# Rozwiazywanie równań nieliniowych

Joanna Klimek

Informatyka, WIET

28.04.2021r.

Nie udało mi się wykonać zadania pierwszego, ani bitowej dokładności w zadaniu 2.

**Zadanie 2.**

Napisz schemat iteracji wg metody Newtona dla każdego z następujących równań nieliniowych:

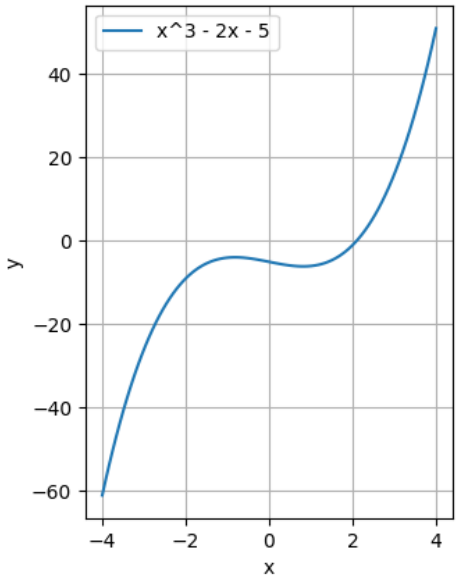
a) $x^3 - 2x - 5 = 0$
b) $e^{-x} = x$
c) $x\,sin(x) = 1$

Jeśli $x_0$ jest przybliżeniem pierwiastka z dokładnością 4 bitów, ile iteracji należy wykonać aby osiągnąć:

- 24-bitowa dokładność
- 53-bitowa dokładność?

a) Aby móc zastosować metodę Netwona funkcja powinna spełniać następujące warunki na przedziale $I = [a, b]$:
   - $f(a) \cdot f(b) < 0$ – funkcja ma różne znaki na końcach przedziału = ma min. 1 pierwiastek w przedziale
   - $f'(x) \neq 0, x \in I$ – pierwiastek jest jednokrotny
   - $f''(x) \geq 0\ lub\ f''(x) \leq 0, dla\ wszystkich\ x \in I$

Najpierw sprawdzamy wykres badanej funkcji:




Widać, że szukane miejsce zerowe mieści się w przedziale [1,3].

Sprawdzamy czy faktycznie tak jest ze wzoru: $f(a) \cdot f(b) < 0$

$$f(1) \cdot f(3) = (1^3 - 2 \cdot 1 - 5) \cdot (3^3 - 2 \cdot 3 - 5) = (1 - 2 - 5) \cdot (27 - 6 - 5) = -6 \cdot 16 = -96 < 0$$

A więc warunek jest spełniony.

Sprawdzimy jeszcze dla pewności krotność pierwiastka:

$$f'(x) = 3x^2 - 2$$

$$f'(1) \cdot f'(3) = (3 \cdot 1^2 - 2) \cdot (3 \cdot 3^2 - 2) = 1 \cdot 25 = 25 > 0$$

Dowodzi to, że nasza pierwsza pochodna jest stałego znaku, co potwierdza drugi warunek.

Z następnej kolejności zbadamy drugą pochodną:

$$f''(x) = 6x$$

$$f''(1) \cdot f''(3) = (6 \cdot 1) \cdot (6 \cdot 3) = 6 \cdot 18 = 108 > 0$$

Co udowadnia, że druga pochodna również jest stałego znaku, a więc spełniony jest warunek trzeci.

Ostatecznie wzór Newtona będzie zbieżny na wybranym przedziale do miejsca zerowego.

Wzór wykorzystywany przy metodzie Newtona:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

Przy założeniu, że $f(x) = 0$:

$$0 = f(x_0) + f'(x_0)(x - x_0)$$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Podstawiam funkcje do wzoru na iteracje Newtona:

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

Wyniki otrzymane w programie dla punktu startowego $x = 1$:

```
1 : x = 7.0
2 : x = 4.76551724137931
3 : x = 3.348702759480283
4 : x = 2.5315996410025097
5 : x = 2.173915884939232
6 : x = 2.097883686441764
7 : x = 2.094557715850057
8 : x = 2.0945514815642077
9 : x = 2.0945514815423265
10 : x = 2.094551481542327
```

Ostateczny wynik: x = 2.094551481542327

Jest on zgodny z obserwacjami na wykresie.

