POLSKA AKADEMIA NAUK

INSTYTUT MASZYN PRZEPŁYWOWYCH

TRANSACTIONS OF THE INSTITUTE OF FLUID-FLOW MACHINERY

PRACE INSTYTUTU MASZYN PRZEPŁYWOWYCH

97

WYDAWNICTWO IMP PAN

GDAŃSK 1993

PRACE INSTYTUTU MASZYN PRZEPŁYWOWYCH

poświęcone są publikacjom naukowym z zakresu teorii i badań doświadczalnych w dziedzinie mechaniki i termodynamiki przepływów, ze szczególnym uwzględnieniem problematyki maszyn przepływowych

*

THE TRANSACTIONS OF THE INSTITUTE OF FLUID-FLOW MACHINERY

exist for the publication of theoretical and experimental investigations of all aspects of the mechanics and thermodynamics of fluid-flow with special reference to fluid-flow machines

REDAKCJA – EDITORIAL OFFICE

Wydawnictwo Instytutu Maszyn Przepływowych
Polskiej Akademii Nauk
ul. Gen. Józefa Fiszera 14, 80-952 Gdańsk, skr. poczt. 621,
tel. (0-58) 41-12-71 wew. 141, fax: (0-58) 41-61-44,
e-mail: tjan@imppan.imp.pg.gda.pl

ISBN 83-01-94115-2
ISSN 0079-3205

PRACE INSTYTUTU MASZYN PRZEPŁYWOWYCH
1994 Zeszyt 97

JACEK M. ELSZKOWSKI[1]

On an effective algorithm for solving non-linear equations and their systems

A new algorithm for solving a single non-linear equation is proposed in this paper. The algorithm constitutes a combination of three elementary algorithms based on the *regula falsi* method, the half-interval method, and the method of secants. The combined algorithm is globally convergent provided that bounds of the root are known. Generalisation of the algorithm on a system of non-linear equations is also proposed.

1. Introduction

Solving non-linear equations represents very often an auxiliary "sub problem" appearing in more complicated algorithms. Determination of the mapping function [1] yields a good example of this kind. In particular, verification of the obtained function reduces to solving of a rather nasty non-linear equation several hundred times.

Therefore, there is still a need for general methods of solving non-linear equations ensuring reliability, fast convergence, and low cost.

Consideration of the present paper will be confined to algorithms for solving the non-linear equation:

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

– the algorithms which are globally convergent in an interval:

$$x \in (a, b), \tag{1.2}$$

where:

$$f(a)f(b) \leq 0. \tag{1.3}$$

The symbol f in (1.1) and (1.3) denotes a continuous function of the real variable x.

[1]Samodzielna Pracownia Numerycznej Mechaniki Płynów, Instytut Maszyn Przepływowych PAN, ul. Fiszera 14, 80-952 Gdańsk

From the assumption of continuity and from the nonequality (1.3) there follows that Eq. (1.1) has at least one root x_*, in the interval (1.2), i.e.

$$f(x_*) = 0. \tag{1.4}$$

Let us introduce now two definitions:
First: any interval

$$x \in [x_1, x_2] \tag{1.5}$$

satisfying the condition

$$f(x_1)f(x_2) \leq 0 \tag{1.6}$$

will be referred to as *bounds of the root* of Eq. (1.1).
Second: the number

$$d = | x_1 - x_2 | \tag{1.7}$$

will be called *magnitude of the bounds.*

We are interested only in such algorithms for solving Eq.(1.1), which possess two properties. First of all, they should yield sequences of bounds (1.5) of decreasing magnitude. Moreover, they should be not expensive, i.e. they should involve computation of the function $f(x)$ only, but not of its derivatives.

It can be easily seen that for example the Newton method does not possess any of these two properties.

2. Three elementary methods of solving

Three elementary methods for solving Eq. (1.1) will be recalled here, and their particular properties will be discussed.

2.1. The half interval method

Let us assume that some bounds $[x_1, x_2]$ of the root of the equation (1.1) are known.

The half-interval method consists in computation of a new approximation of the root by halving the known bounds:

$$x_N = \frac{1}{2}(x_1 + x_2), \tag{2.1}$$

and in investigation which one of the intervals:

$$[x_1, x_N], [x_N, x_2] \tag{2.2}$$

represents new bounds of the root.

