NONLINEAR EQUATIONS IN MATLAB

Sanda Micula[1]

ABSTRACT

In this paper, we explore numerical methods for solving nonlinear equations using MATLAB. We present the most widely used iterative methods for nonlinear equations and MATLAB features for finding numerical solutions. The paper concludes with numerical examples.

AMS Subject Classification: 12E12, 39B12, 65D17, 65H04, 65K05, 65H04.
Keywords: Numerical Analysis, nonlinear equations, Newton's method, numerical approximations, polynomial equations, MATLAB.

1. ITERATIVE NUMERICAL METHODS FOR NONLINEAR EQUATIONS

1.1 Preliminaries

We want to analyze a nonlinear equation of the form

$$f(x) = 0, \tag{1.1}$$

where $f: \mathbb{R}^m \to \mathbb{R}^n$ is a nonlinear function, which will be assumed to have a certain degree of smoothness. If $n \neq 1$, then (1.1) represents a system of n equations (at least one nonlinear) with m unknowns. In this paper, we will restrict our discussion to the case $m = n = 1$, although many of the procedures we describe can very easily be generalized to the multidimensional case.

Definition 1.1. *A number $\alpha \in \mathbb{C}$ satisfying equation* (1.1) *is called a **zero** or a **root** of f.*

In most cases, an exact solution of equation (1.1) cannot be found explicitly, so numerical methods must be employed instead. An iterative method will produce a sequence $\{x_n\}_{n\in\mathbb{N}}$ of approximations of α, starting with one or more initial values $(x_0, x_1, \dots)$, such that $\lim_{n\to\infty} x_n = \alpha$. These initial values will be determined, in general, from the context of the problem or from the graph of the function. What makes an iterative method better than another is how fast it converges to the desired solution. Regarding the speed of convergence, we define the following:

Definition 1.2. *We say that a sequence of iterates $\{x_n\}_{n\in\mathbb{N}}$ converges to α with **order of convergence** $p \geq 1$, if*

[1] Department of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, smicula@math.ubbcluj.ro

$$|x_{n+1} - \alpha| \leq c\,|x_n - \alpha|^p, \text{ for all } n \in \mathbb{N},$$

(1.2)

where $c > 0$ is a constant independent of n. For $p = 1$, we require also that $c < 1$ and in that case, c is called the ***rate of linear convergence*** *of x_n to α.*

In the next sections we present the most popular iterative methods for nonlinear equations.

1.2 Bisection method

The simplest of iterative methods, the *bisection method* is derived from the Intermediate Value Theorem, which states that if a continuous function *f*, with an interval $[a, b]$ as its domain, takes values $f(a)$ and $f(b)$ at each end of the interval, then it also takes any value between $f(a)$ and $f(b)$,at some point within the interval. So, assume $f: [a, b] \to \mathrm{R}$ is a continuous function and that $f(a)f(b) < 0$. Also, assume that the interval $[a_n, b_n]$ contains only one root α of *f*. Then the idea is that at each step, we bisect the interval, so that *f* has opposite signs at the endpoints of the interval. This will produce a sequence of embedded intervals $[a_n, b_n]$, such that for every $n \in \mathrm{N}$, $\alpha \in [a_n, b_n]$ and $f(a_n)f(b_n) < 0$. The procedure is presented in Algorithm 1.1 and illustrated in Figure 1.

Algorithm 1.1.

$a_0 = a$, $b_0 = b$, f;
for $n = 1,2, \ldots$
$x_n = \dfrac{a_n + b_n}{2}$
if $b_n - a_n <$ err **then**
sol $= x_n$;
return

