Metaheurystyki i ich zastosowania 2023/24

Zadanie 5 – rój cząstek

Autorzy:
Michał Ferdzyn 242383
Artur Grzybek 242399

1. Zasada działania programu

Wybrane funkcje:

- W pliku *funkcje.py* zdefiniowane są dwie funkcje adaptacji: *ackley_function* i *mccormic_function*. Wybierana funkcja adaptacji jest przekazywana jako argument do programu.
- Ackley function:

$$f(x,y) = -20\exp\left[-0.2\sqrt{0.5\left(x^2+y^2\right)}\right] - \exp[0.5\left(\cos 2\pi x + \cos 2\pi y\right)] + e + 20$$

  Dla x i y:

- McCormic function:

$$f(x,y) = \sin(x+y) + (x-y)^2 - 1.5x + 2.5y + 1$$

  Dla x i y:

$$-5 \leq x, y \leq 5$$

Klasa „cząstka”:

- W pliku *Czastka.py* znajduje się klasa *Czastka*, która reprezentuje pojedynczą cząstkę w algorytmie PSO.
- Każda cząstka przechowuje swoją aktualną pozycję (x, y), prędkości w kierunku osi x i y,

$$-1.5 \leq x \leq 4,$$
$$-3 \leq y \leq 4$$

  wartość funkcji adaptacji dla aktualnej pozycji oraz najlepszą pozycję i adaptację, jakie osiągnęła dotychczas.

Główna pętla algorytmu:

- Początkowo generowana jest populacja cząstek o losowych pozycjach.
- Następnie program przechodzi przez określoną liczbę iteracji, w każdej z nich aktualizując pozycje, prędkości i wartości adaptacji cząstek.
- Na koniec każdej iteracji rysowany jest aktualny stan populacji na wykresie.

Parametry Algorytmu:

- ***LICZBA_ITERACJI:*** Liczba iteracji algorytmu.
- ***ROZMIAR_POPULACJI:*** Liczba cząstek w populacji.
- ***BEZWLADNOSC:*** Współczynnik bezwładności.

- ***STALA_POZNAWCZA i STALA_SPOLECZNA:*** Współczynniki wpływu doświadczenia najlepszej cząstki lokalnej i globalnej na ruch cząstki.
- ***DZIEDZINA:*** Zakres przeszukiwania przestrzeni funkcji.
- ***FUNKCJA_ADAPTACJI:*** Wybór funkcji adaptacji spośród ackley_function i mccormic_function.

Proces optymalizacji:

- Każda cząstka porusza się w przestrzeni poszukując maksimum funkcji adaptacji.
- W każdej iteracji aktualizowane są pozycje cząstek na podstawie ich prędkości i doświadczeń.
- Wykonywane jest porównanie aktualnych adaptacji cząstek z ich najlepszymi osiągnięciami.
- Na koniec program wypisuje najwyższą adaptację w populacji oraz szczegóły dla każdej cząstki.

Zasada działania algorytmu

Program implementuje algorytm optymalizacji rojem cząstek (Particle Swarm Optimization, PSO) w celu znalezienia maksimum globalnego funkcji adaptacji. Algorytm ten jest inspirowany zachowaniem roju zwierząt, gdzie każda cząstka porusza się w przestrzeni poszukując optymalnego rozwiązania. W kontekście programu, cząstki reprezentują potencjalne rozwiązania, a ich ruchy są kierowane przez ich własne doświadczenia oraz przez doświadczenia najlepszych cząstek w populacji.

2. Opisać wybrane miejsca implementacji rozwiązania.

Funkcje.py:

```
def ackley_function(x, y):
    return -(-20.0 * exp(-0.2 * sqrt(0.5 * (pow(x, 2) + pow(y, 2)))) -
exp(0.5 * (cos(2 * pi * x) + cos(
        2 * pi * y))) + e + 20)
```

- Funkcja przyjmuje dwie zmienne x i y jako argumenty.
- Wykorzystuje funkcje matematyczne z modułów numpy do obliczeń: exp: Oblicza funkcję eksponencjalną, sqrt: Oblicza pierwiastek kwadratowy, cos: Oblicza funkcję cosinus.
- Implementuje wzór funkcji Ackley, która jest znana ze swojej nieregularnej struktury i stosowana jest do testowania algorytmów optymalizacyjnych.

```
def mccormic_function(x, y):
    return sin(x + y) + pow(x - y, 2) - 1.5 * x + 2.5 * y + 1
```

- Funkcja również przyjmuje dwie zmienne x i y jako argumenty.
- Wykorzystuje funkcje matematyczne: sin: Oblicza funkcję sinus, pow: Podnosi liczbę do określonej potęgi.
- Implementuje wzór funkcji McCormick, znanej ze swojej nieregularnej i nietrywialnej do optymalizacji struktury.

Czastka.py:

```
class Czastka:
    def __init__(self, x, y, bezwladnosc, stala_poznawcza, stala_spoleczna,
funkcja_adaptacji, dziedzina):
        self.bezwladnosc = bezwladnosc
        self.stala_poznawcza = stala_poznawcza
        self.stala_spoleczna = stala_spoleczna
        self.funkcja_adaptacji = funkcja_adaptacji
        self.dziedzina = dziedzina

        self.predkosc_x = 0
        self.predkosc_y = 0

        self.x = x
        self.y = y
        self.aktualna_adaptacja = self.oblicz_adaptacje()

        self.najlepszy_x = x
        self.najlepszy_y = y
        self.najlepsza_adaptacja = self.aktualna_adaptacja
```

- Inicjalizacja: Konstruktor klasy *__init__* jest wywoływany podczas tworzenia nowego obiektu klasy.
- Przyjmuje on parametry, takie jak początkowe pozycje *x* i *y*, współczynniki algorytmu PSO, funkcję adaptacji i dziedzinę przeszukiwania.
- Inicjalizuje atrybuty obiektu, przypisując im wartości podane jako argumenty.
- Parametry: ***bezwladnosc:*** Współczynnik bezwładności cząstki, ***stala_poznawcza****:* Współczynnik wpływu doświadczenia najlepszej pozycji lokalnej, ***stala_spoleczna****:* Współczynnik wpływu doświadczenia najlepszej pozycji globalnej, ***funkcja_adaptacji****:* Funkcja adaptacji, której minimum szukamy, ***dziedzina:*** Zakres przeszukiwania przestrzeni dla cząstki.
- Prędkości i pozycje: Inicjalizowane są prędkości cząstki (*predkosc_x i predkosc_y*) oraz jej początkowe pozycje (*x i y*).
- Obliczenie Początkowej Wartości Funkcji Adaptacji: Metoda *oblicz_adaptacje* oblicza początkową wartość funkcji adaptacji dla cząstki.
- Najlepsze Pozycje i Adaptacja: Inicjalizowane są atrybuty *najlepszy_x, najlepszy_y* i *najlepsza_adaptacja* na podstawie początkowych pozycji i wartości funkcji adaptacji.

```
def oblicz_adaptacje(self):
    return self.funkcja_adaptacji(self.x, self.y)
```

- Metoda *oblicz_adaptacje* jest odpowiedzialna za obliczenie wartości funkcji adaptacji dla aktualnych współrzędnych cząstki.
- *self.funkcja_adaptacji* to funkcja adaptacji, która została dostarczona jako jeden z parametrów podczas inicjalizacji obiektu klasy Czastka. Ta funkcja adaptacji jest używana do oceny jak dobrze dana pozycja cząstki jest dopasowana do optymalnej wartości.
- Wartości *self.x i self.y* to aktualne współrzędne cząstki.
- Wywołanie *self.funkcja_adaptacji(self.x, self.y)* zwraca wartość funkcji adaptacji dla danej pozycji cząstki.
- Ta wartość funkcji adaptacji jest następnie zwracana przez metodę.

```
def aktualizuj_adaptacje(self):
    self.aktualna_adaptacja = self.oblicz_adaptacje()
    if self.aktualna_adaptacja > self.najlepsza_adaptacja:
```

```
        self.najlepsza_adaptacja = self.aktualna_adaptacja
        self.najlepszy_x = self.x
        self.najlepszy_y = self.y
```

- Metoda *aktualizuj_adaptacje* aktualizuje informacje dotyczące funkcji adaptacji cząstki.
- Pierwszym krokiem jest obliczenie nowej wartości funkcji adaptacji za pomocą metody *oblicz_adaptacje*, a następnie przypisanie jej do atrybutu *self.aktualna_adaptacja.*
- Następnie sprawdzane jest, czy aktualna wartość funkcji adaptacji *(self.aktualna_adaptacja)* jest większa od dotychczas najlepszej wartości funkcji adaptacji *(self.najlepsza_adaptacja).*
- Jeśli tak, to aktualizowane są informacje o najlepszej pozycji i adaptacji cząstki: *self.najlepsza_adaptacja* przyjmuje wartość *self.aktualna_adaptacja, self.najlepszy_x* przyjmuje aktualną wartość współrzędnej *self.x, self.najlepszy_y* przyjmuje aktualną wartość współrzędnej *self.y.*

```
def aktualizuj_predkosc(self, najlepszy_x_w_populacji,
najlepszy_y_w_populacji):
    self.predkosc_x = self.bezwladnosc * self.predkosc_x + \
                      (self.stala_spoleczna * random()) *
(najlepszy_x_w_populacji - self.x) + \
                      (self.stala_poznawcza * random()) *
(self.najlepszy_x - self.x)

    self.predkosc_y = self.bezwladnosc * self.predkosc_y + \
                      (self.stala_poznawcza * random()) *
(self.najlepszy_y - self.y) + \
                      (self.stala_spoleczna * random()) *
(najlepszy_y_w_populacji - self.y)
```

- Metoda *aktualizuj_predkosc* aktualizuje prędkości cząstki na podstawie wpływu najlepszych pozycji w populacji.
- Wzory na aktualizację prędkości opierają się na trzech składnikach:

a) ***Bezwładność (self.bezwladnosc):*** Reprezentuje tendencję cząstki do kontynuowania swojego ruchu z dotychczasową prędkością. *self.bezwladnosc * self.predkosc_x i self.bezwladnosc * self.predkosc_y* odpowiadają składnikowi bezwładności w kierunkach x i y.

b) ***Wpływ najlepszej pozycji lokalnej (self.stala_poznawcza * random() * (self.najlepszy_x - self.x) i self.stala_poznawcza * random() * (self.najlepszy_y - self.y)):*** Reprezentuje tendencję cząstki do poruszania się w kierunku swojej dotychczasowej najlepszej pozycji. *(self.najlepszy_x - self.x) i (self.najlepszy_y - self.y)* to różnice między aktualną pozycją a najlepszą pozycją.

c) ***Wpływ najlepszej pozycji globalnej (self.stala_spoleczna * random() * (najlepszy_x_w_populacji - self.x) i self.stala_spoleczna * random() * (najlepszy_y_w_populacji - self.y)):*** Reprezentuje tendencję cząstki do poruszania się w kierunku najlepszej pozycji w całej populacji. *(najlepszy_x_w_populacji - self.x) i (najlepszy_y_w_populacji - self.y)* to różnice między aktualną pozycją a najlepszą pozycją w populacji.

- Wszystkie te składniki są uwzględniane w obliczeniach nowych predkości cząstki w kierunkach x i y.
- Metoda wykorzystuje funkcję *random()* do generowania losowych wartości, co wprowadza losowy element w proces aktualizacji prędkości.

```
def aktualizuj_pozycje(self):
    if self.x + self.predkosc_x < self.dziedzina[0]:
        self.x = self.dziedzina[0]
    elif self.x + self.predkosc_x > self.dziedzina[1]:
        self.x = self.dziedzina[1]
    else:
        self.x += self.predkosc_x

    if self.y + self.predkosc_y < self.dziedzina[0]:
        self.y = self.dziedzina[0]
    elif self.y + self.predkosc_y > self.dziedzina[1]:
        self.y = self.dziedzina[1]
    else:
        self.y += self.predkosc_y
```

- Metoda *aktualizuj_pozycje* aktualizuje współrzędne cząstki na podstawie jej predkości.
- W pierwszej części (dotyczącej współrzędnej x), sprawdzane są warunki graniczne dla nowej pozycji. Jeśli przemieszczenie po osi x w wyniku predkości spowoduje wyjście poza zakres określony przez *self.dziedzina[0]* (dolna granica) lub *self.dziedzina[1]* (górna granica), to pozycja jest ustawiana na odpowiednią granicę. W przeciwnym razie pozycja x jest aktualizowana o wartość predkości.
- Analogicznie, w drugiej części (dotyczącej współrzędnej y), sprawdzane są warunki graniczne dla nowej pozycji na osi y, i pozycja jest aktualizowana analogicznie.
- Dzięki temu, cząstka nie opuszcza określonej dziedziny przeszukiwania, co jest istotne w kontekście algorytmu optymalizacji, aby utrzymać poprawność rozwiązania.

