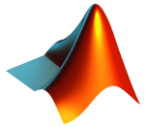
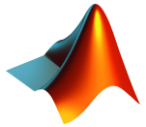
MATLAB 프로그래밍 및 실습

14강. 프로그래밍 스킬의 이해 2



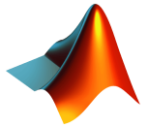
오늘 배울 내용

- Symbolic math
- 예외처리
- global을 지양해야 하는 이유
- eval을 지양해야 하는 이유
- 강의 마무리



Symbolic math*

*Symbolic Math Toolbox required



대수학 (algebra)

- 지금까지 배운 것

```
>> x = 2;
>> y1 = x^2
y1 =
     4
>>
>> y2 = (x+1)*(x+2)
y2 =
    12
```

```
>> x = pi;
>> sin(x)
ans =
   1.2246e-16
```

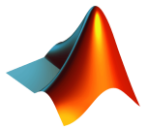
- 있으면 좋을 것 같은 것

```
>> x
x =
x
>> y1 = x^2
y1 =
x^2
>>
>> y2 = (x+1)*(x+2)
y2 =
(x + 1)*(x + 2)
>> expand(y2)
ans =
x^2 + 3*x + 2
```

```
>> x = sym(pi);
>> sin(x)
ans =
0
```

```
>> sym(0.1) + sym(0.2) -sym(0.3)
ans =
0
```

※ 변수를 "값"이 아닌 "변수" 그대로 둘 수 있다면?
※ double의 "근사값"이 아닌 "참값" 그대로 둘 수 있다면?



symbolic expressions

```
syms a b c x y

f = a*x^2 + b*x + c;
g = exp(x) + log(x*y) + pi*sin(x/y);
h = x*3/7 + y*5/9 + y*1/3;
```

```
>> f
f =
a*x^2 + b*x + c
>>
>> g
g =
log(x*y) + exp(x) + pi*sin(x/y)
>>
>> h
h =
(3*x)/7 + (8*y)/9
>>
>> [3/7, 8/9]
ans =
    0.4286    0.8889
```

a, b, c, x, y가 symbolic
-> f, g, h도 symbolic

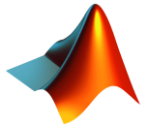
pi를 근사값으로
쓰지 않음

5/9 + 1/3
-> 8/9로 계산해줌

3/7, 8/9 등을
분수 그대로 표현함

```
>> p = (exp(1))*x
p =
(3060513257434037*x)/1125899906842624
```

완벽하지는 않음…



symbolic expressions – numbers

```
>> a = sym(3);
>> b = sym(5);
>>
>> e = b/a + sqrt(a)
e =
3^(1/2) + 5/3
>>
>> a = 3;
>> b = 5;
>> e = b/a + sqrt(a)
e =
    3.3987
```

숫자도 symbolic이 될 수 있다.

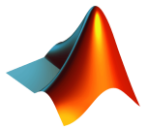
근사값을 구하지 않는다.

```
>> sym(2)^1000
ans =
1071508607186267320948425049060001810561404811705533607443750388370351(
```

double은 근사값으로 표현한다.

```
>>
>> a = sym(3);
>> a+3/4
ans =
15/4
```

피연산자에 symbolic이 있으면
연산 결과도 symbolic이 된다.

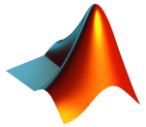


expand, collect

```
syms x

f = (x + 3)*(x^2 + 2*x + 2)*(x^3-1);
g = (x - 1)*(x + sin(x))*(x^2 + 1);
```

```
>> f
f =
(x^3 - 1)*(x + 3)*(x^2 + 2*x + 2)
>> expand(f)
ans =
x^6 + 5*x^5 + 8*x^4 + 5*x^3 - 5*x^2 - 8*x - 6
>>
>> g
g =
(x^2 + 1)*(x + sin(x))*(x - 1)
>> expand(g)
ans =
x^3*sin(x) - sin(x) - x^2*sin(x) - x + x*sin(x) + x^2 - x^3 + x^4
>> collect(g)
ans =
x^4 + (sin(x) - 1)*x^3 + (1 - sin(x))*x^2 + (sin(x) - 1)*x - sin(x)
>>
```

```
>> syms x y
>> expand(sin(x+y))
ans =
cos(x)*sin(y) + cos(y)*sin(x)
```