elseif $f(a_n)f(x_n) < 0$ **then**
$a_{n+1} = a_n, b_{n+1} = x_n;$

else
$a_{n+1} = x_n, b_{n+1} = b_n;$
end if
end if
end for

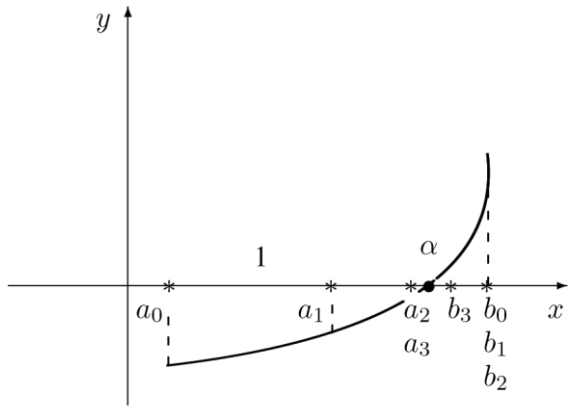


Figure 1. Bisection Method

The sequence $\{a_n\}_{n\in\mathbb{N}}$ is monotonely increasing, sequence $\{b_n\}_{n\in\mathbb{N}}$ is monotonely decreasing and $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n = \lim_{n\to\infty} x_n = \alpha$. Also, we have

$$\begin{aligned} |x_n - \alpha| &\leq b_n - a_n = \left(\frac{1}{2}\right)^n (b-a) < err, \\ |x_{n+1} - \alpha| &\leq \frac{1}{2}|x_n - \alpha|, \end{aligned} \tag{1.3}$$

which shows that the bisection method converges linearly (order of convergence $p = 1$) with a rate of convergence $c = 1/2$.

Remark 1.1. The advantages of the bisection method are: it always converges (no initial guess of the solution is necessary) and the error bound (1.3) can be used as a stopping criterion. As disadvantages, we mention the following: it converges slowly (only linear) and it only approximates real roots.

1.3 Secant method

We consider two initial values x_0, x_1 and we approximate the graph of $y = f(x)$ with the secant line determined by the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$. The root α of f is then approximated by x_2, the point of intersection of the secant line with the x-axis. Recursively, we obtain a sequence of iterates given by

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \quad n = 1, 2 \ldots \tag{1.4}$$

For the convergence of the method, we have the following result:

Theorem 1.1. *Assume f, f' and f'' are continuous on an interval $I_\varepsilon = (\alpha - \varepsilon, \alpha + \varepsilon)$ containing the simple root α ($f'(\alpha) \neq 0$). Then, for starting values x_0 and x_1 sufficiently close to α, the iterates in* (1.4) *converge to α, with order of convergence $p = (1 + \sqrt{5})/2 \approx 1.618033$ ... (the "golden ratio").*

Remark 1.2.

1. To understand what "sufficiently close" means in the theorem above, let

$$M_\varepsilon = \frac{\max\limits_{I_\varepsilon} |f''(x)|}{2 \min\limits_{I_\varepsilon} |f'(x)|},\ e_0 = |x_0 - \alpha|,\ \ e_1 = |x_1 - \alpha| \tag{1.5}$$

Then the method converges if $x_0, x_1 \in I_\varepsilon,$ with $\varepsilon > 0$ chosen so that

$\max\{M_\varepsilon e_0,\ M_\varepsilon e_1\} < 1.$

2. It is obvious that the secant method does not always converge, but when it does, it does so faster than the bisection method.
3. Another advantage is that the secant method can be used to approximate complex roots, as well, if the initial values x_0 and x_1 are taken to be complex numbers satisfying the conditions above.
4. A simple computation shows that for sufficiently large *n*, $|x_n - \alpha| \approx |x_{n+1} - x_n|$, which can be used as a stopping criterion.

Below, we present the algorithm for the secant method.

Algorithm 1.2.

$x_0, x_1, f;$

for $n = 0,1, \ldots$

if $|x_{n+1} - x_n| < err$ **then**

sol=x_n;

