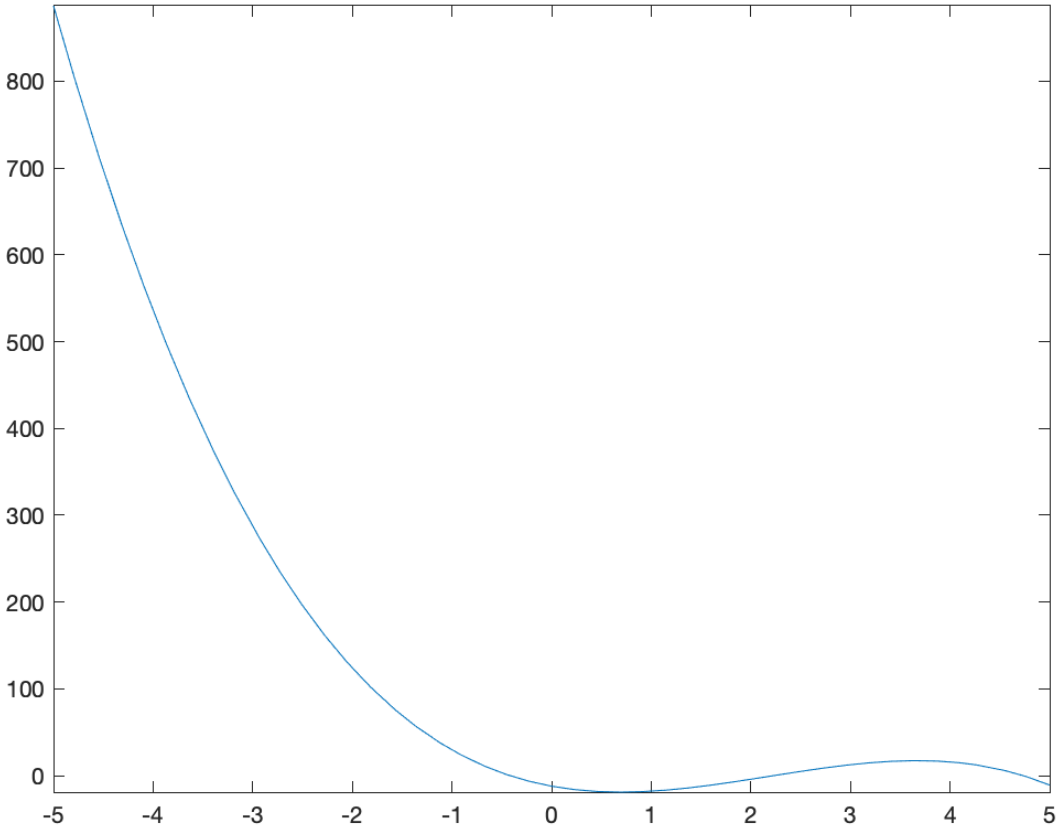
```
clc; clear all;


%%%%%%%%%%%%%%% Problem 1 %%%%%%%%%%%%%%%
func = @(x) -12 -21*x + 18*x^2 -2.75*x^3;
xl=-1; xu=0; error=0.01; iter = 10;
```




```
% (a)
syms x;
y=-12 -21*x + 18*x^2 -2.75*x^3;
solve(y==0, x)
figure(1)
fplot(y)


print -dpng 'p1.png'
```

```
% (b)
bisection(func, xl, xu, error, iter);

% (c)
false_loc(func, xl, xu, error, iter);



%%%%%%%%%%%%%%% Problem 2 %%%%%%%%%%%%%%%
syms x;
func = x^3 -6*x^2+11*x-6.1;
xi=3.5; xl=2.5; xu=3.5; delta=0.01; iter=4;

% (a)
y=x^3 -6*x^2+11*x-6.1;
solve(y==0,x)
figure(2)
fplot(func)

print -dpng 'p2.png'

% (b)
NR(func, diff(func), xi, error, iter);

% (c)
func = @(x) x^3 -6*x^2+11*x-6.1;
SC(func, xl, xu, error, iter);

% (d)
SC(func, xi, delta, error, iter);

% (e)

%%%%%%%%%%%%%%% Problem 3 %%%%%%%%%%%%%%%
```