This method is illustrated in Fig. 1. The half-interval

$$q = \frac{d}{2}$$

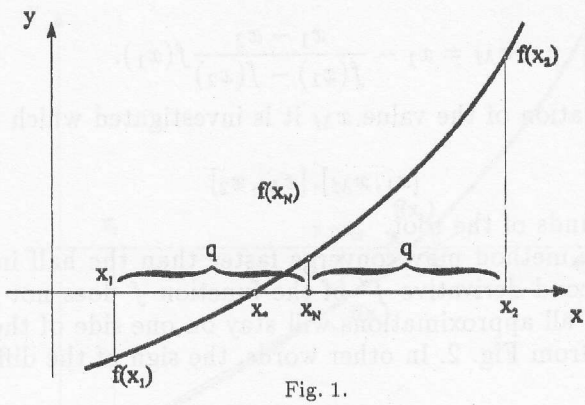

Fig. 1.

is also indicated in the Figure.

The method is linearly convergent, and it involves solely evaluation of the function $f(x)$ during the investigation of the intervals (2.2). Moreover it is possible to calculate the number of iterations necessary for determination of the root with given accuracy represented by the final magnitude of the bounds.

2.2. The *regula falsi* method

For given bounds $[x_1, x_2]$, the *regula falsi* method determines a new approximation of the root x as the ordinate x_M of a point of intersection of the ordinate axis with a straight line connecting the points:

$$(x_1, f(x_1)), (x_2, f(x_2)). \tag{2.3}$$

This operation is illustrated in Fig. 2. The sought-for ordinate can be calculated

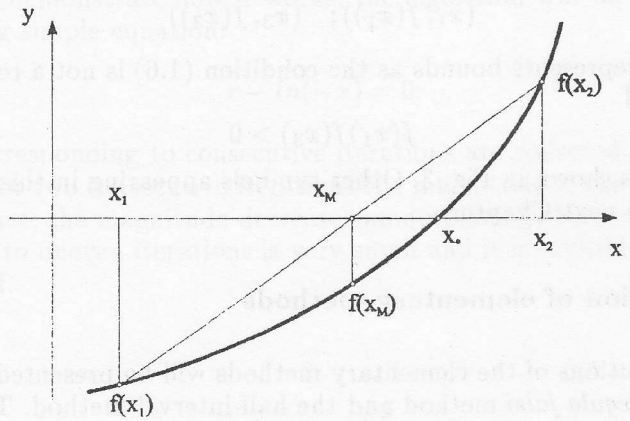

Fig. 2.

from the formula:

$$x_M = x_1 - \frac{x_1 - x_2}{f(x_1) - f(x_2)} f(x_1). \tag{2.4}$$

After the determination of the value x_M it is investigated which one of the intervals

$$[x_1, x_M], [x_M, x_2] \tag{2.5}$$

represents new bounds of the root.

The *regula falsi* method may converge faster than the half interval method. However, if the second derivative f'' of the function f does not change its sign within the bounds, all approximations will stay on one side of the root x_*, what can be easily seen from Fig. 2. In other words, the sign of the difference

$$x_M - x_* \tag{2.6}$$

will not change during the iterative process. Consequently, the smallest distance of the bounds:

$$d_{min} = | x_* - x_2 | \tag{2.7}$$

may be quite large so that it can not be accepted as a measure of accuracy of the final approximation of the root. This is not the case as far as the former method is concerned.

2.3. The method of secants

The method of secants, called also the false position method [2], is closely related to the *regula falsi* method and a new approximation x_P of the root x_*, is computed from the following formula:

$$x_P = x_1 - \frac{x_1 - x_3}{f(x_1) - f(x_3)} f(x_1) \tag{2.8}$$

formally identical with Eq. (2.4). However, the points

$$(x_1, f(x_1)); \quad (x_3, f(x_3)) \tag{2.9}$$

do not have to represents bounds as the condition (1.6) is not a requirement and it can be as well

$$f(x_1) f(x_3) > 0 \tag{2.10}.$$

Such situation is shown in Fig. 3. Other symbols appearing in this Figure will be explained in the next Chapter.

3. Combination of elementary methods

Two combinations of the elementary methods will be presented. The first one consists of the *regula falsi* method and the half-interval method. The second one involves all three elementary methods recalled in the former Chapter.