factor, simplify, pretty

```
syms x y

h = x^3 + 6*x^2 + 11*x + 6;
factor(h)
pretty(h)

p = (x^2 + 5*x + 6)/(x + 2);
q = (x + y)/(1/x+1/y);

simplify(p)
simplify(q)
```

$$h = x^3 + 6x^2 + 11x + 6$$
$$= (x + 3)(x + 2)(x + 1)$$

$$p = \frac{x^2 + 5x + 6}{(x + 2)}$$

$$q = \frac{(x + y)}{\left(\frac{1}{x} + \frac{1}{y}\right)}$$

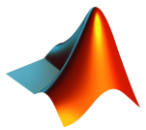
```
>> h
h =
x^3 + 6*x^2 + 11*x + 6
>> factor(h)
ans =
[ x + 3, x + 2, x + 1]
>>
>> pretty(h)
 3      2
x  + 6 x  + 11 x + 6

>>
>> p
p =
(x^2 + 5*x + 6)/(x + 2)
>> simplify(p)
ans =
x + 3
>> q
q =
(x + y)/(1/x + 1/y)
>> simplify(q)
ans =
x*y
```

matrix operations

```
>> A = sym('A%d%d', [2,2])
A =
[ A11, A12]
[ A21, A22]
>> B = sym('B%d%d', [2,2])
B =
[ B11, B12]
[ B21, B22]
>> A*B
ans =
[ A11*B11 + A12*B21, A11*B12 + A12*B22]
[ A21*B11 + A22*B21, A21*B12 + A22*B22]
>> A.*B
ans =
[ A11*B11, A12*B12]
[ A21*B21, A22*B22]
>> inv(A)
ans =
[  A22/(A11*A22 - A12*A21), -A12/(A11*A22 - A12*A21)]
[ -A21/(A11*A22 - A12*A21),  A11/(A11*A22 - A12*A21)]
```

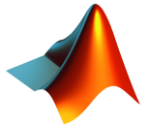
$$B = \frac{1}{2EI_b}\begin{bmatrix} 3l_b^2 & \frac{8}{3}l_b^3 & l_b^3 \\ l_b^2 & l_b^3 & \frac{8}{3}l_b^3 \\ 4l_b & 3l_b^2 & l_b^2 \end{bmatrix}$$

$$\begin{bmatrix} & & & & & & & & & -1 & 0 & b_1 \\ B & & 0 & & 0 & & 0 & & & 0 & -1 & -a_1 \\ & & & & & & & & & 0 & 0 & -1 \\ & & & & & & & & & -1 & 0 & b_2 \\ 0 & & B & & 0 & & 0 & & & 0 & 1 & a_2 \\ & & & & & & & & & 0 & 0 & -1 \\ & & & & & & & & & 1 & 0 & -b_3 \\ 0 & & 0 & & B & & 0 & & & 0 & 1 & a_3 \\ & & & & & & & & & 0 & 0 & -1 \\ & & & & & & & & & 1 & 0 & -b_4 \\ 0 & & 0 & & 0 & & B & & & 0 & -1 & -a_4 \\ & & & & & & & & & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & & \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & & 0 & \\ 1 & -b_1 & a_1 & 1 & -b_2 & -a_2 & 1 & b_3 & -a_3 & 1 & b_4 & a_4 & & & \end{bmatrix}$$



matrix operations

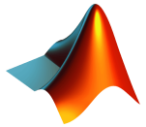
```
>> C = sym('C%d%d', [3, 3])
C =
[ C11, C12, C13]
[ C21, C22, C23]
[ C31, C32, C33]
>> iC = inv(C)
iC =
[  (C22*C33 - C23*C32)/(C11*C22*C33 - C11*C23*C32 - C12*C21*C33 + C12*C23*C
[ -(C21*C33 - C23*C31)/(C11*C22*C33 - C11*C23*C32 - C12*C21*C33 + C12*C23*C
[  (C21*C32 - C22*C31)/(C11*C22*C33 - C11*C23*C32 - C12*C21*C33 + C12*C23*C
>> C*iC
ans =
[ (C13*(C21*C32 - C22*C31))/(C11*C22*C33 - C11*C23*C32 - C12*C21*C33 + C12*
[ (C23*(C21*C32 - C22*C31))/(C11*C22*C33 - C11*C23*C32 - C12*C21*C33 + C12*
[ (C33*(C21*C32 - C22*C31))/(C11*C22*C33 - C11*C23*C32 - C12*C21*C33 + C12*
>> simplify(C*iC)
ans =
[ 1, 0, 0]
[ 0, 1, 0]
[ 0, 0, 1]
```