Kod programu wykorzystany do narysowania wykresu:

```
import numpy as np
import matplotlib.pyplot as plt

fx = lambda x: x**3 - (2*x) - 5

x = np.linspace(-4, 4, 100)

fig, ax = plt.subplots()
ax.plot(x, fx(x), label="x^3 - 2x - 5")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_aspect(0.1)
ax.legend()
ax.grid()
plt.show()
```

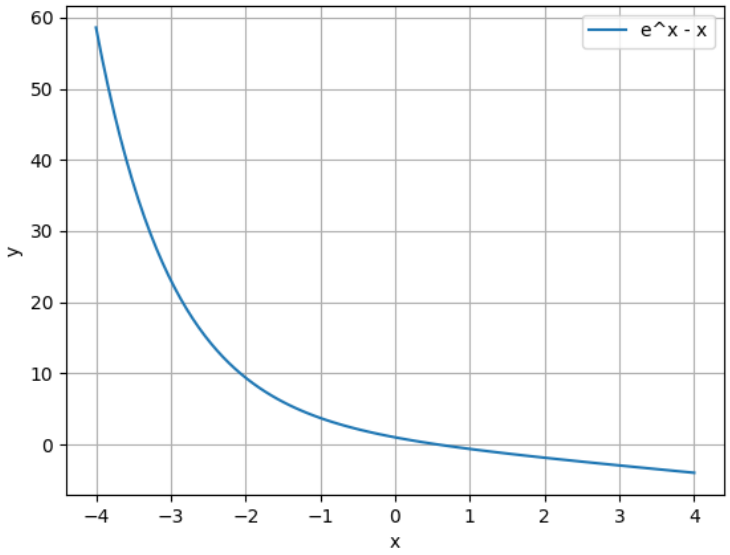
Kod programu wykorzystany do obliczeń:

```
import numpy as np

def newton_method(x, i):
    # badana funkcja
    fx = lambda x: (2 * x * x * x + 5) / (3 * x * x - 2)

    for i in range(0, i):
        x = fx(x)
        print(i + 1, ": x =", x)

# punkt startowy
startx = 1

# iteracje
iter = 10

newton_method(startx, iter)
```

b) Funkcję przekształcam następująco:

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

Podobnie jak poprzednio najpierw sprawdzamy wykres badanej funkcji:




Widać, że szukane miejsce zerowe mieści się w przedziale [0,1].

Sprawdzamy czy faktycznie tak jest ze wzoru: $f(a) \cdot f(b) < 0$

$$f(0) \cdot f(1) = (e^{-0} - 0) \cdot (e^{-1} - 1) = (1 - 0) \cdot (e^{-1} - 1) = e^{-1} - 1 < 0$$

A więc warunek jest spełniony.

Sprawdzimy jeszcze dla pewności krotność pierwiastka:

$$f'(x) = -e^{-x} - 1$$

$$f'(0) \cdot f'(1) = (-e^{-0} - 1) \cdot (-e^{-1} - 1) = -2 \cdot (-e^{-1} - 1) = 2e^{-1} + 2 > 0$$

Dowodzi to, że nasza pierwsza pochodna jest stałego znaku, co potwierdza drugi warunek.

Z następnej kolejności zbadamy drugą pochodną:

$$f''(x) = e^{-x}$$

$$f''(0) \cdot f''(1) = (e^{-0}) \cdot (e^{-1}) = 1 \cdot e^{-1} = e^{-1} > 0$$

Co udowadnia, że druga pochodna również jest stałego znaku, a więc spełniony jest warunek trzeci.

Ostatecznie wzór Newtona będzie zbieżny na wybranym przedziale do miejsca zerowego.