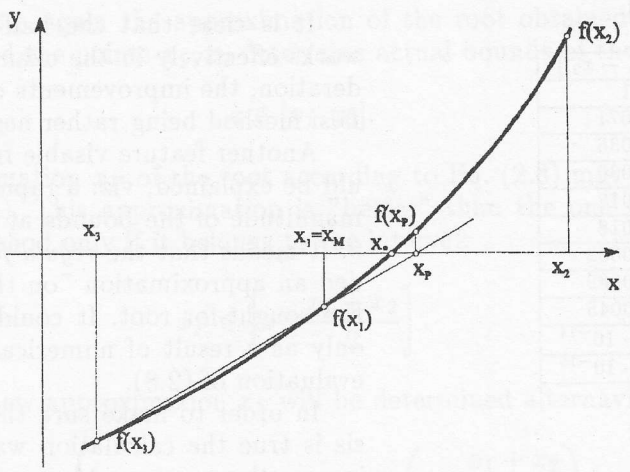

Fig. 3.

3.1. The classical algorithm

Considering the *regula falsi* method and the half interval method one can see that their effectiveness depends on the form of the line, representing the function

$$y = f(x) \tag{3.1}$$

in the vicinity of the root (1.4). Usually this form is not known. Hence, an obvious idea occured to the author of the present paper to apply boths methods alternatively. By virtue of such combination at least one of the methods will work effectively in any extremal case.

An algorithm based on this combination is published in [3]. Because of its numerous and diverse applications it will be referred to as the *classical* one.

In order to demonstrate how it works, the algorithm will be applied to solve of the following simple equation:

$$x - ln(-x) = 0. \tag{3.2}$$

Bounds corresponding to consecutive iterations are collected in Table 1. It seems that for two consecutive iterations the magnitude of the bounds remains the same. In fact, the magnitude decreases monotonously. However, the decrease corresponding to uneven iterations is very small and it is invisible in Table 1 due to rounding-off.

Table 1

Iteration	$\lvert x_1 - x_2 \rvert$
0	0.01
1	0.0071
2	0.0036
3	0.0036
4	0.0018
5	0.0018
6	0.00089
7	0.00089
8	0.00045
9	$1.1 \cdot 10^{-14}$
10	$5.8 \cdot 10^{-15}$

Table 2

Iteration	$\lvert x_1 - x_2 \rvert$
0	0.01
1	0.0071
2	0.0036
3	0.0036
4	0.0018
5	0.0018
6	0.00089
7	0.00089
8	0.00045
9	0.00045
10	0.00022
20	$6.9 \cdot 10^{-6}$
30	$2.1 \cdot 10^{-7}$

It is clear that the half-interval method works effectively in the example under consideration, the improvements due to the *regula falsi* method being rather negligible.

Another feature visable from Table 1 should be explained, viz. a rapid decrease in the magnitude of the bounds at the iteration No 9. It means that the *regula falsi* method yielded an approximation "on the other side" of the sought-for root. It could have happened only as a result of numerical inaccuracies in evaluation of (2.8).

In order to make sure that this hypothesis is true the calculation was repeated with increased accuracy. More exactly, the type *extended* was applied instead of the type *double* in the respective procedure written in Turbo Pascal. The results collected in Table 2 are identical to those in Table 1 as far as the iterations numbered 0 to 8 are concerned. However no "jump" in the magnitude of the bounds appears in Table 2, and the regular pattern is sustained up to the iteration No 30 when the iterative process was terminated. Comparing the bounds corresponding to the final iterations in both Tables one can say that paradoxically, in the case under consideration, larger round-off errors resulted in fewer iterations necessary for determination of the root with the accuracy of 10^{-14}.

3.2. The improved algorithm

Considering results contained in Tables 1, and 2 one must arrive at the conclusion that the classical algorithm is, in a sense, wasteful as far as its application to solve Eq. (3.2) is concerned. Especially, incorporation of the *regula falsi* method into the algorithm seems to be unnecessary. These conclusions stem from just one example, nevertheless, some improvements in the classical algorithm seem to be advisable.

