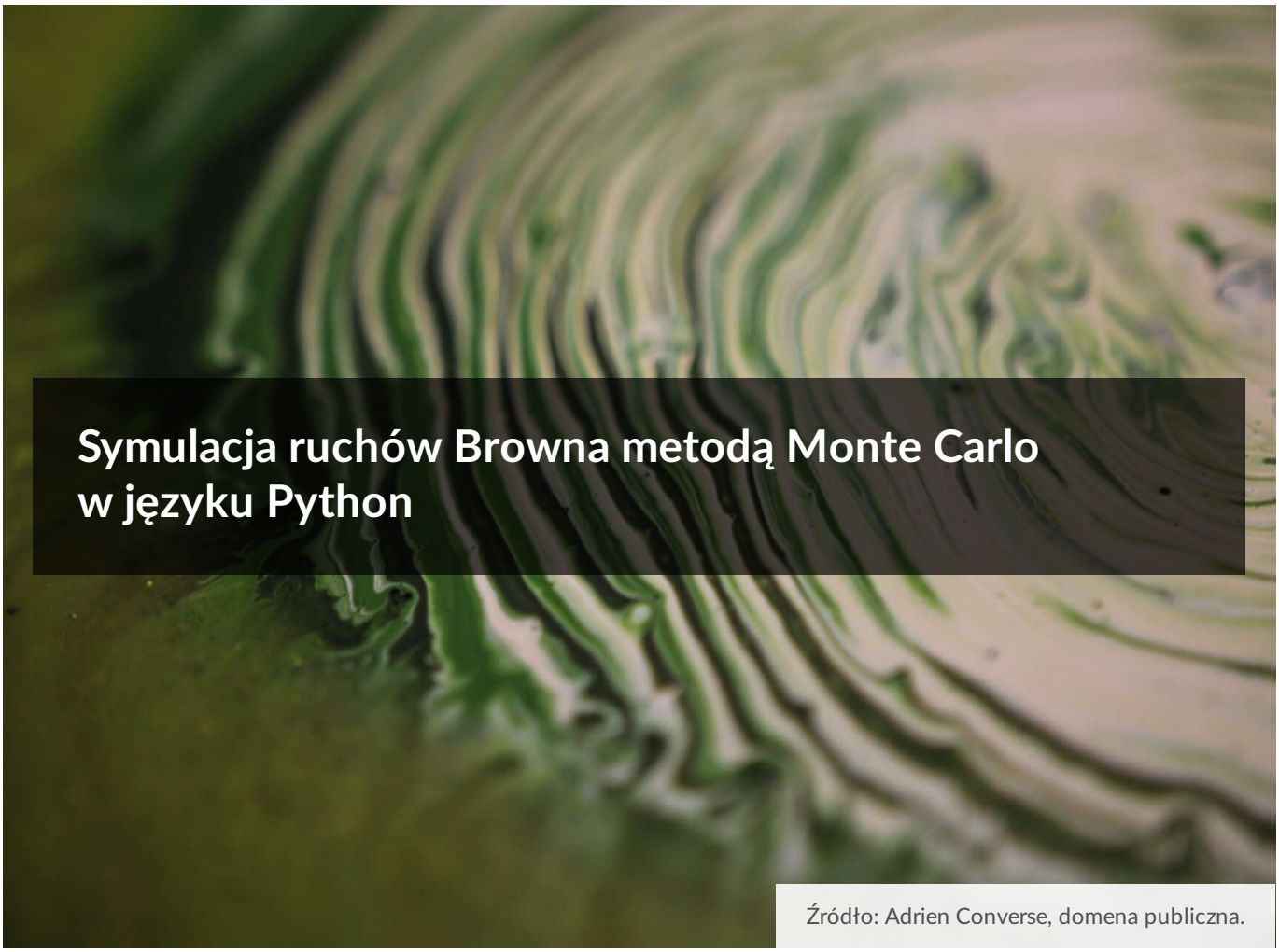
Symulacja ruchów Browna metodą Monte Carlo w języku Python



Symulacja ruchów Browna metodą Monte Carlo w języku Python

Źródło: Adrien Converse, domena publiczna.

Znasz już teorię ruchów Browna, nadszedł więc moment, aby prześledzić poruszanie się cząstki na ekranie. W tym e-materiale zajmiemy się realizacją symulacji ruchów cząsteczki w języku Python. Aby to osiągnąć, wykorzystamy połączenie teorii dotyczącej procesu Wienera oraz praktycznej metody Monte Carlo.

Z informacjami na temat wyznaczania liczby π metodą Monte Carlo można się zapoznać w e-materiale Wyznaczanie liczby π metodą Monte Carlo w języku Python.

Więcej teorii oraz ćwiczeń znajdziesz w:

- Symulacja ruchów Browna metodą Monte Carlo,
- Symulacja ruchów Browna metodą Monte Carlo – zadania maturalne.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Omówiono je w dwóch pozostałych materiałach z tej serii:

- Symulacja ruchów Browna metodą Monte Carlo w języku Java,
- Symulacja ruchów Browna metodą Monte Carlo w języku C++.

