Scientific Issues
Jan Długosz University
in Częstochowa
Mathematics XXII (2017)
31–45
DOI `http://dx.doi.org/10.16926/m.2017.22.03`

APPLICATIONS OF PYTHON PROGRAMS IN SOLVING OF EQUATIONS BASED ON SELECTED NUMERICAL METHODS

MARCIN ZIÓŁKOWSKI, LIDIA STĘPIEŃ, MARCIN RYSZARD STĘPIEŃ, AND ARTUR GOLA

Abstract

In this paper we present the mathematical background of the four most used numerical methods of solving equations and few examples of Python applications that find the approximations of the roots of the given equations. We also compare the exact and approximate solutions of polynomial equations of third degree. Exact solutions are obtained with usage of Cardano formulae by the help of Mathematica environment, the approximate ones – based on the selected numerical methods by the help of applications written in Python language.

1. Introduction

From the mathematical point of view, it is always very important to find the exact solution of a given equation. Unfortunately, the effective algorithms exist only for a small group of equations. For example, we can solve only polynomial equations whose degree is less than five. But even in this case, the obtained formulae may be very complicated and they cannot be used in practice, especially in the case of third or fourth degree polynomial equations. For the other groups of equations, that contain in addition logarithmic, exponential or trigonometric functions, it is usually impossible to find exact solutions. In such situations, we use numerical methods to find approximations of solutions. Of course, almost all numerical methods have

• *Marcin Ziółkowski* — e-mail: marcin_ ziolkowski@sggw.pl
Warsaw University of Life Sciences.
• *Lidia Stępień* — e-mail: l.stepien@ajd.czest.pl
Jan Długosz University in Częstochowa.
• *Marcin Ryszard Stępień* — e-mail: mstepien@tu.kielce.pl
Kielce University of Technology.
• *Artur Gola* — e-mail: a.gola@ajd.czest.pl
Jan Długosz University in Częstochowa.

their own limitations that relate to the properties of the function appearing in the equation. Moreover, we usually must know in which intervals the roots of a given equation are. On the other hand, we luckily may use many different methods in the process of finding the approximations. The most important methods are: bisection, secant method, tangent method and iteration method [1]. Computations that are made in this case are also very laborious because of many loops appearing. Therefore, we should use computer programs. One of the most effective programming language that let write short and simple applications that solve mathematical problems is Python language. This language additionally has no limitation of the introduced numbers [4].

All analysed numerical methods are based on Bolzano – Cauchy theorem:

Theorem 1. *Let $f(x)$ be a function which is continuous on bounded interval $[a, b]$. If $f(a) \cdot f(b) < 0$, then there exists in interval (a, b) at least one root of the equation $f(x) = 0$.*

2. Bisection method

Now we discuss the first and simplest method which is called bisection. Let f be a continuous function defined on the interval $[a, b]$ and let $f(a) \cdot f(b) < 0$ what means that the sign of the values of this function changes in interval $[a, b]$. We will look for zero of the function f (or the root of an equation $f(x) = 0$). The idea of bisection is connected with the division the analysed interval onto two identical parts using the midpoint of an interval. Each time we choose this part of an interval where the value of the function changes the sign. The cut is continued until we get the exact solution or the required precision of the approximation is reached. If the function f fulfils the assumptions mentioned above, then the root of an equation exists and the bisection method can be used for finding it. In the next steps we determine the point x_0, which is a center of an interval $[a, b]$, taking: $x_0 = \frac{a+b}{2}$. Next, we count the value of the function at this point. If the $f(x_0)$ is sufficiently close to 0, i.e. $|f(x_0)| < \epsilon$, then x_0 can be accepted as a proper approximation of the root of an equation and the algorithm ends. In the other case, we take a new interval for searching the root – it will be the interval $[a, x_0]$ or $[x_0, b]$ depending on in which interval the function f changes the sign at its endpoints. The algorithm is repeated from the beginning until required precision is reached.

