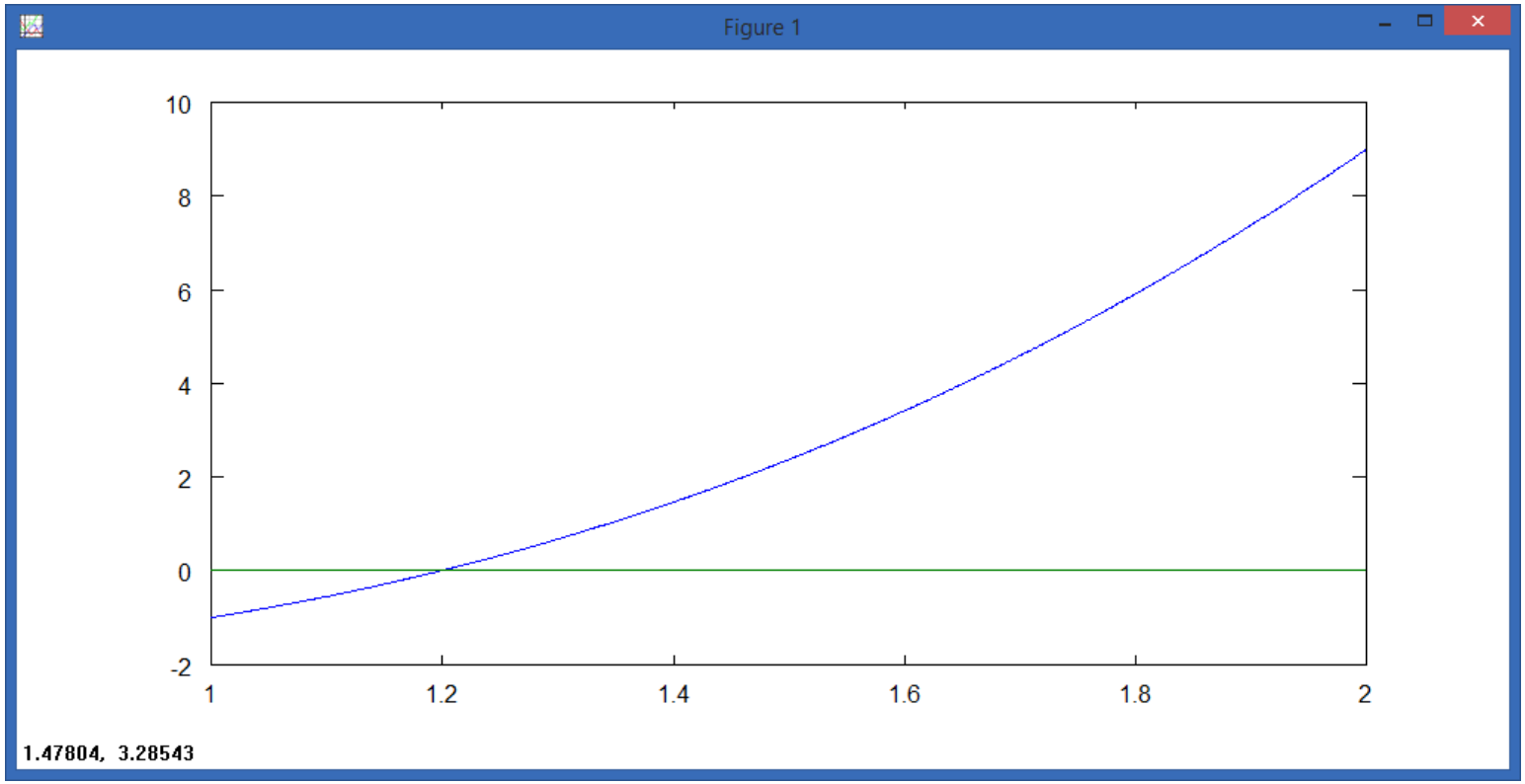
ΕΠΙΛΥΣΗ ΕΞΙΣΩΣΕΩΝ

Σχεδιασμός γραφικής παράστασης

```
>> x=1:0.001:2;
>> y=x.^3+2.*(x.^2)-3.*x-1;
>> t=0*x;
>> plot(x,y,x,t)
```