return

else

$$x_{n+2} = x_{n+1} - f(x_{n+1}) \frac{x_{n+1} - x_n}{f(x_{n+1}) - f(x_n)};$$

end if

end for

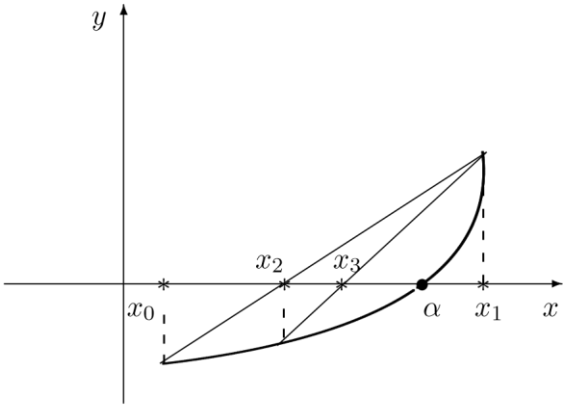


Figure 2. Secant Method

1.4 Newton's method

We start with one initial value x_0 and we approximate the graph of $y = f(x)$ with the line tangent to the graph of f at the point $\big(x_0, f(x_0)\big)$. The root α of f is then approximated by x_1, the point of intersection of the tangent line with the x-axis. Recursively, we obtain a sequence of iterates given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1 \ldots \tag{1.6}$$

The convergence result:

Theorem 1.2. *Assume f, f' and f'' are continuous on an interval $I_\varepsilon = (\alpha - \varepsilon, \alpha + \varepsilon)$ containing the simple root α ($f'(\alpha) \neq 0$). Then, if the initial value x_0 is sufficiently close to α, the iterates in* (1.6) *converge to α and*

$$\lim_{n\to\infty} \frac{x_{n+1} - \alpha}{(x_n - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)}, \tag{1.7}$$

which shows the order of convergence is $p = 2$.

Remark 1.3.

1. Similarly with Remark 1.2, "sufficiently close" means if $x_0 \in I_\varepsilon$, where ε is chosen so that $M_\varepsilon e_0 < 1$, with M_ε and e_0 defined in (1.5).
2. Again, as before, Newton's method does not always converge, but when it does, it does so faster ($p = 2$) than the bisection method ($p = 1$) and the secant method ($p = (1 + \sqrt{5})/2$).
3. Also, Newton's method can be used to approximate complex roots, as well, if the initial value x_0 is a complex number satisfying the conditions above.
4. Again, for sufficiently large n, $|x_n - \alpha| \approx |x_{n+1} - x_n|$, which can be used as a stopping criterion.

The algorithm is described below.

Algorithm 1.3.

x_0, f, f';

$x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$

for $n = 0,1, \ldots$

  if $|x_{n+1} - x_n| <$ err **then**

    sol $= x_n$;

    return

  else

$x_{n+2} = x_{n+1} - \dfrac{f(x_{n+1})}{f'(x_{n+1})}$

  end if

end for

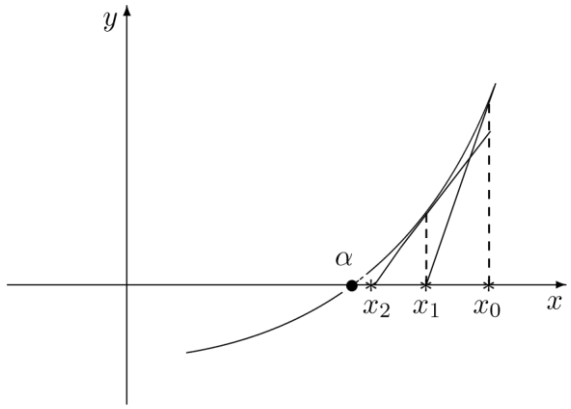


Figure 3. Newton's Method

1.5 Newton-Fourier method

The Newton-Fourier method is an improvement on Newton's method, in the sense that it gives both lower and upper bounds for a simple root. There will be two sequences of iterates, $\{x_n\}_{n\in\mathbb{N}}$ and $\{z_n\}_{n\in\mathbb{N}}$ approaching the root both from above and from below. The sequence $\{x_n\}_{n\in\mathbb{N}}$ is defined as in Newton's method (see (1.6)). The other sequence is defined as follows: Start with one initial value z_0 and construct the line parallel to the line tangent to the graph of f at the point $(x_0, f(x_0))$ and passing through the point $(z_0, f(z_0))$. The next iterate, z_1 will be the point of intersection of that line with the *x*-axis. For simplicity, we present a particular case of this method.

