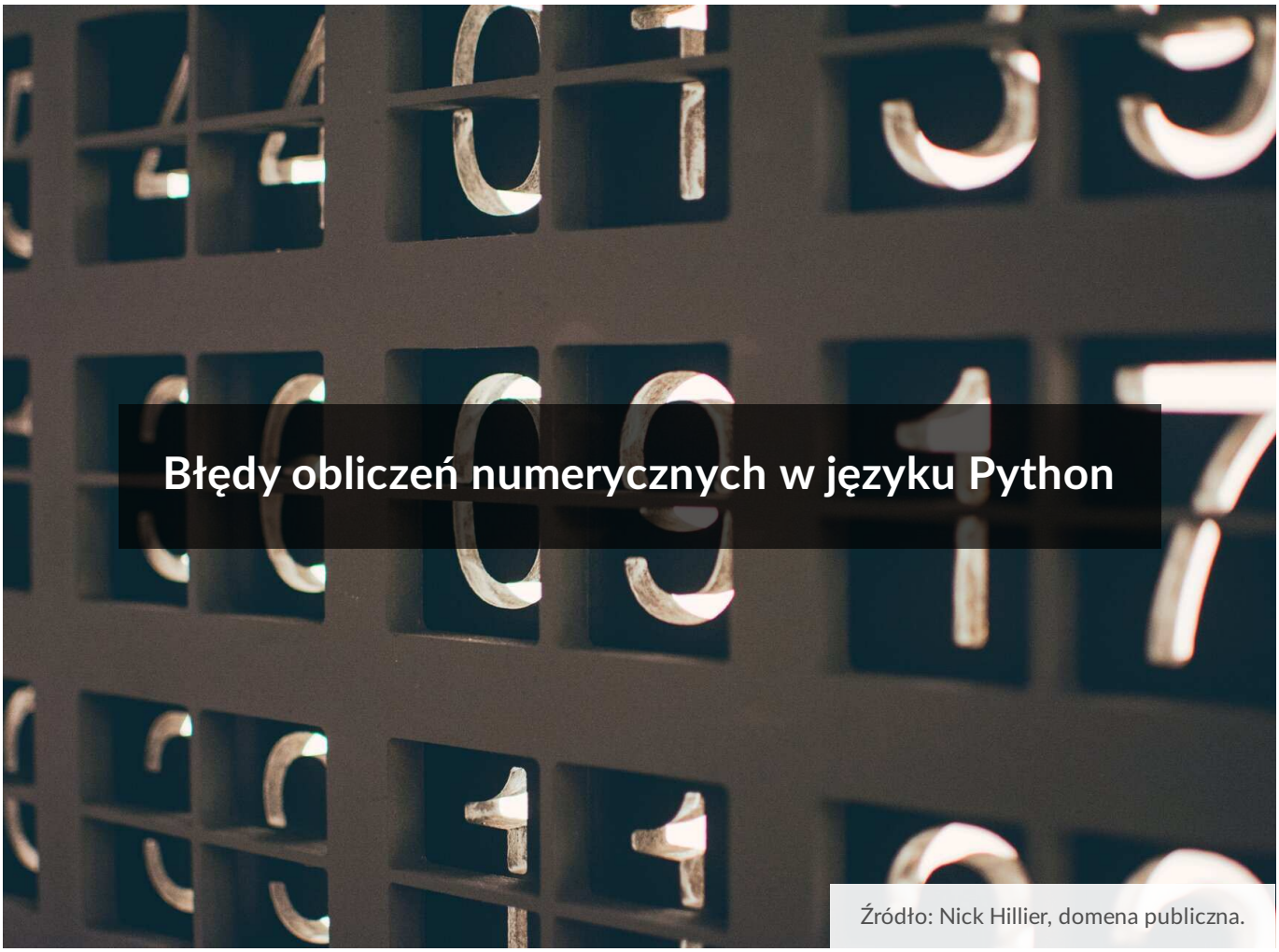
Błędy obliczeń numerycznych w języku Python



Źródło: Nick Hillier, domena publiczna.