> **Twoje cele**

- Utworzysz szkic przykładowej trasy cząsteczki, która porusza się zgodnie z ruchami Browna symulowanymi metodą Monte Carlo, wykorzystując możliwości graficzne języka Python.
- Prześledzisz, jak można przeprowadzić prezentację wyników symulacji w programie.
- Przeanalizujesz sposób wykorzystania rozkładu normalnego Gaussa w języku Python.

Przeczytaj

Przypomnienie wiadomości o symulowaniu ruchów Browna metodą Monte Carlo

W e-materiale Modelowanie ruchów Browna przeprowadziliśmy proste modelowanie, w którym zakładaliśmy, że w każdej jednostce czasu cząsteczka przemieszcza się o określoną z góry odległość. Zaimplementujemy ten algorytm w języku Python.

Aby zobrazować proces losowego poruszania się cząsteczki w płynie, zakładamy, że każda kolizja powoduje jej przemieszczenie się o określoną długość w dowolnym kierunku. Wybieramy losowy kąt z przedziału $\langle 0, 2\pi)$, a następnie przesuwamy cząsteczkę o wybraną długość pod wylosowanym kątem.

Zaimplementujmy algorytm, który pozwoli utworzyć regularne trasy ruchu cząsteczki. Przyjmujemy następujące założenia:

- cząsteczka na początku znajduje się w środku układu współrzędnych `start_xy = (0, 0)`,
- każdy ruch cząsteczki ma taką samą długość o wartości `wektor = 1`,
- kierunek ruchu wyznaczony jest kątem `fi` losowanym tylko spośród wielokrotności kąta podstawowego,
- wielokrotność kąta podstawowego będzie wyliczana na podstawie przyjętej wartości liczby całkowitej `k`,
- kolejne współrzędne `(x, y)` cząstki będą wyliczane według wzorów:

$$X_n = X_{n-1} + (wektor \cdot \cos{(\varphi)})$$

$$Y_n = Y_{n-1} + (wektor \cdot \sin{(\varphi)})$$

Wielokrotność kąta podstawowego $\frac{2\pi}{k}$ uzyskamy w następujący sposób:

- wylosujemy liczbę naturalną z przedziału `<0; k)` (nie włączając `k`),
- podzielimy kąt 2π przez `k` i pomnożymy przez wylosowaną liczbę.

> **Przykład 1**
>
> Napiszemy program, który wylicza losowe współrzędne poruszającej się cząsteczki i obrazuje jej ruch.

Współrzędne kolejnych punktów wygeneruje funkcja `ruchy_browna()`. Aby uzyskać takie same zestawy liczb pseudolosowych, użyjemy funkcji `seed()` z modułu `random`, która zainicjuje wartość początkową generatora liczb argumentem `start_seed`. Funkcja `randint()` pozwoli losować wartości całkowite z zadanego przedziału. Losowy kąt ruchu o wielkości `360 / k`, dostosowany do uzyskania regularnych tras, będzie wyliczany w instrukcji: `fi:float = randint(0, k-1) * 2 * pi / k`.

Wyliczone współrzędne kolejnych punktów, w których znajdzie się cząsteczka, zapisywać będziemy w dwóch listach `(lista_x, lista_y)` o tej samej długości. Funkcja `ruchy_browna` zwróci te listy, abyśmy mogli wykreślić ruch cząsteczki.

Do pokazania ruchu cząsteczki posłużymy się funkcją `wizualizuj_ruchy_browna()` i biblioteką matplotlib, która pozwala m.in. na rysowanie punktów na płaszczyźnie przy użyciu list zawierających ich współrzędne `(x, y)`. Dzięki zastosowaniu odpowiedniej opcji punkty te zostaną automatycznie połączone liniami.

```
start_xy = (0, 0)  # początkowe położenie cząsteczki
n = 150  # liczba punktów
k = 3  # obrót nastąpi o wielokrotność kąta 360/k stopni
wektor = 1.0  # długość przemieszczenia
start_seed = 234567  # wartość inicjująca generator liczb

def ruchy_browna(start_xy: tuple, n: int, wektor: float, start_
    from math import sin, cos, pi
    from random import randint, seed

    seed(start_seed)
    lista_x: list = [start_xy[0]]
    lista_y: list = [start_xy[1]]

    for krok in range(1, n):
        fi:float = randint(0, k-1) * 2 * pi / k  # losowa wielo
        # wyliczenie nowych współrzędnych
        dx = lista_x[krok - 1] + wektor * cos(fi)
        dy = lista_y[krok - 1] + wektor * sin(fi)
        # zapisanie nowego położenia w listach
        lista_x.append(dx)
        lista_y.append(dy)

    return (lista_x, lista_y)


```

```python
def wizualizuj_ruchy_browna(wspolrzedne: tuple) -> None:
    import matplotlib.pyplot as plt

    X: list = wspolrzedne[0]
    Y: list = wspolrzedne[1]
    plt.plot(X, Y, 'o-', color='red', linewidth=1)
    plt.xlabel('Współrzędne X')
    plt.ylabel('Współrzędne Y')
    plt.title('Ruchy Browna')
    plt.grid(True)
    plt.show()

# wywołanie funkcji generujących współrzędne punktów
punkty = ruchy_browna(start_xy, n, wektor, start_seed)

# wypisanie wyliczonych współrzędnych
print(punkty)

# wywołanie funkcji rysującej punkty
wizualizuj_ruchy_browna(punkty)
```

Dla zainteresowanych

W definicjach funkcji po liście parametrów umieściliśmy kod `-> tuple` oraz `-> None`. To przykład użycia `type hints`, które omówiono w dokumencie `PEP-484` języka Python. Sposób ten – opcjonalny – umożliwia określenie oczekiwanych typów danych, co jest pomocne podczas przeglądania i modyfikowania kodu. W omawianym wypadku wskazujemy, że pierwsza funkcja zwraca tuplę (krotkę), natomiast druga nie zwraca niczego.

