Algorytmy numeryczne i przybliżone w języku Python

- Wprowadzenie
- Film samouczek
- Przeczytaj
- Sprawdź się
- Dla nauczyciela

Źródło: Aswin Anand, domena publiczna.

W tym e-materiale zajmiemy się tzw. metodami numerycznymi. Dzięki nim otrzymujemy przybliżone wyniki, a dokładność obliczeń – w zależności od potrzeb – możemy z góry określić.

Spróbujemy wyznaczyć miejsce zerowe funkcji ciągłej, korzystając z twierdzenia Bolzana-Cauchy'ego. Zdefiniujemy funkcję w języku Python, która wykona niezbędne obliczenia i zaprezentuje ich wynik w postaci graficznej. Więcej informacji na ten temat znajdziesz w e-materiale: Algorytmy numeryczne i przybliżone.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Algorytmy numeryczne i przybliżone w języku C++,
- Algorytmy numeryczne i przybliżone w języku Java.

Więcej zadań? Sięgnij do Algorytmy numeryczne i przybliżone – zadania maturalne.

Twoje cele

- Przeanalizujesz sposób obliczania wartości wyrażenia z wykorzystaniem funkcji `eval()`.

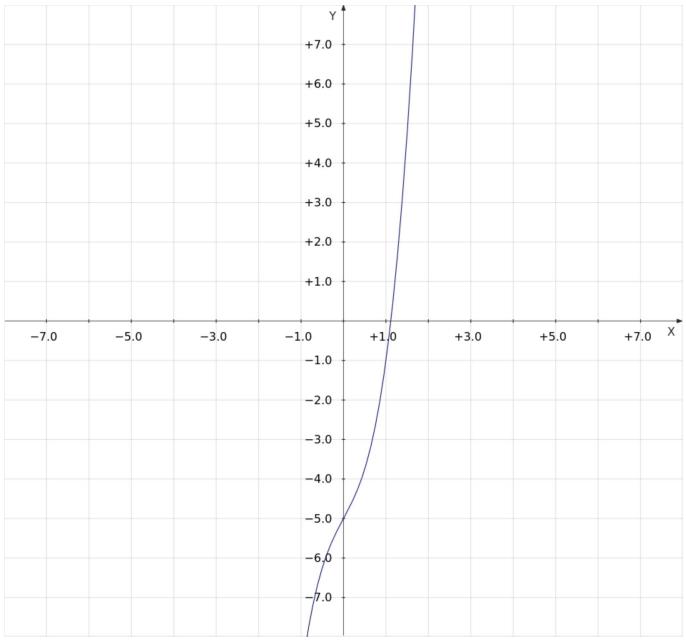
- Wyznaczysz przybliżoną wartość miejsca zerowego funkcji ciągłej w przedziale $\langle a, b \rangle$.
- Zdefiniujesz funkcję wizualizującą graficznie miejsce zerowe funkcji.

Film samouczek

Polecenie 1

Przeanalizuj prezentację objaśniającą kolejne kroki wyznaczania miejsca zerowego funkcji z zastosowaniem metody bisekcji.

1



$f_{(x)} = 2x^3 + 2x\ - 5$

Przeanalizujmy przykładową funkcję. Na wykresie możemy zaobserwować, że równość $f\,(x) = 0$ jest spełniona dla wartości x należącej do przedziału $[1, 2\ ]$. Funkcja w przedziale $[1, 2]$ ma miejsce zerowe. Jako przedział wejściowy przyjmiemy przedział $[-6, 8]$, dlatego używane zmienne przyjmą wartości `a = -6`, `b = 8`, a dokładność niech wynosi `delta = 0.001` i `epsilon = 0.001`.

Przedstawioną funkcję zdefiniujemy w języku Python jako:

```
1 def f(x):
```

```
    return (2 * x ** 3) + (2 * x) - 5
```

Zdefiniujmy funkcję obliczającą miejsce zerowe jako:

```
def metoda_bisekcji(a, b, delta, epsilon):
```

Teraz możemy rozpocząć badanie funkcji.

2

W programie będziemy się odwoływać do omawianej funkcji poprzez `f(x)`. Rozważamy przedział domknięty, zatem weryfikujemy, czy $f(x)$ spełnia warunek zakończenia dla krańców przedziału. Warunek zakończenia jest spełniony, gdy odległość funkcji od osi OX dla danego punktu jest mniejsza od dokładności `epsilon`.

```
def metoda_bisekcji(a, b, delta, epsilon):

    if abs(f(a)) < epsilon:
        return a
    if abs(f(b)) < epsilon:
        return b
```

3

Po zbadaniu wartości funkcji dla krańców przedziałów sprawdzamy, czy możemy kontynuować wykonywanie algorytmu. Zbadajmy, czy wartości funkcji na końcach przedziału mają różne znaki:

```
def metoda_bisekcji(a, b, delta, epsilon):
```

```
2
3       if abs(f(a)) < epsilon:
4           return a
5       if abs(f(b)) < epsilon:
6           return b
7
8       if (f(a)) * (f(b)) > 0:
9           print("Wartości funkcji na krańcach przedziału mają ten sam znak")
10          print("Nie możemy zastosować twierdzenia Bolzana-Cauchy'ego")
11          return None
```

W przypadku niespełnienia tego warunku przerywamy działanie funkcji i zwracamy wartość `None`. Sprawdzamy, czy wartości `f(a)` i `f(b)` mają różne znaki. Jeżeli tak, możemy zastosować twierdzenie Bolzana-Cauchy'ego i kontynuować wykonywanie algorytmu. Twierdzenie to mówi, że jeśli funkcja jest ciągła w przedziale i przyjmuje dla krańców przedziału wartości o przeciwnych znakach, to w przedziale tym istnieje punkt, dla którego funkcja $f(x) = 0$. W przeciwnym wypadku wypisujemy odpowiedni komunikat oraz kończymy działanie programu.

