Metaheurystyki i ich zastosowania 2023/24

Zadanie 2 - symulowane wyżarzanie

Autorzy:
Michał Ferdzyn 242383
Artur Grzybek 242399

Wybranie przykłady funkcji:

Rozdział 3:

Przykład 1
Dana jest funkcja f(x) w przedziale [-150, 150]. Określona jest ona wzorem

$$f(x) = \begin{cases} -2\,|\,x+100\,|+10 & dla \quad x \in (-105, -95) \\ -2.2\,|\,x-100\,|+11 & dla \quad x \in (95, 105) \\ 0 \quad dla \quad x \notin (-105, -95) \cup (95, 105) \end{cases}$$

Dla ustalonych parametrów:
- T = 500
- $\delta(T)$ = 0.999
- k = 0.1
- M = 3000

, gdzie: początkowa temperatura T, współczynnik stygnięcia δ, stała Boltzmanna k, liczba iteracji obliczeń M

Rozdział 4:

Przykład 4
Należy określić maksimum funkcji f(x)=x·sin(10πx)+1 w przedziale [-1, 2].

Dla ustalonych parametrów:
- T = 5
- $\delta(T)$ = 0.997
- k = 0.1
- M = 1200

1. Założenia i działanie algorytmu

Algorytm symulowanego wyżarzania jest techniką heurystyczną, która jest często stosowana do rozwiązywania problemów optymalizacyjnych, zwłaszcza tych, które mają wiele maksimów lokalnych. Algorytm jest w stanie uniknąć utknięcia w lokalnym minimum i przeszukiwać przestrzeń rozwiązań, starając się znaleźć globalne minimum lub maksimum funkcji oceny. Jego wydajność i skuteczność zależą od dobrania odpowiednich parametrów i charakterystyki funkcji oceny.

Założenia:

- **Funkcje optymalizacji:** Algorytm umożliwia optymalizację dwóch różnych funkcji: f1(x) oraz f2(x). Wybór funkcji jest dokonywany przez użytkownika przy użyciu menu.
- **Funkcja oceny:** Algorytm symulowanego wyżarzania zakłada obecność funkcji oceny, która przyjmuje pewne rozwiązanie i zwraca wartość opisującą jakość tego rozwiązania. Celem algorytmu jest znalezienie rozwiązania, które minimalizuje lub maksymalizuje tę funkcję w zależności od problemu optymalizacji.
- **Przedział rozwiązań:** Algorytm operuje na określonym przedziale wartości rozwiązań. Przedział ten jest często określany na podstawie wiedzy o problemie i może zawierać wartości graniczne (włączając w to wartości na końcach przedziału) lub być otwarty.
- **Parametry algorytmu:** Algorytm ma ustalone parametry, takie jak początkowa temperatura T, współczynnik stygnięcia δ, stała Boltzmanna k, liczba iteracji obliczeń M, które można dostosować w zależności od problemu i oczekiwanego działania algorytmu.

Działanie algorytmu:

1. **Inicjalizacja i menu:** Program rozpoczyna się od wyboru funkcji przez użytkownika i inicjalizacji odpowiednich parametrów dla wybranej funkcji.
2. **Główna pętla:** Algorytm wchodzi w główną pętlę, która będzie wykonywana przez określoną liczbę iteracji M.
3. **Generowanie rozwiązania sąsiedniego:** W każdej iteracji generowane jest losowe rozwiązanie sąsiednie, które różni się nieznacznie od aktualnego rozwiązania.
4. **Obliczenie różnicy kosztów:** Obliczana jest różnica kosztów między nowym rozwiązaniem a aktualnym rozwiązaniem.
5. **Akceptacja nowego rozwiązania:** Jeśli różnica kosztów jest dodatnia, nowe rozwiązanie jest akceptowane jako nowe rozwiązanie bieżące. W przeciwnym razie, jest szansa na zaakceptowanie gorszego rozwiązania w zależności od prawdopodobieństwa exp(delta / (k * T)), gdzie T to aktualna temperatura.
6. **Aktualizacja temperatury:** Temperatura jest aktualizowana zgodnie z funkcją T = alpha * T, co oznacza, że z czasem temperatura maleje, co pomaga algorytmowi w zbieżności do optymalnego rozwiązania.
7. **Zapamiętanie najlepszego rozwiązania:** Algorytm śledzi i zapamiętuje najlepsze rozwiązanie znalezione do tej pory.
8. **Warunek zakończenia:** Algorytm powtarza te kroki aż do osiągnięcia maksymalnej liczby iteracji M.
9. **Wynik:** Po zakończeniu obliczeń algorytm zwraca najlepsze znalezione rozwiązanie wraz z jego wartością.