Symulacje utworzone za pomocą powyższego skryptu będą wyglądały następująco:

- dla `k = 3`

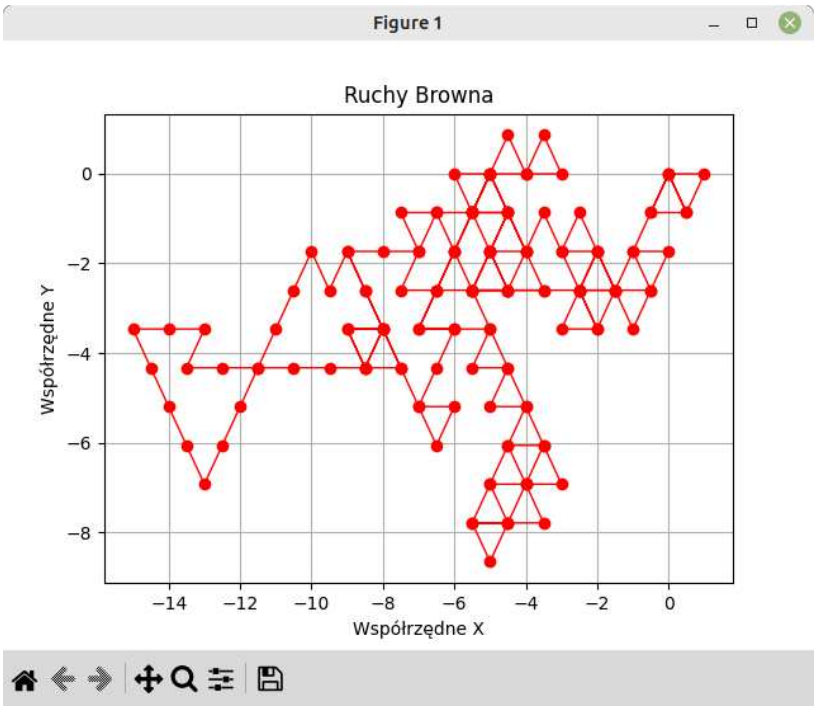


Figure 1
Ruchy Browna
Współrzędne Y
Współrzędne X
0
−2
−4
−6
−8
−14
−12
−10
−8
−6
−4
−2
0

- dla k = 4

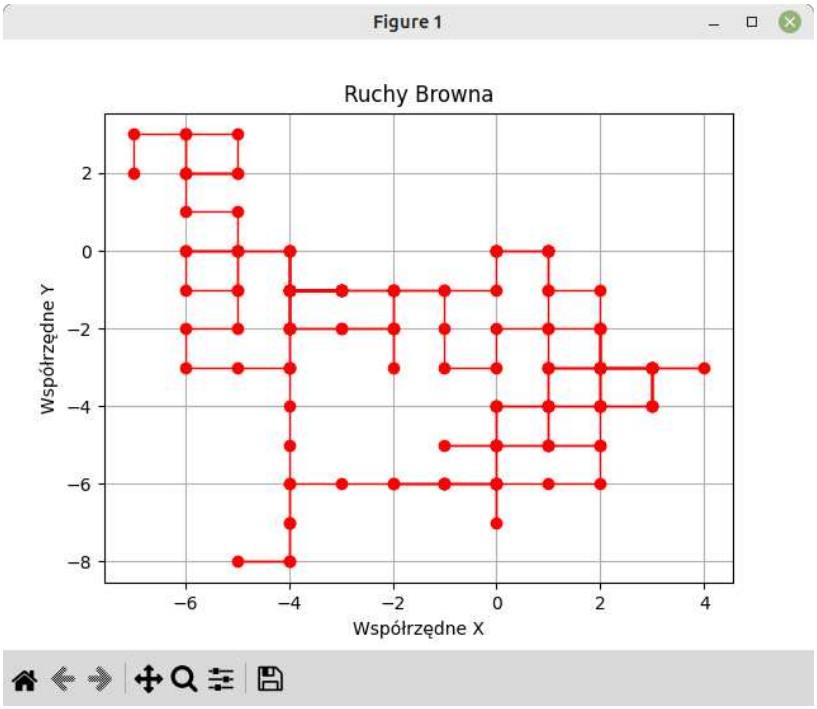
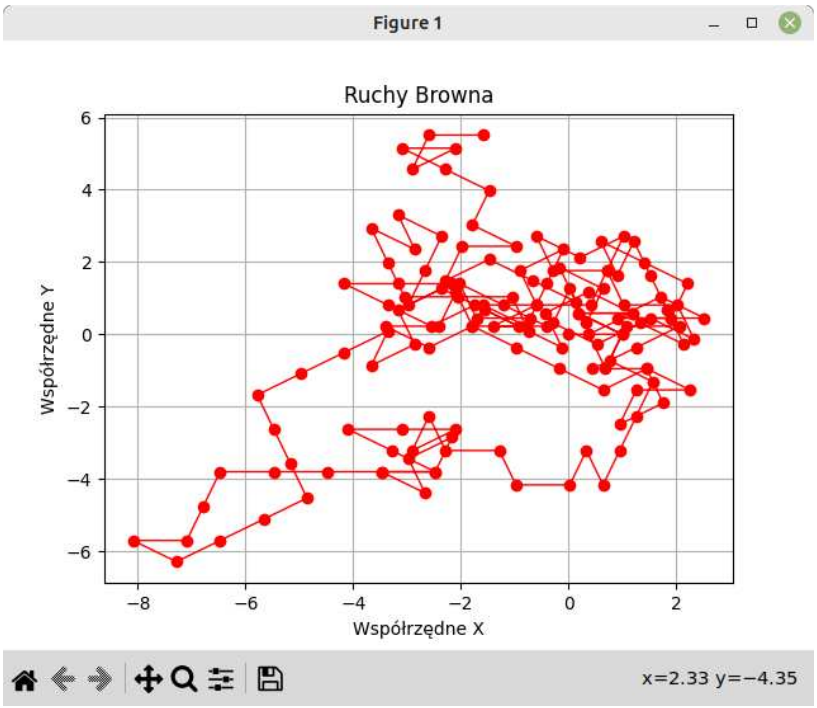