solve

```
syms a b c x y

solve(exp(2*x) - 5)
solve(x^2 - x - 6)
solve(cos(2*x)+3*sin(x)-2)
solve(a*x^2 + b*x + c)
```

```
>> solve(exp(2*x) - 5)
ans =
log(5)/2
>>
>> solve(x^2 - x - 6)
ans =
 -2
  3
>>
>> solve(cos(2*x)+3*sin(x)-2)
ans =
     pi/2
     pi/6
 (5*pi)/6
>>
>> solve(a*x^2 + b*x + c)
ans =
 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```



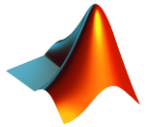
differentiation, integration

```
syms x

f = (x + sin(x))*(x^2 + 1);
g = x^3 + 6*x^2 + 11*x + 6;
```

$$f = (x + \sin(x))(x^2 + 1)$$

$$g = x^3 + 6x^2 + 11x + 6$$

```
>> diff(f)
ans =
2*x*(x + sin(x)) + (x^2 + 1)*(cos(x) + 1)
>> collect(diff(f))
ans =
(cos(x) + 3)*x^2 + 2*sin(x)*x + cos(x) + 1
>> diff(g)
ans =
3*x^2 + 12*x + 11
>>
>> int(f)
ans =
cos(x) - x^2*cos(x) + 2*x*sin(x) + x^2/2 + x^4/4
>> int(g)
ans =
x^4/4 + 2*x^3 + (11*x^2)/2 + 6*x
>> int(g, 0, 1)
ans =
55/4
```



적분의 결과도 symbolic

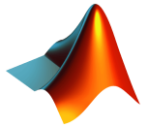
subs, double

```
syms x

f = (x + sin(x))*(x^2 + 1);
g = x^3 + 6*x^2 + 11*x + 6;

G = int(g, 0, 1);
double(G)

subs(f, x, pi/2)
double(subs(f, x, pi/2))
```

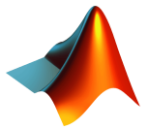
```
>> G = int(g, 0, 1)
G =
55/4
>> double(G)
ans =
   13.7500
>>
>> subs(f, x, pi/2)
ans =
(pi/2 + 1)*(pi^2/4 + 1)
>> double(subs(f, x, pi/2))
ans =
    8.9140
```

double을 하지 않으면 symbolic

double()을 해야 값이 됨



예외처리



자주 뜨는 에러들

- 자주 뜨는 에러들
  - 행렬의 인덱스가 범위를 벗어나거나 자연수가 아닐 때
  - 행렬 연산 시 행렬 크기가 안 맞을 때
  - 정의되지 않은 함수나 변수를 읽으려고 할 때
  - 스크립트를 함수처럼 쓰려고 했을 때
  - 함수 인자가 안 맞을 때
  - 열 수 없는 파일을 열 때 (파일이 없거나 권한이 없을 때)

- 방법1: 발생 가능한 모든 에러에 대해 if문으로 분기
  - 문제점: 예상치 못한 에러가 발생했다면?

- 방법2: **"용서가 허락보다 쉽다."**

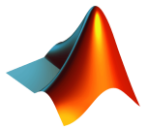
try-catch

```
% roll 3 dices 100 times
diceRolled = randi(6, [100, 3]);

idx = input('Which roll # do you want to see?: ');
disp(diceRolled(idx,:))
% error occurs when...
%       idx>100
%       idx is not a positive integer.
```

```
% utilizing try-catch for any errors
try
    disp(diceRolled(idx,:))
catch
    if idx>100 || idx<1
        warning('index out of range')
    elseif ~(floor(idx)==ceil(idx))
        warning('index must be a positive integer')
    else
        warning('unknown error')
    end
end