W e-materiale Błędy obliczeń numerycznych poznaliśmy najważniejsze informacje dotyczące błędów obliczeń.

W tym e-materiale przeanalizujemy błąd względny jako kryterium zakończenia procedury iteracyjnej w języku Python.

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

- Błędy obliczeń numerycznych w języku C++,
- Błędy obliczeń numerycznych w języku Java.

Więcej zadań? Przejdź do Błędy obliczeń numerycznych – zadania maturalne.

> **Twoje cele**
>
> - Napiszesz w języku Python program obliczający wartość liczby pi.
> - Powtórzysz wiadomości dotyczące błędów względnych i bezwzględnych oraz ich obliczania.
> - Wykonasz ćwiczenia sprawdzające twoją wiedzę dotyczącą błędów obliczeń.

Przeczytaj

Wiemy już, że istnieją błędy przybliżeń, wśród których wyróżniamy błąd bezwzględny oraz błąd względny. W tym e-materiale użyjemy języka Python do obliczenia i wizualizacji tych błędów.

Przykład 1

Znając definicje błędów bezwzględnego oraz względnego, możemy napisać funkcje obliczające odpowiednie wartości. Przetestujemy je dla wartości zmierzonych: 225 i 250 oraz wartości dokładnej 242.

```
from math import fabs

def blad_bezwzgledny(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""
    return fabs(x-x0)

def blad_wzgledny(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""

    d = blad_bezwzgledny(x,x0)
    return d / x

def blad_wzgledny_proc(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""

    d = blad_bezwzgledny(x,x0)
    return round((d / x) * 100, 2)

# przykładowe wywołanie
print("Wartość błędu przybliżenia dla 250:", blad_bezwzgledny(2
print("Wartość błędu przybliżenia dla 225:", blad_bezwzgledny(2

print("Wartość błędu względnego dla 250:", blad_wzgledny(242, 2
print("Wartość błędu względnego dla 225:", blad_wzgledny(242, 2

```

```
print("Wartość błędu względnego procentowo dla 250:", blad_wzgl
print("Wartość błędu względnego procentowo dla 225:", blad_wzgl
```

Wynikiem wykonania kodu dla wartości zmierzonych 225 i 250 oraz wartości dokładnej 242 jest:

```
Wartość błędu przybliżenia dla 250: 8.0
Wartość błędu przybliżenia dla 225: 17.0
Wartość błędu względnego dla 250: 0.03305785123966942
Wartość błędu względnego dla 225: 0.07024793388429752
Wartość błędu względnego procentowo dla 250: 3.31
Wartość błędu względnego procentowo dla 225: 7.02
```

Przeanalizujmy teraz sytuację, w której symulujemy „ręczne" obliczenie wartości za pomocą zaokrąglenia. Prześledźmy błąd bezwzględny oraz względny dla kolejnych zaokrągleń liczby pi.

$$\pi \approx 3,14159265358979323846264338327950288419716 9\ldots$$

W przykładzie mamy podaną wartość liczby pi z dokładnością do 42 miejsc po przecinku. W języku Python w module `math` wartość liczby pi jest zapisana jako `math.pi` z dostępną precyzją – w tym przypadku `3.141592653589793`. Będziemy obliczali błąd bezwzględny i względny oraz błąd względny wyrażony w procentach dla kolejnych przybliżeń: od dwóch miejsc po przecinku, czyli wartości `3.14`, do 15 miejsc po przecinku, a więc dokładnej wartości, kiedy błąd będzie wynosił 0. Sprawdzimy kilka obliczonych wartości.