Example 1. *Find the approximation of solution of the equation* $x^3 - 17x - 13 = 0$ *in the interval* $[4, 5]$

We will use the Python application that is written on the base of the bisection method. The program is listed below.

```
BISECTION METHOD - PROGRAM IN PYTHON LANGUAGE
def fun(x):
    return x*x*x-17*x-13
def main():
    a=float(input("Enter the left endpoint of interval: "))
    b=float(input("Enter the right endpoint of interval: "))
    precision=float(input("Enter precision of solution: "))
    x_0=(a+b)/2
    while abs(fun(x_0)) >= precision:
        x_0=(a+b)/2
        if fun(x_0)==0:
            break
        elif fun(a)*fun(x_0)<0:
            b=x_0
        else:
            a=x_0
        print(x_0)
    print("Approximation of solution is:",x_0)
main()
```

Analysed equation has three real solutions. For example, the only positive solution can be found in the interval $[4, 5]$. The exact solution is equal to (*Mathematica environment*):

$$x_0 = \frac{\sqrt[3]{\frac{1}{2}\left(117+\sqrt{45267}i\right)}}{\sqrt[3]{9}} + \frac{17}{\sqrt[3]{\frac{3}{2}\left(117+\sqrt{45267}i\right)}}.$$

As it is seen the solution has a very complicated form consisting complex numbers and we are not able to find even the approximation of it. Using the Python program we find an approximation of the chosen precision (the precision is understood as the difference between the value $f(x_0)$ and zero). The sequence of the approximations of the above solution is presented in Table 1. Precision of the approximation is 10^{-8}.

The advantage of using the bisection method is its simplicity. This method is always convergent, but slowly converges. The interval in which the root is located is always divided into half. If the determined approximation is close to the real root, then the speed of the method decreases.

3. Secant method

Now we remind the second numerical method of finding the roots of non-linear equations which is called secant method. This method uses (during

STEP	**APPROXIMATION**
1	4.5
2	4.25
3	4.375
4	4.4375
5	4.46875
6	4.453125
7	4.4609375
8	4.46484375
9	4.462890625
10	4.4619140625
11	4.46240234375
12	4.46264648438
13	4.46252441406
14	4.46246337891
15	4.46243286133
16	4.46241760254
17	4.46242523193
18	4.46242141724
19	4.46242332458
20	4.46242237091
21	4.46242284775
22	4.46242260933
23	4.46242272854
24	4.46242266893
25	4.46242263913
26	4.46242262423
27	4.46242263168
28	4.46242263541
29	4.46242263354
30	4.46242263447
31	4.46242263494

Table 1. Approximations of the solution of the equation $x^3 - 17x - 13 = 0$ in the $[4, 5]$ interval – bisection method

generating successive approximations of the value of the sought root of an equation) the linear interpolation. The linear interpolation strategy is built on the basis of the known values of the two recently calculated ordinates of function f. Assume that we have a given function f, two starting points x_1

and x_2, and an interval $[a,b]$ in which we search the root, where the points x_1 and x_2 belong. In this interval, the function must fulfil the following conditions:

(1) f is differentiable on the interval $[a,b]$ and it is continuous there,

(2) f has different signs at the endpoints of the interval $[a,b]$ (this does not apply to the points x_1 and x_2),

Since the function is continuous, then by Darboux theorem it accepts in the interval $[a,b]$ all values between $f(a)$ and $f(b)$. These values have different signs (or they lie on different sides of OX axis), so there must be a point x_0 in the interval $[a,b]$, for which one of the two possibilities holds: $f(a) < f(x_0) < f(b)$ or $f(b) < f(x_0) < f(a)$.

(3) $f'(x) \neq 0$ for $x \in [a,b]$,

Therefore, there is no local minimum or maximum. This condition assures that the secant will not be parallel to the OX axis, which would make it impossible to determine its intersection with this OX axis.