4

Kolejnym krokiem będzie rozpoczęcie pętli `while`. W jej środku będziemy aktualizować zmienne oznaczające obecnie rozważany przedział. Będzie ona działać dopóty, dopóki wartość bezwzględna różnicy zmiennych `a` i `b` jest większa niż dokładność (`delta`):

```
def metoda_bisekcji(a, b, delta, epsilon):

    if abs(f(a)) < epsilon:
        return a
    if abs(f(b)) < epsilon:
        return b

    if (f(a)) * (f(b)) > 0:
        print("Wartości funkcji na krańcach przedziału mają ten sam znak")
        print("Nie możemy zastosować twierdzenia Bolzana-Cauchy'ego")
        return None

    while abs(b - a) > delta:
```

5

Następnie wyznaczymy punkt środkowy i zweryfikujemy drugi warunek przerwania pętli, czyli sprawdzimy, czy wartość funkcji w obliczonym punkcie środkowym jest mniejsza niż dokładność przybliżenia wartości funkcji zawarta w zmiennej `epsilon`:

```
def metoda_bisekcji(a, b, delta, epsilon):

    if abs(f(a)) < epsilon:
        return a
    if abs(f(b)) < epsilon:
        return b
```

```
7
8       if (f(a)) * (f(b)) > 0:
9           print("Wartości funkcji na krańcach przedziału mają ten sam znak")
10          print("Nie możemy skorzystać z twierdzenia Bolzana-Cauchy'ego")
11          return None
12
13      while abs(b - a) > delta:
14          x = (a + b) / 2
15          if abs(f(x)) < epsilon:
16              break
```

Jeżeli taka sytuacja ma miejsce, osiągnęliśmy satysfakcjonujący wynik i możemy przerwać wykonywanie funkcji.

6

Jeśli warunek nie został spełniony, kontynuujemy wykonywanie algorytmu poprzez sprawdzenie, czy iloczyn wartości funkcji $f\left(a\right)$ i $f\left(x\right)$ jest ujemny. To pozwoli ograniczyć przedział rozważany w następnej iteracji do przedziału spełniającego warunek twierdzenia Bolzana-Cauchy'ego. Jeżeli iloczyn jest ujemny, wartości funkcji w badanych punktach mają różne znaki, ustawiamy zatem wartość b, czyli prawy kraniec przedziału, na wartość punktu środkowego. Podobnie w przypadku iloczynu dodatniego, czyli tych samych znaków wartości funkcji. Zmieniamy wartość lewego krańca rozważanego przedziału na wartość środkową. Ograniczymy w ten sposób przedział rozważany w kolejnej iteracji do przedziału, w którym znajduje się miejsce zerowe:

```python
def metoda_bisekcji(a, b, delta, epsilon):

    if abs(f(a)) < epsilon:
        return a
    if abs(f(b)) < epsilon:
        return b

    if (f(a)) * (f(b)) > 0:
        print("Wartości funkcji na krańcach przedziału mają ten sam znak")
        print("Nie możemy skorzystać z twierdzenia Bolzana-Cauchy'ego")
        return None

    while abs(b - a) > delta:
        x = (a + b) / 2
        if abs(f(x)) < epsilon:
            print("break")
            break
        if f(a) * f(x) < 0:
            b = x
        else:
            a = x
```


Przeanalizujmy działanie algorytmu dla przykładowej funkcji podanej w pierwszym kroku.

Iloczyn wartości $f(a)$ oraz $f(x)$ był dodatni, więc wartość x przypisujemy do zmiennej a.

W ten sposób zawężamy zakres przeszukiwania do przedziału $[1, 8]$:

```
# a = 1
# b = 8
```

8

Po zmianie krańców przedziału algorytm rozpocznie kolejną iterację pętli od ponownego wyznaczenia punktu środkowego:

```
# x = (1 + 8) / 2
# x = 4.5
```

9

Sprawdzamy, czy iloczyn wartości funkcji $f(a)$ i $f(x)$ jest ujemny. W obecnym obiegu pętli wartość ta wynosi $-1*186.25 = -186.25$.

10

Obliczony iloczyn jest ujemny, więc wartość x przypisujemy do zmiennej b. W ten sposób zawężamy zakres przeszukiwania do przedziału $[1, 4.5]$:

```
# a = 1
# b = 4.5
```

oraz rozpoczynamy kolejny obieg pętli.

11

Wyznaczamy punkt środkowy:

```
# x = (1 + 4.5) / 2
```

```
# x = 2.75
```

12

Sprawdzamy, czy iloczyn wartości funkcji $f\left(a\right)$ i $f\left(x\right)$ jest ujemny. W obecnym obiegu pętli wartość ta wynosi $-1*42.09375 = -42.09375$.