Theorem 1.3. *Assume f, f' and f'' are continuous on an interval $[a, b]$ containing the simple root α. Assume that $f(a) < 0$, $f(b) > 0$ and that $f'(x)$, $f''(x) > 0$, for all $x \in [a, b]$. Define the sequences*

$$z_0 = a,\ z_{n+1} = z_n - \frac{f(z_n)}{f'(x_n)},\ n = 0, 1 \ldots$$

$$x_0 = b,\ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},\ n = 0, 1 \ldots . \tag{1.8}$$

Then the iterates $\{z_n\}_{n\in\mathbb{N}}$ are strictly increasing to α, the iterates $\{x_n\}_{n\in\mathbb{N}}$ are strictly decreasing to α and

$$\lim_{n\to\infty} \frac{x_{n+1} - z_{n+1}}{(x_n - z_n)^2} = \frac{f''(\alpha)}{2f'(\alpha)}, \tag{1.9}$$

which shows the order of convergence is p = 2.

Remark 1.4.

1. The assumptions of Theorem 1.3 can be relaxed to $f \in C^2$ and $f'(\alpha)f''(\alpha) \neq 0$ in a neighborhood of α. The conclusion still holds in that neighborhood (with the possible interchange of the monotonicity of the two sequences, i.e. with $\{x_n\}_{n\in\mathbb{N}}$ being strictly decreasing to α and $\{z_n\}_{n\in\mathbb{N}}$ being strictly increasing to α).
2. From (1.9), for sufficiently large *n*, we get the error bound $|x_n - \alpha| \approx |x_n - z_n|$, which can be used as a stopping criterion.

Below, we give the algorithm and an illustration of the Newton-Fourier method.

Algorithm 1.4.
$z_0 = a, x_0 = b, f, f'$;
for $n = 0,1, \ldots$

if $|x_n - z_n| < err$ **then**

$sol = x_n$;
return

else

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(x_n)}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

end if
end for

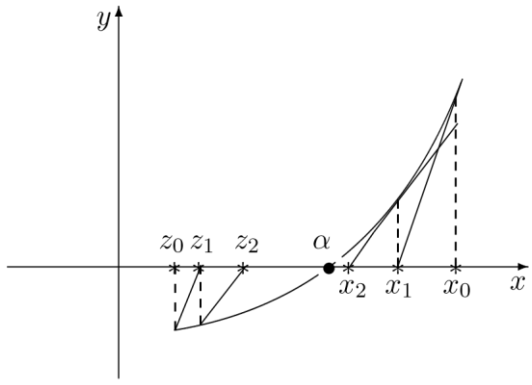


Figure 4. Newton-Fourier Method

2. MATLAB FUNCTIONS AND PROCEDURES FOR SOLVING NONLINEAR EQUATIONS

MATLAB[2] (**MAT**rix **LAB**oratory) is a multi-paradigm, high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB, one can analyze data, develop algorithms and create models and applications in signal processing and communications, image and video processing, control systems, test and measurement, computational finance, computational biology, etc. Easy matrix manipulations makes it an excellent tool for applied mathematics, especially for programming iterative processes.

[2] MATLAB® is a trademark of MathWorks Inc.

3. MATLAB FUNCTIONS FZERO AND ROOTS

Function **fzero** finds a root of a continuous function of one variable, using a combination of bisection, secant, and inverse quadratic interpolation methods. Its syntax is

$$[x, fval] = \textbf{fzero}(fun, x_0),$$

where *fun* is a function handle and x_0, is a scalar close to a root. The value x returned by **fzero** is near a point where *fun* changes sign, or NaN ("not a number") if the search fails, while *fval* (which is optional) is the value of the objective function *fun* at the solution x. If x_0 is a vector of length two, **fzero** assumes it is an interval on which *fun* changes sign (otherwise, an error occurs) and returns a value near a point where *fun* changes sign. More options (having to do with the display of the solution, the tolerance, the number of iterations needed to get to that tolerance, etc.) may be given, both as input and as output (for more details, use MATLAB Help or see [9]).