If the function f satisfies the given conditions, there exists a root of a given equation in the interval $[a,b]$. In the secant method, two previously designated points are used to determine the next approximation of the function root, as follows: we create in a given interval $[a,b]$ the sequence of secant lines whose zeroes converge to the solution of the equation $f(x) = 0$. Depending on the signs of the function $f(x)$ and $f''(x)$ at the points a and b we use two different formulae. If $f(b) \cdot f''(b) > 0$, then we are looking for solution with the formula:

$$x_0 = a,\ x_{n+1} = x_n - \frac{f(x_n)}{f(b) - f(x_n)}\,(b - x_n)\,,\ n \geq 1. \tag{1}$$

However, if $f(a) \cdot f''(a) > 0$, then we find solution basing on the following formula:

$$x_0 = b,\ x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(a)}\,(x_n - a)\,,\ n \geq 1. \tag{2}$$

Now we will illustrate the method using some example.

Example 2. *Find the approximation of the solution of an equation* $x^3 + 7x^2 - 11x + 4 = 0$ *in the interval* $[-9,-8]$.

We will use the Python application that is written on the base of the secant method and formulae (1) and (2). Here is the listing of the program.

```
SECANT METHOD - PROGRAM IN PYTHON LANGUAGE
def fun(x):
    return x*x*x+7*x*x-11*x+4
def diff2(x):
    return 6*x+14
def main():
    a=float(input("Enter the left endpoint of interval: "))
    b=float(input("Enter the right endpoint of interval: "))
    precision=float(input("Enter precision of solution: "))
    if fun(b)*diff2(b)>0:
        x_0=a
        while abs(fun(x_0))>=precision:
            x_0=x_0-(fun(x_0)/(fun(b)-fun(x_0)))*(b-x_0)
            print(x_0)
    elif fun(a)*diff2(a)>0:
        x_0=b
        while abs(fun(x_0))>=precision:
            x_0=x_0-(fun(x_0)/(fun(x_0)-fun(a)))*(x_0-a)
            print(x_0)
    print("Approximation of the solution is:",x_0)
main()
```

Analysed equation has a real solution located in the interval $[-9, -8]$. The exact solution is equal to (*Mathematica environment*):

$$x_0 = \frac{1}{3}\left(-82\sqrt[3]{\frac{2}{1487-3\sqrt{633}}} - \sqrt[3]{\frac{1}{2}\left(1487-3\sqrt{633}\right)}\right) - \frac{7}{3}.$$

As it is seen the solution has also a very complicated form. Using the Python program we find an approximation of the chosen precision. The sequence of the approximations of the above solution is presented in Table 2. The precision is 10^{-12}.

The advantage of the secant method is that we do not need to know the analytical form of the derivative of the function. It is not as fast as the Newton method (tangent method that will be discussed in the next section), but usually faster than the bisection method as its convergence takes into

STEP	**APPROXIMATION**
1	-8.32183908045977
2	-8.364885832734824
3	-8.370338145447704
4	-8.371023837640712
5	-8.371109993976804
6	-8.371120818185480
7	-8.371122178060274
8	-8.371122348904741
9	-8.371122370368353
10	-8.371122373064878
11	-8.371122373403650
12	-8.371122373446212
13	-8.371122373451557
14	-8.371122373452230
15	-8.371122373452314
16	-8.371122373452325

Table 2. Approximations of the solution of the equation $x^3 + 7x^2 - 11x + 4 = 0$ in the [-9,-8] interval – secant method

account the shape of the graph of the function (the interval is not divided on two identical parts).