13

Iloczyn był ujemny, więc do zmiennej b przypisujemy wartość x. W ten sposób zawężamy zakres przeszukiwania do przedziału $[1, 2.75]$:

```
# a = 1
# b = 2.75
```

14

Kroki te będą powtarzane do momentu, aż zostanie osiągnięty któryś z opisanych wcześniej warunków stopu algorytmu. Wartości x wyznaczane w kolejnych iteracjach będą następujące:

```
# x = 1.875
# x = 1.4375
# x = 1.21875
# x = 1.109375
# x = 1.1640625
# x = 1.13671875
# x = 1.123046875
# x = 1.1162109375
# x = 1.11279296875
# x = 1.114501953125
# x = 1.1153564453125
```

Przedziały wartości, w których szukamy miejsca zerowego, również były odpowiednio modyfikowane.

W ostatniej iteracji pętli otrzymujemy przedział $[1.114501953125, 1.1153564453125]$, dla którego:

```
# b - a = 1.1153564453125 - 1.114501953125 = 0.0008544921875
# 0.0008544921875 < 0.001
# 0.0008544921875 < delta
```

Oznacza to, że został spełniony jeden z warunków zakończenia wykonywania pętli i możemy zwrócić obliczoną wartość x jako obliczone z podaną dokładnością miejsce zerowe analizowanej funkcji $f(x) = 2x^3 + 2x - 5$ w przedziale $[-6, 8]$.

```
def f(x):
    return (2 * x ** 3) + (2 * x) - 5

def metoda_bisekcji(a, b, delta, epsilon):

    if abs(f(a)) < epsilon:
        return a
    if abs(f(b)) < epsilon:
        return b

    if (f(a)) * (f(b)) > 0:
        print("Wartości funkcji na krańcach przedziału mają ten sam znak")
```

```
13          print("Nie możemy
skorzystać z twierdzenia
Bolzana-Cauchy'ego")
14          return None
15
16      while abs(b - a) >
delta:
17          x = (a + b) / 2
18          if abs(f(x)) <
epsilon:
19              break
20          if f(a) * f(x) <
0:
21              b = x
22          else:
23              a = x
24      return x
```

Problem 1

Napisz program, który wyznaczy miejsca zerowe funkcji $f\left(x\right)$ metodą bisekcji w zadanym przedziale, korzystając z warunków końcowych przedstawionych w prezentacji.

Przetestuj działanie programu dla funkcji $f\left(x\right) = 5\mathrm{x}^3 + 2\mathrm{x}^2 + 3x + 4$, przedziału $[-3, 4]$, przy współczynniku $\mathrm{epsilon}$ równym 0.001 i współczynniku delta równym 0.001.

Specyfikacja problemu:

Dane:

- `f(x)` – funkcja rzeczywista zmiennej rzeczywistej
- `a` – lewy kraniec przedziału
- `b` – prawy koniec przedziału
- `delta` – dokładność przybliżenia miejsca zerowego
- `epsilon` – dokładność przybliżenia wartości funkcji dla miejsca zerowego

Wynik:

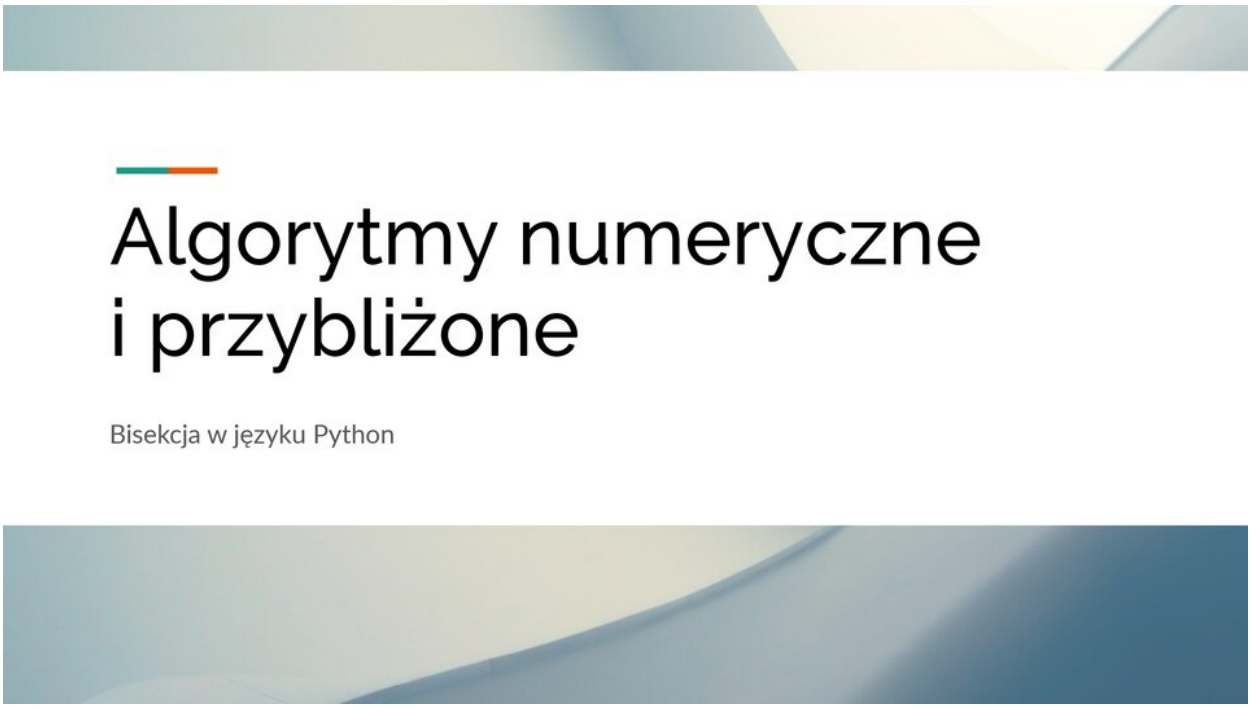
Program wyświetla miejsca zerowe funkcji lub komunikat o braku miejsc zerowych.

```
# Tutaj dodaj własny kod.
# Do wypisania wyniku użyj funkcji print()
```

```
1
```

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.