2. Działanie programu - mini instrukcja

1. Program rozpocznie działanie od wyświetlenia menu, w którym użytkownik może wybrać jedną z dwóch funkcji do optymalizacji. Użytkownik wybiera funkcję, podając numer 1 lub 2 (w przypadku innej komendy następuje zakończenie pracy programu)

```
--------------------------------------------------------------------------------
Wybierz funkcję do optymalizacji:
1. f(x) = x * sin(10πx) + 1 , dla przedziału [-1,2]
2. f(x) = -2 * |x + 100| + 10 dla x należącego do (-105, -95)
   f(x) = -2.2 * |x - 100| + 11 dla x należącego do (95, 105)
   f(x) = 0 dla reszty x z przedziału [-150,150]
Inna komenda - zakończenie pracy programu
Podaj numer funkcji (1 lub 2):
```

2. Po dokonaniu wyboru, program inicjalizuje parametry algorytmu symulowanego wyżarzania, takie jak początkową temperaturę, współczynnik zmiany temperatury, współczynnik wygaszania, liczbę iteracji, przedział wartości rozwiązań oraz zakres generacji rozwiązań sąsiednich.
3. Algorytm symulowanego wyżarzania rozpoczyna obliczenia. W każdej iteracji, program generuje losowe rozwiązanie sąsiednie, oblicza różnicę kosztów między nowym a aktualnym rozwiązaniem i decyduje, czy nowe rozwiązanie zostanie zaakceptowane na podstawie różnicy kosztów oraz prawdopodobieństwa. Jeśli nowe rozwiązanie jest korzystniejsze lub spełniony jest warunek losowego wyboru, staje się nowym rozwiązaniem bieżącym.
4. Algorytm aktualizuje temperaturę i kontynuuje obliczenia przez określoną liczbę iteracji.
5. Po zakończeniu obliczeń, program zwraca wynik, który obejmuje znalezione rozwiązanie i jego wartość (maksimum globalne funkcji) dla wybranej funkcji.
6. Wynik jest wyświetlany w konsoli wraz z informacją o wybranej funkcji.

```
--------------------------------------------------------------------------------
Wybierz funkcję do optymalizacji:
1. f(x) = x * sin(10πx) + 1 , dla przedziału [-1,2]
2. f(x) = -2 * |x + 100| + 10 dla x należącego do (-105, -95)
   f(x) = -2.2 * |x - 100| + 11 dla x należącego do (95, 105)
   f(x) = 0 dla reszty x z przedziału [-150,150]
Inna komenda - zakończenie pracy programu
Podaj numer funkcji (1 lub 2): 2
Maksimum globalne funkcji f2: x = 100.00313929696978, f(x) = 10.99309354666649, liczba korekcji = 8
--------------------------------------------------------------------------------
```

7. Użytkownik może ponownie uruchomić program i wybrać inną funkcję lub wybrać te same lub zmienić parametry algorytmu, aby dostosować go do swoich potrzeb.

3. Wybrane miejsca implementacji rozwiązania

Menu – wyświetlanie

```
def menu():
    while True:
        print("_________________________________________________________________________________________")
        print("Wybierz funkcję do optymalizacji:")
        print("1. f(x) = x * sin(10πx) + 1 , dla przedziału [-1,2]")
        print("2. f(x) = -2 * |x + 100| + 10 dla x należącego do (-105, -95)")
        print("   f(x) = -2.2 * |x - 100| + 11 dla x należącego do (95, 105)")
        print("   f(x) = 0 dla reszty x z przedziału [-150,150]")
        print("Inna komenda - zakończenie pracy programu")
        choice = input("Podaj numer funkcji (1 lub 2): ")
```

- Program rozpoczyna działanie od nieskończonej pętli while True, która pozwala użytkownikowi wykonywać wybór funkcji lub zakończyć pracę programu.
- W menu wyboru funkcji użytkownik ma dwie opcje:
  - Opcja 1: f(x) = x * sin(10πx) + 1 w przedziale [-1, 2].
  - Opcja 2: **f(x) = -2 * abs(x + 100) + 10** dla x należącego do (-105, -95)
    f(x) = -2.2 * abs(x - 100) + 11 dla x należącego do (95, 105)
    f(x) = 0 dla reszty x z przedziału [-150, 150].