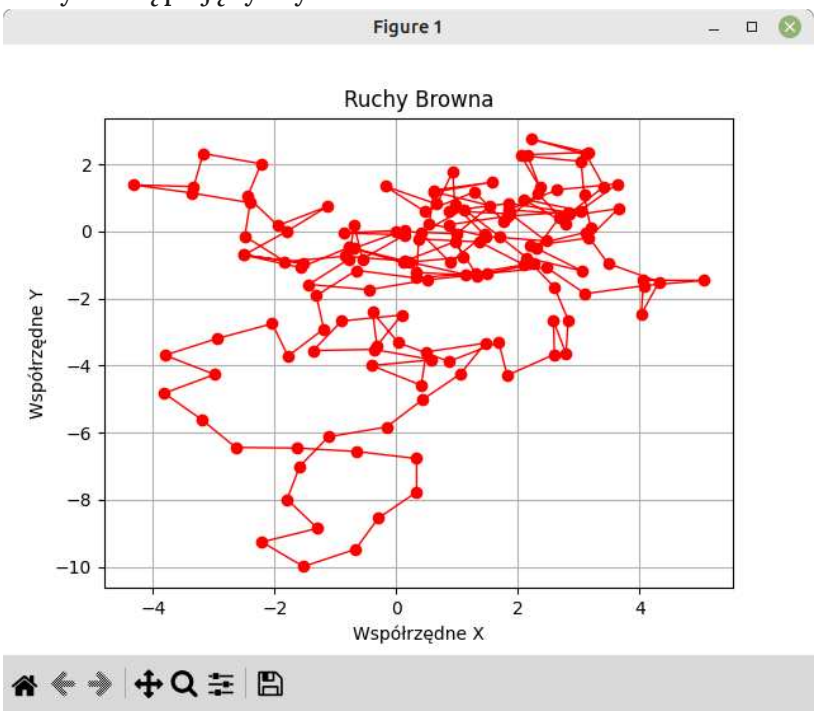
Figure 1
Ruchy Browna
Współrzędne Y
Współrzędne X
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4

- dla k = 5



Wzrost parametru `k` zwiększa losowość kierunku przemieszczania się cząsteczki. Wynika to z faktu, że im większa wartość `k`, tym więcej możliwych wielokrotności kąta wyznaczających kierunek ruchu. Dążąc do jak największej losowości, możemy zwiększać parametr `k` aż do wartości `360`.

Dla `k = 360` uzyskamy następujący wykres:



W przedstawiony sposób możemy symulować ruchy Browna dla różnych danych początkowych, tzn. liczby kroków, liczby wielokrotności oraz długości przemieszczenia.

> **Polecenie 1**
>
> Bazując na poprzednich przykładach, spróbuj wywołać program w taki sposób, aby uzyskać na wykresie tylko punkty początkowy i końcowy podczas wykonywania 300 kolejnych kroków. Jako `wektor` przyjmij wartość `0.73`, a `start_seed` niech wynosi `432432`. Dobierz eksperymentalnie wartość dla parametru `k`.

Symulacja ruchów Browna metodą Monte Carlo z wykorzystaniem procesu Wienera

Przypomnijmy uproszczoną definicję procesu Wienera:

$$W(0) = 0$$

$$W(t + dt) = W(t) + k \cdot N(0,\ dt)$$

gdzie $N\left(\mu, \sigma^2\right)$ jest zmienną losową zgodną z rozkładem normalnym Gaussa, z parametrami: wartość oczekiwana μ oraz odchylenie standardowe σ, przy czym k jest współczynnikiem długości wykonywanych kroków.

