible liquids at low Reynolds number. *SIAM Review* 2000; **42**:417–439.

[4] Harlow FH, Welch JE. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids* 1965; **8**:2182–2189.

[5] Cahn JW, On spinodal decomposition. Acta Metall 1961;9:795-801.

[6] Cahn JW and Hilliard JE, Free energy of a non-uniform system. I. Interfacial free energy. J Chem Phys 1958;28:258-267.

[7] Choi J-W, Lee HG, Jeong D, Kim J. An unconditionally gradient stable numerical method for solving the Allen-Cahn equation. Phys A 2009;388:1791-803.

[8] C.W. Shu, S. Osher, Efficient implementation of essentially non-oscillatory shock capturing schems II, J. Comput. Phys. 83 (1989) 32-78.

[9] R.C.Y. Chin, T.A. Manteuffel and J. de Pillis, ADI as a preconditioning for solving the convection-diffusion equation. *SIAM Journal on Scientific Computing*, **5** (1984) 281-299.

[10] K.J. Hout and S. Foulon, ADI finite difference schemes for option pricing in the Heston model with correlation. *International Journal of Numerical Analysis and Modeling*, **7**(2) (2010) 303-320.

[11] D.J. Duffy, *Finite difference methods in financial engineering:a partial differential equation approach.* John Wiley and Sons, Sussex, 2006.

[12] S. Ikonen and J. Toivanen, Operator splitting methods for American option pricing. *Applied mathematics letters*, 2004, **17**: 809–814.

찾아보기