Menu – inicjalizacja parametrów

```
if choice == "1":
    T = 5
    alpha = 0.997 * T
    k = 0.1
    M = 1200
    f = f1
    s1 = -1
    s2 = 2
    result = simulated_annealing(T, alpha, k, M, f, s1, s2)
elif choice == "2":
    T = 500
    alpha = 0.999 * T
    k = 0.1
    M = 3000
    f = f2
    s1 = -150
    s2 = 150
    result = simulated_annealing(T, alpha, k, M, f, s1, s2)
else:
    print("Zakończenie pracy programu.")
    break
```

- Po wyborze opcji, program inicjalizuje parametry algorytmu symulowanego wyżarzania (temperaturę, współczynniki, itp). Wartości parametrów zostały przyjęte takie jak w podanym artykule.
- Następnie program wywołuje funkcję simulated_annealing z odpowiednimi parametrami, w tym wybraną funkcją optymalizacji i przekazuje jej te parametry.

Menu – wyświetlanie wyniku

```
best_solution, best_cost, correction, solution_changes = result
print(
    f"Maksimum globalne funkcji {f.__name__}: x = {best_solution}, f(x) = {best_cost}, liczba korekcji = {correction}")
# Stwórz wykres funkcji
plot_solution_changes(f, s1, s2, 0.01, solution_changes, (best_solution, best_cost))
```

- Po zakończeniu działania algorytmu, program wyświetla wynik, który obejmuje znalezione maksimum globalne funkcji, wartość x, i wartość funkcji dla tego maksimum oraz ilość poprawek jakich dokonał algorytm w celu znalezienia najlepszego rozwiązania
- Jest wyświetlany wykres funkcji w zadanym przedziale wraz z zaznaczonym ekstremum globalnym oraz punktami korekcji rozwiązania
- Użytkownik ma możliwość powtórnego wyboru funkcji lub zakończenia pracy programu, w zależności od wyboru.

Funkcje – implementacje

```
def f1(x):
    return x * math.sin(10 * math.pi * x) + 1

michalf1703
def f2(x):
    if -105 < x < -95:
        return -2 * abs(x + 100) + 10
    elif 95 < x < 105:
        return -2.2 * abs(x - 100) + 11
    else:
        return 0
```

- Funkcja f1(x) jest wyrażeniem matematycznym, które zależy od pojedynczego parametru x. Jest to funkcja nieliniowa, która jest wynikiem iloczynu x i sinusoidalnej funkcji zawierającej wyrażenie z pi (π). Wynik tej funkcji to suma iloczynu x i sinusoidalnej funkcji oraz 1.
- Funkcja f2(x) jest bardziej złożona i opisuje ją warunki. Działa na parametrze x i ma trzy różne przypadki w zależności od wartości x. Jeśli x należy do przedziału (-105, -95), to funkcja zwraca wartość wynikającą z operacji na x oraz liczbach 100, 2, 10. W przypadku, gdy x należy do przedziału (95, 105), funkcja działa podobnie, ale z innymi wartościami. W pozostałych przypadkach, funkcja zwraca 0.

Algorytm symulowanego wyżarzania

```python
def simulated_annealing(T, alpha, k, M, f, s1, s2):
    current_solution = random.uniform(s1, s2)
    current_cost = f(current_solution)
    best_solution = current_solution
    best_cost = current_cost
    correction = 0
    solution_changes = []

    for i in range(M):
        new_solution = random.uniform(max(s1, current_solution - 2 * T), min(s2, current_solution + 2 * T))
        new_cost = f(new_solution)
        delta = new_cost - current_cost

        if delta > 0 or random.random() < math.exp(delta / (k * T)):
            current_solution = new_solution
            current_cost = new_cost

        if new_cost > best_cost:
            best_solution = new_solution
            best_cost = new_cost
            correction += 1
            solution_changes.append((best_solution, best_cost))

        T = alpha * T

    return best_solution, best_cost, correction, solution_changes
```

