Laboratorium 5

Pierwiastki równań nieliniowych

Jan Rajczyk

19 kwietnia 2021

1 Zadania

1. Mamy równanie:

$$f(x) = x^2 - 2 = 0$$

   (a) zakładając że mamy punkt początkowy $x_0 = 1$, jaką wartość x_1 dostaniemy, jeśli używamy metody Newtona ?

   (b) zakładając że mamy $x_0 = 1$ i $x_1 = 2$ jako punkty początkowe, jaka będzie wartość x_2, jeśli używamy metody siecznych do tego samego problemu ?

2. Napisz iteracje wg metody Newtona do rozwiązywania każdego z następujących równań nieliniowych:

   (a) $x^3 - 2x - 5 = 0$,

   (b) $e^{-x} = x$,

   (c) $xsin(x) = 1$.

3. Zapisz iteracje Newtona do rozwiązywania następującego układu równań nieliniowych.

$$\begin{cases} x_1^2 + x_2^2 = 1 \\ x_1^2 - x_2 = 0 \end{cases}$$

2 Rozwiązania

1. (a) Będziemy korzystali ze wzoru:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$f'(x_k) = 2x_k$$

Mamy więc: $x_{k+1} = x_k - \frac{x_k^2-2}{2x_k}$, stąd dla $x_0 = 1$ otrzymujemy:

$$x_1 = 1 - \frac{1-2}{2} = 1 + \frac{1}{2} = \frac{3}{2}.$$

(b) Będziemy korzystali ze wzoru:

$$x_{k+1} = x_k - \frac{f(x_k)(x_{k-1} - x_k)}{f(x_{k-1}) - f(x_k)}$$

Mamy: $f(x_0) = f(1) = -1$ oraz $f(x_1) = f(2) = 2$. Otrzymujemy więc:

$$x_2 = 2 - \frac{2 \cdot (1-2)}{-1-2} = 2 - \frac{-2}{-3} = 2 - \frac{2}{3} = \frac{4}{3}.$$

Wnioski: Prawdziwe pierwiastki to -2 oraz 2, stąd metoda Newton'a daje nieco lepszy wynik w tym przypadku.
Metody Newton'a oraz siecznych są proste w implementacji i już dla małych k dają bardzo dobre wyniki, nie jest więc dziwne ich powszechne użycie w nauce i technice.

2. Wyniki zostały obliczone za pomocą programu napisanego w języku *Python*:

```
from math import e
from math import sin
from math import cos

#funkcja z zadania (a)
def f_a(x):
    return (2 * x ** 3 + 5) / (3 * x ** 2 - 2)

#funkcja z zadania (b)
def f_b(x):
    return x-(e ** (-x) - x) / (-e ** (-x) - 1)

#funkcja z zadania (c)
def f_c(x):
    return x-(x*sin(x))/(sin(x)+x*cos(x))


def Newton(x0: float, n: int, f):
    x_k = x0
    for k in range(1, n + 1):
        x_k = f(x_k)
        print("k:", k ,"x_k:", x_k)

print("Podpunkt a):")
Newton(1, 10, f_a)
print("Podpunkt b):")
Newton(1, 10, f_b)
print("Podpunkt c):")
Newton(1, 10, f_c)
```

(a) Zauważmy, że:

$$x_{k+1} = x_k - \frac{x_k^3 - 2x_k - 5}{3x_k^2 - 2} = \frac{3x_k^3 - 2x_k - x_k^3 + 2x_k + 5}{3x_k^2 - 2} = \frac{2x_k^3 + 5}{3x_k^2 - 2}$$

Niech $x_0 = 1$ (funkcja oraz jej dwie pochodne są ciągłe, x_0 jest blisko rozwiązania oraz $f'(x_0) \neq 0$. Wyniki przedstawia tabelka:

k	x_k
1	7.0
2	4.76551724137931
3	3.348702759480283
4	2.5315996410025097
5	2.1739158849392317
6	2.097883686441764
7	2.094557715850057
8	2.0945514815642077
9	2.094551481542327
10	2.0945514815423265