`Wartość dokładna to 3.141592653589793`

Wartość zmierzona	Błąd względny	Błąd względny w procentach
`3.14`	0.0005069573828972128	0.05069573828972128
`3.141592`	2.0804409260601893e-07	2.0804409260601892e-05
`3.14159265358`	3.1172263038059322e-12	3.117226303805932e-10
`3.141592653589793`	0.0	0.0

Przykład 2

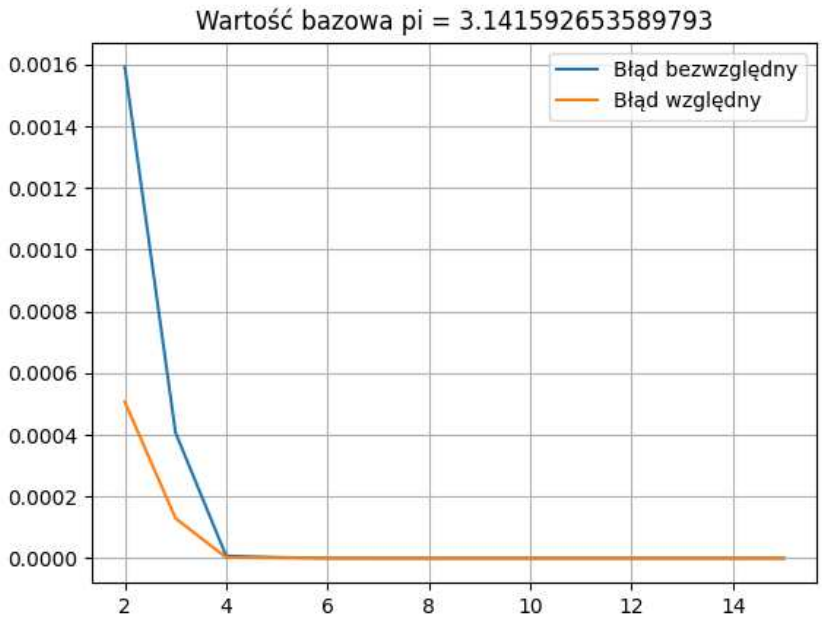
Napiszmy kod, który pozwoli zobrazować błąd w zależności od kolejnych zaokrągleń. Przygotujemy dwa wykresy w celu odpowiedniego doboru skali.

> Bibliotekę `matplotlib` omawialiśmy również w e-materiale Modelowanie ruchów Browna w języku Python.

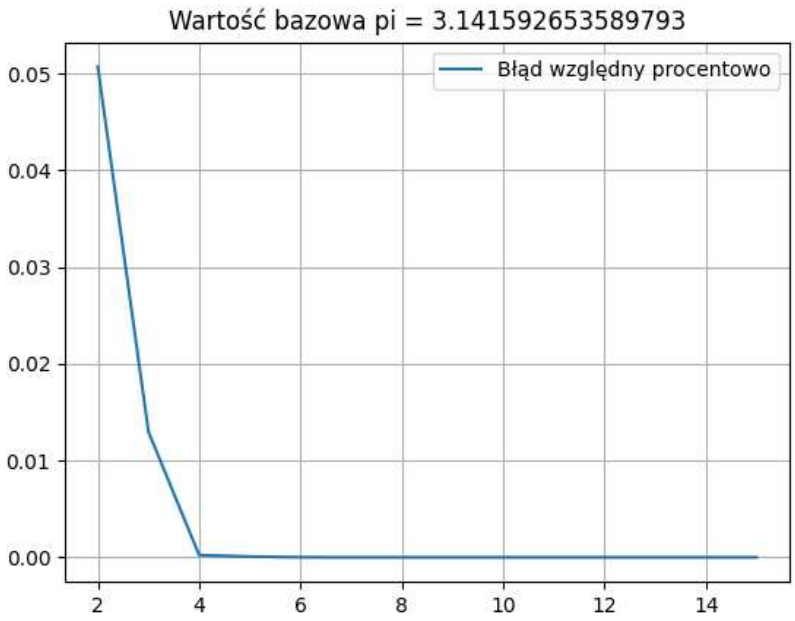
```python
# importujemy niezbędne funkcje i moduły
from math import fabs
import matplotlib.pyplot as plt

# definiujemy funkcje pomocnicze
def blad_bezwzgledny(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""
    return fabs(x - x0)

def blad_wzgledny(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""

    d = blad_bezwzgledny(x,x0)
    return d / x

def blad_wzgledny_proc_dokladn(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""

    d = blad_bezwzgledny(x,x0)
    return (d / x) * 100  # tutaj zwaracamy dokładną wartość, b


# wartości niezbędne dla obliczeń
pi = 3.141592653589793
wartosci = [x for x in range(2, 16)]
bl_bezw = []
bl_wzgl = []
bl_wzgp = []

# w pętli sprawdzamy coraz dokładniejsze przybliżenie wartości
# używamy funkcji round() - która zaokrągla wynik, dając wartoś
for x in range(2, 16):
    base = round(pi, x)
    print("Obliczamy błąd przybliżenia dla wartości: ", base)
    bl_bezw.append(blad_bezwzgledny(pi, base))
    bl_wzgl.append(blad_wzgledny(pi, base))
    bl_wzgp.append(blad_wzgledny_proc_dokladn(pi, base))