- **Funkcja „simulated_annealing”** przyjmuje takie parametry: T (temperatura początkowa), alpha (współczynnik zmiany temperatury), k (współczynnik wygaszania), M (liczba iteracji), f (wybrana przez użytkownika funkcja), s1 (początek przedziału), s2 (koniec przedziału)
- **„current_solution = random.uniform(s1, s2)”:** Inicjalizacja bieżącego rozwiązania current_solution jako losowej wartości z przedziału [s1, s2]. To jest rozwiązanie początkowe, od którego rozpoczynamy proces optymalizacji.
- **„best_solution = current_solution i best_cost = current_cost”:** Inicjalizacja zmiennych best_solution i best_cost jako bieżącego rozwiązania i jego kosztu. Te zmienne będą śledzić najlepsze znalezione rozwiązanie podczas procesu optymalizacji.
- **„correction = 0”**: Inicjalizacja zmiennej correction na 0. Ta zmienna będzie używana do śledzenia liczby poprawek (aktualizacji) najlepszego rozwiązania.
- Inicjalizowana jest lista **„solution_changes”,** która będzie przechowywać zmiany best_solution w trakcie działania algorytmu.
- Następnie przechodzimy do pętli głównej: **„for i in range(M):”.** Ta pętla wykonuje określoną liczbę iteracji M w celu przeszukiwania przestrzeni rozwiązań.
- Losowanie new_solution w okolicy current_solution. Nowe rozwiązanie jest wybierane z przedziału **[current_solution - 2 * T, current_solution + 2 * T]** w celu rozszerzenia obszaru sąsiedztwa (wzór z prezentacji nr 4)

- **„new_cost = f(new_solution)":** Obliczenie wartości funkcji kosztu dla nowego rozwiązania new_solution i przechowanie jej jako new_cost.
- Obliczenie różnicy kosztów: **„delta = new_cost - current_cost".** Ta różnica jest używana do oceny, czy nowe rozwiązanie jest lepsze od bieżącego.
- Warunek akceptacji nowego rozwiązania: Jeśli delta jest mniejsza od zera (delta > 0) lub jeśli warunek losowego wyboru jest spełniony (**random.random() < math.exp(delta / (k * T))**), to bieżące rozwiązanie jest aktualizowane na nowe rozwiązanie (current_solution = new_solution i current_cost = new_cost). To jest kluczowy krok w algorytmie symulowanego wyżarzania, który pozwala na akceptowanie czasami gorszych rozwiązań, aby uniknąć utknięcia w lokalnych minimach.
- Aktualizacja najlepszego rozwiązania: Jeśli new_cost jest większa od best_cost, to best_solution i best_cost są aktualizowane na nowe wartości, a także zmienna correction jest zwiększana o 1.
- Aktualizacja temperatury: **„T = alpha * T".** Temperatura jest aktualizowana w każdej iteracji na podstawie współczynnika alpha. To pomaga w procesie wyżarzania, w którym temperatura maleje z czasem, co wpływa na prawdopodobieństwo akceptacji gorszych rozwiązań.
- Po zakończeniu pętli, algorytm zwraca najlepsze znalezione rozwiązanie (best_solution) oraz jego koszt (best_cost) jako wynik obliczeń oraz liczbę poprawek (correction) oraz listę solution_changes, która zawiera zmiany best_solution w trakcie działania algorytmu.

4. Analiza wyników

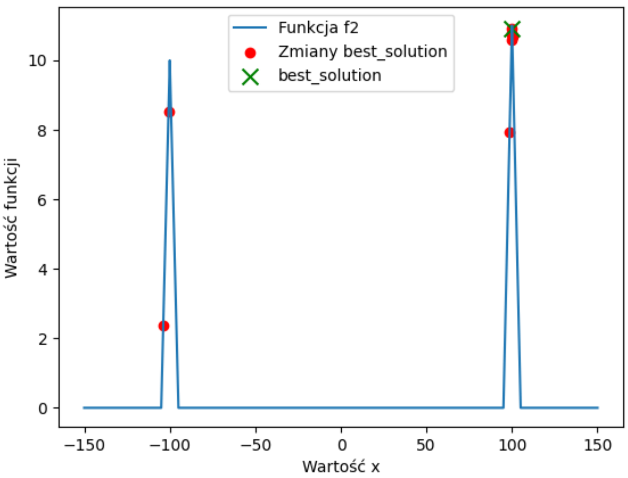
Funkcja z rozdziału 3 (dla przedziału **<-150, 150>**)

$$f(x) = \begin{cases} -2|x+100|+10 \quad dla \quad x \in (-105, -95) \\ -2.2|x-100|+11 \quad dla \quad x \in (95, 105) \\ 0 \quad dla \quad x \notin (-105, -95) \cup (95, 105) \end{cases}$$