Main.py:

```
if __name__ == '__main__':
    LICZBA_ITERACJI = 30
    ROZMIAR_POPULACJI = 50
    BEZWLADNOSC = 0.4
    STALA_POZNAWCZA = 0.3
    STALA_SPOLECZNA = 0.6
    DZIEDZINA = [-5, 5]
    FUNKCJA_ADAPTACJI = ackley_function

    czastki = [Czastka(uniform(DZIEDZINA[0], DZIEDZINA[1]),
                       uniform(DZIEDZINA[0], DZIEDZINA[1]),
                       BEZWLADNOSC,
                       STALA_POZNAWCZA,
                       STALA_SPOLECZNA,
                       FUNKCJA_ADAPTACJI, DZIEDZINA)
               for _ in range(ROZMIAR_POPULACJI)]

    for _ in tqdm(range(LICZBA_ITERACJI), ncols=200):
        najlepsza_aktualna_adaptacja = czastki[0].aktualna_adaptacja
        najlepszy_x = czastki[0].x
        najlepszy_y = czastki[0].y

        for czastka in czastki:
            if czastka.aktualna_adaptacja > najlepsza_aktualna_adaptacja:
                najlepsza_aktualna_adaptacja = czastka.aktualna_adaptacja
                najlepszy_x = czastka.x
                najlepszy_y = czastka.y

        for czastka in czastki:
            czastka.aktualizuj_predkosc(najlepszy_x, najlepszy_y)
```

```
            czastka.aktualizuj_pozycje()
            czastka.aktualizuj_adaptacje()
        rysuj_wykres(DZIEDZINA, czastki, FUNKCJA_ADAPTACJI)
```

- Pętla główna wykonuje się przez określoną liczbę iteracji (*LICZBA_ITERACJI*).
- Wewnątrz pętli obliczane są najlepsze aktualne wartości adaptacji oraz odpowiadające im współrzędne dla całej populacji.
- Następnie dla każdej cząstki w populacji aktualizowane są predkość, pozycja i wartość adaptacji na podstawie najlepszych wartości znalezionych w populacji.
- Po każdej iteracji wywoływana jest funkcja *rysuj_wykres*, która rysuje wykres trójwymiarowy przedstawiający położenie cząstek w przestrzeni przeszukiwania.

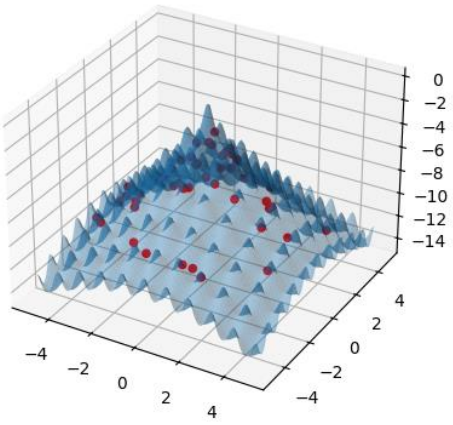
3. Analiza otrzymanych wyników

Funkcja Ackley

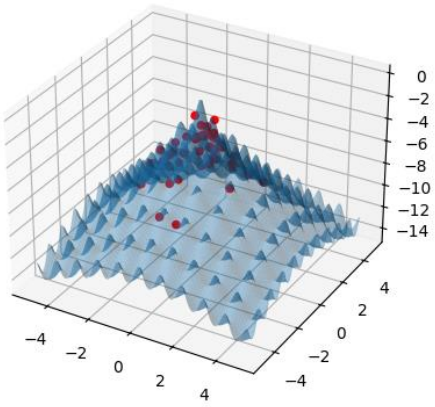
1 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: <-5, 5>

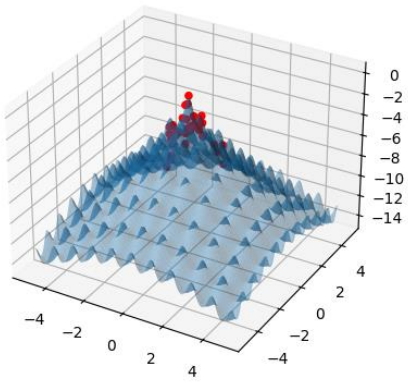
1 iteracja:



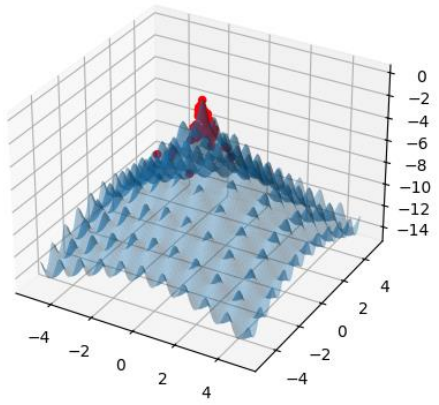
2 iteracja:



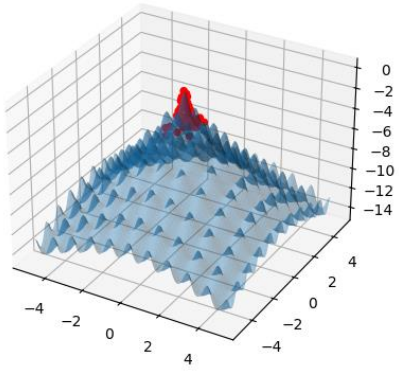
3 iteracja:



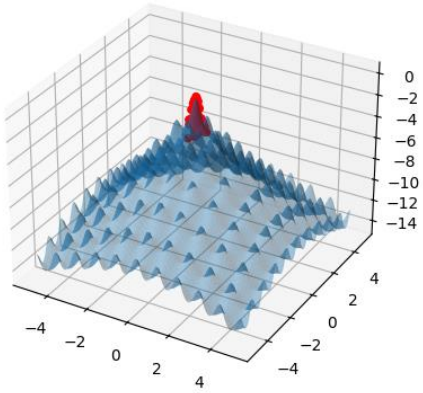
4 iteracja:



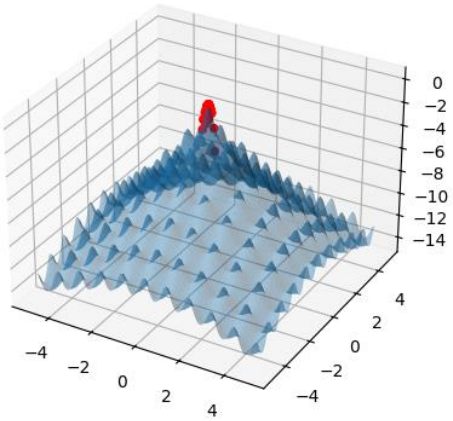
5 iteracja:



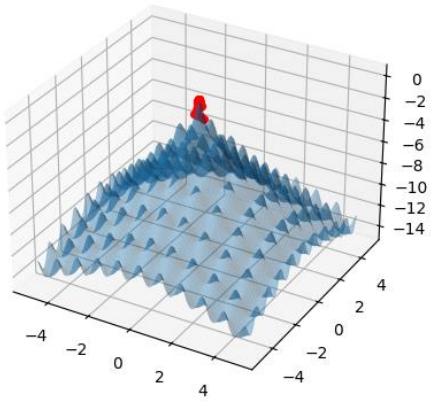
6 iteracja:



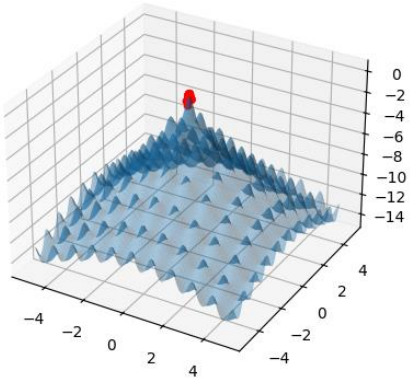
7 iteracja:



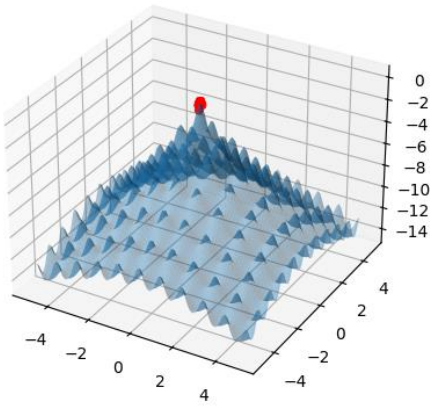
8 iteracja:



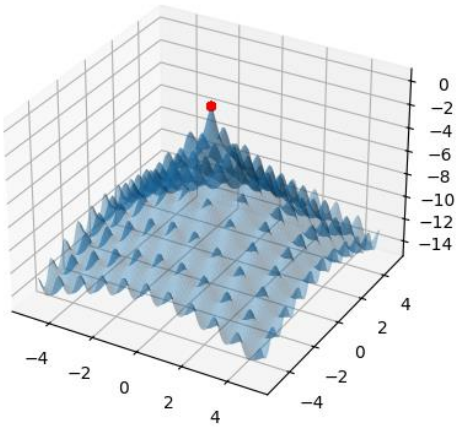
9 iteracja:



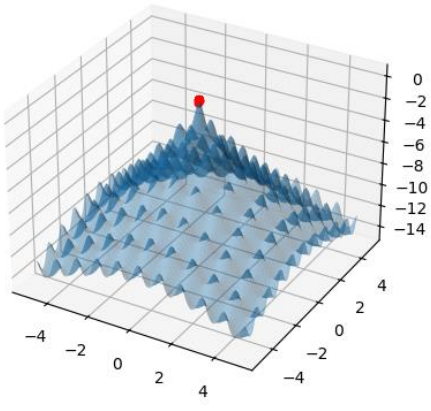
10 iteracja:



11 iteracja:



12 iteracja:



W następnych iteracjach otrzymany wynik już nie ulegał zmianie. Ostateczne wyniki:

- Najlepsza adaptacja = -1.1138855 · 10^{-6}

Wnioski

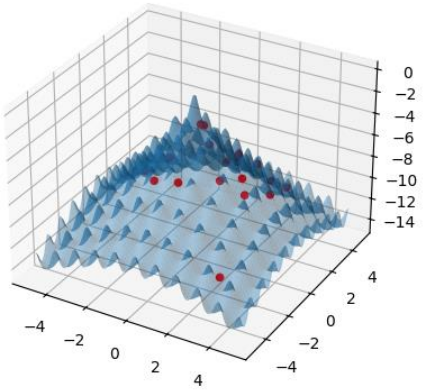
Zauważamy, że około 12 iteracji wynik nie ulega już zmianie. Tym samym liczba iteracji nie musi wynosić dla tej dziedziny, aż 30 iteracji. Dobrym rozwiązaniem byłoby też zastosowanie mechanizmu zatrzymywania, aby przerwać algorytm po osiągnięciu odpowiedniego poziomu optymalizacji.

Badanie wpływu rozmiaru populacji:

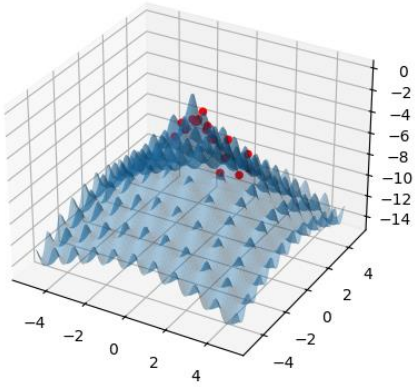
2 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 20
- Inercja (bezwładność): 0.4
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: <-5, 5>

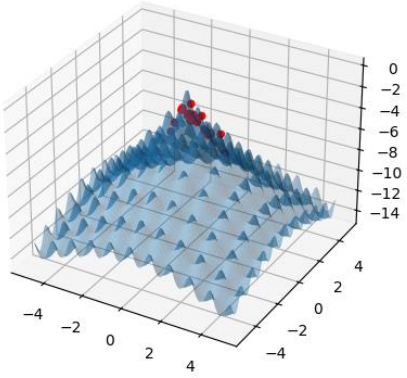
1 iteracja:



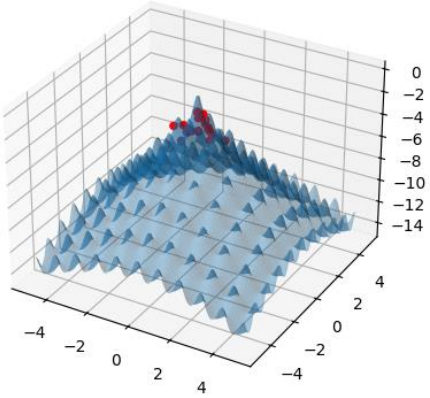
2 iteracja:



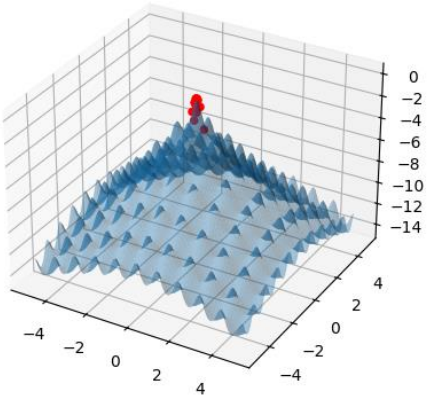
3 iteracja:



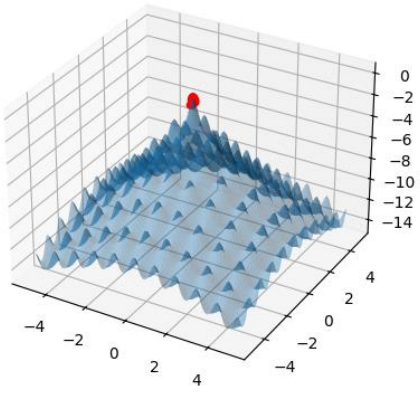
4 iteracja:



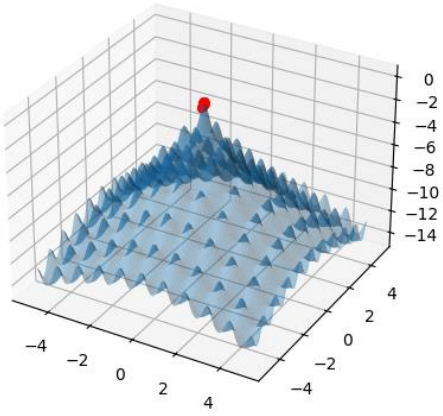
5 iteracja:



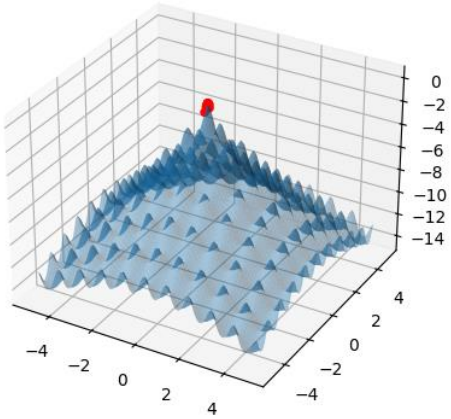
6 iteracja:



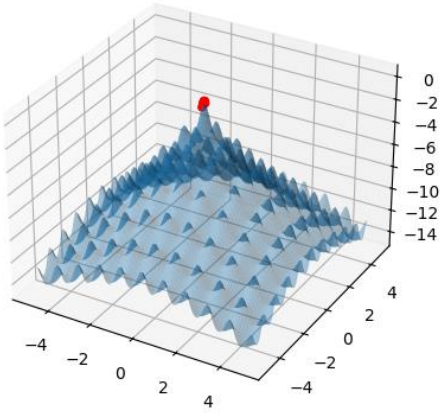
7 iteracja:



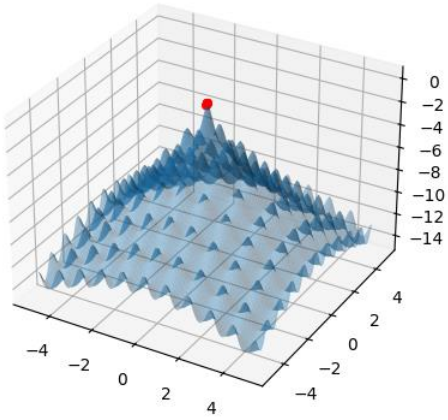
8 iteracja:



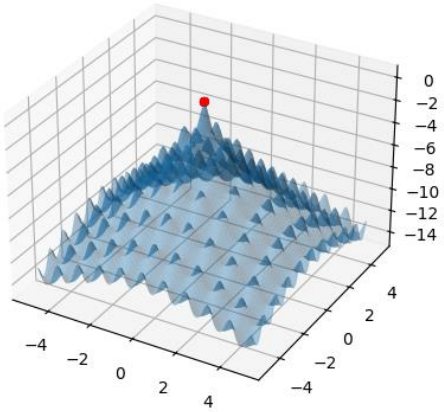
9 iteracja:



10 iteracja:



11 iteracja:



Już od około 11 iteracji wynik nie ulegał już zmianie.

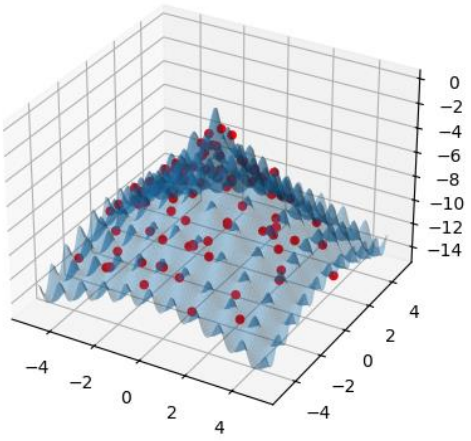
Ostateczne wyniki:

- Najlepsza adaptacja = -1.390042 $\cdot 10^{-5}$

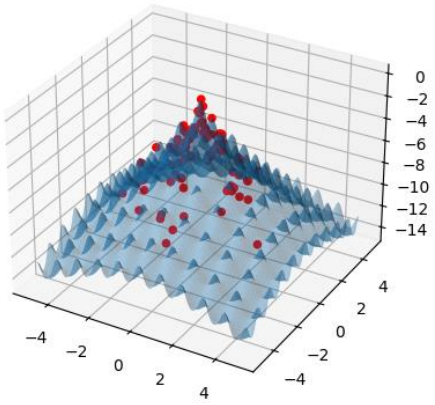
3 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 100
- Inercja (bezwładność): 0.4
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: <-5, 5>

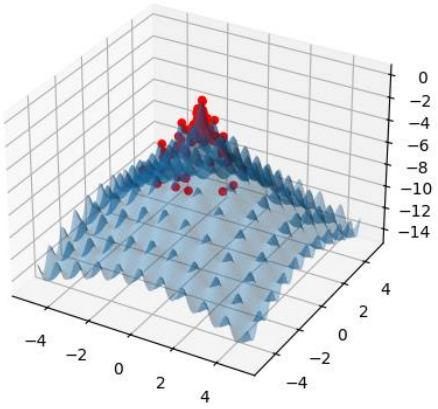
1 iteracja:



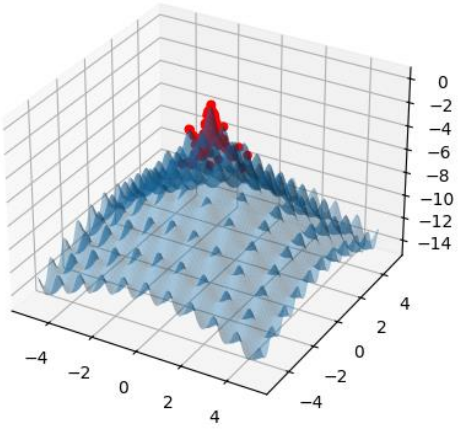
2 iteracja:



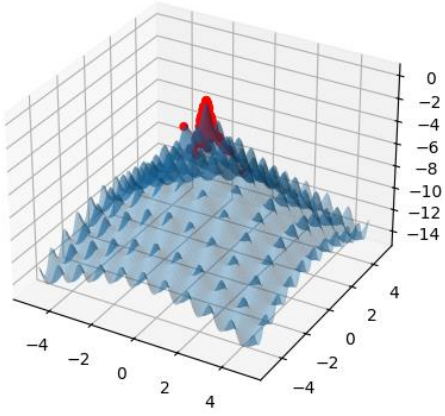
3 iteracja:



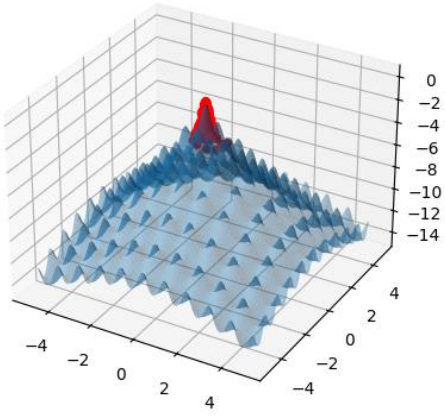
4 iteracja:



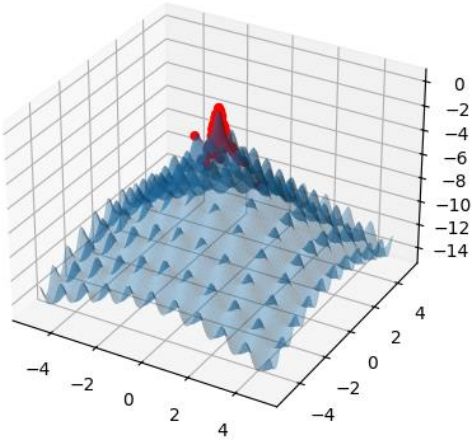
5 iteracja:



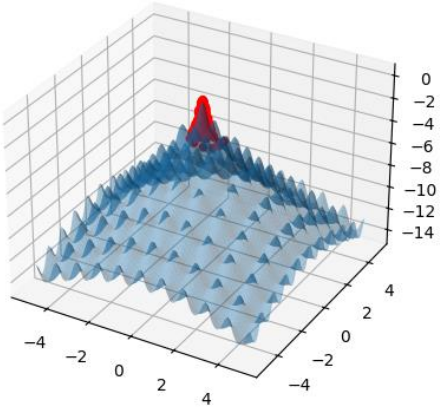
6 iteracja:



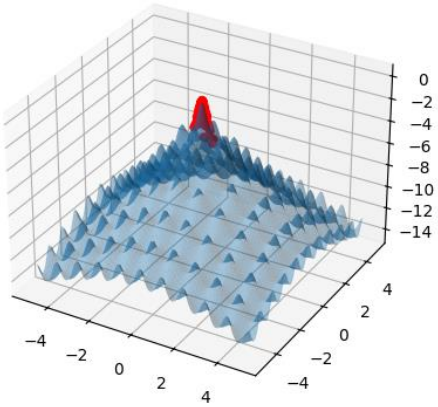
7 iteracja:



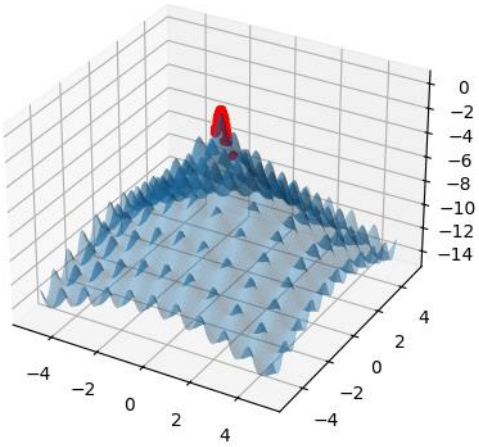
8 iteracja:



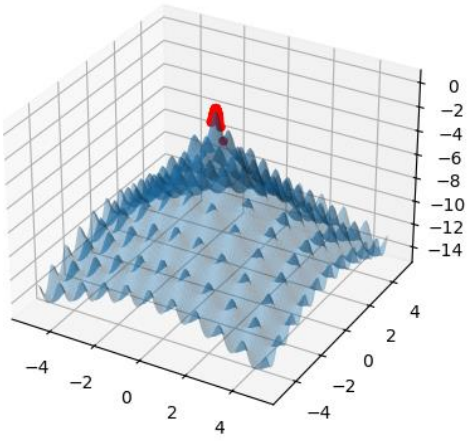
9 iteracja:



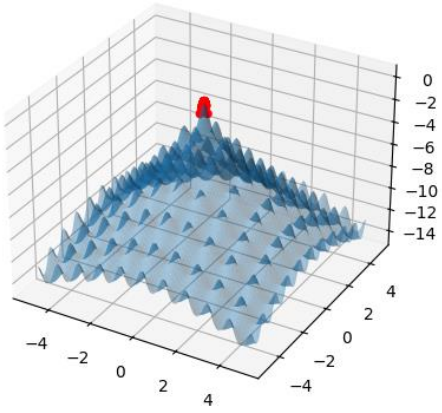
10 iteracja:



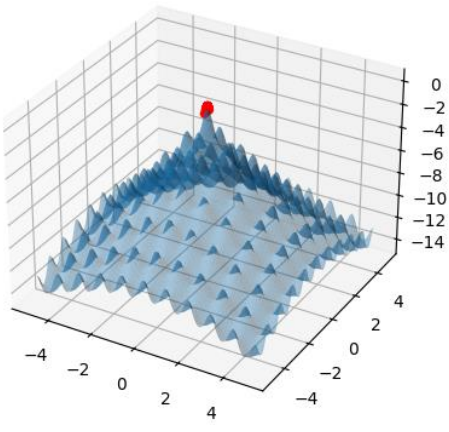
11 iteracja:



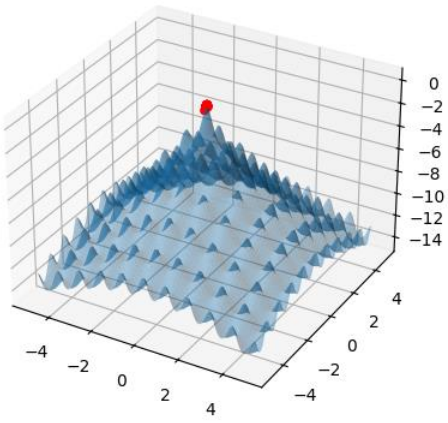
12 iteracja:



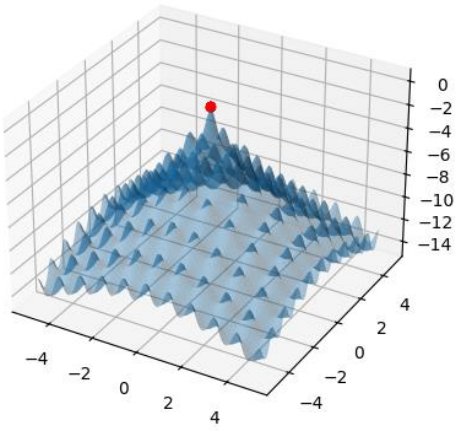
13 iteracja:



14 iteracja:



15 iteracja:



W późniejszych iteracjach wynik już nie ulegał zmianie.