Jak możemy wywnioskować z tej definicji, jest to proces uzależniony nie od liczby iteracji, a od czasu. Odległość cząsteczki od położenia początkowego po wykonaniu stu kroków z parametrem $dt = 0.1$ jest uzależniona od prawdopodobieństwa w taki sam sposób, co odległość po wykonaniu jednego kroku z parametrem $dt = 10$. Innymi słowy, wynik końcowy stu kroków z parametrem $dt = 0.1$ jest statystycznie równy wynikowi końcowemu jednego kroku z parametrem $dt = 10$.

Aby stworzyć symulację wykorzystującą proces Wienera, musimy zastanowić się nad jej warunkami początkowymi. Zaczynamy od umieszczenia cząsteczki w punkcie $(0, 0)$. Przed narysowaniem kolejnej pozycji cząsteczki będziemy czekali 0,05 sekundy, czyli przyrost czasu wyniesie `dt = 0.05`. Dla tak dobranego parametru `dt` ustawimy współczynnik `k = 0.2`, co wpłynie na skalę przemieszczenia.

W języku Python, aby uzyskać losową liczbę zgodną z rozkładem normalnym, wykorzystamy moduł `random` i zawartą w nim funkcję `gauss(μ, σ)`. Pierwszy parametr oznacza wartość oczekiwaną μ, drugi odchylenie standardowe σ. W omawianym przykładzie przyjmujemy wartość oczekiwaną równą `0` i odchylenie równe `1`.

W ten sposób otrzymujemy liczbę $N(0, 1)$. Jak możemy uzyskać $N(0,\ dt)$? Skorzystamy z własności wynikającej ze wzoru funkcji rozkładu normalnego, mianowicie:

$$N\left(\mu, \sigma^2\right) = \frac{1}{\sigma} \cdot N\left(0,\ 1\right) + \mu$$

Do wyznaczenia $N\left(0,\ dt\right)$ możemy więc użyć kodu:

```
from random import gauss
from math import sqrt
# N(0, dt)
N_0_dt = 1 / sqrt(dt) * gauss(0, 1)
```

Wykorzystując tę metodę, kolejne położenia cząsteczek będziemy wyznaczać następująco:

```
x += k * sqrt(dt) * gauss(0, 1)
y += k * sqrt(dt) * gauss(0, 1)
```

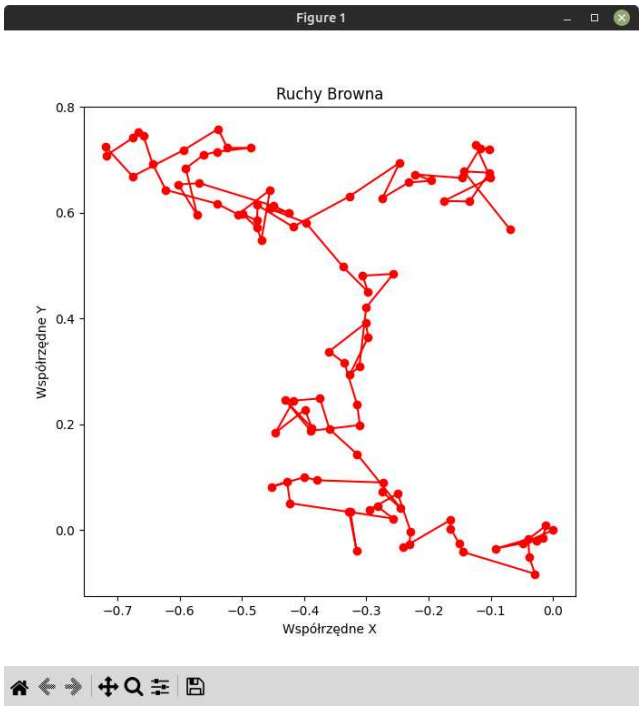
Operację tę będziemy powtarzać do momentu, kiedy cząsteczka wykona `n` kroków. Dodatkowo po każdym wyliczeniu położenia cząsteczki uwzględnimy przyrost czasu wskazany w parametrze `dt`, wstrzymując na ten czas wykonywanie programu. Parametr `dt` wykorzystamy również do opóźnienia rysowania ruchu cząsteczki. Pełny kod skryptu zawierający zmienioną funkcję rysującą `animuj_ruchy_browna()` zamieszczamy poniżej:

```
start_xy = (0, 0)  # początkowe położenie cząsteczki
n = 100  # liczba kroków
k = 0.2  # współczynnik długości kroków
dt = 0.05  # przyrost czasu

def ruchy_browna() -> None:
    from math import sqrt
    from random import gauss
    from time import sleep

    lista_x: list = [start_xy[0]]
    lista_y: list = [start_xy[1]]

    for krok in range(1, n):
        lista_x.append(lista_x[krok - 1] + k * sqrt(dt) * gauss(0
        lista_y.append(lista_y[krok - 1] + k * sqrt(dt) * gauss(0
        sleep(dt)
```