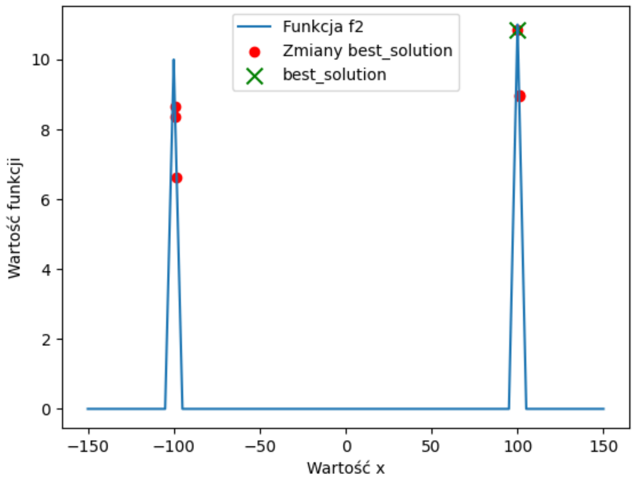
Wyniki:

Uruchomienie (nr)	Wartość x	Wartość f(x)	Ilość korekcji
1	99.95936289582363	10.910598370811988	8
2	99.92854635382119	10.842801978406618	6
3	100.0321585606148	10.929251166647433	6
4	99.9925888609423	10.983695494073071	7
5	100.08172822018487	10.820197915593283	6
6	99.95984425426943	10.91165735939274	9