Podstawiam funkcje do wzoru na iteracje Newtona:

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

Wyniki otrzymane w programie dla punktu startowego $x = 1$:

```
1 : x = 0.5378828427399902
2 : x = 0.5669869914054133
3 : x = 0.567143285989123
4 : x = 0.5671432904097838
5 : x = 0.5671432904097838
6 : x = 0.5671432904097838
7 : x = 0.5671432904097838
8 : x = 0.5671432904097838
9 : x = 0.5671432904097838
10 : x = 0.5671432904097838
```

Ostateczny wynik: x = 0.5671432904097838

Jest on zgodny z obserwacjami na wykresie.

Kod programu wykorzystany do narysowania wykresu:

```
import numpy as np
import matplotlib.pyplot as plt

fx = lambda x: np.exp(-x) - x

x = np.linspace(-4, 4, 100)

fig, ax = plt.subplots()
ax.plot(x, fx(x), label="e^x - x")
ax.set_xlabel("x")
ax.set_ylabel("y")
#ax.set_aspect(0.1)
ax.legend()
ax.grid()
plt.show()
```

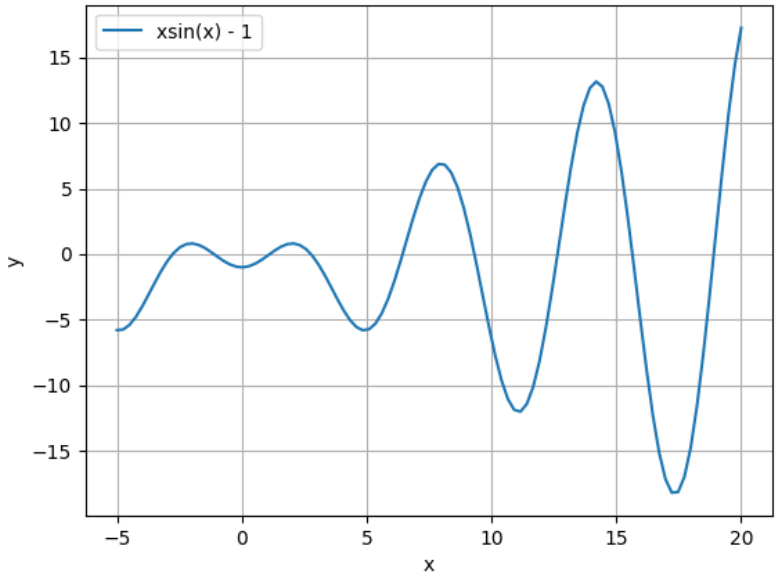
Kod programu wykorzystany do obliczeń:

```
import numpy as np

def newton_method(x, i):
    # badana funkcja
    fx = lambda x: x - ((np.exp(-x) - x) / (- np.exp(-x) - 1))

    for i in range(0, i):
        x = fx(x)
        print(i + 1, ": x =", x)

# punkt startowy
startx = 1

# iteracje
iter = 10

newton_method(startx, iter)
```

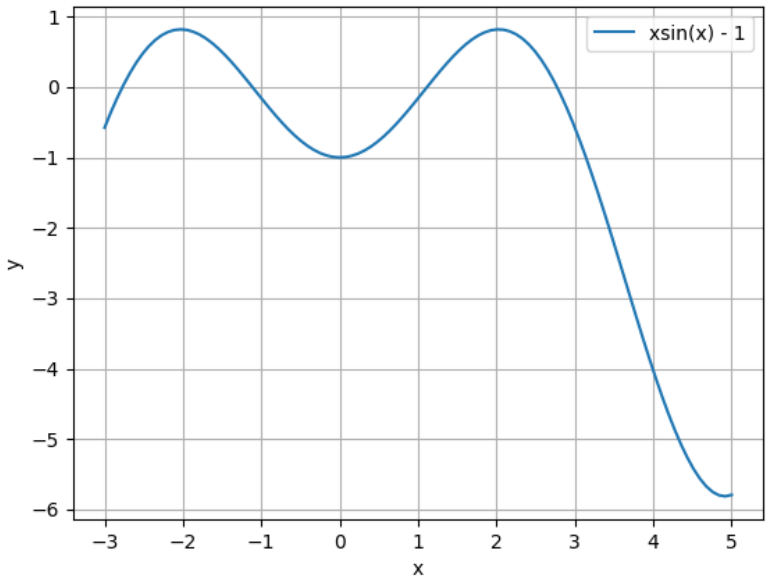
c) Funkcję przekształcam następująco:

$$f(x) = xsin(x) - 1$$

Podobnie jak poprzednio najpierw sprawdzamy wykres badanej funkcji:




Oraz w przybliżeniu:




Widać, że funkcja posiada nieskończenie wiele pierwiastków rozmieszczonych symetrycznie względem osi OY. Ja spróbuję wyznaczyć 2 pierwsze z nich po dodatniej stronie osi OX.

Pierwsze szukane miejsce zerowe mieści się w przedziale $[0, 2]$, a drugie $[2.5, 3]$.

Sprawdzamy czy faktycznie tak jest ze wzoru: $f(a) \cdot f(b) < 0$

$$f(0) \cdot f(2) = (0 \cdot sin(0) - 1)(2 \cdot sin(2) - 1) = -(2 \cdot sin(2) - 1) \approx -0.81859485 < 0$$

$$f(2.5) \cdot f(3) = (2.5 \cdot sin(2.5) - 1)(3 \cdot sin(3) - 1) \approx -0.286117 < 0$$

A więc warunek jest spełniony w obu przypadkach.

Sprawdzimy jeszcze dla pewności krotność pierwiastka:

$$f'(x) = \sin(x) + xcos(x)$$

$$f'(0) \cdot f'(2) = (\sin(0) + 0 \cdot cos(0)) \cdot (\sin(2) + cos(2)) = 0$$

$$f'(2.5) \cdot f'(3) = (\sin(2.5) + 2.5 \cdot cos(2.5)) \cdot (\sin(3) + 3 \cdot cos(3)) = 3.97281 > 0$$

Jak widać pierwsza pochodna jest stałego znaku tylko w drugim przedziale (co potwierdza drugi warunek), jednak spróbujemy przeprowadzić dalsze obliczenia dla obu przedziałów.

Z następnej kolejności zbadamy drugą pochodną:

$$f''(x) = 2cos(x) - x \cdot sin(x)$$

$$f''(0) \cdot f''(2) = (2cos(0) - 0 \cdot sin(0)) \cdot (2cos(2) - 2 \cdot sin(2)) = -5.301777 < 0$$

$$f''(2.5) \cdot f''(3) = (2cos(2.5) - 2.5 \cdot sin(2.5)) \cdot (2cos(3) - 3 \cdot sin(3)) = 7.44669 > 0$$

Ponownie tylko drugi z przedziałów spełnia warunek trzeci.

Ostatecznie wiemy, że wzór Newtona będzie zbieżny tylko na drugim przedziale do miejsca zerowego. Mimo to spróbujemy wykorzystać wzór dla obu z nich.

Podstawiam funkcje do wzoru na iteracje Newtona:

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

Wyniki otrzymane w programie dla punktu startowego $x = 3$:

```
1 : x = 2.7961579968088692
2 : x = 2.772948484224365
3 : x = 2.772604784310486
4 : x = 2.772604708265995
5 : x = 2.772604708265991
6 : x = 2.772604708265991
7 : x = 2.772604708265991
8 : x = 2.772604708265991
9 : x = 2.772604708265991
10 : x = 2.772604708265991
```

Wyniki otrzymane w programie dla punktu startowego $x = 2$:

```
1 : x = -8.63058375708243
2 : x = -9.5969709137395
3 : x = -9.322270122372009
4 : x = -9.317247017715122
5 : x = -9.317242941417522
6 : x = -9.31724294141481
7 : x = -9.31724294141481
8 : x = -9.31724294141481
9 : x = -9.31724294141481
10 : x = -9.31724294141481
```