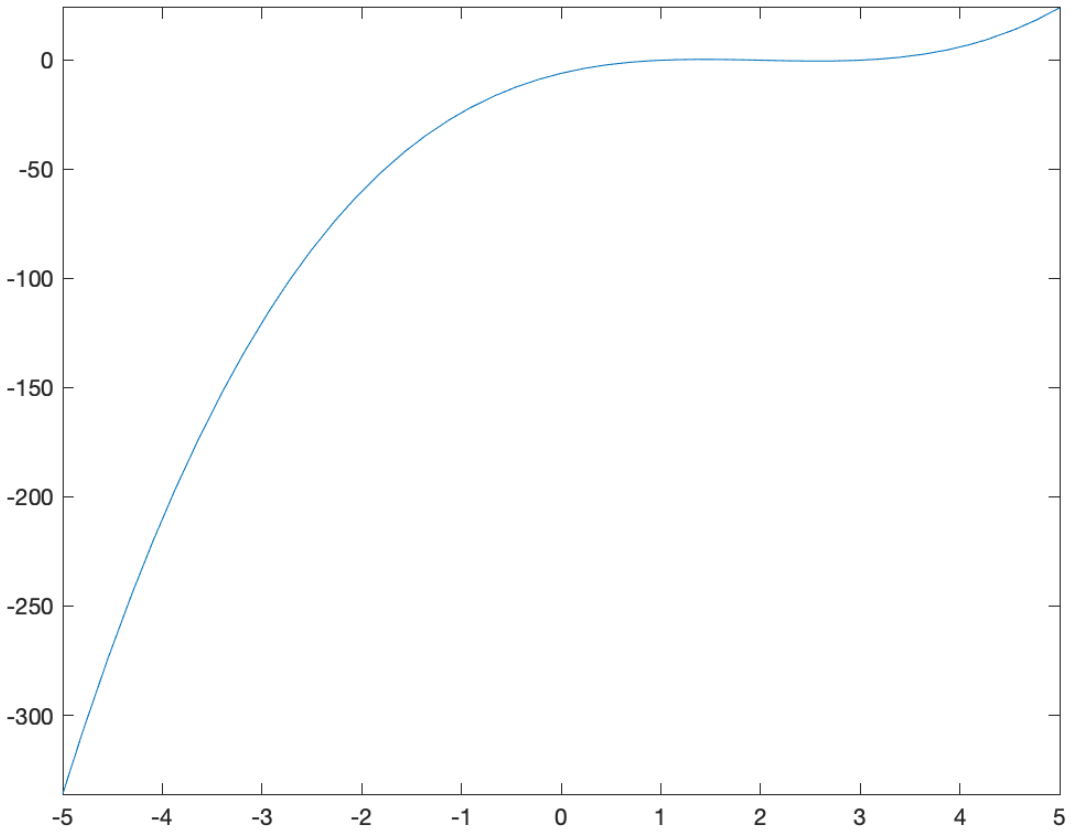
```
syms x1 x2;

y1 = 4*x1 - 8*x2;
y2 = x1 + 6*x2;

X = solve(y1==-24, x1, x2);
4*X.x1 - 8*X.x2 == -24

X = solve(y2==34, x1, x2);
X.x1 + 6*X.x2 == 34



%%%%%%%%%%%%%%% Problem 4 %%%%%%%%%%%%%%%
% (c)
matrix = [8,2,1; 3,7,1; 2,3,9;]
LUNaive(matrix)
```

```
———————————— Result ————————————

Problem 1
(a)ans =

root(z^3 - (72*z^2)/11 + (84*z)/11 + 48/11, z, 1)
root(z^3 - (72*z^2)/11 + (84*z)/11 + 48/11, z, 2)
root(z^3 - (72*z^2)/11 + (84*z)/11 + 48/11, z, 3)

(b)이분법
iteration        xl               xu              root            error
    1        -0.500000        0.000000        -0.500000        -1.000000
    2        -0.500000        -0.250000        -0.250000        -1.000000
```