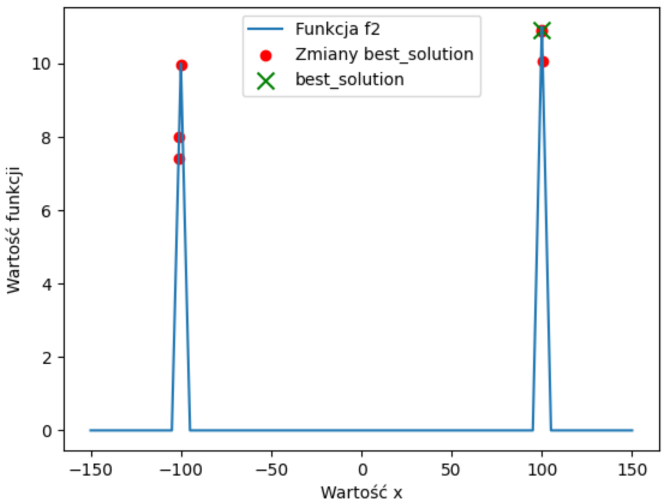
Uruchomienie nr 1



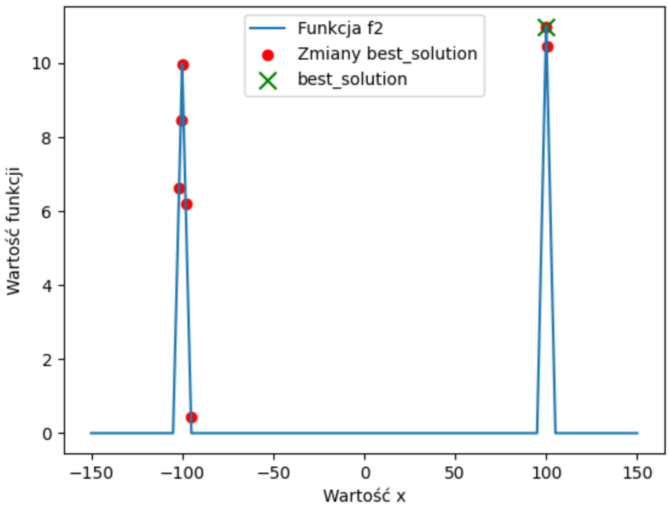
Uruchomienie nr 2



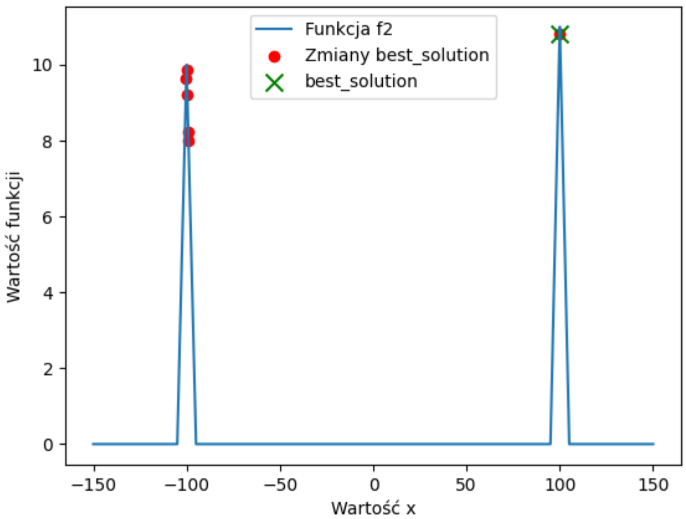
Uruchomienie nr 3



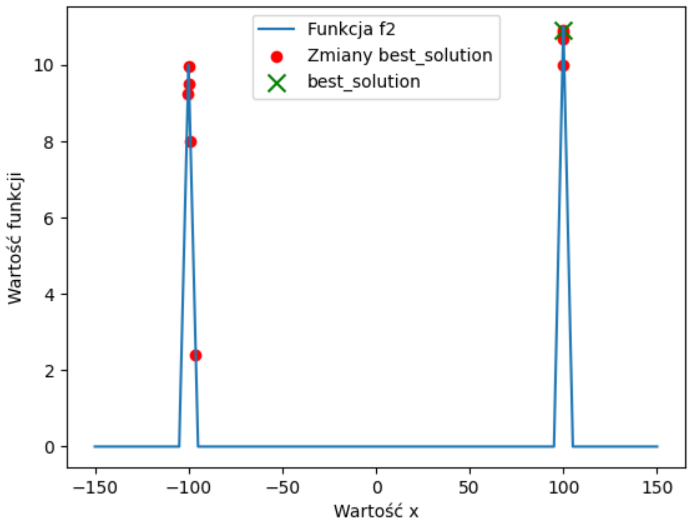
Uruchomienie nr 4



Uruchomienie nr 5



Uruchomienie nr 6



Analiza wyników:

Wynik jaki otrzymano w artykule: **f(x)=10.980462 dla x=100.008881, 7 korekcji.**
Wynik otrzymany, który jest najbardziej zbliżony: **f(x)= 10.983695 dla x=99.9925, 7 korekcji.**
Większość otrzymanych przez nas wyników jest zbliżona do tej wartości. Przeprowadziliśmy kilka prób dla tych samych parametrów, ponieważ wyniki algorytmu mogą zależeć od losowości generowania rozwiązań oraz rozwiązania początkowego. Na 3000 iteracji maksymalnie 9 razy było poprawiane ekstremum na większą wartość. Pozostałe parametry zostały przyjęte takie same jak w artykule (T = 500, k = 0.1, $\alpha(T) = 0.999$). Dokonaliśmy pewnych eksperymentów z wartościami parametrów: przetestowane zostały różne przedziały losowania sąsiadów. Najlepsze zaobserwowane przez nas wyniki otrzymywane były dla przedziału (-15,15). Zdecydowaliśmy się jednak na wykorzystanie wzoru, który jest bardziej optymalny dla różnych funkcji (wzór z wykładu 4), a nie tylko dla tego konkretnego przypadku. Również przy zwiększeniu liczby iteracji nasze wyniki były jeszcze bardziej zbliżone do poprawnych.

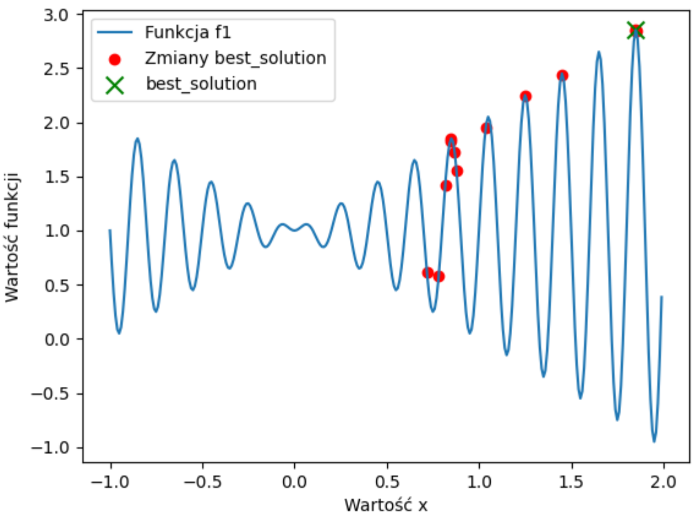
Funkcja z rozdziału 4 (dla przedziału **<-1, 2>**)