```

```
# pierwszy wykres
plt.plot(wartosci, bl_bezw, label = "Błąd bezwzględny")
plt.plot(wartosci, bl_wzgl, label = "Błąd względny")
plt.legend()
plt.title("Wartość bazowa pi = " + str(pi))
plt.grid(True)
plt.show()
# drugi wykres
plt.plot(wartosci, bl_wzgp, label = "Błąd względny procentowo")
plt.legend()
plt.title("Wartość bazowa pi = " + str(pi))
plt.grid(True)
plt.show()
```

Wykres pierwszy – błąd względny oraz bezwzględny wartościowo:



Wykres drugi – błąd względny procentowo:



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

Ważne!

Możemy zauważyć, że błąd przybliżenia (zarówno względny, jak i bezwzględny) zmienia się – maleje wraz ze wzrostem dokładności obserwowanej liczby.

Przykład 3

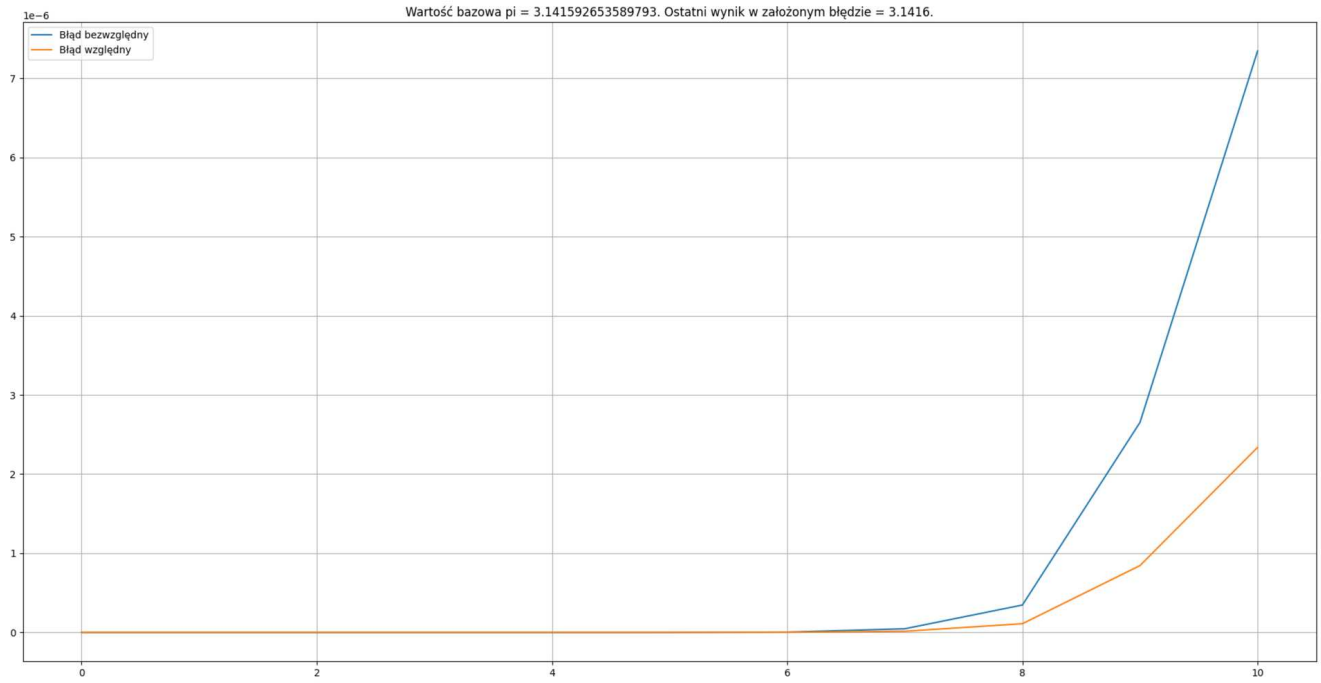
Napiszmy kod, który pozwoli zaokrąglić liczbę pi do takiego momentu, dla którego błąd względny przekroczy 0,01 procenta – wówczas zatrzymamy kolejne zaokrąglenia.

```
# importujemy niezbędne funkcje i moduły
from math import fabs
import matplotlib.pyplot as plt

# definiujemy funkcje pomocnicze
def blad_bezwzgledny(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""
    return fabs(x - x0)