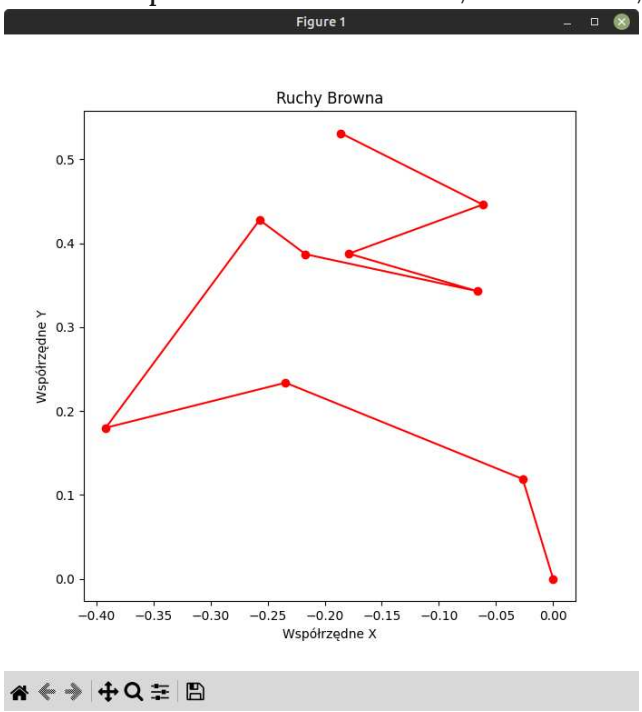
```
18
19      return (lista_x, lista_y)
20
21
22  def animuj_ruchy_browna(wspolrzedne: tuple) -> None:
23      import matplotlib.pyplot as plt
24      from matplotlib.animation import FuncAnimation
25
26      X: list = wspolrzedne[0]
27      Y: list = wspolrzedne[1]
28      fig, ax = plt.subplots()
29      ax.figure.set_size_inches(7, 7)
30      ax.set_xlabel('Współrzędne X')
31      ax.set_ylabel('Współrzędne Y')
32      ax.set_title('Ruchy Browna')
33
34      line, = ax.plot(X, Y, color='r', marker='o')
35
36      def animate(i):
37          line.set_data(X[:i+1], Y[:i+1])
38          return line
39
40      ani = FuncAnimation(fig, animate, interval=dt*1000, repeat=Fa
41      plt.show()
42
43  punkty = ruchy_browna()
44  animuj_ruchy_browna(punkty)
```

Prześledźmy przykładowe wyniki symulacji po uruchomieniu przedstawionego kodu z wybranymi parametrami.

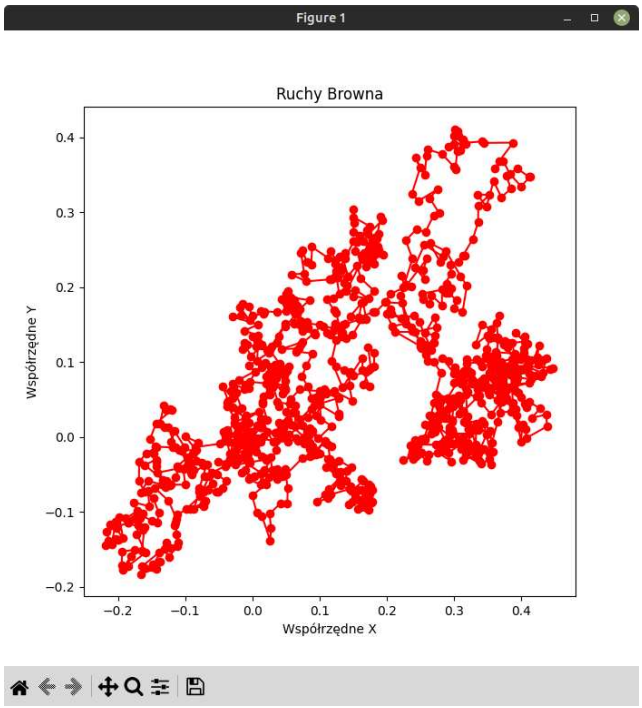
Końcowy wynik symulacji ruchu cząsteczki dla parametrów `n = 100`, `dt = 0.05`, `k = 0.2`:



Wizualizacja ruchu cząsteczki dla parametrów `n = 10`, `dt = 0.5`, `k = 0.2`:



Rezultat symulacji przeprowadzonej dla danych `n = 1000`, `dt = 0.005`, `k = 0.2`:



Zwróć uwagę, że przedstawione wyniki nie różnią się czasem symulacji, a jedynie jej dokładnością.

> **Już wiesz**
>
> Podsumujmy najważniejsze wiadomości:
>
> - metoda Monte Carlo może być przydatna przy szacowaniu wyników na podstawie dużej liczby zmiennych losowych,
> - w języku Python możemy stosować tak zwane type hints,
> - funkcja `seed()` pozwala ustawić wartość początkową (ziarno) dla generatora liczb pseudolosowych.