The method of secants will be applied for this purpose, hence, we return to Fig. 3. Some additional assumptions will be introduced:

$$(x_3 - x_*)(x_M - x_*) > 0; \tag{3.3}$$

$$(x_1 - x_*)(x_3 - x_*) > 0, \tag{3.4}$$

where x_M denotes again the approximation of the root obtained by the *regula falsi* method, and the points x_1, x_2 determine actual bounds of the root:

$$x \in [x_1, x_2]. \tag{3.5}$$

The approximation x_P of the root according to Eq. (2.8) may fall beyond to the bounds (3.5). This approximation is "better" than the one yielded by the half-interval method only if it belongs to the interval:

$$x_P \in \left[x_1, \frac{x_1 + x_2}{2}\right]. \tag{3.6}$$

Therefore, the new approximation x_S will be determined alternavitely:

$$x_S = \begin{cases} x_P, & \text{when } x_P \in \left(x_1, \dfrac{x_1 + x_2}{2}\right); \\ \dfrac{x_1 + x_2}{2}, & \text{when } x_P \notin \left(x_1, \dfrac{x_1 + x_2}{2}\right). \end{cases} \tag{3.7}$$

Acceleration of the iterative process by means of Eq. (3.7) has two reasons. First: the half-interval method is replaced by the faster one. Second: if f'' does not change the sign within the bounds (3.5) the root x_* is always contained between x_S and x_M.

The flow chart of the improved algorithm is presented in Fig. 4.

A main idea consists in alternative application of the *reguła falsi* method and the method of secants.

Boxes 1, 6, 7 serve the purpose of determination of the new bounds of the root (3.5). Moreover, the three points:

$$x_1, x_2, x_3 \tag{3.8}$$

are determined within these Boxes, basing on the three points:

$$x_1, x_2, x_{NEW}, \tag{3.9}$$

where

$$[x_1, x_2]$$

denotes the preliminary bounds of the root, and x_{NEW} – the new approximation. Generally speaking, one of the three points in (3.9), which does not appear in the new bounds is accepted as the point x_3. For example, in Fig. 2 the points x_M and x_2 determine the new bounds. Therefore, the point x_1 would be renamed as x_3. The point x_M denoting now the lower bound would be renamed as x_1. The point x_2 would again denote the upper bound, hence, no renaming would be necessary.

In Boxes 2 and 8 the criterion of convergence is checked: according to this criterion the magnitude of the bounds (1.7) has to be smaller than a given number

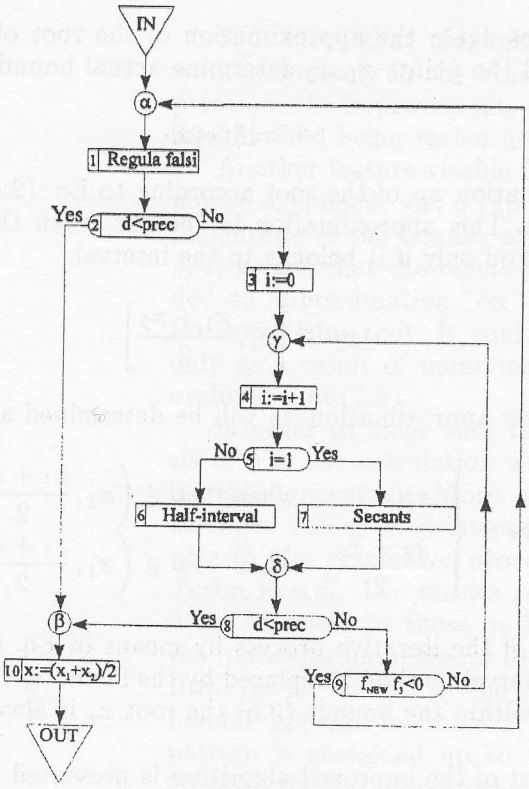

Fig. 4.

prec.

In Box 9 the condition

$$f_{NEW} f_3 < 0 \tag{3.10}$$

is investigated where

$$f_{NEW} = f(x_N),$$

if calculations go through Box 6 and

$$f_{NEW} = f(x_S),$$

if calculations go through Box 7.

If the condition (3.10) is satisfied, calculations return to Box 1, and the new approximation is computed by means of the *regula falsi* method.

If the condition (3.10) is not satisfied, calculations go to Box 6, and the new approximation is computed by means of the half interval method.

The final approximation of the root is computed in Box 10 from Eq. (2.1).