def blad_wzgledny(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""

    d = blad_bezwzgledny(x, x0)
```

```
    return d / x

def blad_wzgledny_proc_dokladn(x, x0):
    """ x - wartość dokładna
        x0 - wartość zmierzona"""

    d = blad_bezwzgledny(x, x0)
    return (d / x) * 100  # tutaj zwaracamy dokładną wartość, b



pi = 3.141592653589793
bl_bezw = []
bl_wzgl = []
bl_wzgp = []
przyblizenia = []
ok = ""

for dokladnosc in range(15, 1, -1):
    wartosc = round(pi, dokladnosc)
    blad_wzgledny_v = blad_wzgledny(pi, wartosc)
    blad_bezwzgledny_v = blad_bezwzgledny(pi, wartosc)
    blad_proc = blad_wzgledny_proc_dokladn(pi, wartosc)
    if blad_proc > 0.01:
        break
    else:
        bl_bezw.append(blad_bezwzgledny_v)
        bl_wzgl.append(blad_wzgledny_v)
        bl_wzgp.append(blad_proc)
        ok = str(wartosc)
        przyblizenia.append(dokladnosc)

# przygotowujemy wartości X, bazując na długości listy przybliż
X = [ x for x in range(len(przyblizenia)) ]

# pierwszy wykres
plt.plot(X, bl_bezw, label = "Błąd bezwzględny")
plt.plot(X, bl_wzgl, label = "Błąd względny")
plt.legend()
plt.title("Wartość bazowa pi = " + str(pi) + ". Ostatni wynik w
plt.grid(True)
plt.show()
```