4. Newton tangent method

Newton method which is called also the tangent method allow to find the approximations of the roots of the continuos function f. We can also use it in the case when we are not able to find exact solutions of the equation $f(x) = 0$ or when exact solutions are complicated. This method is one of the best methods as its convergence is very quick but unfortunately the function f in the equation must satisfy the following strong conditions [2]:

Theorem 2. *If the function f is continuous in the interval $[a, b]$ and:*

- *signs of the function f are different on the endopints of the interval $[a, b]$ i. e. $f(a) \cdot f(b) < 0$,*
- *there exists only one root of the function f in the analysed interval,*
- *first and second derivative of the function f do not change their signs in the interval $[a, b]$,*

then equation $f(x) = 0$ has only one solution $x_z \in [a, b]$.

If the function f has more zeroes in the interval $[a, b]$ we may use this method dividing the interval $[a, b]$ onto smaller ones in which the conditions given in the above theorem are satisfied.

The Newton method means constructing the sequence of the tangents to the graph of the function f. Points of intersection the secants with OX axis are the following approximations of the root of the function f. The method is convergent to this solution x_z independent on the choice of a startpoint $x_0 \in [a, b]$. The idea of the Newton method is presented in Fig. 1. and Fig. 2.

In the first step we choose the starting point x_0 and we construct the first tangent to the graph of the function f in the point $P_0 = (x_0, f(x_0))$. We usually take as x_0 one of the points a or b in which signs of the function f and its second derivative f'' are the same ($f(a) \cdot f''(a) > 0$ or $f(b) \cdot f''(b) > 0$). The tangent to the graph of the function f in the point P_0 has the intersection with the OX axis in the point $(x_1, 0)$. Then we construct the next tangent to the graph of the function f in the point $P_1 = (x_1, f(x_1))$ and we find its intersection with the OX axis finding the point $(x_2, 0)$ and so on. This recurentive procedure let find the approximations of the solution of the equation $f(x) = 0$. The sequence $x_0, x_1, \ldots$ converges to the sought solution.

The procedure can be presented as the recurentive formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{3}$$

The choice of the starting point is very important and has an influence on the convergence of the procedure. The Newton method usually converges faster than secant method. [3].

Every numerical method generates errors. The error of n-th approximation in this case can be found using the following formula :

$$|x_z - x_n| \leq \frac{f(x_n)}{\min_{x\in[a,b]} |f'(x)|}.$$

or

$$|x_z - x_n| \leq \frac{\max_{x\in[a,b]} |f''(x)|}{2\min_{x\in[a,b]} |f'(x)|}(x_z - x_{n-1})^2.$$

where x_z is the exact value of the zero of the function f.

Practically, to end the algorithm we choose one of the following conditions:

- $|f(x_n)| < \epsilon$ – the value of the function f in the point x_n is less than ϵ

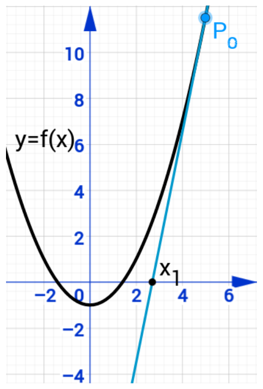


FIGURE 1. First approximation – point x_1 [own work]

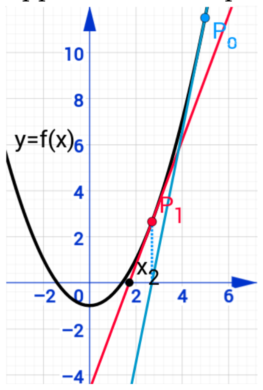


FIGURE 2. Second approximation – point x_2 [own work]

- $|x_{n+1} - x_n| < \epsilon$ – the difference between approximations is less than ϵ
- $\dfrac{M}{2m}(x_{n+1} - x_n)^2 \leq \epsilon$ – the error is less than ϵ.

Example 3. *Find the approximation of the root of the equation* $\frac{1}{10}(x^3 + 4x^2 - 9x + 1) = 0$ *in the interval* $[1, 2]$.