Procedure **Root 2** listed in the Annex represents a computer realisation of

Table 3

Iteration	$\lvert x_1 - x_2 \rvert$
0	0.01
1	0.0071
2	$1.1 \cdot 10^{-5}$
3	$1.8 \cdot 10^{-8}$
4	$1.1 \cdot 10^{-13}$

Table 4

Iteration	$\lvert x_1 - x_1 \rvert$
0	1
1	0.73
2	0.36
3	0. 16
4	0.04
5	0.0088
6	0.0028
7	$9.1 \cdot 10^{-6}$
8	$2.5 \cdot 10^{-9}$
9	$7.1 \cdot 10^{-13}$

the algorithm just described.

One of the two tasks of the program **Equations**, also listed in the Annex, is solving Eq. (3.2) by means of the procedure **Root 2**.

Results corresponding to successive iterations are collected in Table 3. Just four iterations suffice for determination of the root with the accuracy of 10^{-13}. The comparison of Table 3 with Table 1 and 2 does not require any comment.

As a further example of application of the improved algorithm the following equation

$$x\, exp(-x) = 0 \tag{3.11}$$

was solved. Results collected in Table 4 indicate that 9 iterations are only needed to reduce the magnitude of the bounds from 10^0 to 10^{-13} in this case.

4. Application of the above to systems of non-linear equations

Let us consider a system of non-linear equations

$$\begin{cases} f_1(x_1, x_2, \ldots, x_N) = 0; \\ f_2(x_1, x_2, \ldots, x_N) = 0; \\ \quad\vdots \\ f_N(x_1, x_2, \ldots, x_N) = 0, \end{cases} \tag{4.1}$$

where $f_1, f_2, \ldots f_N$ denote continuous functions of their arguments.

In the present Chapter the ideas presented in Section 3.2 will be extended on *systems* of non-linear equations. In other words, an algorithm based on these ideas will be developed.

It should be emphasized that it is not an easy task. Nevertheless, such an algorithm, globally convergent if some conditions are satisfied, will be presented here. However, the computation cost of this algorithm will be high (comp. [3]).

General considerations concerning the sought-for algorithm start with the assumption that an effective, globally convergent algorithm for solving of just one equation is known. By means of this known algorithm an equation

$$f_N(x_1, x_2, \ldots, x_N) = 0 \tag{4.2}$$

can be solved with respect to one unknown, e.g. x_N, provided that the remaining ones, i.e.

$$x_1, x_2, \ldots, x_{N-1}$$

are fixed. Because of the global convergence of the known algorithm the solution of (4.2) always exists; it will be denoted as $\tilde{x}_N$. Therefore, it can be said that a function

$$\tilde{x}_N = g_N(x_1, x_2, ..., x_{N-1}) \tag{4.3}$$

is determined. This function can be applied to eliminate of x_N from the initial $N-1$ equations of the system (4.1):

$$\begin{cases} f_1(x_1, x_2, ..., x_{N-1}, \tilde{x}_N) = 0; \\ f_2(x_1, x_2, ..., x_{N-1}, \tilde{x}_N) = 0; \\ \vdots \\ f_{N-1}(x_1, x_2, ..., x_{N-1}, \tilde{x}_N) = 0. \end{cases} \tag{4.4}$$

The process of elimination can be applied now to the system (4.4). For example, the unknown x_{N-1} can be eliminated by means of the equation:

$$f_{N-1}(x_1, x_2, ..., x_{N-1}, \tilde{x}_N) = 0, \tag{4.5}$$

and, as a result of this operation, the following system can be obtained:

$$\begin{cases} f_1(x_1, x_2, ..., x_{N-2}, \tilde{x}_{N-1}, \tilde{x}_N) = 0; \\ f_2(x_1, x_2, ..., x_{N-2}, \tilde{x}_{N-1}, \tilde{x}_N) = 0; \\ \vdots \\ f_{N-2}(x_1, x_2, ..., x_{N-2}, \tilde{x}_{N-1}, \tilde{x}_N) = 0. \end{cases} \tag{4.6}$$

By continuation of the this process one arrives finally at the following single equation:

$$f_1(x_1, \tilde{x}_2, ..., \tilde{x}_N) = 0, \tag{4.7}$$

which can be solved with respect to x_1 by means of the already mentioned, known algorithm.

In order to express the described sequence of consecutive eliminations in a computer language, one can apply recurrence procedures. Let us consider, for example, a hypothetical procedure **RootN(j,s)** with the following parameters:
j – index;
X – vector of unknowns $x_1, x_2, ..., x_N$.
The task of this procedure consists in solving of the equation:

$$f_j(x_1, x_2, ..., x_j, \tilde{x}_{j+1}, ..., \tilde{x}_N) = 0 \tag{4.8}$$

with respect to x_j.