$$f(x)=x \cdot \sin(10\pi x)+1$$

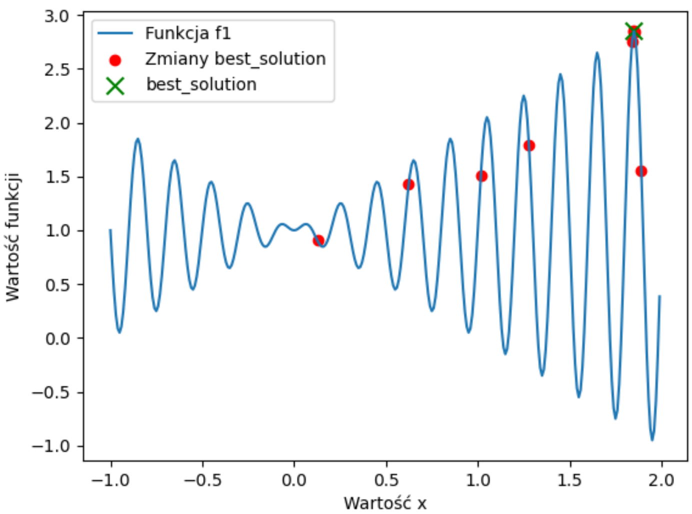
Wyniki:

Uruchomienie (nr)	Wartość x	Wartość f(x)	Ilość korekcji
1	1.8496141811375522	2.849478314605088	13
2	1.8504770042305076	2.8502692309119197	9
3	1.8509861070388305	2.850097957095463	5
4	1.850825841947466	2.8502029613254085	8
5	1.8487179695230198	2.847218704524102	11
6	1.8510100532316738	2.8500782362715134	14