The function f satisfies the conditions of the theorem 2: it is continuous, the values on the endpoints have different signs: $f(1) = -\frac{3}{10}, f(2) = \frac{7}{10}$. First derivative is equal to $f'(x) = \frac{1}{10}(3x^2 + 8x - 9)$ and its zeroes are equal to $x_1 = -\frac{1}{3}(4 + \sqrt{43})$ and $x_2 = \frac{1}{3}(\sqrt{43} - 4) < 1$, so $f'(x) > 0$ for $x \in (-\infty, x_1) \cup (x_2, \infty)$ – first derivative of the function has the same sign in the interval $[1, 2]$. Second derivative is equal to $f''(x) = \frac{1}{5}(3x + 4)$, so $f''(x) > 0$ for $x \in (-\frac{4}{3}, \infty)$ – second derivative of the function has the same sign in the interval $[1, 2]$ (see also Fig. 3.).

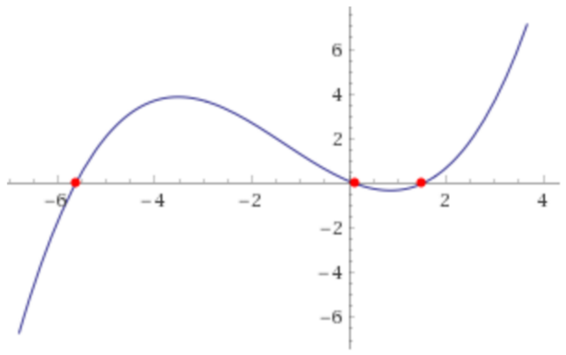


FIGURE 3. Graph of the function $f(x)=\frac{1}{10}(x^3+4x^2-9x+1)$ [www.wolframalpha.com]

We can find the exact solutions of the analysed equation. The solution in the interval $[1,2]$ (*Mathematica environment*) has the form:

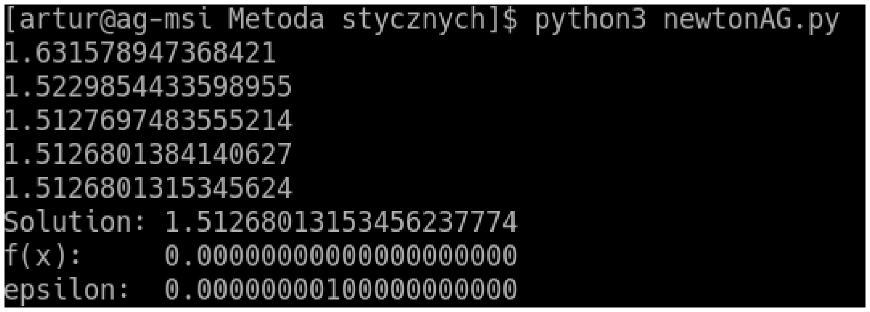
$$x=-\frac{4}{3}+\frac{1}{3}\left(\frac{43}{\sqrt[3]{\frac{1}{2}\left(-479+3i\sqrt{9843}\right)}}+\sqrt[3]{\frac{1}{2}\left(-479+3i\sqrt{9843}\right)}\right)$$

The exact solution is not practical, so we use the Newton method to find the approximation of it.

The Python program can be written using formula (3). Its code is listed below.

```
TANGENT (NEWTON) METHOD - PROGRAM IN PYTHON LANGUAGE
import math
def f(x):
    return (x*x*x+4*x*x-9*x+1) / 10
def dev_f(x):
    return (3*x*x+8*x-9) / 10
def main():
    a = 1
    b = 2
    x_0 = b
    precision = float(input("Determine precision:"))
    while abs(f(x_0))>=precision:
        x_0 = x_0 - f(x_0) /dev_f(x_0)
        print(x_0)
    print("Approximation of the solution is:",x_0)
main()
```