For polynomial equations of the form

$$a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0,$$

function **roots** can be used. Its syntax is

$$r = \textbf{roots}(p),$$

where $p = [a_n, a_{n-1}, \ldots, a_1, a_0]$is a vector containing the coefficients of the polynomial, including the null ones and r is a vector containing the approximations of all its zeros. Function **roots** approximates the zeros of p by computing the eigenvalues of the companion matrix

$$A = \begin{bmatrix} -a_{n-1} & -a_{n-2} & \ldots & -a_1 & -a_0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix}$$

Remark 2.1.
MATLAB has relatively few routines for finding roots of a nonlinear algebraic univariate functions and none for finding solutions of systems of nonlinear equations. However, both problems can be approached by using least squares minimization, in the Optimization Toolbox.

4. ITERATIVE METHODS IMPLEMENTATION IN MATLAB

We now give MATLAB implementations of the algorithms described in Section 1.

Bisection Method

```
function [sol, nr_it] = bisection_method(f, a, b, eps)
% f is a continuous function, whose root is being approximated;
% [a,b} is the interval on which a root exists and f(a)f(b)<0;
% eps is the specified error for the approximation;
% if no eps is specified,
% the default will be 10^(-5);
% sol is the approximation of the solution;
% nr_it is the number of iterations needed for tolerance err;
if nargin < 4
    eps=1e-5;
end n=0;
x = (a + b)/2;
while b - a >= eps
      n = n + 1;
      x = (a + b)/2;
      if f(a)*f(x) < 0
          b = x;
      else
          a = x;
      end
end
sol = x;
nr_it = n;
end
```

Since the next three methods do not converge for all initial values, we introduce a maximum number of iterations allowed, in case the initial values are not within the interval of convergence.
In case that maximum is exceeded, an error message will be displayed.

Secant Method

```
function [sol, nr_it] = secant_method(f, x0, x1, eps, nr_max_it)
% f is a continuous function, whose root is being approximated;
% x0, x1 are two initial values;
% eps is the specified error for the approximation;
% if no eps is specified,
% the default will be 10^(-5);
% nr_max_it is the maximum number of iterations allowed;
% if no nr_max_it is specified,
```

```
% the default will be 100;
% sol is the approximation of the solution;
% nr_it is the number of iterations needed for tolerance err;

if nargin < 5
     nr_max_it = 100;
end
if nargin < 4
    eps=1e-5;
end
for n=1:nr_max_it
     if abs(x1 - x0) < eps
          sol = x0;
          nr_it = n;
          return
     else
          x2 = x1 - f(x1) / (f(x1) - f(x0)) * (x1 - x0); x0 = x1; x1 = x2;
     end
end error('too many iterations, try different initial values')
end
```

Newton's Method

```
function [sol, nr_it] = newton_method(f, fprime, x0, eps, nr_max_it)
% f is a continuous function, whose root is being approximated;
% fprime is the derivative of f;
% x0 is the initial value;
% eps is the specified error for the approximation;
% if no eps is specified,
% the default will be 10^(-5);
% nr_max_it is the maximum number of iterations allowed;
% if no nr_max_it is specified, the default will be 100;
% sol is the approximation of the solution;
% nr_it is the number of iterations needed for tolerance err;

if nargin < 5
    nr_max_it = 100;
end
if nargin < 4
    eps=1e-5;
end
for n=1:nr_max_it
     x1 = x0 - f(x0) / fprime(x0);
     if abs(x1 - x0) < eps
        sol = x0;
```

```
        nr_it = n;
        return
    else
        x0 = x1;
    end
end error('too many iterations, try a different initial value')
end
```

Newton-Fourier's Method

```
function [sol, nr_it ] = newton_fourier_method(f, fprime, a, b, eps)
% f is a continuous function, whose root is being approximated;
% fprime is the derivative of f;
% a, b are two initial values so that f(a) <0, f(b) >0,
% f'(x), f''(x) >0, for all x in [a,b];
% eps is the specified error for the approximation;
% if no eps is specified,
% the default will be 10^(-5);
% sol is the approximation of the solution;
% nr_it is the number of iterations needed for tolerance err;

if nargin < 5
   eps=1e-5;
end
z0 = a;
x0 = b;
n = 0;
while abs( x0 - z0) >= eps
     z0 = z0 - f(z0) / fprime(x0);
     x0 = x0 - f(x0) / fprime(x0);
     n = n +1;
end
sol = x0;
nr_it = n;
end
```