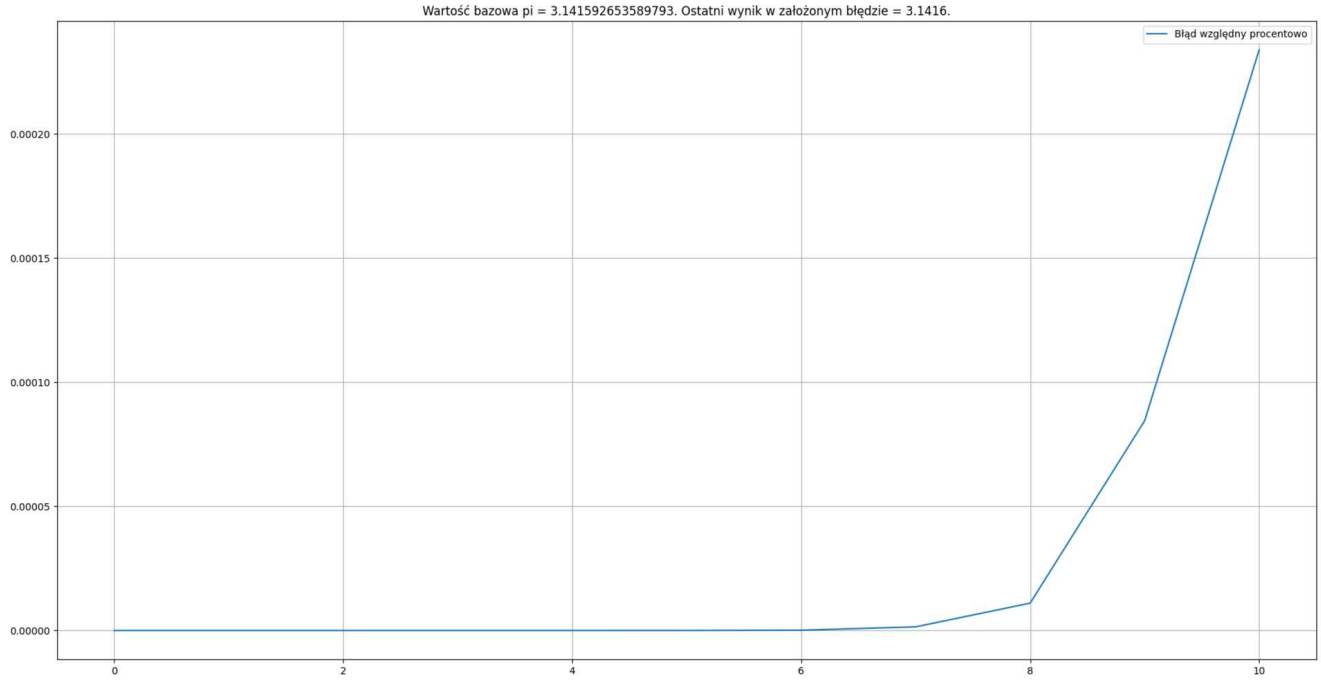
```
58 # drugi wykres
59 plt.plot(X, bl_wzgp, label = "Błąd względny procentowo")
60 plt.legend()
61 plt.title("Wartość bazowa pi = " + str(pi) + ". Ostatni wynik w
62 plt.grid(True)
63 plt.show()
```

Wykres pierwszy – błąd względny oraz bezwzględny wartościowo:



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

Wykres drugi – błąd względny procentowo:



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

Dla zainteresowanych

W języku Python istnieje moduł o nazwie `Decimal`. Zapewnia on obsługę arytmetyki zmiennoprzecinkowej i podaje dokładniejsze wyniki. Prześledźmy to na przykładzie dodawania do siebie dwóch wartości rzeczywistych: `0.1 + 0.2`. Z lekcji matematyki wiemy, że wynik powinien wynosić `0.3`, a jednak program w języku Python wyświetla następującą wartość:

```
print(0.1+0.2)
# 0.30000000000000004
```

Wynik jest większy niż `0.3` - niewiele, ale jednak większy. Jest to związane ze sposobem, w jaki język Python przechowuje w pamięci liczby. Zobaczymy, jak wygląda ta sama sytuacja, kiedy użyjemy modułu `Decimal`. Pamiętajmy, że liczby podajemy jako parametr typu `str`.

```
a = Decimal("0.1")
b = Decimal("0.2")
c = a + b
print(c)
# 0.3
```

Zauważmy, że wartości podane po wykorzystaniu modułu `Decimal` są dokładniejsze.

$$3 \cdot a - b = c$$

$$3 \cdot 0.1 - 0.3 = 0$$

```
a = 0.1
b = 0.3
c = 3 * a -b

if c == 0:
    print("Wynik poprawny = 0.")
else:
    print("Wynik niepoprawny; wartość = ",c)

# Wynik niepoprawny; wartość = 5.551115123125783e-17
```

```python
from decimal import *
x = Decimal("0.1")
y = Decimal("0.3")
z = 3 * x - y

if z == 0:
    print("Wynik poprawny = 0.")
else:
    print("Wynik niepoprawny; wartość = ",z)

# Wynik poprawny = 0.
```

Dla komputera obliczenie `3 * 0.1 - 0.3` daje wynik `5.551115123125783e-17`. Jest on bardzo bliski zeru, jednak nie jest to dokładnie zero. Natomiast zastosowanie do tego obliczenia modułu i typu danych `Decimal` daje oczekiwany wynik.