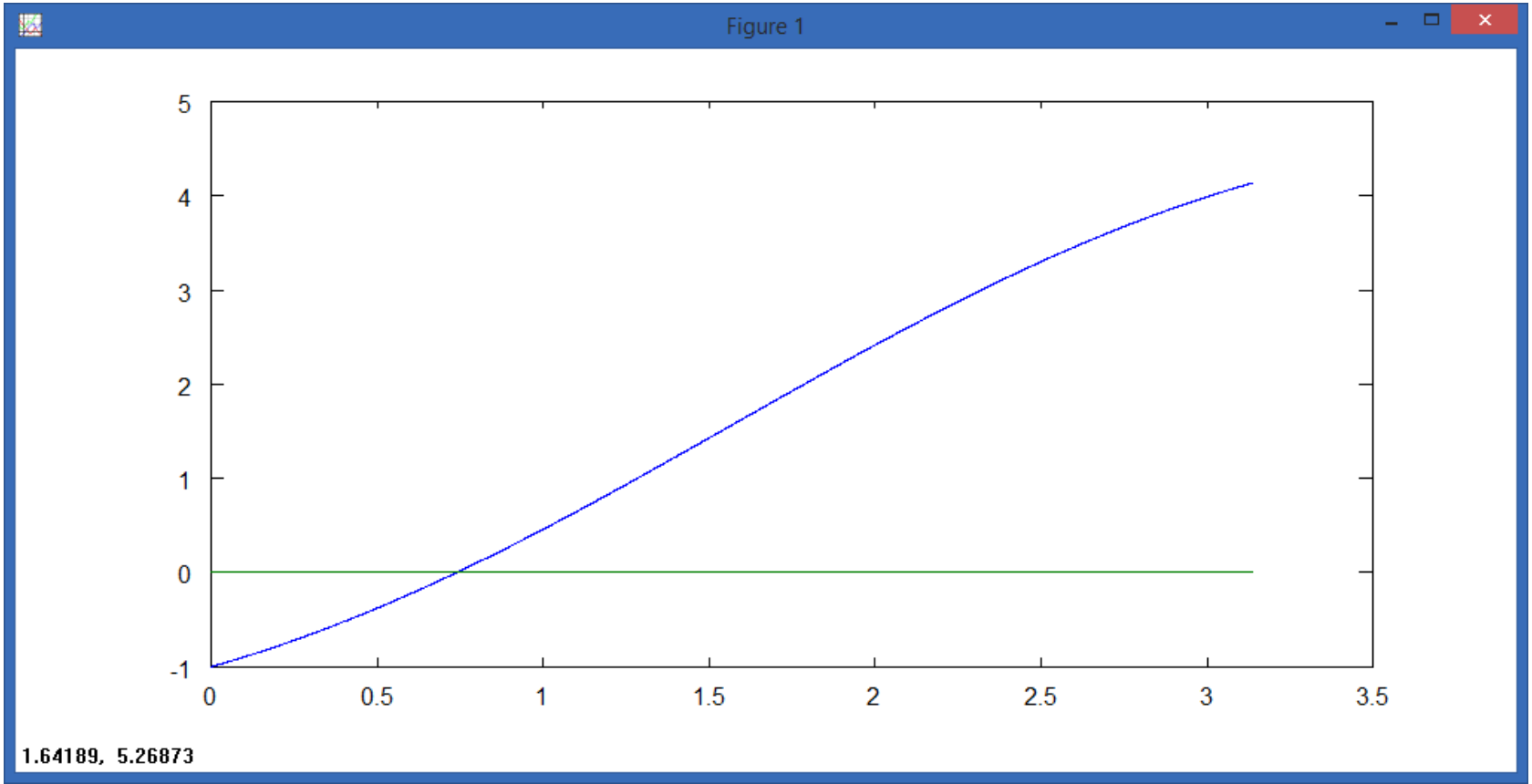
Σχεδιασμός γραφικής παράστασης

>> x=0:0.001:3.14;