Film dostępny pod adresem /preview/resource/R16MBwYJpI30p

Film nawiązujący do algorytmów numerycznych i przybliżonych.

Plik o rozmiarze 937.00 B w języku polskim

Przeczytaj

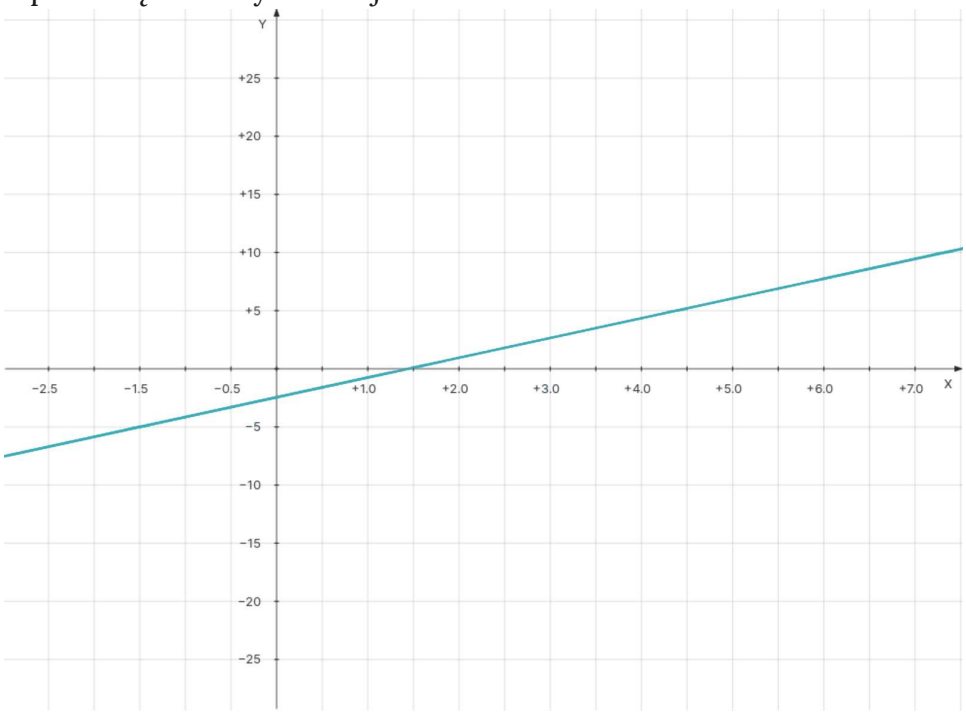
Podsumowanie

Ważne!

Metodę bisekcji możemy stosować dla dowolnej funkcji, o ile jest ona ciągła w badanym przedziale $[a, b]$.

Przykład 1

Oto wykres przykładowej funkcji, której miejsce zerowe z przedziału $[-2.5, 7]$ można wyznaczyć za pomocą metody bisekcji:



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Zapoznajmy się z programem, który obliczy miejsce zerowe funkcji $f(x) = 1.7x - 2.45$ metodą bisekcji w przedziale $[-2, 3]$.

```
def f(x):
    return  (1.7 * x) - 2.45

def wyznaczenie_miejsca_zerowego(a, b, delta, epsilon):
    if f(a) * f(b) < 0:
        while abs(a - b) > delta:
            x = (a + b) / 2
            if abs(f(x)) <= epsilon:
                return x
            else:
```

```
                if f(a) * f(x) < 0:
                    b = x
                else:
                    a = x
        else:
            return x
    else:
        return None


# przykład wykonania
print(wyznaczenie_miejsca_zerowego(-2,3,0.000000001, 0.00000000
```

Wykonanie przedstawionego kodu z podanymi argumentami da następujący wynik:

```
1.4411764703691006
```

Przykład 2

Wyznaczmy miejsce zerowe wielomianu funkcji $f(x) = 2x^3 + 2x - 5$ w przedziale $[-10, 10]$:

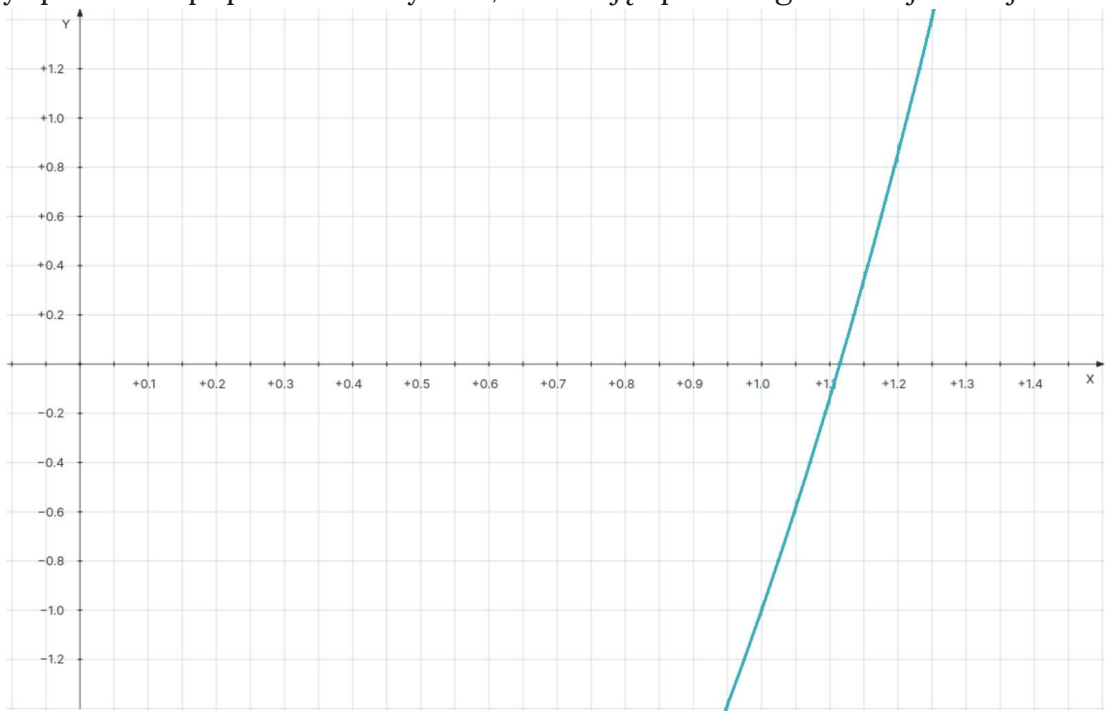
```
def f(x):
    return (2 * x ** 3) + (2 * x) - 5

def wyznaczenie_miejsca_zerowego(a, b, delta, epsilon):
    if f(a) * f(b) < 0:
        while abs(a - b) > delta:
            x = (a + b) / 2
            if abs(f(x)) <= epsilon:
                return x
            else:
                if f(a) * f(x) < 0:
                    b = x
                else:
                    a = x
        else:
            return x
    else:
        return None

```

```
# przykład wywołania
print(wyznaczenie_miejsca_zerowego(-10, 10, 0.00001, 0.00001))
```

Wykonanie przedstawionego kodu z podanymi argumentami da następujący wynik:

```
1.1147403717041016
```

Możemy sprawdzić poprawność wyniku, analizując przebieg badanej funkcji:



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Zmodyfikujmy wcześniejszy kod w taki sposób, aby sprawdzić, jakie wyniki otrzymamy dla coraz mniejszej wartości przybliżenia (czyli dla coraz większej precyzji wyznaczenia miejsca zerowego). W celu wyświetlenia wyniku, zastosujemy konwencję f-string:

```
def f(x):
    return (2 * x ** 3) + (2 * x) - 5

def wyznaczenie_miejsca_zerowego_print(a, b, delta, epsilon):
    if f(a) * f(b) < 0:
        while abs(a - b) > delta:
            x = (a + b) / 2
            if abs(f(x)) <= epsilon:
                return f"Znaleziono wartość x = {x:.15f}"
            else:
                if f(a) * f(x) < 0:
```

```
                    b = x
                else:
                    a = x
        else:
            return f"Ostatnia wartość x = {x:.15f}"
    else:
        return None

# przykład wykonania
print(wyznaczenie_miejsca_zerowego_print(-10, 10, 0.00001, 0.00
print(wyznaczenie_miejsca_zerowego_print(-10, 10, 0.0000000001,
print(wyznaczenie_miejsca_zerowego_print(-10, 10, 0.00000000000
```

Przedstawiony kod po wykonaniu zwróci następujący komunikat:

```
Ostatnia wartość x = 1.114740371704102
Ostatnia wartość x = 1.114747109750169
Znaleziono wartość x = 1.114747109704517
```

Ważne!

Ograniczeniem przykładowego programu jest to, że funkcja `f(x)` została zdefiniowana statycznie. Nie możemy wykorzystywać zapisanego kodu w celu wyznaczania miejsc zerowych innych funkcji. Spróbujmy to zmienić. Użyjemy w tym celu funkcji `eval()`, która służy obliczaniu wartości wyrażenia podanego jej jako parametr tekstowy. Jeżeli w tym wyrażeniu znajdą się nazwy zmiennych, to funkcja `eval()` wykorzysta do wykonania podanego jej wyrażenia zapisane w nim zmienne, bez potrzeby przekazywania ich do funkcji.