Słownik

błąd bezwzględny

wyraża bezwzględną różnicę pomiędzy wartością zmierzoną a dokładną; obliczamy go zgodnie ze wzorem:

$$\Delta x = |x - x_0|$$

gdzie x to wartość dokładna, x_0 – wartość zmierzona, a Δx – błąd bezwzględny

błąd względny

informuje, o ile procent różni się wartość zmierzona od dokładnej; obliczamy go zgodnie ze wzorem:

$$\delta = \frac{|x - x_0|}{x} \cdot 100\% = \frac{\Delta x}{x} \cdot 100\%$$

gdzie x to wartość dokładna, x_0 – wartość zmierzona, Δx – błąd bezwzględny, zaś δ to błąd względny

Film samouczek

Polecenie 1

Napisz program w języku Python, w którym wykorzystasz pętlę warunkową `while` wykonującą obliczenia przybliżenia liczby dziesiętnej. Przetestuj jego działanie dla stałej $e = 2.718281828459045$ i błędu względnego, którego procent graniczny wynosi $0,00005$ jako kryterium zakończenia procedury.

Polecenie 2

Porównaj swoje rozwiązanie z prezentacją omawiającą kolejne kroki powstawania programu, który wykorzystuje pętlę `while` do obliczenia przybliżenia liczby dziesiętnej.

Polecenie 3

Napisz program, który metodą siecznych wyznaczy pierwiastek x_0 dla zadanej funkcji. Przyjmij, że pierwiastek wyznaczany jest dla przedziału z początkiem w punkcie `punkt_startowy` oraz z końcem w punkcie `punkt_koncowy`, a także kolejnych, coraz mniejszych przedziałów, aż do spełnienia warunków:

- | `punkt_startowy` – `punkt_koncowy` | ≤ `epsilon`
- | $f(x_0)$ | < `epsilon0`

gdzie zmienne `epsilon` i `epsilon0` określają dokładność wyznaczania pierwiastka.

Za początek każdego nowego przedziału (przechowywanego w zmiennej `punkt_startowy`) przyjmujemy punkt końcowy poprzedniego przedziału, a za koniec (przechowywany w zmiennej `punkt_koncowy`) ostatni wyznaczony pierwiastek funkcji.

Swój program przetestuj dla funkcji $f(x) = -x^2 + x + 2$, wartości zmiennych `epsilon` oraz `epsilon0` równych odpowiednio 0,01 i 0,0001, a także przedziału liczbowego $[\ -1,25;\ \ -0,9\ ]$.

Specyfikacja problemu:

Dane:

- `punkt_startowy` – początek badanego przedziału; liczba rzeczywista
- `punkt_koncowy` – koniec badanego przedziału; liczba rzeczywista
- `epsilon` – dokładność wyznaczania pierwiastka; liczba rzeczywista
- `epsilon0` – dokładność porównania wartości funkcji dla wytypowanego pierwiastka z zerem; liczba rzeczywista

Wynik:

Program na standardowym wyjściu zwraca wyznaczony metodą siecznych pierwiastek x_0 zadanej funkcji.

Polecenie 4

Porównaj swoje rozwiązanie z filmem przedstawiającym implementację algorytmu wyznaczania pierwiastka za pomocą metody siecznych w języku Python.

Trwa wczytywanie danych ..

Film dostępny pod adresem /preview/resource/R11hnbd2lnFBw

Film nawiązujący do treści lekcji dotyczący implementacji algorytmu wyznaczania pierwiastka przy pomocy metody siecznych w języku Python.