3	-0.500000	-0.375000	-0.375000	0.333333
4	-0.437500	-0.375000	-0.437500	0.142857
5	-0.437500	-0.406250	-0.406250	-0.076923
6	-0.421875	-0.406250	-0.421875	0.037037
7	-0.421875	-0.414062	-0.414062	-0.018868
8	-0.417969	-0.414062	-0.417969	0.009346

(c)가위치법

iteration	xl	xu	root	error



1	-1.000000	-0.287425	-0.287425	-2.479167
2	-1.000000	-0.379449	-0.379449	0.242520
3	-1.000000	-0.405232	-0.405232	0.063626
4	-1.000000	-0.412173	-0.412173	0.016840
5	-1.000000	-0.414022	-0.414022	0.004464

Problem 2

(a)ans =

root(z^3 - 6*z^2 + 11*z - 61/10, z, 1)

root(z^3 - 6*z^2 + 11*z - 61/10, z, 2)

root(z^3 - 6*z^2 + 11*z - 61/10, z, 3)

(b)Newton-Raphson법

iteration	root	error
1	3.191304e+00	-0.096730
2	3.068699e+00	-0.039954
3	3.047317e+00	-0.007017

(c)할선법

iteration	root	error
1	2.527536	0.010894
2	2.555404	0.010906
3	2.583467	0.010862

(d)할선법

iteration	root	error
1	3.193707	-0.095905
2	3.123432	-0.022499
3	3.089878	-0.010859

Problem 3

ans =

-24 == -24

ans =

```
34 == 34
```

```
Problem 4
(c)matrix =

     8     2     1
     3     7     1
     2     3     9


ans =

    1.0000         0         0
    0.3750    1.0000         0
    0.2500    0.4000    1.0000
```

이분법

```
function bisection = bisection(func,xl,xu,error,maxit)

i = 1; % iteration, 반복횟수를 초기화한다.
       % i = 0으로 초기화 할 경우 while문에서 i = i + 1의 위치가 if 문 위로 와야 할 것이다.
xrold = 0; % 오차 계산을 위한 xrold를 정의하고, 초기값은 0으로 준다.
xr = xl;   % 해인 xr은 일단 xl과 같다고 설정해둔다.

fprintf('이분법\n')
fprintf('iteration        xl                xu              root
error\n')

while(1)                        % 주어진 조건을 만족할 때 까지 무한반복시킨다.
```

```
    test = func(xl)*func(xu); % if문에서 사용될 조건을 미리 계산해둔다.
    if test > 0                % 구간의 양 끝값의 곱이 0보다 크다면 해가 없거나 2개 이상
        fprintf('Error : choose different range of function'); % 이분법에서 근을 추정
하기 어렵다.
    else
        xrold = xr;            % xrold의 값을 현재까지 저장된 xr의 값으로 지정한다.
        xr = (xl+xu)/2;        % 이분법에서 해는 구간 양 끝의 중점이다.
        er = (xr-xrold)/xr;    % 근사오차

        test_2 = func(xr) * func(xl);  % 두 개로 나뉜 구간 중 어느 구간을 택할지 정한다.
        if test_2 < 0
            xu = xr;
        else
            xl = xr;
        end
    end
    fprintf('    %d        %f        %f        %f        %f\n',i,xl,xu,xr,er)
    i = i + 1;                         % 반복횟수를 1 증가시킨다.

if i == maxit || abs(er) <= error, break, end % 루프 탈출 조건을 지정한다.
                                              % 허용 오차보다 오차가 적거나
                                              % 최대 반복횟수만큼 반복한 경우
end
```