Ostateczne wyniki:

- Najlepsza adaptacja = -7.476310 · 10^{-7}

Wnioski

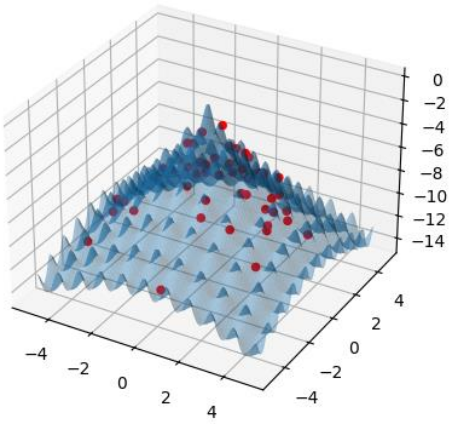
Zauważamy, że wraz ze wzrostem ilości cząstek wydłuża się czas obliczeń naszego programu. Używając dużej ilości cząstek (trzecie uruchomienie algorytmu - 100 cząstek) widoczne jest, że ilość iteracji, po których wynik się stabilizował wzrósł. Wnioskiem jest, że większa liczba cząstek zwykle powoduje, że przyciąganie do globalnego najlepszego rozwiązanie trwa dłużej, ponieważ więcej cząstek jest przyciąganych do lokalnie najlepszych rozwiązań. Najlepszy wynik został odnotowany dla 100 cząstek i wynosi: -7.476310 · 10^{-7}

Badanie wpływu inercji:

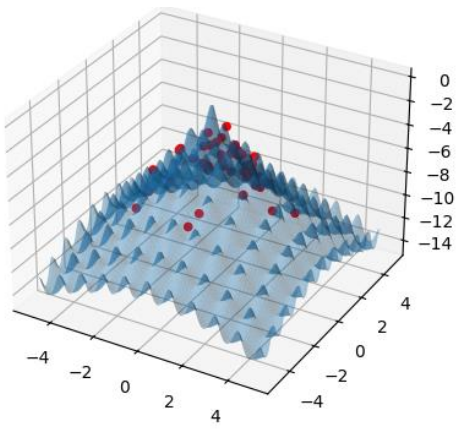
4 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.1
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: <-5, 5>

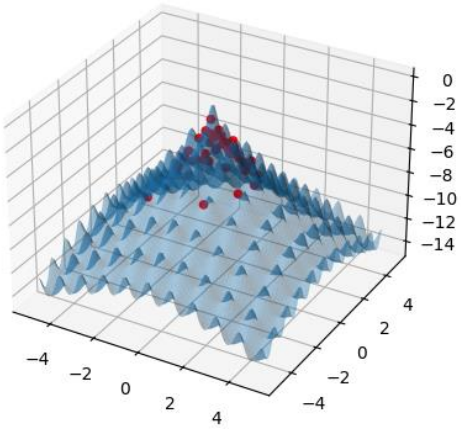
1 iteracja:



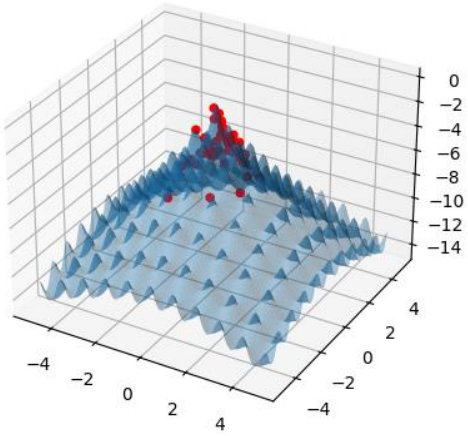
2 iteracja:



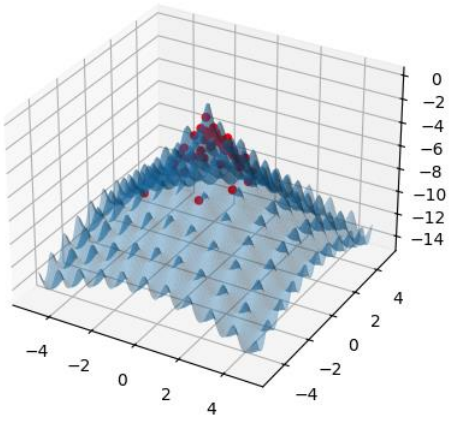
3 iteracja:



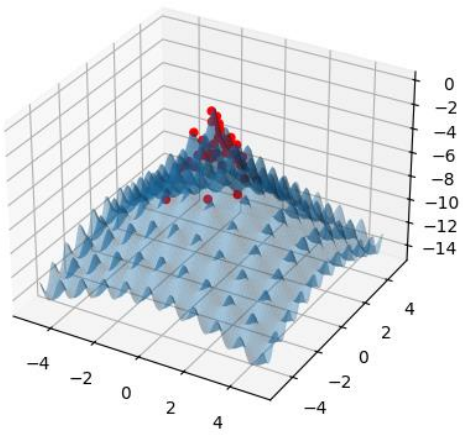
4 iteracja:



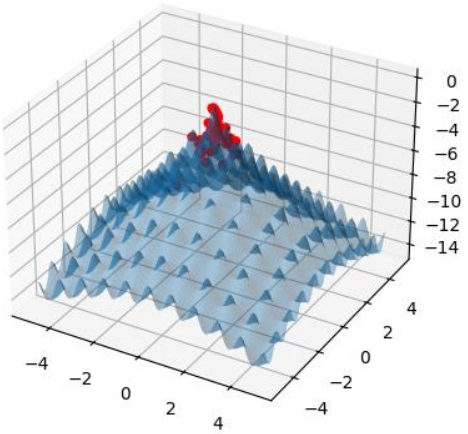
5 iteracja:



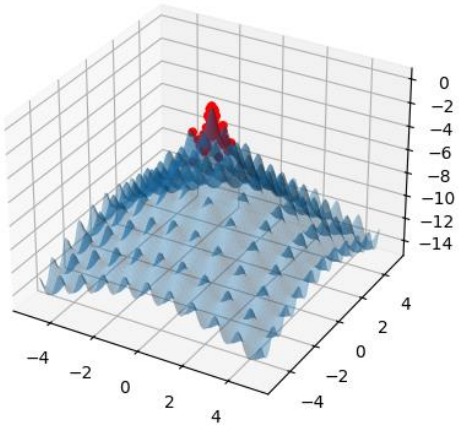
6 iteracja:



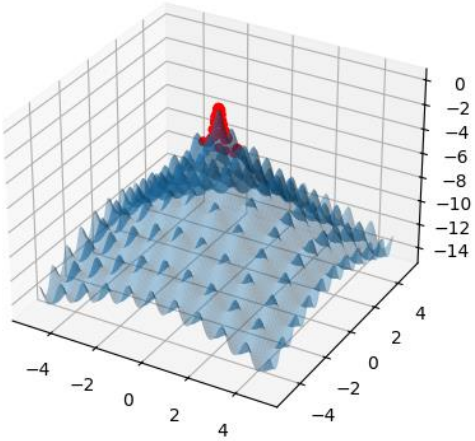
7 iteracja:



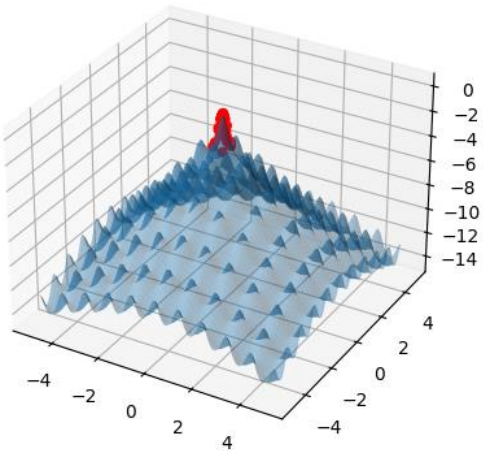
8 iteracja:



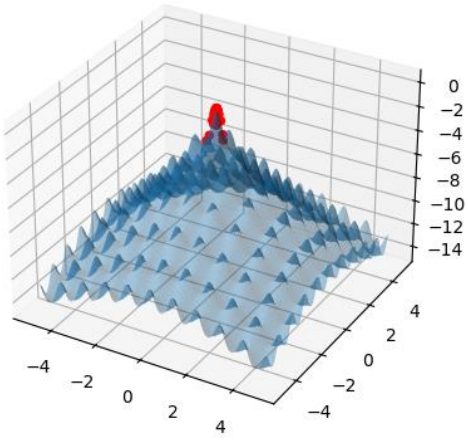
9 iteracja:



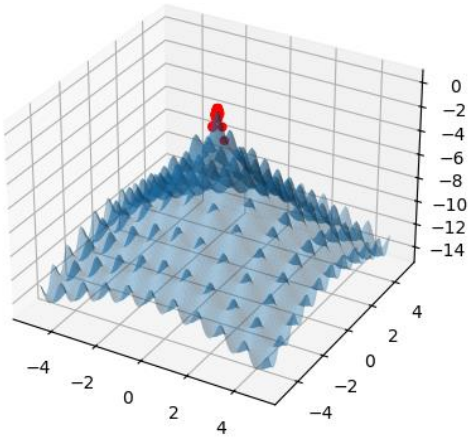
10 iteracja:



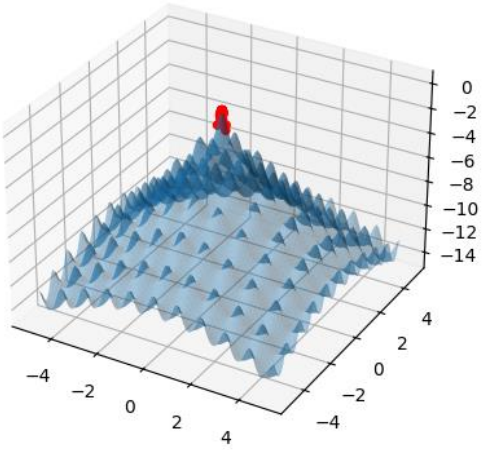
11 iteracja:



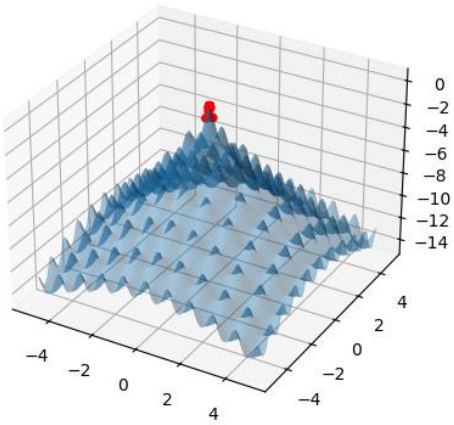
12 iteracja:



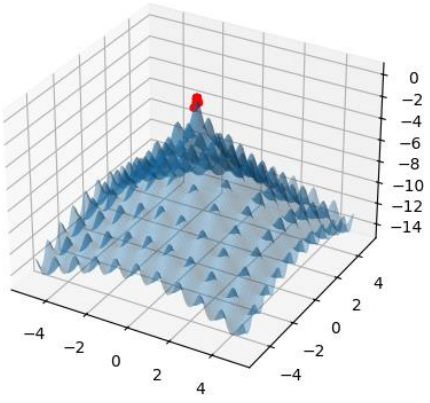
13 iteracja:



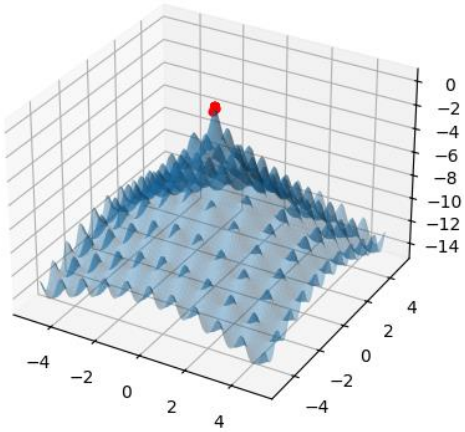
14 iteracja:



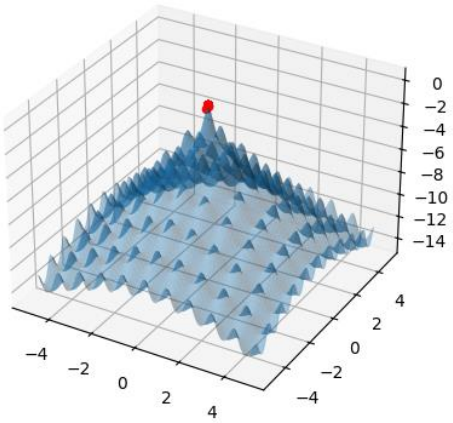
15 iteracja:



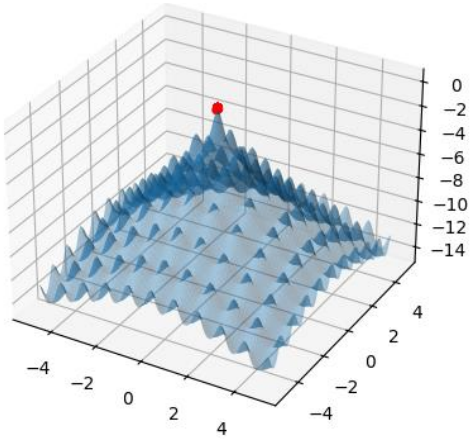
16 iteracja:



17 iteracja:



18 iteracja:



W następnych iteracjach wynik nie ulegał zmianie.