% utilizing try-catch giving specific error type
try
    disp(diceRolled(idx,:))
catch ME
    if strcmpi(ME.identifier, 'MATLAB:badsubscript')
        warning(ME.message)
    else
        warning('unknown error')
    end
end
```

- 용서가 허락보다 쉽다.
- 허락 = if
- 용서 = try-catch

전역변수



https://kr.mathworks.com/matlabcentral/answers/51946-systematic-do-not-use-global-don-t-use-eval
https://matlab.fandom.com/wiki/FAQ#Are_global_variables_bad.3F

전역변수를 쓰는 이유

```
function F = calc_grav_force(m1, m2, r)

global G
F = G*m1*m2/r;

end
```

```
function tau = get_torsion(angle)

global G
G = 26e9;
tau = G*angle

end
```

```
global G
G = 6.67408e-11;

F = calc_grav_force(1, 1, 1);
disp(G)
tau = get_torsion(0.001);
disp(G)
```

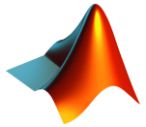
- 함수에 전달해야 할 **상수**가 많을 때
- 모든 함수가 **공통으로 쓰는 파라미터**가 많을 때

- 왜 위험한가?
  - G = 중력상수? 재료의 sheer modulus?
  - h = 플랑스 상수? 허블 상수 (h0)?
  - t = 시간? 온도?
  - v = 부피? 속도?
  - 여러 사람이 협업하는 프로젝트라면?

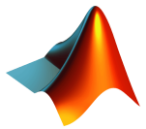
전역변수를 피하는 방법

```
function R = blackbody(wavelength, temperature, constants)

h = constants.h;
c = constants.c;
k = constants.k;

R = ...;

end
```

- 함수는 workspace를 공유하지 않아야 한다!



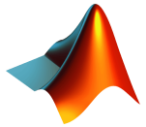
eval = evil



What is eval?

```
for i=1:3
    % data_1 = load('hw7_prob1_data.mat);
    % data_2 = load('hw7_prob2_data.mat);
    % data_3 = load('hw7_prob3_data.mat);

    s = ['data_' num2str(i) '=load(hw7_prob' num2str(i)  '_data.mat;'];
    eval(s);
end
```

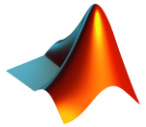
- 장점: 어떤 코드든 만들어낼 수 있다.
- 단점:
  - 실행속도가 느려진다.
    - JIT 컴파일러가 동작하지 않음
  - 가독성이 떨어진다.
    - 위 예제에서 문법오류를 찾아보자.



What is eval?

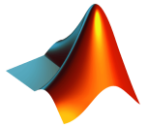
```
for i=1:3
    % data_1 = load('hw7_prob1_data.mat);
    % data_2 = load('hw7_prob2_data.mat);
    % data_3 = load('hw7_prob3_data.mat);

    s = ['data_' num2str(i) '=load(hw7_prob' num2str(i)  '_data.mat;'];
    eval(s);
end
```

- 단점
  - 디버깅이 어려워진다.
    - code helper, syntax highlighting이 동작하지 않는다.
  - 안정성 이슈
    - 의도치 않은 동작이 발생해도 예상 및 방지하기 어렵다.
  - eval은 한번 쓰면 계속 쓰게 된다.



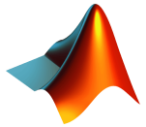
eval의 대안

```
% using sprintf, cell and struct array
for i=1:3
    fname = sprintf('hw7_prob%d_data.mat',i);
    A{i} = load(fname); % using cell array
    B(i).data = load(fname); %using struct array
end


% using cell array of function handles
fns = {@(n)rand(n), @(n)randn(n), @(n)randi(5,[n,n])};
data = zeros(5,5,length(fns));
for i=1:length(fns)
    data(:,:,i) = fns{i}(5);
end


% creating an anonymous function with array output
fn = @(n) [rand(n,1) randn(n,1) randi(5,[n,1])];
data = fn(10);
```



eval의 대안

```
% using feval
x = 1:10;
y = randi(10,size(x));
fns = {@plot, @bar, @stem};
figure,
for i=1:length(fns)
    subplot(3,1,i);
    feval(fns{i},x,y);
end


% struct field name
for i=1:3
    ifile = sprintf('hw7_prob%d_data.mat',i);
    ifield = sprintf('prob%d',i);
    data.(ifield) = load(ifile);
end
```



Q&A