Słownik

> **importowanie**
>
> operacja pozwalająca w programie (funkcji) używać dodatkowych funkcji lub obiektów, dostępnych w zewnętrznych bibliotekach
> **matplotlib**
>
> biblioteka służąca do przedstawienia obrazów złożonych z punktów o współrzędnych `x` oraz `y` (np. wykresów, histogramów, rozkładów); moduł `matplotlib` nie jest dostępny w standardowej instalacji języka Python; należy go zainstalować, korzystając z mechanizmu `pip`
> **rozkład normalny Gaussa**
>
> ciągły rozkład prawdopodobieństwa, zdefiniowany następującą funkcją gęstości prawdopodobieństwa:

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

gdzie μ to wartość oczekiwana zmiennej losowej x, a σ to odchylenie standardowe zmiennej losowej x

seed(x)

(ang. *seed* – ziarno) funkcja pozwalająca ustalić punkt początkowy obliczeń, których rezultatem jest wygenerowanie liczby pseudolosowej przez funkcje takie jak `random()`, `randint()` i podobne; po określeniu wartości zmiennej `x` wyniki działania funkcji losowych są powtarzalne; jeśli nie podano wartości `x` lub ustalono ją jako `None`, za wartość `x` przyjmuje się aktualny czas

`type hints`

informacja o typie danego obiektu zapisana literalnie w kodzie programu; z uwagi na dynamiczną naturę języka Python pomaga w zrozumieniu kodu programu i pozwala na sprawdzanie kodu przez różne środowiska IDE (np. PyCharm, SublimeText lub Atom); przykłady użycia: `zmienna: int = 10`; `type hints` są dokładnie opisane w dokumencie PEP 484 – ich inna nazwa to **`annotations`** (opisane w dokumencie PEP 3107)

zmienna losowa

zmienna, której wartość pozostaje nieznana do momentu losowania; w języku Python można uzyskać ją między innymi dzięki funkcjom należącym do modułu `random`:

- `randint(a, b)` – zwraca liczbę całkowitą z przedziału `<a, b>`
- `randrange(a, b, c)` – zwraca liczbę całkowitą z przedziału `<a, b-1>`, z krokiem `c`
- `random()` – zwraca liczbę rzeczywistą z przedziału `<0.0, 1.0)`

Gra edukacyjna

Polecenie 1

Sprawdź swoją wiedzę, biorąc udział w grze.

Test

Sprawdź swoją wiedzę, biorąc udział w grze

Poziom trudności:

łatwy

Limit czasu:

7 min

Twój ostatni wynik:

-

Uruchom

Polecenie 2

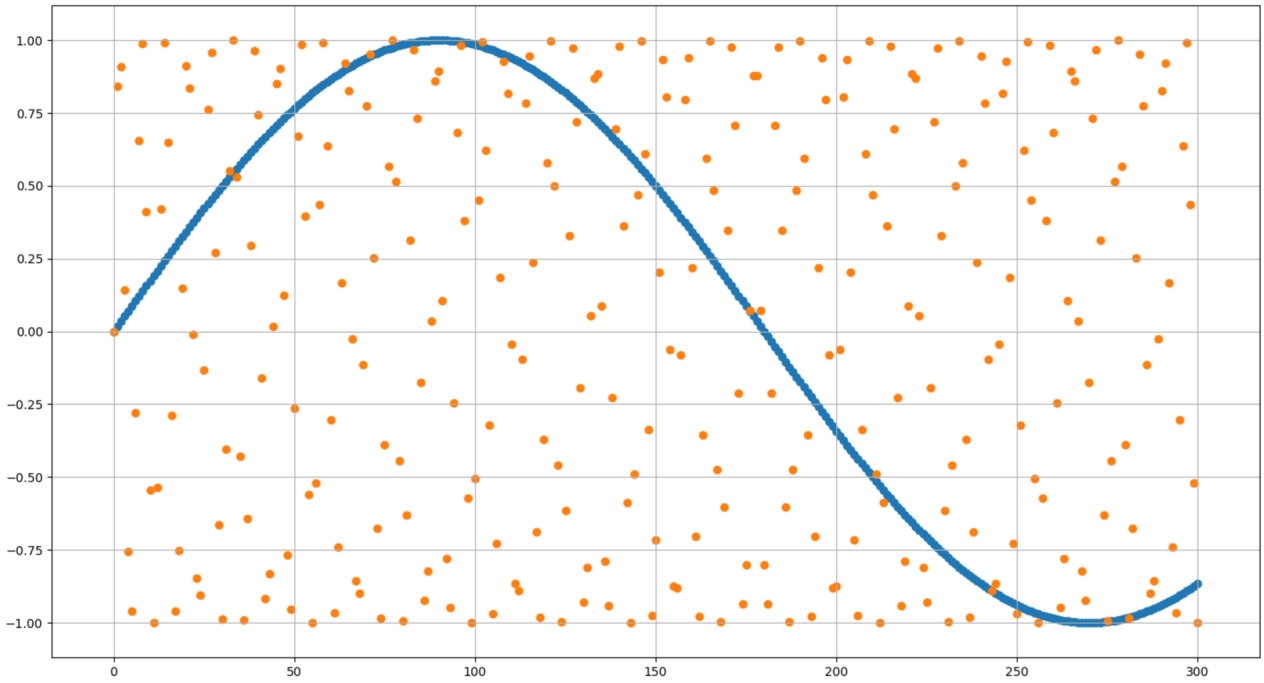
Zastanów się, które pytania sprawiły ci trudność. Wróć do fragmentów e-materiału, w których możesz znaleźć informacje na ten temat.