If

$$j = N,$$

then the procedure **RootN** works identically as the procedure **Root2**. However if,

$$j < N$$

then the procedure *RootN* has to call itself with the value of the index

$$j+1$$

after every modification of the unknown x_j. It can be done in a following manner:

```
procedure RootN(j:integer; var x:array of double);
var
...
begin
...
repeat ...
x[j] :=
if j < N then RootN(j + 1, x);
f[j]:=...
...
until...
...
endRootN;
```

This general structure of the procedure must include further elements to ensure global convergence.

Like in Chapter 1, a definition of the bounds of the root in the case of a N-dimensional space will be formulated.

A set Ω of points

$$X = [x_1, x_2, ...x_N]$$

of the N-dimensional space will be referred to as the **bounds of the root** of the system of equations (4.1) if:

$$\Omega = \{X \in R^N : a_1 \leq x_1 \leq b_1, a_2 \leq x_2 \leq b_2, ..., a_N \leq x_N \leq b_N\}, \tag{4.9}$$

$$f_j(x_1, x_2, ..., x_{j-1}, a_j, x_{j+1}, ...x_N) f_j(x_1, x_2, ..., x_{j-1}, b_j, x_{j+1}, ...x_N) < 0, \quad j = 1, 2, ..., N \tag{4.10}$$

As continuity of the functions f has been assumed, it means, that the root of the system of equations (4.1) belongs to Ω. Moreover, the condition (4.10) ensures that for an arbitrary

$$X \in \Omega$$

the interval

$$[a_j, b_j]$$

represents the bounds of the equation (4.8) with the unknown x_j.

Finally, it can be said that the modified procedure *RootN*2 will be globally convergent in the domain Ω provided that the values

$$a_i, b_i \qquad i = 1, 2, ..., N$$

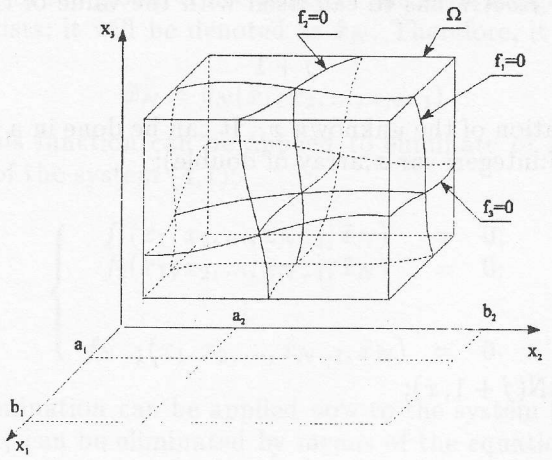

Fig. 5.

satisfying the conditions (4.9) and (4. 10) are known.

Interpretation of the conditions (4.9), (4. 10) in the 3-dimensinal space ($N = 3$) is shown in Fig. 5. Determination of the domain Ω is, in general, a difficult problem. In some cases transformation of independent variables:

$$X = AY,$$

may be helpful, Y denoting vector of new variables in this transformation and A – a suitable matrix. For example, if the gradients of the functions f_j are known in a point, representing an approximation of the root of the system (4.1), then the matrix A can be chosen in such a manner that the direction of the vector ∇f_j would coincide with the direction of the axis x_j.

Example. The second task of the program Equations listed in the Annex consists in solving of the following system of equations:

$$\begin{cases} x_1^2 + x_2^2 + x_3^2 - 1 = 0; \\ x_1 - 2x_2 = 0; \\ x_3 = 0, \end{cases} \tag{4.11}$$

representing a section of a sphere with two planes. In order to satisfy the conditions (4.9), (4. 10) the following bounds of the root will be accepted:

$$a_1 = 0, b_1 = 0.5, a_2 = 0, a_3 = -0.02, b_3 = -0.02$$

Postulating the accuracy

$$prec = 1.0 \cdot 10^{-7}$$

one obtains the following solution:

$$x_1 = 8.944 \cdot 10^{-1} x_2 = 4.472 \cdot 10^{-1} x_3 = 3.814 \cdot 10^{-8}$$

The computational cost is proportional to the number of callings of the functional procedure evaluating left – hand – sides of equations in the system (4.11). In the example presented this number turned out to be rather high, namely 75 11.