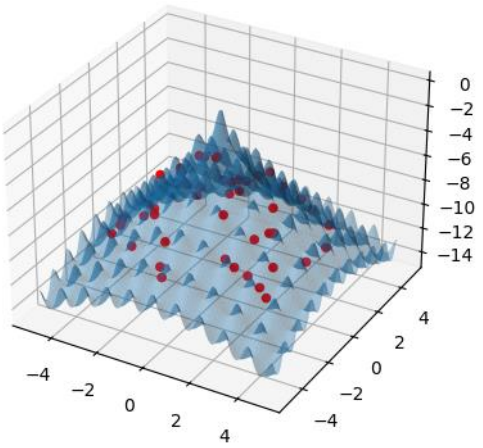
Ostateczne wyniki:

- Najlepsza adaptacja: -3.891980 · 10^{-6}

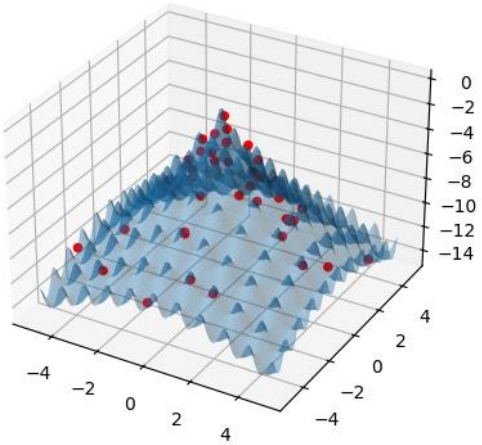
5 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.9
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: <-5, 5>

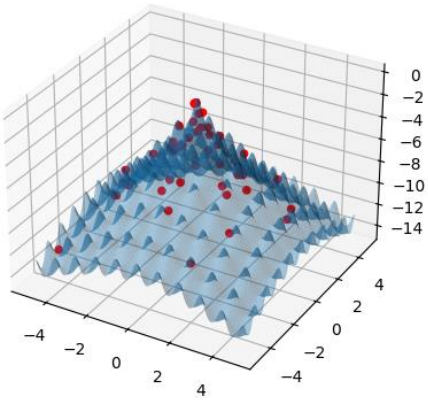
1 iteracja:



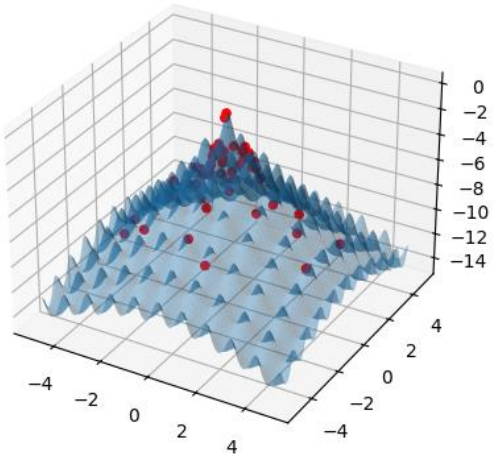
4 iteracja:



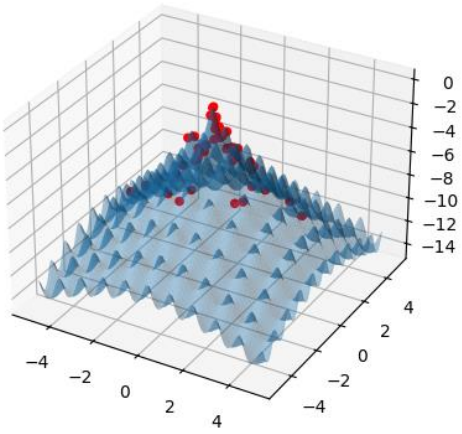
8 iteracja:



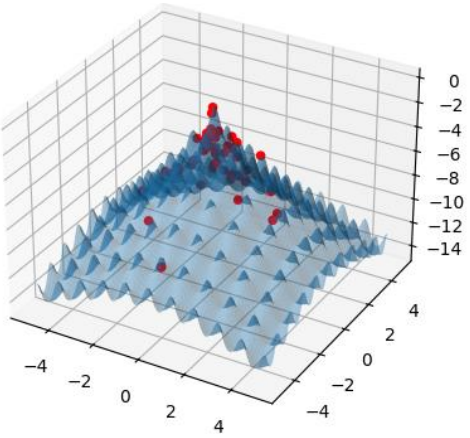
12 iteracja:



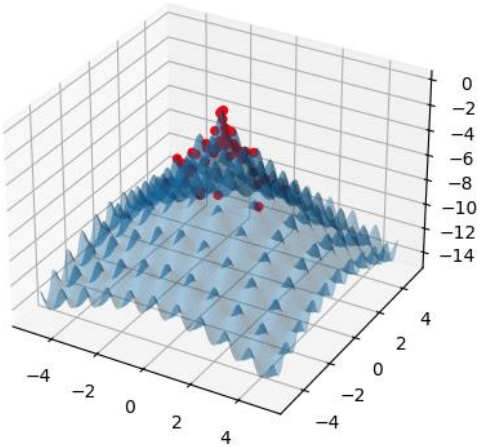
16 iteracja:



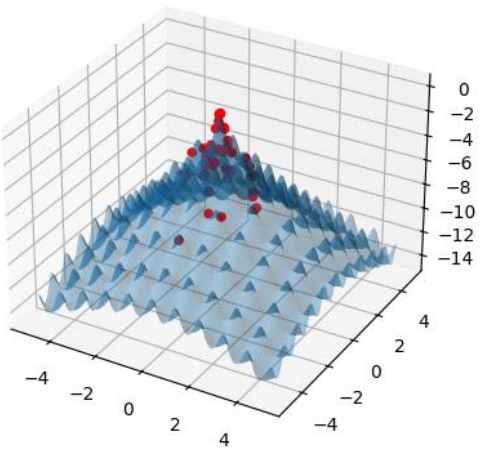
20 iteracja:



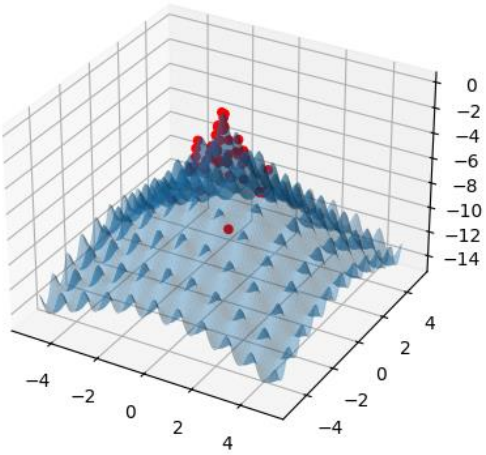
24 iteracja:



28 iteracja:



30 iteracja:



Ostateczne wyniki:

- Najlepsza adaptacja: -0.066075

Wnioski

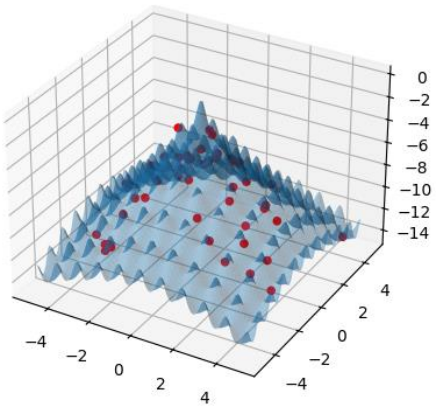
Zauważamy, że wraz ze wzrostem współczynnika inwersji znacząco pogarsza się wartość najlepszej osiągniętej adaptacji. Na wykresach zauważamy, że w przypadku wysokiej wartości tej zmiennej, większość cząstek po skończonej liczbie iteracji nie zbliżyło się do optymalnego rozwiązania. Spowodowane jest to tym, że wartość współczynnika inercji, która jest zbliżona do 1 powoduje szybszą globalną eksploracje w mniejszej ilości iteracji, natomiast wartość zbliżona do 0 sprzyja lokalnej eksploracji i może wymagać większej ilości iteracji.

Badanie wpływu stałej poznawczej:

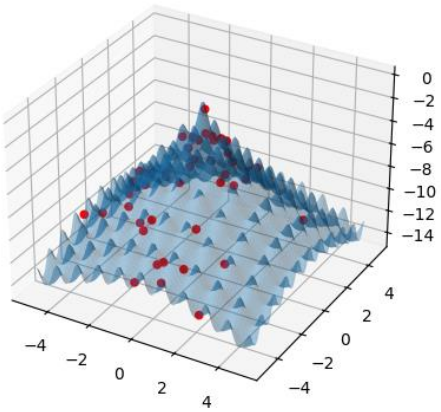
6 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała(komponent) poznawcza: 0.1
- Stała społeczna: 0.6
- Badana dziedzina: <-5, 5>

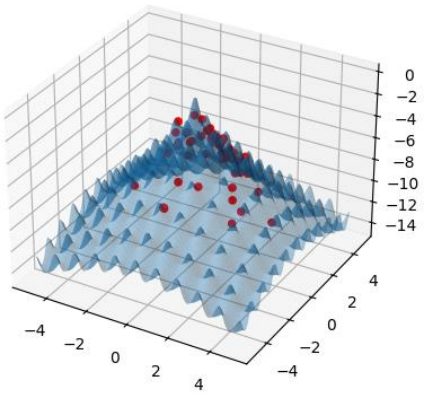
1 iteracja:



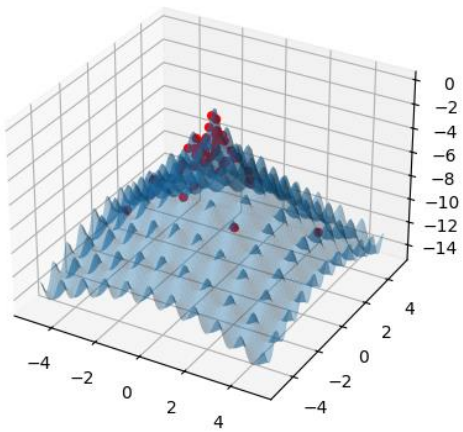
4 iteracja:



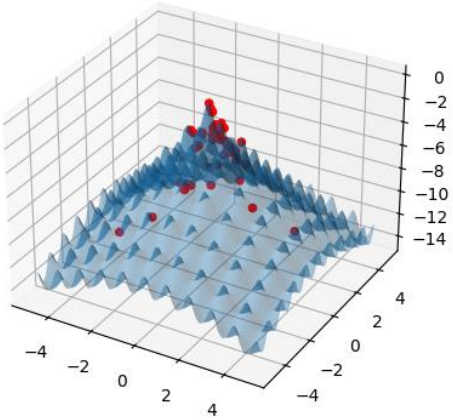
8 iteracja:



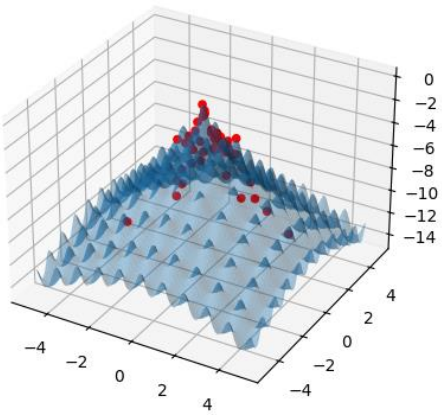
12 iteracja:



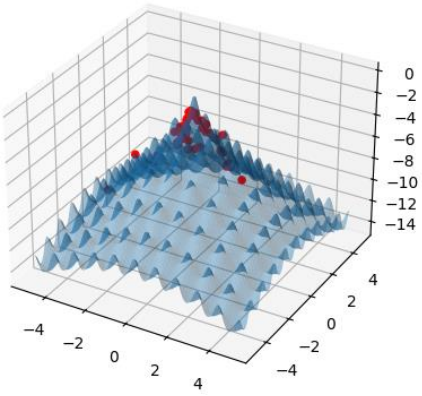
16 iteracja:



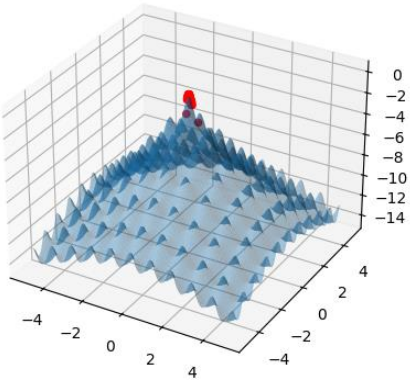
20 iteracja:



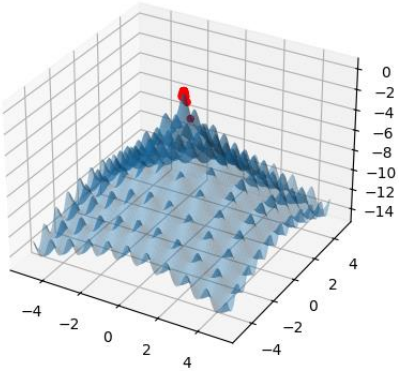
24 iteracja:



28 iteracja:



30 iteracja:



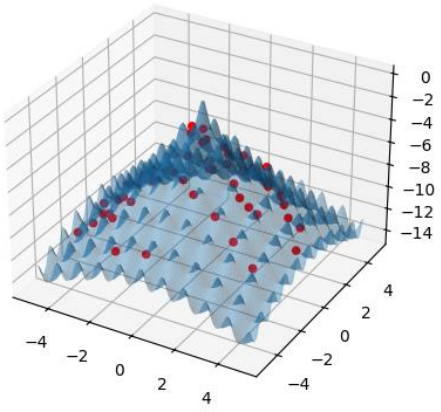
Ostateczne wyniki:

- Najlepsza adaptacja: -3.04966 · 10^{-6}

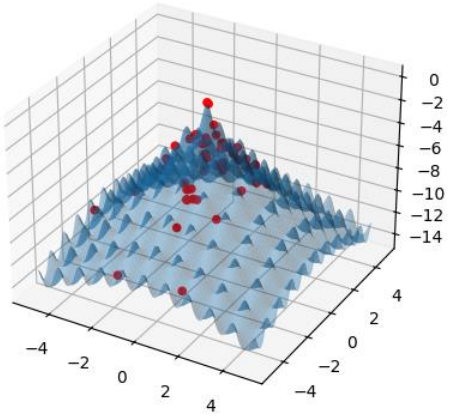
7 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała(komponent) poznawcza: 1.8
- Stała społeczna: 0.6
- Badana dziedzina: <-5, 5>

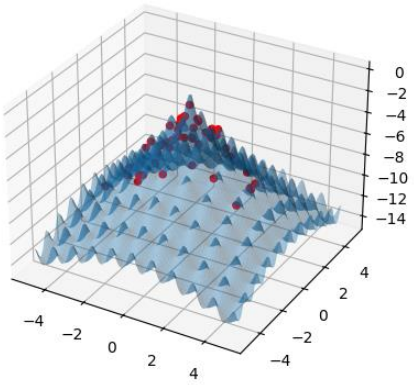
1 iteracja:



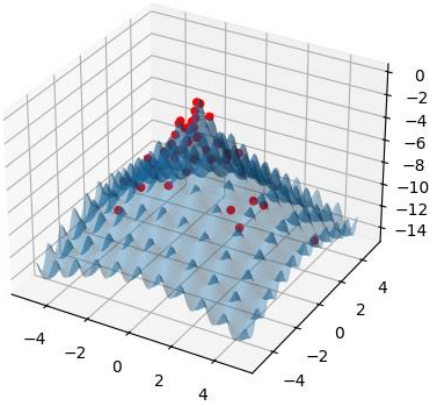
2 iteracja:



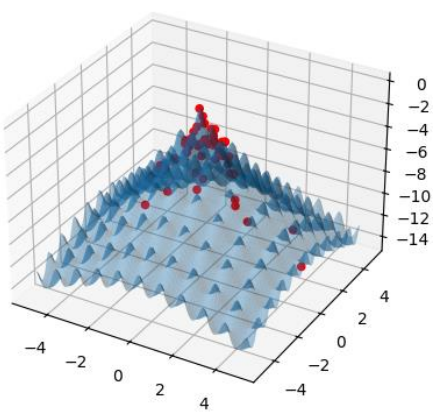
3 iteracja:



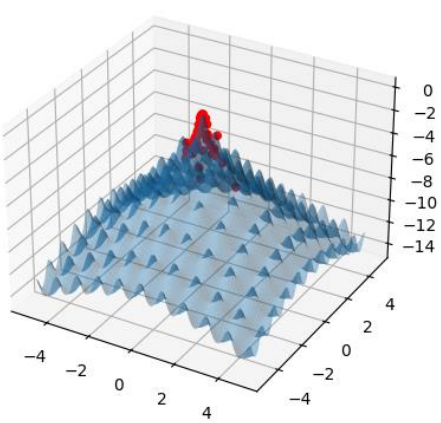
4 iteracja:



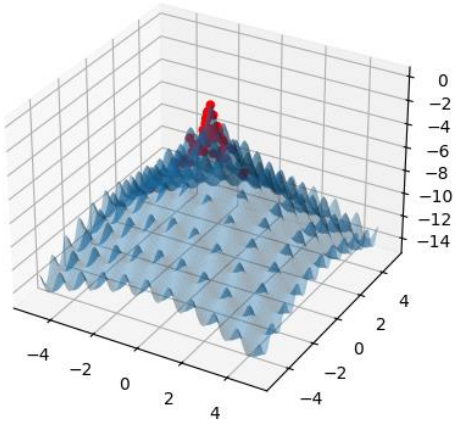
5 iteracja:



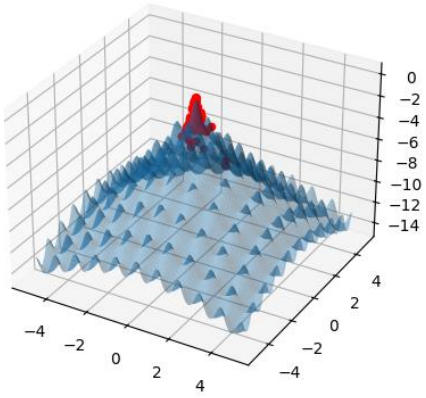
6 iteracja:



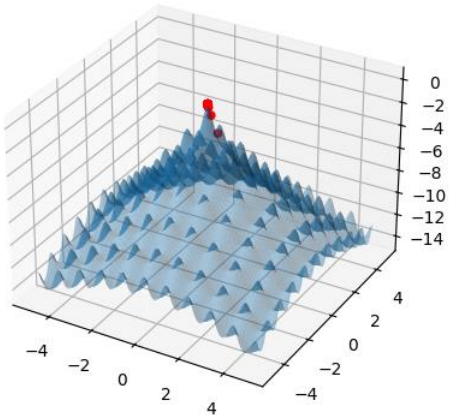
7 iteracja:



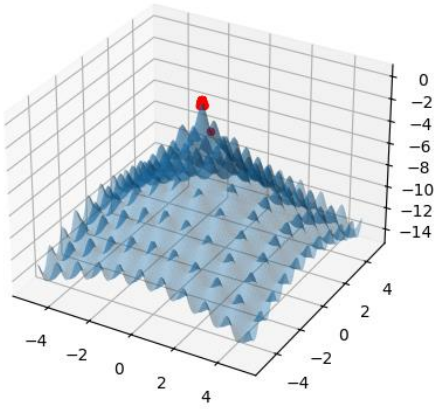
8 iteracja:



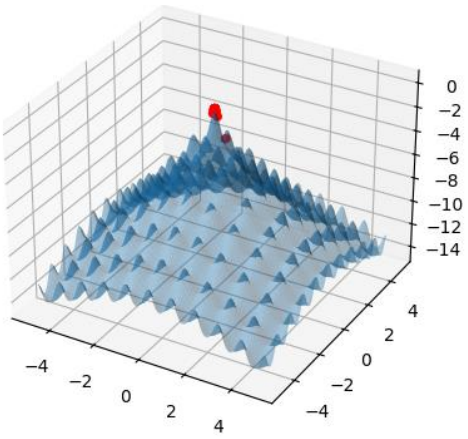
9 iteracja:



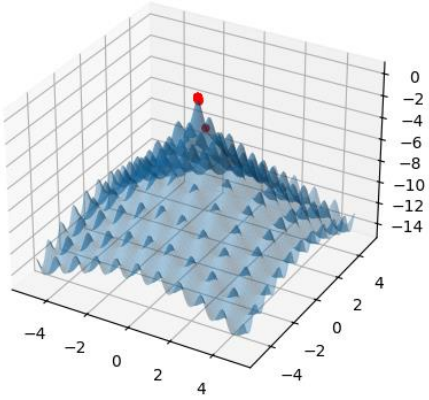
10 iteracja:



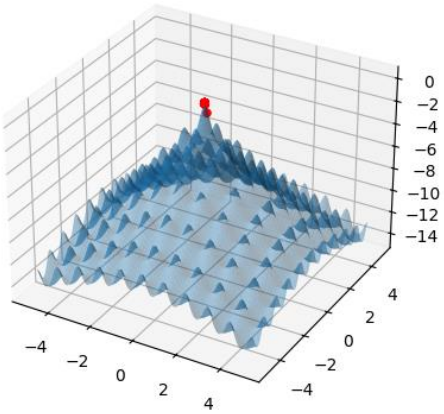
11 iteracja:



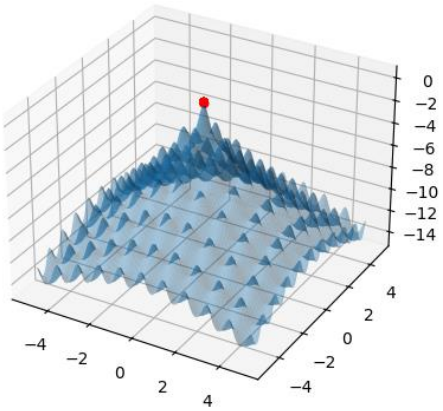
12 iteracja:



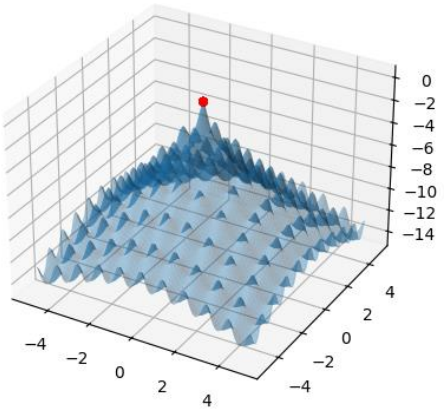
13 iteracja:



14 iteracja:



15 iteracja:



W następnych iteracjach wynik nie ulegał zmianie.

Ostateczne wyniki:

- Najlepsza adaptacja: -1.264965 · 10^{-6}

Wnioski:

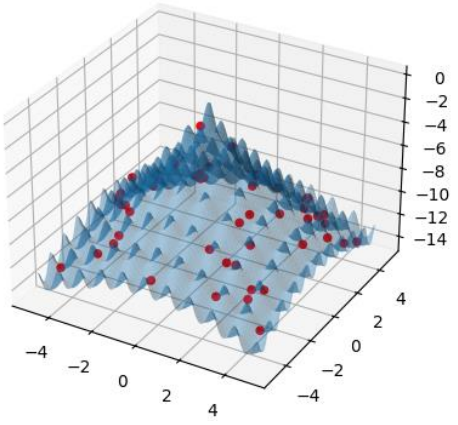
Zauważamy, że przy bardzo niskiej wartości komponentu poznawczego (równego np. 0.1), niektóre cząstki nawet po skończonej liczbie iteracji są w pewnym stopniu oddalone od optymalnego wyniku. Przy wysokiej wartości tego współczynnika cząstki szybciej „docierają" do optymalnego rozwiązana (ponieważ już od około 15 iteracji wynik nie ulegał znaczącej poprawie). Wyższa wartość tego współczynnika powoduje również wyższą eksploatacje. Ostatecznie minimalną lepszą adaptacje uzyskaliśmy w przypadku wyższej wartości tego współczynnika.

Badanie wpływu stałej społecznej:

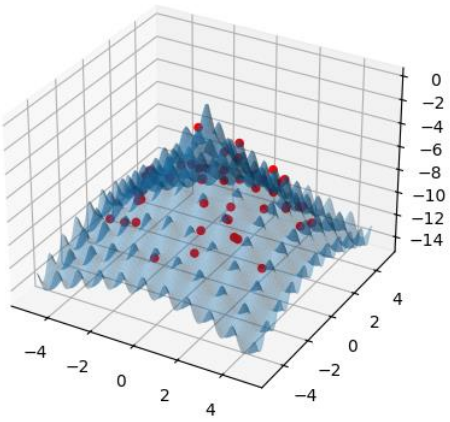
8 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała(komponent) poznawcza: 0.3
- Stała społeczna: 0.1
- Badana dziedzina: <-5, 5>

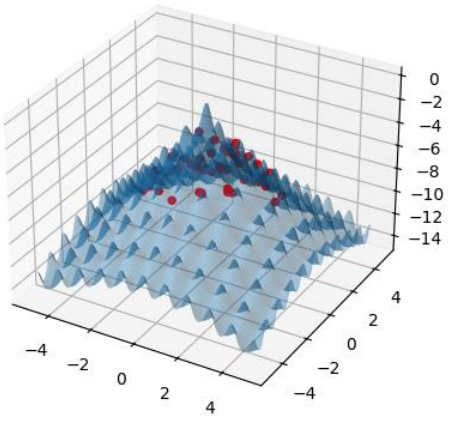
1 iteracja:



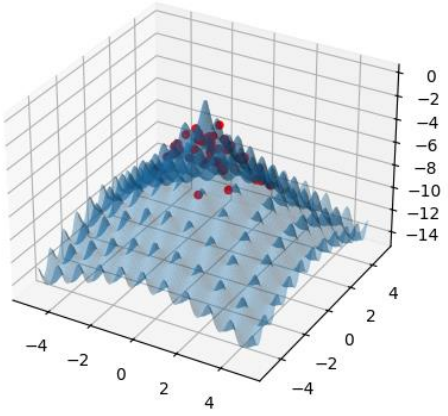
5 iteracja:



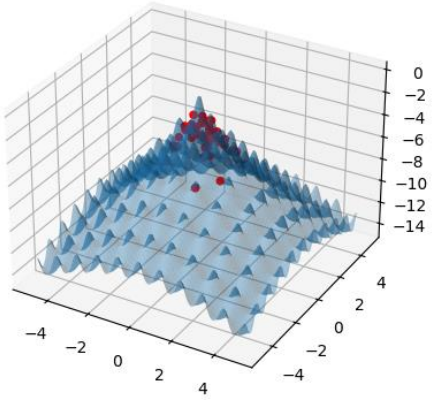
10 iteracja:



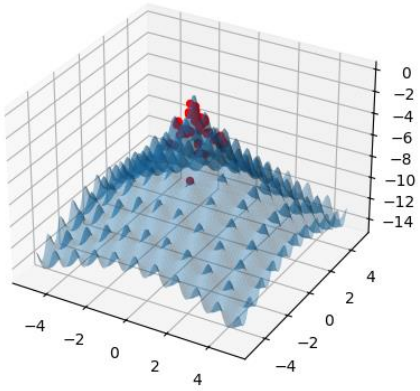
15 iteracja:



20 iteracja:



30 iteracja:



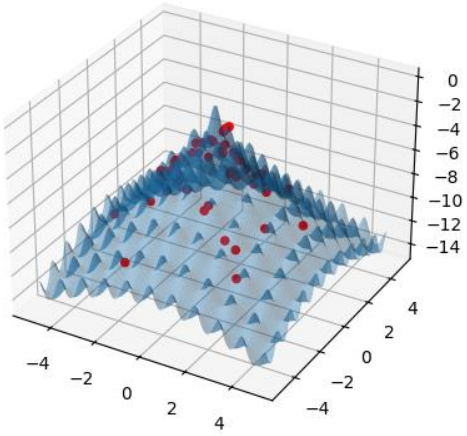
Ostateczne wyniki:

- Najlepsza adaptacja: -1.09796

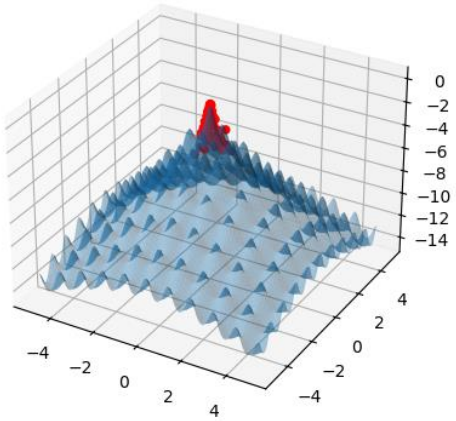
9 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała(komponent) poznawcza: 0.3
- Stała społeczna: 0.9
- Badana dziedzina: <-5, 5>

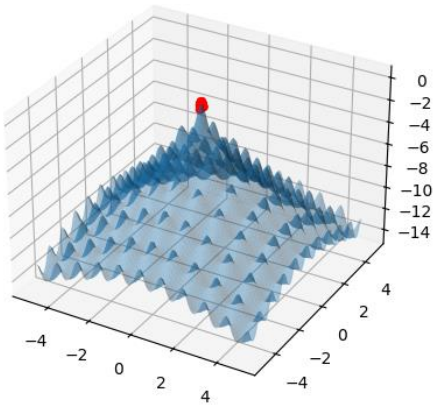
1 iteracja:



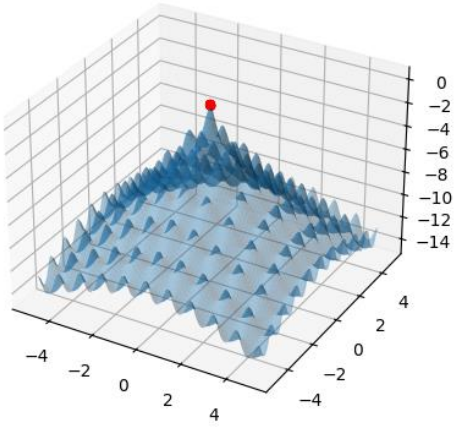
5 iteracja:



10 iteracja:



13 iteracja:



W kolejnych iteracjach wynik nie ulegał znaczącemu polepszeniu.