The approximation of the solution with precision 10^{-15} we find in fifth iteration. The approximations are presented in Fig. 4.



```
[artur@ag-msi Metoda stycznych]$ python3 newtonAG.py
1.631578947368421
1.5229854433598955
1.5127697483555214
1.5126801384140627
1.5126801315345624
Solution: 1.51268013153456237774
f(x):     0.00000000000000000000
epsilon:  0.00000000100000000000
```

FIGURE 4. The following approximations of the equation [own work]

5. Iteration method

Iteration method is a very practical application of Banach fixed-point theorem to finding the approximations of the solutions of nonlinear equations. The basic advantage of this method is its fast convergence. So we do not need many iterations to find the satisfactionable approximation. But unfortunatelly this method can be used only if the following conditions are satisfied:

(1) The nonlinear equation $f(x) = 0$ must be presented in the form $g(x) = x$, where g is differentiable in the analysed interval $[a, b]$;
(2) The function g is a contraction mapping in the interval $[a, b]$.

Let us remind the definition of the contraction mapping.

Definition 1. *The function f is said to be a contraction mapping in interval $[a, b]$ if there exists a constant $M < 1$, such that for all $x, y \in [a, b]$ the following condition is satisfied:*

$$|f(x) - f(y)| \leq M\,|x - y|\,.$$

Remark. It is clear (and can be eaisily proved) that if the function f is differentiable in the interval $[a, b]$ and $|f'(x)| < 1$ for all $x \in [a, b]$ then the function f is a contraction mapping. We also use this practical condition to check if the function is a contraction map.

Now we remind the theorem which is the base of the iteration method.

Theorem 3. *If the function f is a contraction mapping in the interval $[a, b]$ then there exists only one point x_0 such that $f(x_0) = x_0$. In addition the sequence given by the following recurentive formula: $a, f(a), f(f(a)), \dots$ is convergent to x_0 independent on the choice of the starting point a.*

As it may be noticed, in this case there exists only one solution of the equation $f(x) = x$.

Basing on the above theorem we can present the following algorithm of the iteration method:

(1) We rewrite the equation $f(x) = 0$ in the form $g(x) = x$;
(2) We check if in the interval $[a, b]$ the function f is such that $f(a) \cdot f(b) < 0$ and function g is a contraction mapping;
(3) We create the sequence: $t, g(t), g(g(t)), g(g(g(t))), \dots$ that is convergent to the sought solution, the starting point $t \in [a, b]$ can be chosen arbitraly.

Now we will illustarate the method with two examples.

Example 4. *Find the solution of the equation $\cos x = 2x$ in the interval $[0, \frac{\pi}{2}]$ using iteration method .*

Let us notice that this equation is equivalent to the equation $\cos x - 2x = 0$. Let f be the function given by the formula $f(x) = \cos x - 2x$. It is clear that this function has at least one zero in the interval $[0, \frac{\pi}{2}]$ as $f(0) = 1 > 0$ and $f(\frac{\pi}{2}) = -\pi < 0$. In addition it is not a polynomial equation, we cannot find the exact solution. Let us rewrite the equation in the form $\frac{\cos x}{2} = x$. The function g given by the formula $g(x) = \frac{\cos x}{2}$ is a contraction mapping in every interval as we have an inequality $|g'(x)| = \frac{1}{2} |\sin x| < 1$. Now we can find the approximation of the only one solution using the Python program that is listed below.

```
ITERATION METHOD - PROGRAM IN PYTHON LANGUAGE
import math
x=0
print("Determine precision:")
precision=float(input())
while abs(math.cos(x)-2*x)>=precision:
    x=0.5*math.cos(x)
    print(x)
print("Approximation of the solution is:",x)
```

In fifth step we obtain the approximation $x \approx 0,45$ with precision 10^{-3} and in ninth step approximation $x \approx 0,450184$ with precision 10^{-6}.

Example 5. *Find the solution of the equation* $x^3+4x-1=0$ *in the interval* $[0,1]$.