2 EXAMPLES

Example 3.1. Consider the nonlinear equation

$$x^3 - x - 1 = 0\,. \tag{3.1}$$

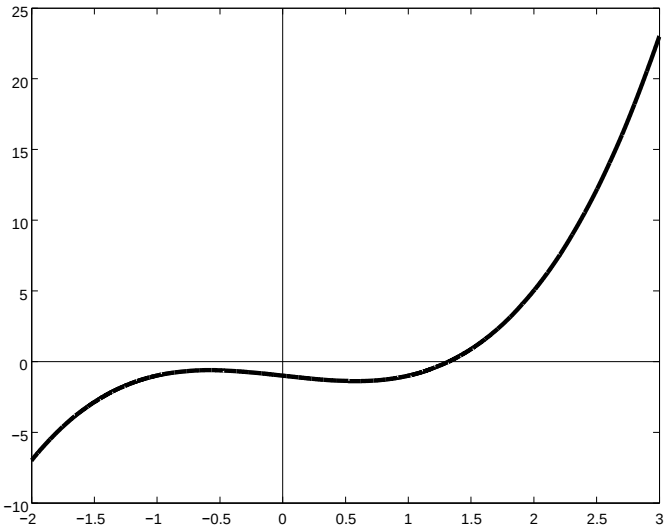
Then

$$\begin{aligned} f(x) &= x^3 - x - 1 \\ f'(x) &= 3x^2 - 1. \end{aligned}$$

The number of sign changes of f at the endpoints and at the roots of the derivative

x	$-\infty$	$-\sqrt{3}/3$	$\sqrt{3}/3$	∞
$f(x)$	$-$	$-$	$-$	$+$

is 1, so there is one real root and two conjugate complex roots. From the graph of f below, we estimate that the real root is in the interval $(1, 2)$.



Function $f(x) = x^3 - x - 1$

Since (3.1) is a polynomial equation, we can find its zeros in MATLAB with

```
>> p=[1 0 -1 -1];
>> alpha=roots(p)
alpha =
    1.3247
−0.6624 + 0.5623i
−0.6624 − 0.5623i
```

We can also find the real root (with initial guess $x_0 = 1$) with

```
>> x = fzero(f,1)
x = 1.3247
```

Now we use the routines described in the previous section. We define the function and its derivative in MATLAB:

```
>> f=@(x) x.^3 - x -1
f =
@(x)x.^3-x-3

>> fprime=@(x) 3*x.^2 - 1
fprime =
@(x)3*x.^2-1
```

In the tables below, we give the approximations of the roots $\alpha_1 = 1.3247, \alpha_{2,3} = -0.6624 \pm 0.5623i$, the starting values and the number of iterations needed for precision $\varepsilon = 10^{-5}$.

α_1	Starting value(s)	Approximate solution	Number of iterations
Bisection	$a = 1, b = 2$	1.3247	17
Secant	$x_0 = 1, x_1 = 2$	1.3247	7
Newton	$x_0 = 1$	1.3247	5
Newton−Fourier	$a = 1, b = 2$	1.3247	5

α_2	Starting value(s)	Approximate solution	Number of iterations
Secant	$x_0 = -1 - 0.2i,$ $x_1 = -1 - i$	$-0.6624 - 0.5623i$	7
Newton	$x_0 = -1 - 0.2i$	$-0.6624 - 0.5623i$	6
Newton−Fourier	$a = -1 - 0.2i,$ $b = -1 - i$	$-0.6624 - 0.5623i$	5

α_3	Starting value(s)	Approximate solution	Number of iterations
Secant	$x_0 = -1 + 0.2i,$ $x_1 = -1 + i$	$-0.6624 + 0.5623i$	7
Newton	$x_0 = -1 + 0.2i$	$-0.6624 + 0.5623i$	5
Newton−Fourier	$a = -1 + 0.2i,$ $b = -1 + i$	$-0.6624 + 0.5623i$	5