Plik o rozmiarze 673.00 B w języku polskim

Polecenie 5

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program zaokrąglający podaną liczbę x tak, aby błąd względny nie przekroczył ustalonej wartości krytycznej `blad_krytyczny`.

Przetestuj działanie programu, zaokrąglając liczbę 0,054256, dopóki błąd względny nie przekroczy wartości 5%.

Specyfikacja problemu:

Dane:

- x – zaokrąglana wartość; liczba rzeczywista z przedziału [0, 1]
- `blad_krytyczny` – wartość krytyczna; liczba rzeczywista z przedziału [0, 100]

Wynik:

- wartość zaokrąglenia liczby

Ćwiczenie 2

Korzystając ze zdefiniowanej w programie stałej `PIERWIASTEK_Z_2`, sprawdź, z dokładnością do ilu miejsc po przecinku należy wypisać jej przybliżenie, aby błąd względny między wartością pierwotną a przybliżoną wynosił mniej niż 0,01%. Wypisz wyznaczoną liczbę cyfr.

Przykład:

Błąd względny między przybliżeniem pierwiastka z liczby 2 do dwóch cyfr po przecinku a wartością 1,41421356237 wynosi:

$$\frac{1,41421356237 - 1,41}{1,41421356237} \cdot 100\% = 0,29794385247\%$$

Potrzeba zatem przybliżenia do dwóch cyfr po przecinku, aby błąd względny wyniósł mniej niż 0,3%.

Ćwiczenie 3

Liczba Eulera może być zdefiniowana przez sumę następującego szeregu:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \dots$$

Sprawdź, dla jakiego n błąd względny wyznaczonego przybliżenia liczby e będzie mniejszy niż 0,0000005. Jako wartość dokładną przyjmij stałą matematyczną `math.e`. Wypisz minimalną wartość n.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Błędy obliczeń numerycznych 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. Rozumienie, analizowanie i rozwiązywanie problemów.

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

7) wyjaśnia, jakie może być źródło błędów pojawiających się w obliczeniach komputerowych: błąd zaokrąglenia, błąd przybliżenia;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

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

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

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):

- Napiszesz w języku Python program obliczający wartość liczby pi.
- Powtórzysz wiadomości dotyczące błędów względnych i bezwzględnych oraz ich obliczania.
- Wykonasz ćwiczenia sprawdzające twoją wiedzę dotyczącą błędów obliczeń.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

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

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 „Błędy obliczeń numerycznych w języku Python". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj" dotyczącymi programowania.
2. Uczniowie przypominają sobie najważniejsze informacje związane z błędami obliczeń numerycznych.

Faza wstępna:

1. Nauczyciel sprawdza przygotowanie uczniów do lekcji.
2. Nauczyciel wprowadza uczniów szczegółowo w temat lekcji i jej cele. Może posłużyć się wyświetloną na tablicy zawartością sekcji „Wprowadzenie".
3. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel prosi uczniów, aby metodą burzy mózgów wymienili najważniejsze informacje, które zdobyli, zapoznając się przed lekcją z sekcją „Przeczytaj". Wybrany uczeń wynotowuje je na tablicy. W razie konieczności nauczyciel dopowiada brakujące informacje. W kolejnym kroku uczniowie analizują przykłady z sekcji „Przeczytaj" i powtarzają zaprezentowane rozwiązania na swoim komputerze.
2. **Praca z multimedium** Uczniowie zapoznają się z poleceniem 1 w sekcji „Film samouczek". Następnie analizują prezentację.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1–3 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność rozwiązań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń z programowania w języku Python.

Praca domowa:

1. Uczniowie wykonują polecenie 3–5 z sekcji „Film samouczek" i porównują swoje rozwiązanie z filmem. Zapisują swoje wnioski.

Materiały pomocnicze:

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

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać multimedia w sekcji „Film samouczek" do przygotowania się do lekcji powtórkowej.