Tablica 1: Wyniki dla pierwszych dziesięciu punktów

(b) Mamy:

$$x_{k+1} = x_k - \frac{e^{-x_k}}{-e^{-x_k} - 1}$$

Niech $x_0 = 1$ (funkcja oraz jej dwie pochodne są ciągłe, x_0 jest blisko rozwiązania oraz $f'(x_0) \neq 0$. Wyniki przedstawia tabelka:

k	x_k
1	0.5378828427399902
2	0.5669869914054133
3	0.567143285989123
4	0.5671432904097838
5	0.567143290409784
6	0.5671432904097838
7	0.567143290409784
8	0.5671432904097838
9	0.567143290409784
10	0.5671432904097838

Tablica 2: Wyniki dla pierwszych dziesięciu punktów

(c) Mamy:

$$x_{k+1} = x_k - \frac{x_k \sin x_k}{\sin x_k + x_k \cos x_k}$$

Niech $x_0 = 1$ (funkcja oraz jej dwie pochodne są ciągłe, x_0 jest blisko rozwiązania oraz $f'(x_0) \neq 0$. Wyniki przedstawia tabelka:

k	x_k
1	0.391020950769569
2	0.19034374168959914
3	0.0945922783938496
4	0.047225459684434624
5	0.02360394824726822
6	0.011800878076633858
7	0.00590030208396525
8	0.0029501339242978884
9	0.0014750648224886335
10	0.0007375321437883491

Tablica 3: Wyniki dla pierwszych dziesięciu punktów

Wnioski: Wartość wybranego przez nas x_0 nie ma zbytniego znaczenia, ponieważ już po kilku iteracjach kolejne wartości są bardzo zbliżone.

3. Zapiszmy zadany układ jako:

$$\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ x_1^2 - x_2 = 0 \end{cases}$$

Korzystając ze wzorów z prezentacji napisałem program w języku *Python* na podstawie którego udało mi się wyliczyć x oraz y dla 10 iteracji.

```
def f1(x, y):
    return x**2 + y**2 - 1

def f2(x, y):
    return x**2 - y

def f1x(x, y):
    return 2*x

def f2x(x, y):
    return 2*x

def f1y(x, y):
    return 2*y

def f2y(x, y):
    return -1

def Jacob(x, y):
    return f1x(x, y)*f2y(x, y) - f1y(x, y)*f2x(x, y)

def Newton(x0, y0, n):
    x = x0
    y = y0
    for i in range(1, n+1):
        delta_x = (-f1(x, y)*f2y(x, y)+f2(x, y)*f1y(x, y))/Jacob(x, y)
        delta_y = (-f2(x, y)*f1x(x, y)+f1(x, y)*f2x(x, y))/Jacob(x, y)
        x = x + delta_x
        y = y + delta_y
        print("i:", i, "x:", x, "y:", y)

Newton(1, 1, 10)
```

Wyniki przedstawiłem w tabelce:

i	x_i	y_i
1	0.8333333333333334	0.6666666666666667
2	0.7880952380952381	0.6190476190476191
3	0.7861540663060879	0.6180344478216818
4	0.7861513777620804	0.618033988749989
5	0.7861513777574233	0.6180339887498948
6	0.7861513777574233	0.6180339887498948
7	0.7861513777574233	0.6180339887498948
8	0.7861513777574233	0.6180339887498948
9	0.7861513777574233	0.6180339887498948
10	0.7861513777574233	0.6180339887498948

Tablica 4: Wyniki dla pierwszych dziesięciu iteracji

Wnioski: Widzimy, że już po kilku pierwszych iteracjach wyniki są do siebie bardzo zbliżone. Metoda ta, choć nie najprostsza w zrozumieniu, jest bardzo prosta w implementacji i daje naprawdę dobre rezultaty.

3 Bibliografia

- Katarzyna Rycerz: *Wykład z przedmiotu Metody Obliczeniowe w Nauce i Technice*
- `https://en.wikipedia.org/wiki/Newton%27s_method`
- `https://www.wolframalpha.com/`
- Piotr Fronczak *Metody numeryczne*