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Przeanalizuj zawartość wykresu i wskaż kod, którego użyto do jego wygenerowania:



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

```
# kod 1
def obliczenia(start: int = 0, koniec: int = 301):
    from math import sin, pi, radians
    import matplotlib.pyplot as plt

    def r(i):
        return sin(radians(float(i)))

    X = [x for x in range(start, koniec)]
    Y = [r(q) for q in X]
    Z = [sin(q) for q in X]

    plt.scatter(X, Y)
    plt.scatter(X, Z)
    plt.grid(True)
    plt.show()

# wykonanie
obliczenia(1, 310)
```

```python
# kod 2
def obliczenia(start: int = 0, koniec: int = 301):
    from math import sin, pi, radians
    import matplotlib.pyplot as plt

    def r(i):
        return sin(radians(float(i)))

    X = [x for x in range(start, koniec)]
    Y = [r(q) for q in X]
    Z = [sin(q) for q in X]

    plt.scatter(X, Y)
    plt.scatter(X, Z)
    plt.grid(True)
    plt.show()

# wykonanie
obliczenia(0, 301)
```

```python
# kod 3
def obliczenia(start: int = 0, koniec: int = 301):
    from math import sin, pi, radians
    import matplotlib.pyplot as plt

    def r(i):
        return sin(float(i))

    X = [x for x in range(start, koniec)]
    Y = [r(q) for q in X]
    Z = [sin(q) for q in X]

    plt.scatter(X, Y)
    plt.scatter(X, Z)
    plt.grid(True)
    plt.show()

# wykonanie
obliczenia(0, 301)
```

Wskaż, który z przedstawionych kodów został użyty.

- ◯ kod nr 3
- ◯ żadna z odpowiedzi – te kody generują błąd: `ZeroDivisionError: float division by zero`
- ◯ kod nr 1
- ◯ kod nr 2

Uzasadnij swoją odpowiedź.

Pokaż ćwiczenia:

Ćwiczenie 2

Ustawiono chodziarza na osi liczbowej w punkcie 0. W każdym ruchu chodziarz porusza się losowo w lewo lub w prawo. Metodą Monte Carlo przeprowadź n symulacji, w których sprawdzisz, w jakim punkcie znalazł się chodziarz po wykonaniu k kroków. Następnie dla każdej pozycji parzystej wypisz słupek ze znaków `*`, który będzie wizualizował procentowy udział danej pozycji we wszystkich pozycjach końcowych. Każdy znak `*` ma oznaczać 1 punkt procentowy. Łącznie należy wypisać `m` znaków `*`. W celu wylosowania tego samego rozkładu kroków dla każdego wywołania programu posłuż się ziarnem (`seed`) jako argumentem funkcji losującej.

Działanie programu sprawdź dla `n = 100 000`, `k = 10`, `m = 100`, `seed = 3`.

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba symulacji
- `k` – liczba naturalna; liczba wykonanych kroków
- `m` – liczba naturalna; liczba punktów procentowych zaznaczonych za pomocą znaku *, oznaczających procentowy udział pozycji we wszystkich pozycjach końcowych
- `seed` – liczba naturalna; argument funkcji losującej

Wynik:

- `m` znaków `*`; „wykres” procentowego udziału kolejnych pozycji we wszystkich pozycjach końcowych

Poprawne wyjście dla `n = 100 000`, `k = 10`, `m = 100`, `seed = 3` powinno wyglądać następująco:

```
1 *
2 ****
3 ************
```

```
**********
**********
**********
**********
****
*
```

Twoje zadania

1. Program tworzy „wykres” procentowego udziału kolejnych pozycji we wszystkich pozycjach końcowych.

```
def main():
    # tu wpisz swój kod


main()
```

```
```

Dla nauczyciela

Autor: Robert Bednarz
Przedmiot: Informatyka

Temat: Symulacja ruchów Browna metodą Monte Carlo 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

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;

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

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

h) metodę Monte Carlo (obliczanie przybliżonej wartości liczby π, symulacja ruchów Browna),

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):

- Utworzysz szkic przykładowej trasy cząsteczki, która porusza się zgodnie z ruchami Browna symulowanymi metodą Monte Carlo, wykorzystując możliwości graficzne języka Python.
- Prześledzisz, jak można przeprowadzić prezentację wyników symulacji w programie.
- Przeanalizujesz sposób wykorzystania rozkładu normalnego Gaussa w języku Python.

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: „Symulacja ruchów Browna metodą Monte Carlo w języku Python”. Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj”.

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Wspólnie z uczniami ustala kryteria sukcesu.
2. Nauczyciel sprawdza przygotowanie uczniów do lekcji.
3. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, np.
    - jakie zjawisko nosi nazwę ruchów Browna?
    - co to jest zmienna losowa?
    - na czym polega modelowanie z wykorzystaniem tzw. metody Monte Carlo?

   Chętni lub wybrani uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie analizują w parach przykład z sekcji „Przeczytaj", powtarzają i testują zaprezentowane rozwiązanie na swoich komputerach.
2. **Praca z multimedium.** Nauczyciel prosi uczniów, aby indywidualnie wzięli udział w grze z sekcji „Gra edukacyjna". Po ustalonym wcześniej czasie wskazane osoby prezentują swoje odpowiedzi.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1 z sekcji „Sprawdź się", a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.

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.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 2 z sekcji „Sprawdź się".
2. Uczniowie wykonują polecenie 2 z sekcji „Gra edukacyjna".

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 „Gra edukacyjna" można wykorzystać na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.