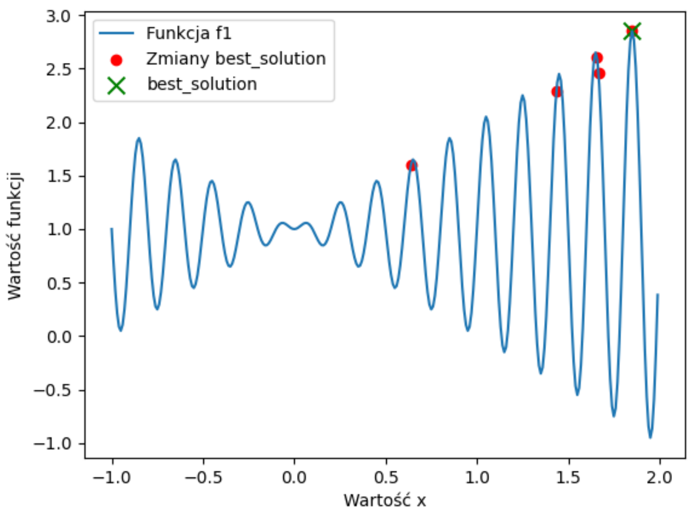
Uruchomienie nr 1



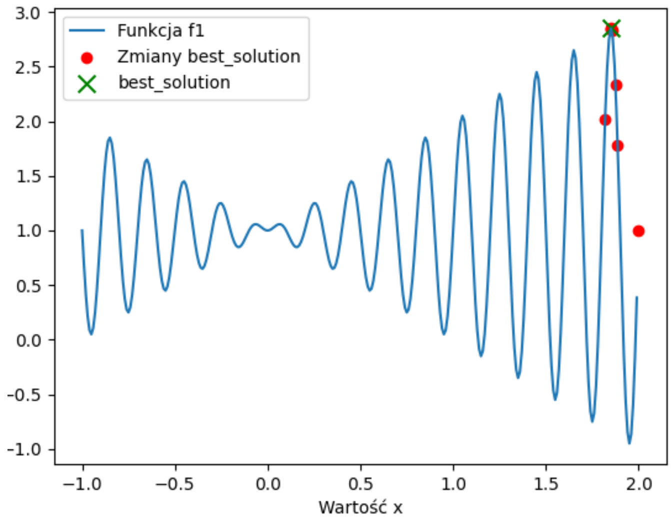
Uruchomienie nr 2



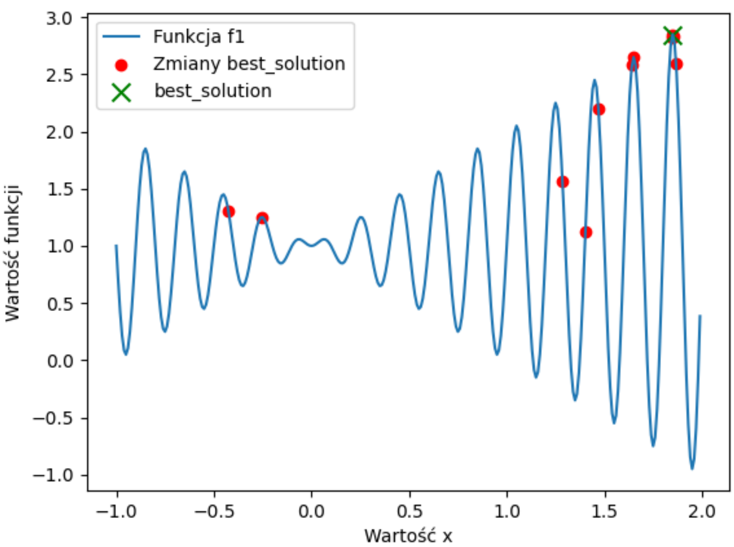
Uruchomienie nr 3



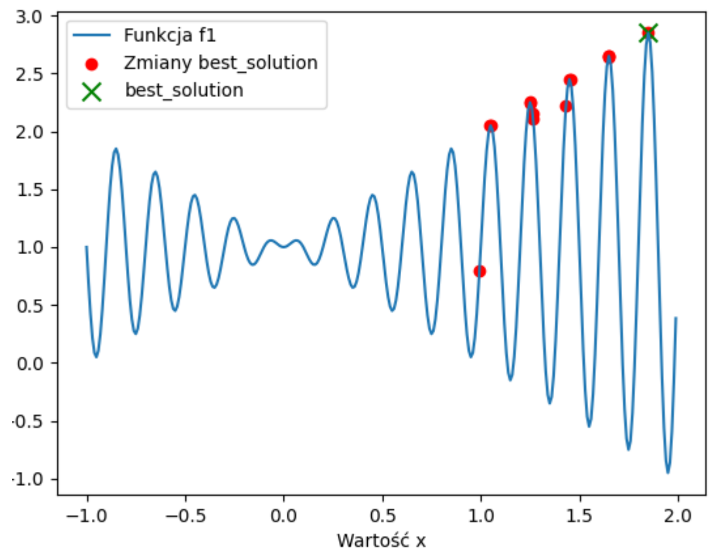
Uruchomienie nr 4



Uruchomienie nr 5



Uruchomienie nr 6



Analiza wyników:

Wynik jaki otrzymano w artykule**: f(x)= 2.850273253 dla x= 1.85052376133, 11 korekcji.** Wynik otrzymany, który jest najbardziej zbliżony: **f(x)= 2.850269230 dla x=1.850477004230, 9 korekcji.** Większość otrzymanych przez nas wyników jest zbliżona do tej wartości. Z dokładnością 4 miejsc po przecinku otrzymano dokładną wartość tej funkcji (oraz dokładną wartość otrzymaną w artykule), natomiast argument x różni się dopiero na 4 miejscu po przecinku. Taki rezultat otrzymaliśmy po 9 korelacjach przy 1200 iteracjach. Pozostałe parametry zostały przyjęte takie same jak w artykule (T = 5, k = 0.1, $\alpha(T) = 0.997$).

Dokonaliśmy pewnych eksperymentów z wartościami parametrów: przetestowane zostały różne przedziały losowania sąsiadów. Najlepsze zaobserwowane przez nas wyniki otrzymywane były dla przedziału (-0.1, 0.1). Zdecydowaliśmy się jednak na wykorzystanie wzoru, który jest bardziej optymalny dla różnych funkcji (wzór z wykładu 4), a nie tylko dla tego konkretnego przypadku. Również przy zwiększeniu liczby iteracji nasze wyniki były jeszcze bardziej zbliżone do poprawnych.

5. Wnioski końcowe

- Algorytm wymaga ustawienia kilku parametrów, takich jak temperatura początkowa T, współczynnik zmniejszania temperatury $\alpha(T)$, parametr k oraz liczba iteracji M. Wybór tych parametrów może mieć wpływ na efektywność algorytmu.
- Algorytm jest w stanie znaleźć globalne ekstremum nawet w przypadku funkcji, które zawierają wiele lokalnych ekstremów. Dzięki mechanizmowi akceptacji gorszych rozwiązań z pewnym prawdopodobieństwem na początku, algorytm jest w stanie uniknąć utknięcia w lokalnych minimach.
- Dla różnych funkcji celu i zestawów parametrów algorytm może zachowywać się inaczej, dlatego eksperymentacja i dostosowanie parametrów do konkretnej sytuacji mogą być konieczne.
- Wnioskiem ogólnym jest to, że algorytm symulowanego wyżarzania stanowi użyteczne narzędzie do rozwiązywania różnorodnych problemów optymalizacji, szczególnie tam, gdzie istnieje potrzeba znalezienia globalnych ekstremów w trudnych funkcjach.