Przykład 3

Napiszmy kod funkcji pobierającej jako argument inną funkcję (taką, której miejsce zerowe chcemy znaleźć).

```
def wyznaczenie_miejsca_zerowego_fun(fun, a, b, delta, epsilon)
    """fun należy podać z _x_ jako parametrem, będzie zamienian

    # tworzymy dwie wersje ciągów dla eval aby móc liczyć dla d
    fun_x = fun.replace("_x_", "x")
    fun_a = fun.replace("_x_", "a")
    fun_b = fun.replace("_x_", "b")

```

```
    if eval(fun_a) * eval(fun_b) < 0:
        while abs(a - b) > delta:
            x = (a + b) / 2
            if abs(eval(fun_x)) <= epsilon:
                return x
            else:
                if eval(fun_a) * eval(fun_x) < 0:
                    b = x
                else:
                    a = x
        else:
            return x
    else:
        return None

# przykładowe wykonanie funkcji
print(wyznaczenie_miejsca_zerowego_fun("(2*_x_**3) + (2*_x_) -
print(wyznaczenie_miejsca_zerowego_fun("(3*_x_**3) - (1.4*_x_)
```

Jako wynik działania przedstawionego kodu otrzymamy:

```
1.1147403717041016
-1.1638355255126953
```

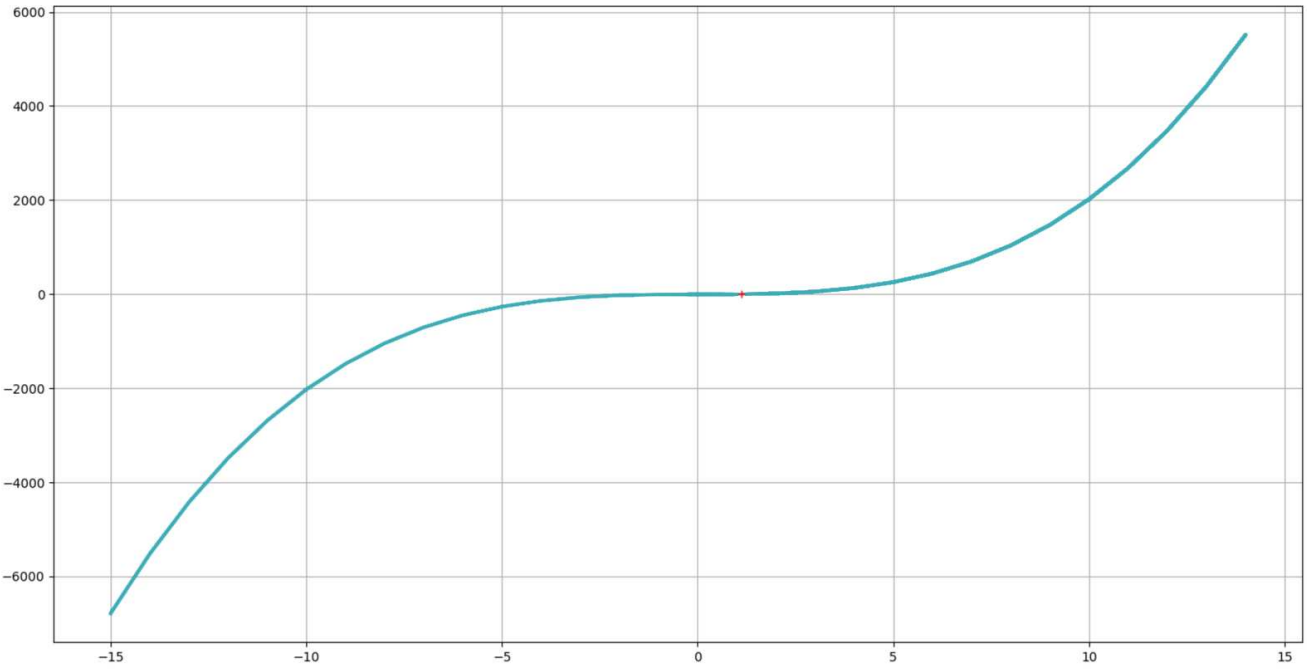
Przykład 4

Napiszmy program, który wyznaczy miejsce zerowe badanej funkcji z wykorzystaniem zaprezentowanego kodu oraz przedstawi rozwiązanie w postaci graficznej.

```
def stworz_wykres_fun(fun, x_min, x_max):
    import matplotlib.pyplot as plt

    fun_x =  fun.replace("_x_", "x")
    # tworzymy listę x
    ile_punktow = abs(x_min) + abs(x_max)
    X = []
    for punkt in range(x_min, x_min+ile_punktow):
        X.append( punkt )
    # tworzymy wartości funkcji y
    Y = [ eval(fun_x) for x in X ]
    # obliczamy punkt zerowy
```

```
    z = wyznaczenie_miejsca_zerowego_fun(fun, x_min, x_max, 0.0
    plt.plot(X,Y)
    # dodajemy do wykresu wartość x określająca miejsce zerowe
    # parametr 'r+' oznacza red (czerowny)
    plt.plot(z,0,'r+')
    plt.grid(True)
    plt.show()

# przykład wywołania
stworz_wykres_fun("(2*_x_**3) + (2*_x_) - 5", -15, 15)
```

Możemy w prosty sposób zwizualizować działanie programu – w podanym przykładzie miejsce zerowe oznaczone jest czerwonym znakiem +:



Już wiesz

Podsumujmy najważniejsze informacje:

- Miejsca zerowe funkcji ciągłych można wyznaczać metodą bisekcji.
- Moduł `matplotlib` można zastosować do przedstawiania wykresów funkcji.
- Funkcja `eval()` umożliwia obliczenie wartości wyrażeń zapisanych w postaci tekstowej.