Wnioski:

- Dla $x = 3$ otrzymaliśmy zadowalający nas wynik $x = 2.772604708265991$, zgodny z obserwacjami na wykresie.
- Dla $x = 2$ otrzymany wynik $x = -9.31724294141481$ nie jest zadowalający i nie mieści się w badanym przedziale $[0, 2]$. Co potwierdza, że niespełnione warunki dla iteracji metodą Newtona dają nieoczekiwane wyniki.
- Co ciekawe, dla $x = 1$ wynik wynosi $x = 1.1141571408719302$, czyli wynik zgodny z naszymi obserwacjami. Najprawdopodobniej powinniśmy ograniczyć wybrany przedział, aby otrzymać poprawny wynik.

Kod programu wykorzystany do narysowania wykresu:

```
import numpy as np
import matplotlib.pyplot as plt

fx = lambda x: x * np.sin(x) - 1

x = np.linspace(-5, 20, 100)

fig, ax = plt.subplots()
ax.plot(x, fx(x), label="xsin(x) - 1")
ax.set_xlabel("x")
ax.set_ylabel("y")
#ax.set_aspect(0.1)
ax.legend()
ax.grid()
plt.show()


x = np.linspace(-3, 5, 100)

fig, ax = plt.subplots()
ax.plot(x, fx(x), label="xsin(x) - 1")
ax.set_xlabel("x")
ax.set_ylabel("y")
#ax.set_aspect(0.1)
ax.legend()
ax.grid()
plt.show()
```

Kod programu wykorzystany do obliczeń:

```
import numpy as np

def newton_method(x, i):
    # badana funkcja
    fx = lambda x: x - ((x * np.sin(x) - 1) / (x * np.cos(x) + np.sin(x)))

    for i in range(0, i):
        x = fx(x)
        print(i + 1, ": x =", x)

# punkt startowy
startx = 2
#startx = 3

# iteracje
iter = 10

newton_method(startx, iter)
```

**Zadanie 3.**

Napisz schemat iteracji wg metody Newtona dla następującego układu równań nieliniowych.

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

Przekształcam układ równań, aby po lewej stronie równań mieć zera:

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

Stąd otrzymuję dwie funkcje:

$$f_1(x) = x_1^2 + x_2^2 - 1$$

$$f_2(x) = x_1^2 - x_2$$

Wzór wykorzystywany przy metodzie Newtona:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

Przy założeniu, że $f(x) = 0$:

$$0 = f(x_0) + f'(x_0)(x - x_0)$$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

W przypadku wielowymiarowej funkcji Newtona, przy założeniu, że $F(x) = 0$, korzystamy z analogicznego wzoru:

$$x = x_0 - F'(x_0)^{-1} F(x_0)$$

gdzie:

$$F(x) = \begin{bmatrix} f_1(x_1, x_2 \dots x_n) \\ f_2(x_1, x_2 \dots x_n) \\ \vdots \\ f_n(x_1, x_2 \dots x_n) \end{bmatrix},$$

$$F'(x_0) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$ i jest macierzą odwracalną,

$$x = [x_1, x_2 \dots x_n].$$

Otrzymane rekurencyjne równanie:

$$x_k = x_{k-1} - F'(x_{k-1})^{-1} F(x_{k-1})$$

Dla naszego przykładu:

$$F(x) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} x_1^2 + x_2^2 - 1 \\ x_1^2 - x_2 \end{bmatrix}$$