>> y=x-cos(x)

>> t=0*x;

>> plot(x,y,x,t)



Μέθοδος διχοτόμησης

```
function [root,fx,ea,iter]=bisect(func,xl,xu,es,maxit)
% input:
% func = name of function
% xl, xu = lower and upper guesses
% es = desired relative error (default = 0.0001%)
% maxit = maximum allowable iterations (default = 50)
% output:
% root = real root
% fx = function value at root
% ea = approximate relative error (%)
% iter = number of iterations
if nargin<3,error('at least 3 input arguments required'),end
test = func(xl)*func(xu);
if test>0,error('no sign change'),end
if nargin<4|isempty(es), es=0.0001;end
if nargin<5|isempty(maxit), maxit=50;end
```

Μέθοδος διχοτόμησης (2)

```
iter = 0; xr = xl; ea = 100;
while (1)
xrold = xr;
xr = (xl + xu)/2;
iter = iter + 1;
if xr ~= 0,ea = abs((xr - xrold)/xr) * 100;end
test = func(xl)*func(xr);
if test < 0
xu = xr;
elseif test > 0
xl = xr;
else
ea = 0;
end
if ea <= es | iter >= maxit,break,end
end
root = xr; fx = func(xr);
```

Μέθοδος διχοτόμησης (χρήση)

```
>> f = @(x) x-cos(x);

>> [root,fx,ea,iter]=bisect(f,0,1,0.00001,40)
root =  0.73909
fx = 7.7470e-009
ea = 8.0647e-006
iter =  24
```

Μέθοδος εσφαλμένης θέσης

```
function [c,yc,err,P] = regula(f,a,b,delta,epsilon,max1)
% Inputs
%   f         name of the function
%   a         left endpoint of the initial interval
%   b         right endpoint of the initial interval
%   delta     convergence tolerance for c
%   epsilon   convergence tolerance for yc
%   max1      maximum number of iterations
% Return
%   c         solution: the root
%   yc        solution: the function value
%   err       error estimate in the solution c
%   P         History vector of the iterations
P = [a b];
ya = feval(f,a);
yb = feval(f,b);
```

```
if ya*yb > 0, break, end
for k=1:max1,
  dx = yb*(b - a)/(yb - ya);
  c  = b - dx;
  ac = c - a;
  yc = feval(f,c);
  if  yc == 0,
    break;
  elseif  yb*yc > 0,
    b = c;
    yb = yc;
  else
    a = c;
    ya = yc;
  end
  P = [P;a b];
  dx = min(abs(dx),ac);
  if abs(dx) < delta,   break, end
  if abs(yc) < epsilon, break, end
end
err = abs(dx);
```

Μέθοδος εσφαλμένης θέσης (χρήση)

```
>> f = @(x) x-cos(x);
>> [c,yc,err,P] = regula(f,0,1,0.0001,0.0001,40)
c =  0.73908
yc = -1.1719e-005
err = 1.3277e-004
P =
   0.00000   1.00000
   0.68507   1.00000
   0.73630   1.00000
   0.73895   1.00000
   0.73908   1.00000
```

Μέθοδος σταθερού σημείου

```
function y = fixed_point ( g, x0, TOL, Nmax )
%           g       iteration function
%           x0      initial approximation to location of fixed point
%           TOL     absolute error convergence tolerance
%           Nmax    maximum number of iterations to be performed
%           y       approximate value of fixed point
old = x0
for i = 1 : Nmax
   new = feval(f,old);
          if ( nargout == 0 )
             disp ( sprintf ( '\t\t %3d \t %.10f \n', i, new ) )
          end
```