Let W be a polynomnial given by the formula $W(x)=x^3+4x-1$. We easily notice that $W(0)=-1$ and $W(1)=4$. So at least one solution of the equation exists in the interval $[0,1]$. We can use Cardano formulae and by the help of *Mathematica environment* find the exact solution:

$$x_0=\frac{\sqrt[3]{\frac{1}{2}\left(9+\sqrt{849}\right)}}{\sqrt[3]{9}}-4\sqrt[3]{\frac{2}{3\left(9+\sqrt{849}\right)}}.$$

The exact solution has very complicated form so we use iteration method in this case. We can rewrite the equation in the form $\frac{1-x^3}{4}=x$. The functiong given by the formula $g(x)=\frac{1-x^3}{4}$ is a contraction mapping in the interval $[0,1]$ as we have the inequality: $|g'(x)|=\frac{3}{4}x^2<1$.

Now we can find the approximations of the only one solution using Python program.

```
ITERATION METHOD - PROGRAM IN PYTHON LANGUAGE
import math
def fun(x):
    return 0.25*(1-x*x*x)
def main():
    x=0
    print("Determine precision:")
    precision=float(input())
    while abs(x*x*x+4*x-1)>=precision:
        x=fun(x)
        print(x)
    print("Approximation of the solution is:",x)
main()
```

It is interesting that we obtain the approximation $x\approx 0,246$ with precision 10^{-3} in the second step, and the approximation $x\approx 0,246266$ with precision 10^{-6} in the fifth step, so the iteration method in this case is fast convergent.

6. Final remarks

In this article we have presented the possiblilities of using the computer programs written in Python language in solving of nonlinear equations. In many real situations it is the best way to find at least approximations of the solutions of such equations as in many cases there does not exist the way to obtain the exact form of the solutions or exact solutions have very compilcated impractical forms consisting for example complex numbers. Then we may use numerical methods to find approximations. We remind four important methods: bisection method, secant method, tangent method and iteration method and we show how to use them to write computer programs that find the approximations of the solutions machanically. On one hand Python language programs have a very simple structure so writing programs is very quick and simple, on the other hand programs let to avoid long and complicated computations. Of course, every method have its own advantages and disadvantages. Bisection method is very simple and we can use it to every equation, but it is slowly convergent, secant and tangent methods are usually faster but the functions present in equations must satisfy adiitional, often strict, conditions. The iteration method seems to be the fastest in many cases but we can use it very seldom as it is not always possible to present the equation in a form $g(x) = x$ and even if we do that, the function g must be a contraction mapping in the analysed interval.

References

[1] Povstenko J. *Wprowadzenie do metod numerycznych*, Wydawnictwo EXIT, Warszawa 2005.
[2] Fortuna Z., Macukow B., Wąsowski J., *Metody numeryczne*, Wydawnictwo Naukowo-Techniczne, Warszawa 2002.
[3] Ryaben'kii V.S.; Tsynkov S.V., *A Theoretical Introduction to Numerical Analysis*, CRC Press, Boca Raton 2007.
[4] Dawson M. *Python dla każdego. Podstawy programowania*, Helion, Warszawa 2014.

Received: October 2017

Marcin Ziółkowski
Warsaw University of Life Sciences,
Department of Informatics,
Ul. Nowoursynowska 159, 02-787 Warszawa, Poland
E-mail address: `marcin_ ziolkowski@sggw.pl`

Lidia Stępień
Jan Długosz University in Częstochowa,
Institute of Mathematics and Computer Science,
Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland
E-mail address: `l.stepien@ajd.czest.pl`

Marcin Ryszard Stępień
Kielce University of Technology,
Department of Mathematics and Physics,
Al. Tysiąclecia Państwa Polskiego 7, 25-314 Kielce, Poland
E-mail address: mstepien@tu.kielce.pl

Artur Gola
Jan Długosz University in Częstochowa,
Institute of Mathematics and Computer Science,
Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland
E-mail address: a.gola@ajd.czest.pl