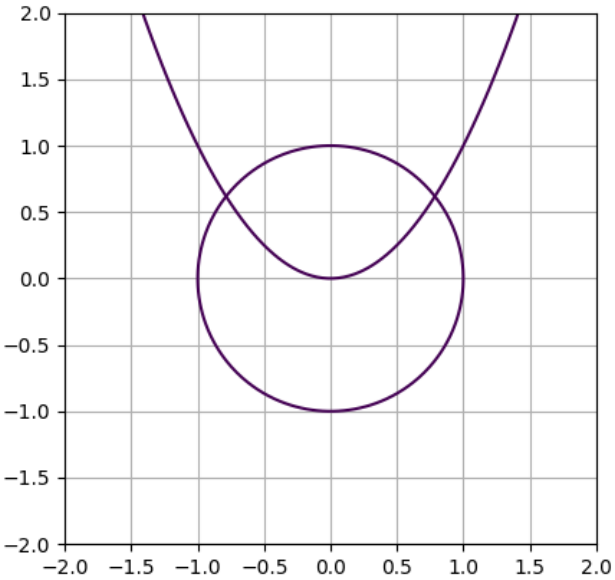
$$F'(x_0) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 & 2x_2 \\ 2x_1 & -1 \end{bmatrix}$$

$$x = [x_1, x_2]$$

Stąd równanie rekurencyjne:

$$x_k = x_{k-1} - \begin{bmatrix} 2x_1 & 2x_2 \\ 2x_1 & -1 \end{bmatrix}^{-1} \begin{bmatrix} x_1^2 + x_2^2 - 1 \\ x_1^2 - x_2 \end{bmatrix}$$

Przy użyciu programu w Pythonie rysujemy wykresy funkcji $f_1(x)$ $i$ $f_2(x)$:




Można zauważyć, że pierwiastki mieszczą się w przedziałach $[-1, 0.5] \times [0.5, 1]$ oraz $[0.5, 1] \times [0.5, 1]$. Jednocześnie są one symetryczne względem osi OY, więc wystarczy znaleźć jedno z nich.

Jako punkt startowy wybieramy (1,1). Po obliczeniach wyznaczone punkty:

```
1 : (x, y) = ( 0.8333333333333333 , 0.6666666666666667 )
2 : (x, y) = ( 0.7880952380952381 , 0.6190476190476191 )
3 : (x, y) = ( 0.7861540663060879 , 0.6180344478216818 )
4 : (x, y) = ( 0.7861513777620804 , 0.618033988749989 )
5 : (x, y) = ( 0.7861513777574233 , 0.6180339887498948 )
6 : (x, y) = ( 0.7861513777574233 , 0.6180339887498948 )
7 : (x, y) = ( 0.7861513777574233 , 0.6180339887498948 )
8 : (x, y) = ( 0.7861513777574233 , 0.6180339887498948 )
9 : (x, y) = ( 0.7861513777574233 , 0.6180339887498948 )
10 : (x, y) = ( 0.7861513777574233 , 0.6180339887498948 )
```

Widać, że po 5 iteracji wynik się już nie zmienia.

Ostateczne wyniki:

$$(x, y) = (0.7861513777574233, 0.6180339887498948)$$

$$(x, y) = (-0.7861513777574233, 0.6180339887498948)$$

Kod wykorzystany do wykonania wykresów:

```
import numpy as np
import matplotlib.pyplot as plt

xlin = np.linspace(-2, 2, 100)
ylin = np.linspace(-2, 2, 100)

x, y = np.meshgrid(xlin, ylin)

f1 = x ** 2 + y ** 2 - 1
f2 = x ** 2 - y

figure, axes = plt.subplots()

axes.contour(x, y, f1, [0], )
axes.contour(x, y, f2, [0])
axes.set_aspect(1)
axes.grid()

plt.show()
```

Kod programu do obliczeń:

```
import numpy as np
from numpy.linalg import inv

def newton_method(x, y, n):
    F = lambda x, y: np.array([[x*x + y*y - 1], [x*x - y]], dtype='float')
    dF = lambda x, y: np.array([[2*x, 2*y], [2*x, -1]], dtype='float')
    X = np.array([[x], [y]], dtype='float')

    for i in range(0, n):
        a, b = X[0][0], X[1][0]
        X = X - (inv(dF(a, b)) @ (F(a, b)))
        print(i + 1, ": (x, y) = (", X[0][0], ",", X[1][0], ")")

# punkt startowy
startx = 1
starty = 1

newton_method(startx, starty, 10)
```