Acknowledgement. The author renders best thanks to Professor Włodzimierz J.Prosnak for his encouragement to create this paper and for his help to compose it.

Manuscript received: February 18, 1994

References

[1] M. E. Klonowska, W. J. Prosnak: *Jednostopniowe odwzorowanie obszaru jednospójnego na obszar kołowy*, Zeszty Naukowe IMP PAN, 405/1367/93

[2] B. Carnahan, H. Luther, J. Wilkes: *Applied numerical methods*, John Wiley and Sons, 1969.

[3] W. J. Prosnak: *Wprowadzenie do numerycznej mechaniki płynów. Część A. Podstawowe metody numeryczne*, Zakład Narodowy im. Ossolińskich – Wydawnictwo, Wrocław 1993.

O efektywnym algorytmie do rozwiązywania równań nieliniowych i ich układów

Streszczenie

Niniejsza praca jest poświęcona metodzie rozwiązywania równania nieliniowego, otrzymanej w wyniku połączenia metod bisekcji, *reguły falsi* i metody siecznych – globalnie zbieżnej przy pewnych założeniach. Opisany jest także sposób uogólnienia tej metody na przypadek układu równań nieliniowych.

Annex

Annex contains Listings of the program **Equations**, and of the unit **rq_no_l2** which consists of the procedures **Root2, RootN2** and **fun_j**. The program contains two further procedures: the procedure **f**, and the proedure **fn**, computing left-hand-sides of Eqs. (1.1) and (4.11), respectively. The program and all procedures are written in Turbo Pascal.

A user of program is guided by instructions appearing on the computer screen. The first input parameter data allows him to decide whether Eq. (3.2) or the system of equations (4.11) will to be solved by the program. In the second case

number 1 should be typed, otherwise any other number can be introduced.

The second input parameter is a value of the variable **prec**, which refers to accuracy of determination of the root. More exactly, it determines the largest acceptable magnitude of the bounds (2.7), so the iterative process will terminate if the following criterion is satisfied:

$$| x_1 - x_2 | < prec$$

in the case of single equation. An analogous criterion applies in the case of the system of equations.

The final input data consist of pairs of numbers determining the bounds of the root. In the case of single equation just one pair has to be written-in; three such pairs are necessary in the case of the system of equations.

When the computations are terminated the program displays the solution on the screen.