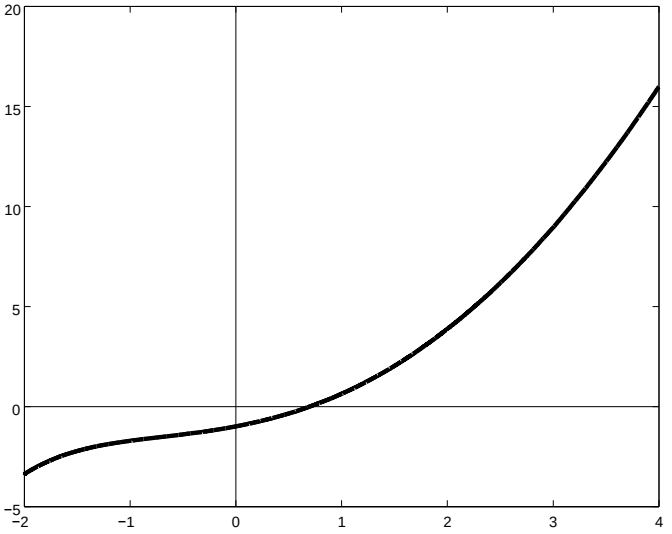
Example 3.2. Let us consider the transcendental equation

$$x^2 - e^{-x} = 0. \qquad (3.2)$$

We have

$$\begin{aligned} f(x) &= x^2 - \mathrm{e}^{-x} \\ f'(x) &= 2x + \mathrm{e}^{-x}. \end{aligned}$$

After some calculations, we find that f has only one sign change between 0 and 1 (so there is only one real root $\alpha \in (0,1)$) and that on that interval, f is increasing ($f'(x) > 0$) and concave up ($f''(x) > 0$). The graph below confirms our assertion:



Function $f(x) = x^2 - e^{-x}$

We define f and f':

```
>> f=@(x) x.^2 - exp(-x)
f =
@(x)x.^2-exp(-x)

>> fprime=@(x) 2*x +exp(-x)
fprime =
@(x)2*x+exp(-x),
```

find the solution α,

```
>> alpha = fzero(f,1)
alpha =
0.7035
```

and then approximate it with the four numerical methods, with precision $\varepsilon = 10^{-4}$. The results are found in the following table.

α	Starting value(s)	Approximate solution	Number of iterations
Bisection	$a = 0, b = 1$	0.7034	14
Secant	$x_0 = 0, x_1 = 1$	0.7035	5
Newton	$x_0 = 1$	0.7035	4
Newton−Fourier	$a = 0.5, b = 1$	0.7035	3

REFERENCES

[1] K. E. Atkinson, An introduction to Numerical Analysis, second edition, John Wiley & Sons, Inc. New York, 1989.

[2] P. Deuflhard, Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms, Springer Series in Comp. Math. 35, 2004.

[3] P. Deuflhard, *A Short History of Newton's Method*, Documenta Mathematica Extra Volume ISMP (2012), 25-30.

[4] C. T. Kelley, Iterative Methods for Linear and Nonlinear Equations, Frontiers in Appl. Math 16, SIAM, Philadelphia, 1995.

[5] S. Micula, R. Sobolu, M. Micula, Analiză numerică cu Maple, Editura Academic Press, Cluj-Napoca, 2008.

[6] S. Micula, R. Sobolu, *Numerical Solutions of Nonlinear Algebraic Equations with Maple*, Didactica Mathematica, Vol. 26 (2009), Nr. 1, 121-128.

[7] M. Shacham, N. Brauner, *Numerical solution of non-linear algebraic equations with discontinuities*, Computers and Chemical Engineering 26 (2002), 1449-1457.

[8] D. D. Stancu, Gh. Coman, P. Blaga: Analiză numerică şi Teoria aproximării, vol. II, Presa Univ. Clujeană, 2002.

[9] R. Trîmbiţaş: Numerical Analysis in MATLAB, Cluj Univ. Press, 2005.