Słownik

eval()

wbudowana funkcja języka Python, która może służyć do dynamicznego wyliczania wartości wyrażenia; parsuje ona podane jej wyrażenie do kodu w języku Python i zwraca rezultat wykonania tego kodu

f-string

sposób zapisywania zmiennych w łańcuchu znaków przeznaczonym do wyświetlenia za pomocą funkcji `print()`, dostępny począwszy od wersji 3.6 języka Python; polecenie wykorzystujące mechanizm `f-string` ma postać:
`f"Napis, a w nim {zmienna} do wypisania"`; dokładnie opisany w dokumencie PEP 498 – *Literal String Interpolation*

funkcja jako parametr

w języku Python istnieje możliwość przekazywania funkcji jako parametru innej funkcji; funkcja może też być zwracana po użyciu słowa kluczowego `return`

przybliżenie

wartość lub liczba niezupełnie dokładna, zaokrąglona

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program wyznaczający miejsce zerowe funkcji $f\,(x)$, wykorzystując algorytm bisekcji.

Przetestuj działanie programu dla funkcji $f(x) = \mathrm{x}^2 - 2$, przedziału $[0, 5]$, przy współczynniku $\mathrm{epsilon}$ równym $0,001$ i współczynniku delta równym $0,001$.

Specyfikacja problemu:

Dane:

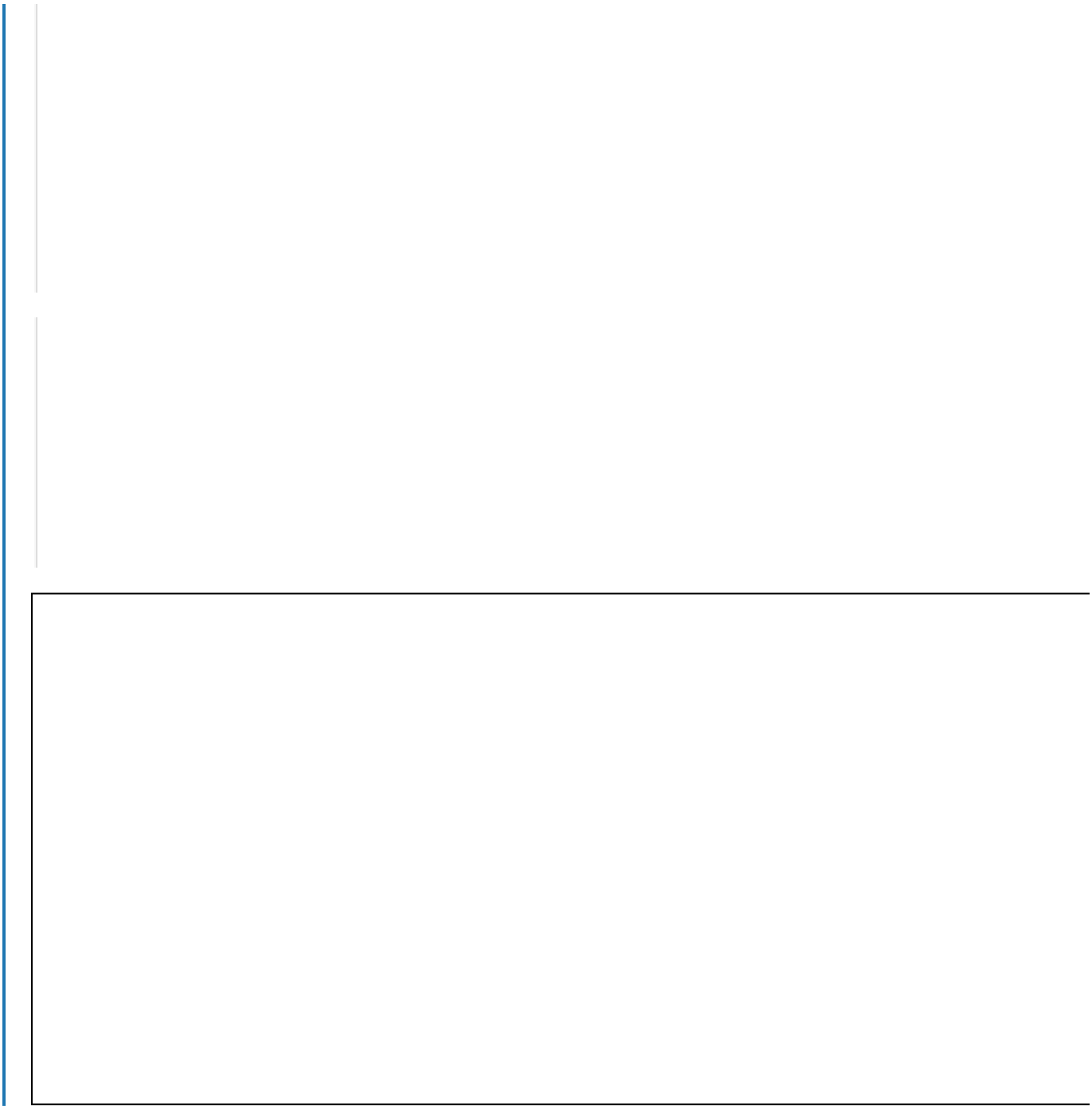
- `f(x)` – funkcja rzeczywista, której miejsce zerowe mamy obliczyć
- `a` – liczba rzeczywista; początek przedziału
- `b` – liczba rzeczywista; koniec przedziału
- `delta` – liczba rzeczywista, przybliżenie określające maksymalną długość przedziału
- `epsilon` – liczba rzeczywista; dokładność przybliżenia wartości funkcji w punkcie x_0

Wynik:

Program wyznacza i wypisuje wartość miejsca zerowego funkcji, zaokrąglone z dokładnością do trzech miejsc po przecinku.