Μέθοδος σταθερού σημείου (2)

```
        if ( abs(new-old) < TOL )
           if ( nargout == 1 )
              y = new;
           end
           return
        else
           old = new;
        end

end

disp('Maximum number of iterations exceeded')
if ( nargout == 1 ) y = new; end
```

Μέθοδος σταθερού σημείου (χρήση)

```
>> g = @(x) cos(x);
>> fixed_point ( g, 0, 0.00001, 50 )
old = 0
```

1	1.0000000000
2	0.5403023059
3	0.8575532158
4	0.6542897905
5	0.7934803587
6	0.7013687736
7	0.7639596829
8	0.7221024250
…	
29	0.7390893414
30	0.7390822985

Μέθοδος Newton Raphson (1)

```
function [root,ea,iter]=newtraph(func,dfunc,xr,es,maxit)
% func = name of function
% dfunc = name of derivative of function
% xr = initial guess
% es = desired relative error (default = 0.0001%)
% maxit = maximum allowable iterations (default = 50)
% output:
% root = real root
% ea = approximate relative error (%)
% iter = number of iterations
if nargin<3,error('at least 3 input arguments required'),end
if nargin<4|isempty(es),es=0.0001;end
if nargin<5|isempty(maxit),maxit=50;end
```

Μέθοδος Newton Raphson (2)

```
iter = 0;
while (1)
xrold = xr;
xr = xr - func(xr)/dfunc(xr);
iter = iter + 1;
if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
if ea <= es | iter >= maxit, break, end
end
root = xr;
```

Μέθοδος Newton Raphson (χρήση)

>> f = @(x) x-cos(x);

>> df = @(x) 1+sin(x);

>> [root,ea,iter]=newtraph(f,df,0,0.0001,20)

root = 0.73909

ea = 2.3018e-008

iter = 5

Μέθοδος χορδής (1)

```
function [p1,y1,err,P] = secant(f,p0,p1,delta,epsilon,max1)
%   f         name of the function
%   p0        starting value
%   p1        starting value
%   delta     convergence tolerance for p1
%   epsilon   convergence tolerance y1
%   max1      maximum number of iterations
%   p1        solution: the root
%   y1        solution: the function value
%   err       error estimate in the solution p1
%   P         History vector of the iterations
P(1) = p0;
P(2) = p1;
y0 = feval(f,p0);
y1 = feval(f,p1);
```

Μέθοδος χορδής (2)

```
for k=1:max1,
  df = (y1-y0)/(p1-p0);
  if df == 0,
    dp = 0;
  else
    dp = y1/df;
  end
  p2 = p1 - dp;
  y2 = feval(f,p2);
  err = abs(dp);
  relerr = err/(abs(p2)+eps);
  p0 = p1;
  y0 = y1;
  p1 = p2;
  y1 = y2;
  P = [P,p2];
  if (err<delta)|(relerr<delta)|(abs(y2)<epsilon), break, end
end
```

Μέθοδος χορδής (χρήση)

>> >> f = @(x) x-cos(x);

>> [p1,y1,err,P] = secant(f,0,1,0.0001,0.0001,40)

p1 = 0.73912

y1 = 5.7286e-005

err = 0.0028204

P =

0.00000 1.00000 0.68507 0.73630 0.73912

Ενσωματωμένες συναρτήσεις στο matlab

```
>> f = @(x) x-cos(x);

>> fzero(f,0)
ans =  0.73909
```

Ενσωματωμένες συναρτήσεις στο matlab

$p(x)=3x^3+2x^2+1$

```
>> p = [3 2 0 1];
>> roots(p)
ans =
  -1.00000 + 0.00000i
   0.16667 + 0.55277i
   0.16667 - 0.55277i
```