Listing of the unit

```
unit rq_no_12;

interface

const
Km=10;

type
vec            =array[1..Km] of double;
frame          =array[1..2] of vec;
f_type         =function(x:double):double;
functions      =function(var x:vec;j:integer):double;

procedure Root2(x1,x2,f1,f2,prec:double;f:f_type;var s:double);

function fun_j(var x:vec;j,N:integer;f:functions;var ab:frame;
               var prec:double):double;

procedure RootN2(var xk:vec;j,N:integer;f:functions;
                 var ab:frame;var prec:double);

implementation

procedure Root2(x1,x2,f1,f2,prec:double;f:f_type;var s:double);
var
x,y                    :array[1..3] of double;
xx,yy,xs,xb            :double;
```

```
opos                    :boolean;
k_p,k_m,k,fsb,fsbz,it   :integer;
begin
x[1]:=x1; y[1]:=f1;
x[2]:=x2; y[2]:=f2;
if y[1]>0 then k_p:=1 else k_p:=2;
k_m:=3-k_p;
fsbz:=2; opos:=true; k:=1; it:=0;
repeat
it:=it+1;
 if opos then fsb:=1 else fsb:=3;
 if fsbz=1 then fsb:=2;
 fsbz:=fsb;
 if fsb<>1 then begin
   xb:=(x[1]+x[2])/2;
   if abs(y[k])<abs(y[3]) then
   xs:=x[k]-y[k]*(x[k]-x[3])/(y[k]-y[3])
   else xs:=xb;
   if abs(x[k]-xs)>abs(x[k]-xb) then xs:=xb;
   end{fsb<>1};
 if fsb=1 then xx:=x[1]-y[1]*(x[1]-x[2])/(y[1]-y[2]);;
 if fsb=2 then xx:=xs;
 if fsb=3 then xx:=xb;
 yy:=f(xx);
 opos:=(y[k]*yy<0);
 if yy>0 then k:=k_p else k:=k_m;
 x[3]:=x[k];  y[3]:=y[k];
 x[k]:=xx;    y[k]:=yy;
until abs(x[1]-x[2])<=prec;
s:=(x[1]+x[2])/2;
end{Roots2};

function fun_j(var x:vec;j,N:integer;f:functions;var ab:frame;
               var prec:double):double;
begin
if j<N then RootN2(x,j+1,N,f,ab,prec);
fun_j:=f(x,j);
end;

procedure RootN2(var xk:vec;j,N:integer;f:functions;
                 var ab:frame;var prec:double);
var
x,y                     :array[1..3] of double;
xx,yy,xs,xb             :double;
opos                    :boolean;
```

```
k_p,k_m,k,fsb,fsbz,it   :integer;
begin
x[1]:=ab[1,j];
xk[j]:=x[1];
y[1]:=fun_j(xk,j,N,f,ab,prec);
x[2]:=ab[2,j];
xk[j]:=x[2];
y[2]:=fun_j(xk,j,N,f,ab,prec);
k:=1;
while y[1]*y[2]>0 do begin
   k:=k+1;
   if odd(k) then x[1]:=x[1]+(ab[1,j]-ab[2,j])
   else x[2]:=x[2]+(ab[2,j]-ab[1,j]);
   xk[j]:=x[1];
   y[1]:=fun_j(xk,j,N,f,ab,prec);
   xk[j]:=x[2];
   y[2]:=fun_j(xk,j,N,f,ab,prec);
   end;
if y[1]>0 then k_p:=1 else k_p:=2;
k_m:=3-k_p;
fsbz:=2; opos:=true; k:=1; it:=0;
repeat
it:=it+1;
 if opos then fsb:=1 else fsb:=3;
 if fsbz=1 then fsb:=2;
 fsbz:=fsb;
 if fsb<>1 then begin
   xb:=(x[1]+x[2])/2;
   if abs(y[k])<abs(y[3]) then
   xs:=x[k]-y[k]*(x[k]-x[3])/(y[k]-y[3])
   else xs:=xb;
   if abs(x[k]-xs)>abs(x[k]-xb) then xs:=xb;
   end{fsb<>1};
 if fsb=1 then xx:=x[1]-y[1]*(x[1]-x[2])/(y[1]-y[2]);
 if fsb=2 then xx:=xs;
 if fsb=3 then xx:=xb;
 xk[j]:=xx;
 yy:=fun_j(xk,j,N,f,ab,prec);
 opos:=(y[k]*yy<0);
 if yy>0 then k:=k_p else k:=k_m;
 x[3]:=x[k];  y[3]:=y[k];
 x[k]:=xx;    y[k]:=yy;
until abs(x[1]-x[2])<=prec;
xk[j]:=(x[1]+x[2])/2;
end{RootsN2};
```

```
end.{unit}
```

Listing of the program

```
program Equations;
uses RQ_No_12;

var
iter,i,j       :integer;
x1,x2,s,prec   :double;
ab             :frame;
x              :vec;
sys            :boolean;

function f(x:double):double; far;
begin
f:=x-ln(-x);
end;

function fn(var x:vec;j:integer):double; far;
begin
iter:=iter+1;
case j of
1: fn:=sqr(x[1])+sqr(x[2])+sqr(x[3])-1;
2: fn:=x[1]-2*x[2];
3: fn:=x[3];
else fn:=0;
end{case};
end{fn};

function chos(var prec:double):boolean;
var
i:integer;
begin
writeln('Write-in 1 if you wish to solve the system of equations,');
write(' otherwise write-in any other integer:');
readln(i);
if i=1 then chos:=true else chos:=false;
write('Precision :prec=');readln(prec);
end;

begin
```

```
if not chos(prec) then begin
writeln('Write-in first approximationof the bounds of the solution')
writeln('x1<0,  x2<0');
write('x1='); readln(x1);
write('x2='); readln(x2);
root2(x1,x2,f(x1),f(x2),prec,f,s);
writeln('Solution:');
writeln('x=',s);
end else begin
writeln('Write-in first approximation of the bounds of the solution')
for i:=1 to 2 do
for j:=1 to 3 do begin
if i=1 then write('a') else write('b');
write('[',j,']='); readln(ab[i,j]);
end;
rootN2(x,1,3,fn,ab,prec);
writeln('Solution:');
for i:=1 to 3 do
writeln('x[',i,']=',x[i]);
end;
end.{Equations}
```