가위치법

```
function false_loc = false_loc(func,xl,xu,error,maxit)

i = 1; % iteration, 반복횟수를 초기화한다.
       % i = 0으로 초기화 할 경우 while문에서 i = i + 1의 위치가 if 문 위로 와야 할 것이다.
xrold = 0; % 오차 계산을 위한 xrold를 정의하고, 초기값은 0으로 준다.
xr = xl;   % 해인 xr은 일단 xl과 같다고 설정해둔다.
```

```
fprintf('가위치법\n')
fprintf('iteration        xl                xu                root
error\n')


while(1)                          % 주어진 조건을 만족할 때 까지 무한반복시킨다.
    test = func(xl)*func(xu); % if문에서 사용될 조건을 미리 계산해둔다.
    if test > 0                   % 구간의 양 끝값의 곱이 0보다 크다면 해가 없거나 2개 이상
        fprintf('Error : choose different range of function'); % 가위치법에서 근을 추
정하기 어렵다.
    else
        xrold = xr;              % xrold의 값을 현재까지 저장된 xr의 값으로 지정한다.
        a = (func(xu)-func(xl))/(xu-xl) ;         % 구간의 양 끝을 잇는 직선의 기울기를 구
한다.
        xr = -func(xl)/a + xl ;
        er = (xr-xrold)/xr;     % 근사오차

        test_2 = func(xr) * func(xl);  % 두 개로 나뉜 구간 중 어느 구간을 택할지 정한다.
        if test_2 < 0
            xu = xr;
        else
            xl = xr;
        end
    end
    fprintf('    %d         %f         %f         %f         %f\n',i,xl,xu,xr,er)
    i = i + 1;                        % 반복횟수를 1 증가시킨다.

if i == maxit || abs(er) <= error, break, end % 루프 탈출 조건을 지정한다.
                                              % 허용 오차보다 오차가 적거나
                                              % 최대 반복횟수만큼 반복한 경우
end
```

Newton-Raphson법

```
function NR = NR(func,diff,xi,error,maxit)

i = 1; % iteration, 반복횟수를 초기화한다.
       % i = 0으로 초기화 할 경우 while문에서 i = i + 1의 위치가 if 문 위로 와야 할 것이다.
xrold = 0; % 오차 계산을 위한 xrold를 정의하고, 초기값은 0으로 준다.
xr = xi;   % 해인 xr은 일단 xi과 같다고 설정해둔다.

fprintf('Newton-Raphson법\n')
fprintf('iteration              root             error\n')

while(1)                        % 주어진 조건을 만족할 때 까지 무한반복시킨다.

    xrold = xr;                 % xrold는 이전에 계산해둔 xr과 같은 값이다.
    xr = xr - subs(func,xr)/subs(diff,xr);    % NR법의 기본 식을 넣는다.
    er = (xr-xrold)/xr;                       % 상대오차를 계산한다.

    fprintf('    %d          %e         %f\n',i,double(xr),double(er))
    i = i + 1;                          % 반복횟수를 1 증가시킨다.

if i == maxit || abs(er) <= error, break, end % 루프 탈출 조건을 지정한다.
                                              % 허용 오차보다 오차가 적거나
                                              % 최대 반복횟수만큼 반복한 경우
end
```

할선법

```
function SC = SC(func,xi,delta,error,maxit)

i = 1; % iteration, 반복횟수를 초기화한다.
       % i = 0으로 초기화 할 경우 while문에서 i = i + 1의 위치가 if 문 위로 와야 할 것이다.
xrold = 0; % 오차 계산을 위한 xrold를 정의하고, 초기값은 0으로 준다.
xr = xi;   % 해인 xr은 일단 xi과 같다고 설정해둔다.
```

```
fprintf('할선법\n')
fprintf('iteration        root            error\n')

while(1)                        % 주어진 조건을 만족할 때 까지 무한반복시킨다.

    xrold = xr;                 % xrold는 이전에 계산해둔 xr과 같은 값이다.
    xr = xr - func(xr)/((func(xi+delta) - func(xi))/delta);     % 할선법의 기본 식을
넣는다.
    er = (xr-xrold)/xr;                                 % 상대오차를 계산한다.

    fprintf('    %d          %f         %f\n',i,double(xr),double(er))
    i = i + 1;                          % 반복횟수를 1 증가시킨다.

if i == maxit || abs(er) <= error, break, end % 루프 탈출 조건을 지정한다.
                                              % 허용 오차보다 오차가 적거나
                                              % 최대 반복횟수만큼 반복한 경우
end
```

LU분해

```
function [L, U] = LUNaive(A)

[m,n] = size(A);
if m~=n, error('Matrix A must be square'); end
L = eye(n);
U = A;

for k = 1:n-1
     for i = k+1:n
        L(i,k) = U(i,k)/U(k,k);
        U(i,k) = 0;
        U(i,k+1:n) = U(i,k+1:n)-L(i,k)*U(k,k+1:n);
     end
end
```