Twoje zadania

1. Program na standardowe wyjście wypisuje miejsce zerowe podanej funkcji, zaokrąglone z dokładnością do trzech miejsc po przecinku.



Ćwiczenie 2

Napisz funkcję wyznaczającą miejsce zerowe funkcji $f(x)$ z wykorzystaniem metody bisekcji, a następnie zmodyfikuj ją tak, by zwracała krotkę w postaci `(c, w, k)`, gdzie `c` to miejsce zerowe funkcji, `w` to wyznaczona wartość funkcji w miejscu zerowym, a `k` to liczba iteracji wykonanych przez algorytm bisekcji (liczba pełnych obiegów pętli). Wykorzystaj warunki zakończenia algorytmu zaprezentowane w prezentacji.

Przetestuj działanie programu dla funkcji $f(x) = \mathrm{x}^2 - 2$, przedziału $[0,\ 5]$, przy współczynniku $\mathrm{epsilon}$ równym 10^{-6} i współczynniku delta równym 10^{-6}.

Specyfikacja problemu:

Dane:

- `f(x)` – funkcja rzeczywista, której miejsce zerowe mamy obliczyć
- `a` – liczba rzeczywista; początek przedziału
- `b` – liczba rzeczywista; koniec przedziału
- `delta` – liczba rzeczywista, przybliżenie określające maksymalną długość przedziału
- `epsilon` – liczba rzeczywista; dokładność przybliżenia wartości funkcji w punkcie x_0

Wynik:

Program wyznacza i wypisuje wartość miejsca zerowego funkcji, wartość funkcji w wyznaczonym miejscu zerowym i liczbę iteracji algorytmu w postaci krotki `(c, w ,k)`, gdzie `c` jest miejscem zerowym funkcji, `w` jest wyznaczoną wartością funkcji w miejscu zerowym, a `k` jest liczbą iteracji wykonanych przez algorytm bisekcji. Wartości `c` i `w` mają być zaokrąglone z dokładnością do pięciu miejsc po przecinku.

Twoje zadania

1. Program na standardowe wyjście drukuje krotkę zawierającą miejsce zerowe funkcji $f(x) = \mathrm{x}^2 - 2$, wartość funkcji w wyznaczonym miejscu zerowym oraz liczbę iteracji

algorytmu bisekcji.

Ćwiczenie 3

Napisz funkcję wyznaczającą miejsce zerowe metodą bisekcji, a następnie wykorzystaj ją do wyznaczenia liczby π.

W programie zastosuj funkcję trygonometryczną sinus, przedział $[3, 3.5]$. Wartości współczynników delta i $\mathrm{epsilon}$ dobierz tak, by zwracana wartość mogła być poprawnie zaokrąglona z dokładnością do dwóch miejsc po przecinku.

Specyfikacja problemu:

Dane:

- `f(x)` – funkcja trygonometryczna, którą wykorzystamy do obliczenia wartości liczby π.
- `a` – liczba rzeczywista; początek przedziału
- `b` – liczba rzeczywista; koniec przedziału
- `delta` – liczba rzeczywista, przybliżenie określające maksymalną długość przedziału, do samodzielnego dobrania
- `epsilon` – liczba rzeczywista; dokładność przybliżenia wartości funkcji w punkcie x_0, do samodzielnego dobrania

Wynik:

Program wyznacza i wypisuje wartość liczby π zaokrągloną z dokładnością do dwóch miejsc po przecinku.

Twoje zadania

1. Program drukuje na standardowe wyjście obliczoną wartość liczby pi, zaokrągloną z dokładnością do dwóch miejsc po przecinku.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Algorytmy numeryczne i przybliżone w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

f) wyznaczania miejsc zerowych funkcji metodą połowienia,

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz sposób obliczania wartości wyrażenia z wykorzystaniem funkcji `eval()`.

- Wyznaczysz przybliżoną wartość miejsca zerowego funkcji ciągłej w przedziale $\langle a, b \rangle$.
- Zdefiniujesz funkcję wizualizującą graficznie miejsce zerowe funkcji.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Algorytmy numeryczne i przybliżone w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Wyświetlenie przez nauczyciela tematu i celów zajęć, przejście do wspólnego ustalenia kryteriów sukcesu.

Faza realizacyjna:

1. **Praca z tekstem.** Jeżeli przygotowanie uczniów do lekcji jest niewystarczające, nauczyciel prosi o indywidualne zapoznanie się z treścią zawartą w sekcji „Przeczytaj". Każdy uczestnik zajęć, podczas cichego czytania, wynotowuje najważniejsze kwestie poruszane w tekście.

2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie wspólnie analizują prezentację multimedialną, przedstawiającą sposób działania algorytmu bisekcji. Następnie próbują dokonać implementacji tego algorytmu w języku Python, rozwiązując Problem 1, po czym porównują swoje rozwiązanie z filmem
3. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że będą rozwiązywać ćwiczenia nr 1–2 z sekcji „Sprawdź się". Uczniowie wykonują je w parach. Po ustalonym czasie następuje porównanie napisanych kodów, podczas wspólnego omówienia rozwiązań.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania w języku Python.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 3 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.