Ostateczne wyniki:

- Najlepsza adaptacja: -1.92232 · 10^{-6}

Wnioski

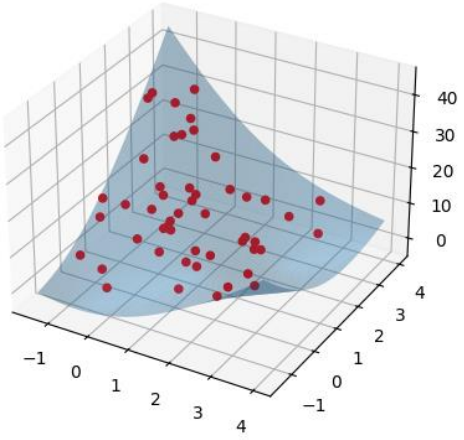
Zauważamy, że w przypadku bardzo niskiej wartości komponentu społecznego (przykładowo wartość = 0.1) to nawet po skończonej liczbie iteracji, niektóre cząstki nie zbliżyły się do ekstremum. Przy użyciu zwiększeniu tej wartości obserwujemy, że wszystkie cząstki bardzo szybko znalazły zbliżone rozwiązanie. Komponent społeczny określa jak bardzo dana cząstka powinna być „zainspirowana” przez sąsiadów. Znacząco lepszą adaptacje otrzymaliśmy w przypadku wysokiej wartości tego współczynnika.

Funkcja McCormick

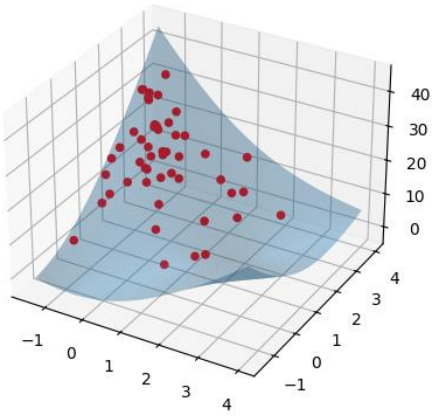
1 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: -1.5 <= x <=4, -3 <= y <= 4

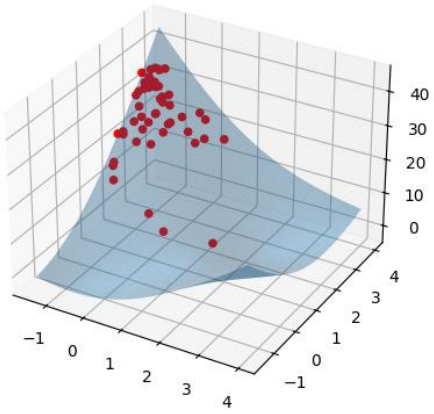
1 iteracja:



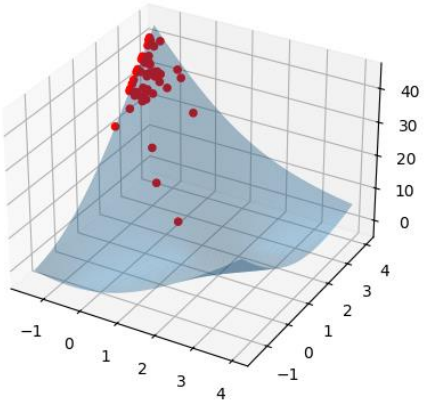
2 iteracja:



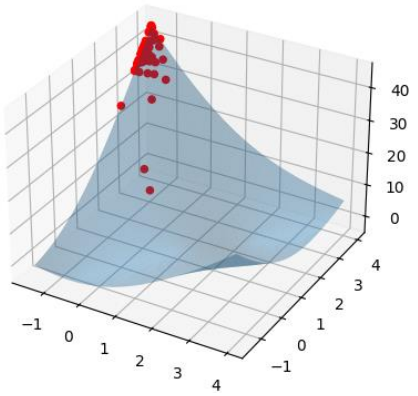
3 iteracja:



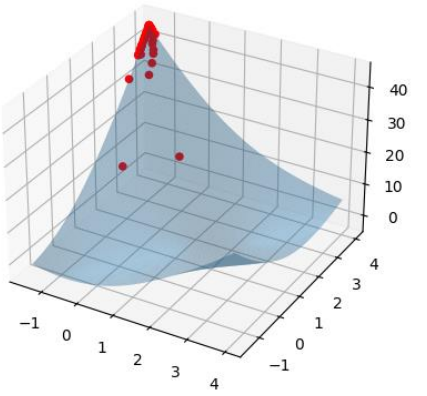
4 iteracja:



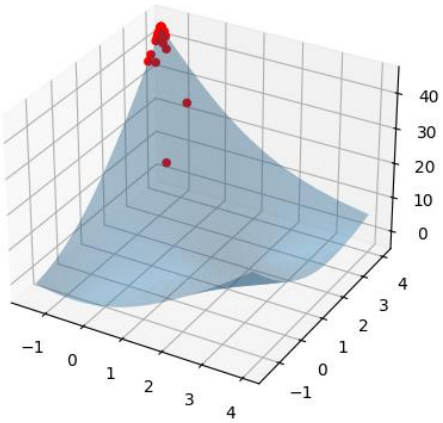
5 iteracja:



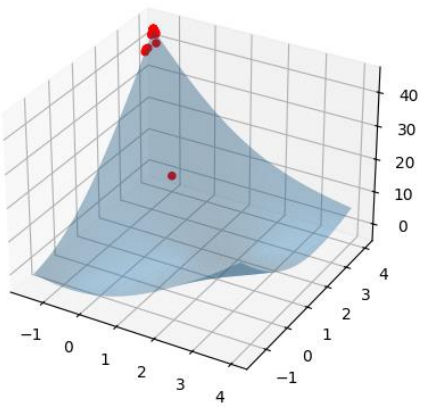
6 iteracja:



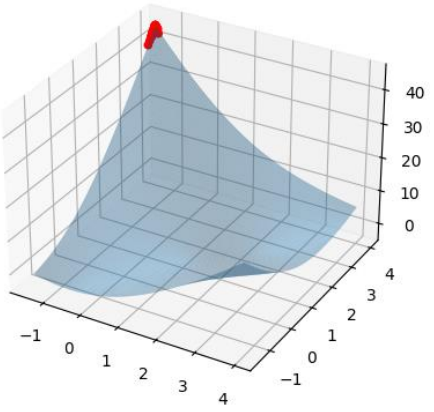
7 iteracja:



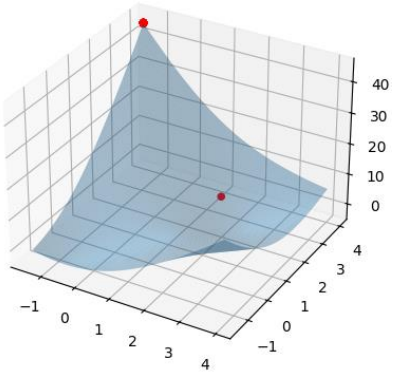
8 iteracja:



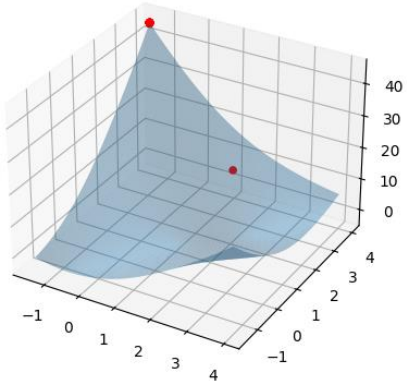
9 iteracja:



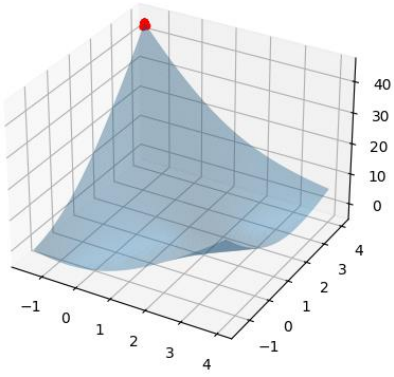
10 iteracja:



11 iteracja:



12 iteracja:



W następnych iteracjach rozwiązanie nie ulegało znaczącej poprawie.

Ostateczne wyniki:

- Najlepsza adaptacja: 44.09847

Wnioski

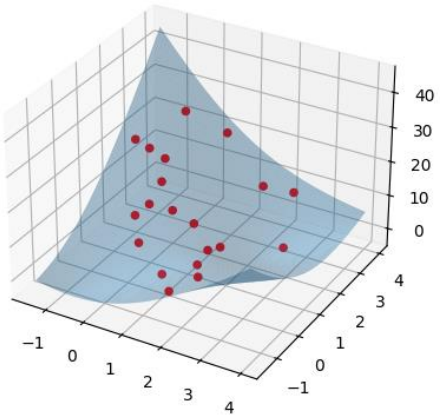
Zauważamy, że około 12 iteracji wynik nie ulega już zmianie. Tym samym liczba iteracji nie musi wynosić dla tej dziedziny, aż 30 iteracji. Dobrym rozwiązaniem byłoby też zastosowanie mechanizmu zatrzymywania, aby przerwać algorytm po osiągnięciu odpowiedniego poziomu optymalizacji.

Badanie wpływu rozmiaru populacji:

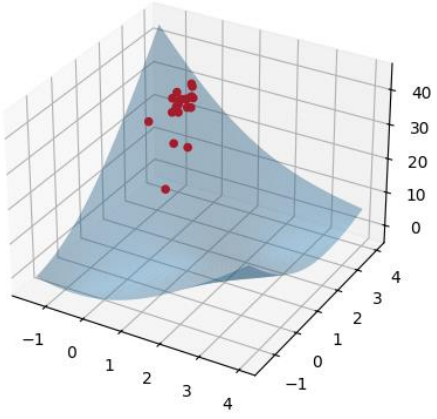
2 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 20
- Inercja (bezwładność): 0.4
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: -1.5 <= x <=4, -3 <= y <= 4

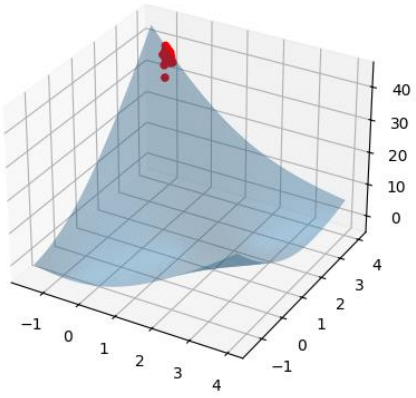
1 iteracja:



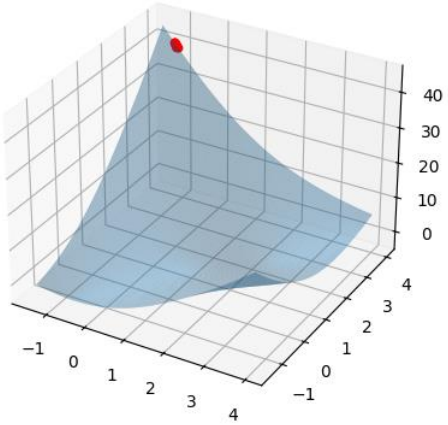
4 iteracja:



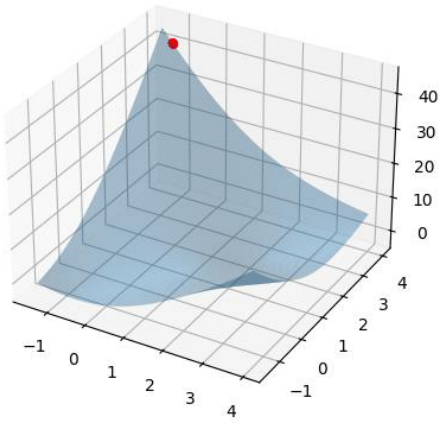
8 iteracja:



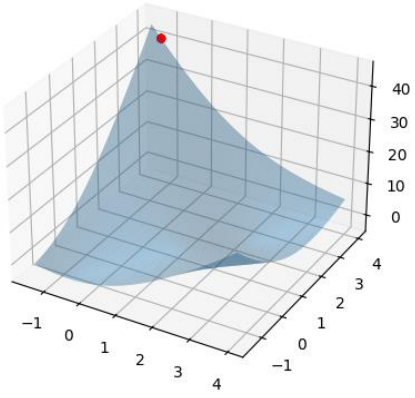
12 iteracja:



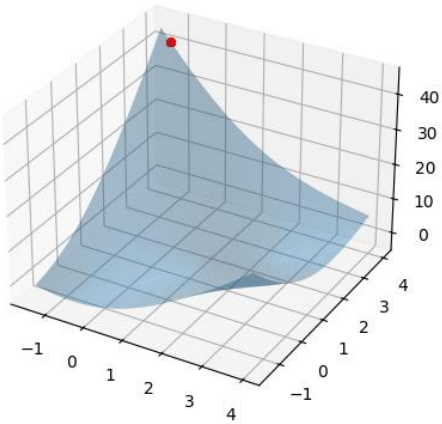
16 iteracja:



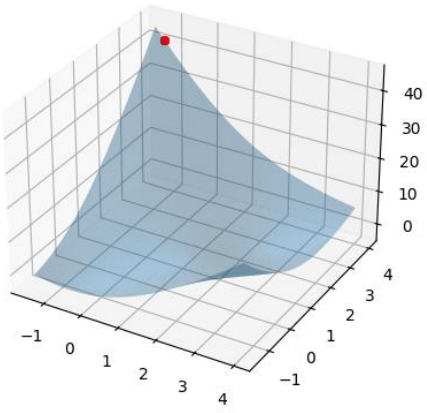
20 iteracja:



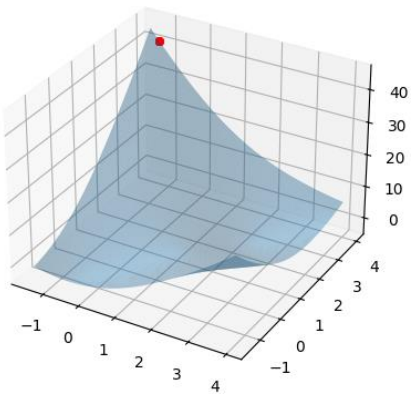
24 iteracja:



28 iteracja:



30 iteracja:



Ostateczne wyniki:

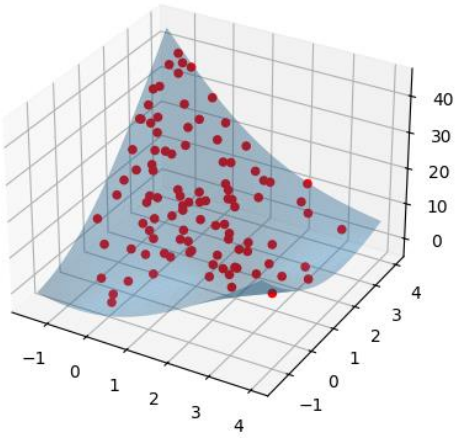
- Najlepsza adaptacja: 40.603949

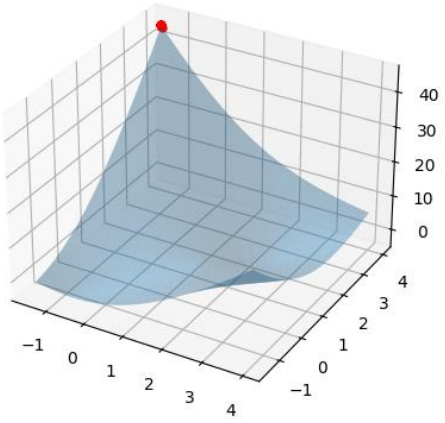
3 uruchomienie algorytmu:

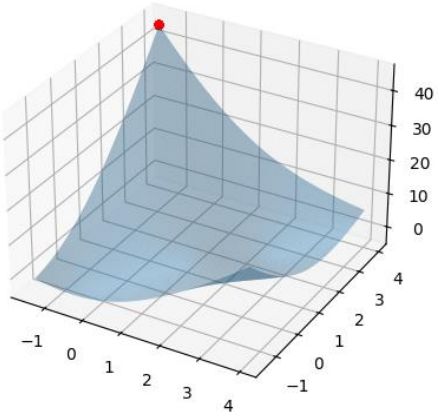
- Liczba iteracji: 30
- Rozmiar populacji: 100
- Inercja (bezwładność): 0.4

- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: -1.5 <= x <=4, -3 <= y <= 4

1 iteracja: **10 iteracja:** **20 iteracja:**







Ostateczne wyniki:

- Najlepsza adaptacja: 44.09847

Wnioski

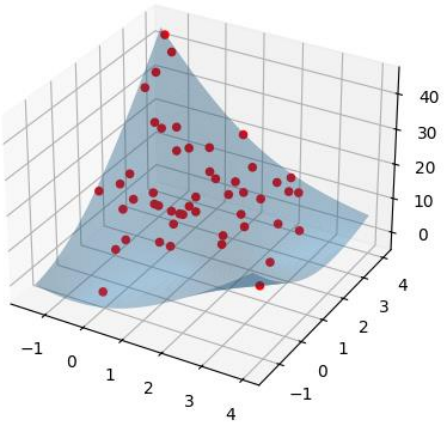
Zauważamy, że wraz ze wzrostem ilości cząstek wydłuża się czas obliczeń naszego programu. Używając dużej ilości cząstek (trzecie uruchomienie programu: 100 cząstek) widoczne jest, że ilość iteracji, po których wynik się stabilizuje uległ zmianie. Wnioskiem jest, że większa liczba cząstek zwykle powoduje, że przyciąganie do globalnego najlepszego rozwiązanie trwa dłużej, ponieważ więcej cząstek jest przyciąganych do lokalnie najlepszych rozwiązań. Wynik najlepszy uzyskaliśmy dla większej ilości cząstek (100 cząstek).

Badanie wpływu inercji:

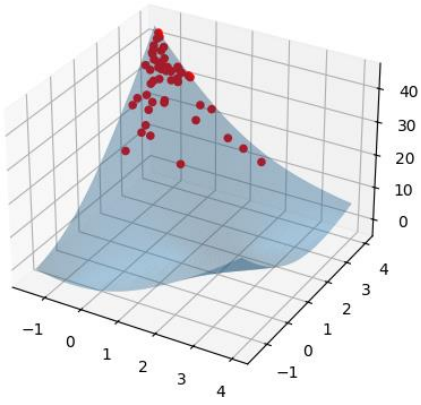
4 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.1
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: -1.5 <= x <=4, -3 <= y <= 4

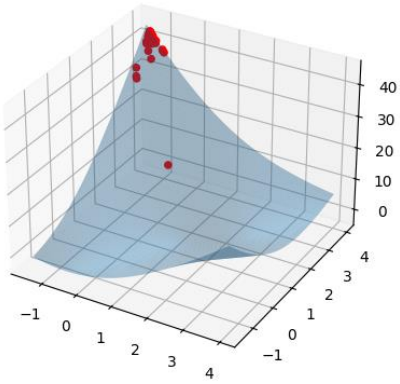
1 iteracja:



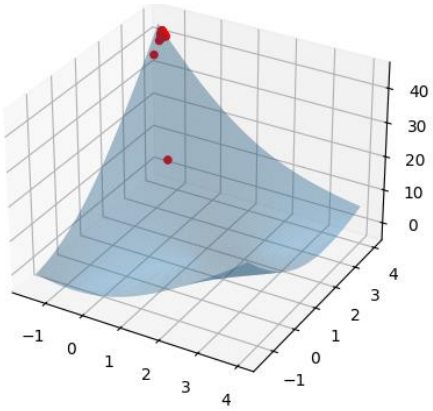
4 iteracja:



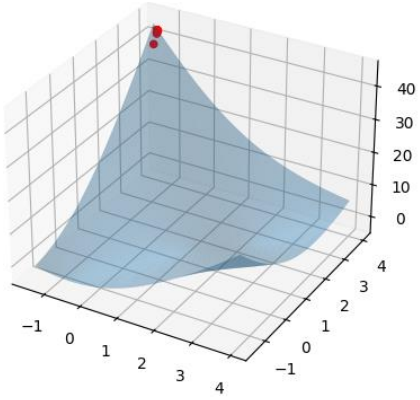
8 iteracja:



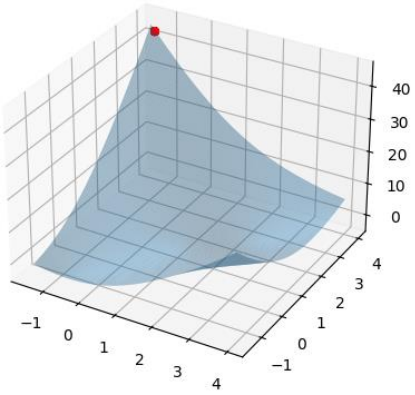
12 iteracja:



16 iteracja:



20 iteracja:



W następnych iteracjach rozwiązanie nie ulegało znaczącej poprawie.

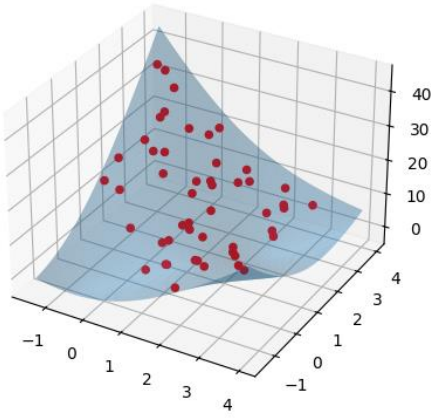
Ostateczne wyniki:

- Najlepsza adaptacja: 42.319102

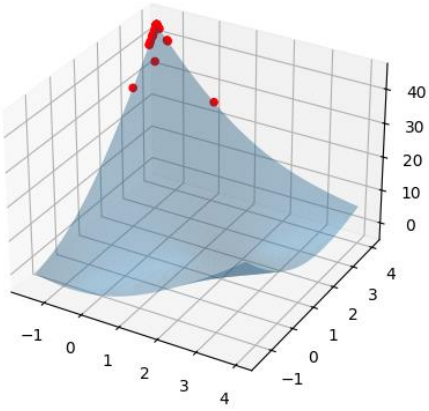
5 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.9
- Stała poznawcza: 0.3
- Stała społeczna: 0.6
- Badana dziedzina: -1.5 <= x <=4, -3 <= y <= 4

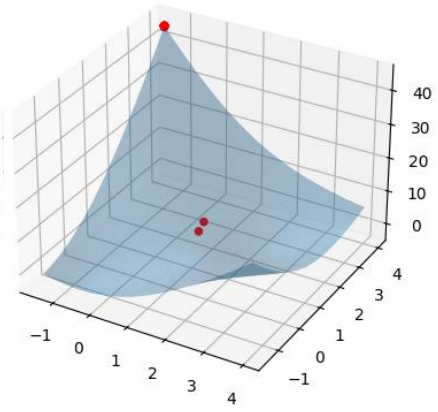
1 iteracja:



4 iteracja:



8 iteracja:



W następnych iteracjach rozwiązanie nie ulegało znaczącej poprawie.

Ostateczne wyniki:

- Najlepsza adaptacja: 44.09847

Wnioski

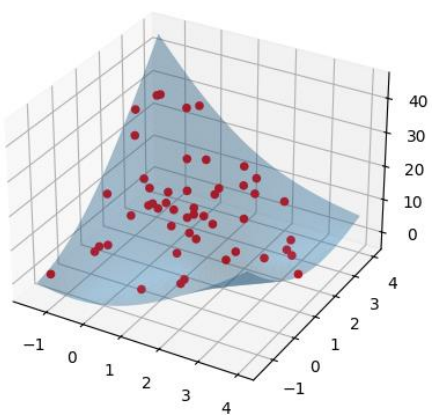
Zauważamy, że wraz ze wzrostem współczynnika inercji polepsza się wartość najlepszej osiągniętej adaptacji. Na wykresach zauważamy, że w przypadku wysokiej wartości tej zmiennej, już po 8 iteracjach uzyskujemy optymalny rezultat (jest to sytuacja odmienna, gdy porównamy wyniki do poprzednio testowanej funkcji, gdzie wraz ze wzrostem tego współczynnika wynik się pogarszał).

Badanie wpływu stałej poznawczej:

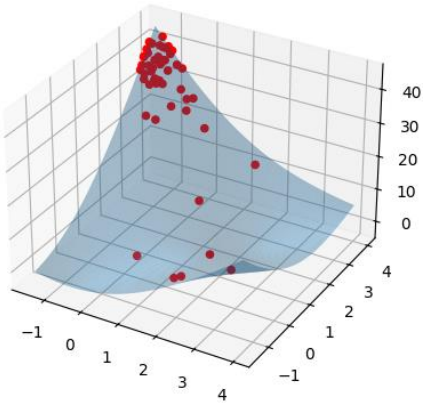
6 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała(komponent) poznawcza: 0.1
- Stała społeczna: 0.6
- Badana dziedzina: -1.5 <= x <=4, -3 <= y <= 4

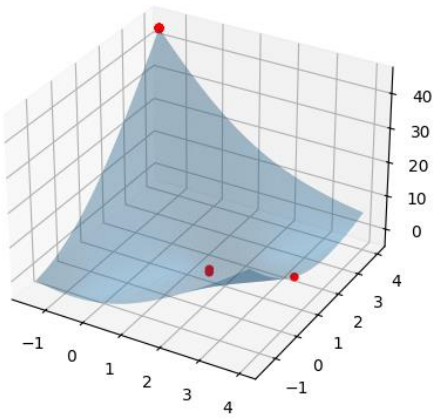
1 iteracja:



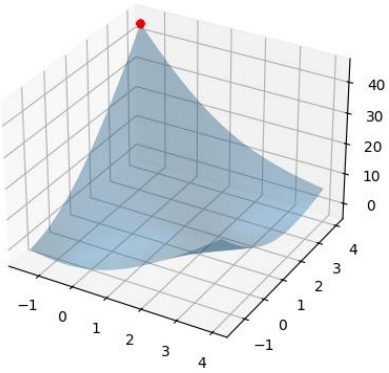
4 iteracja:



8 iteracja:



12 iteracja:



W następnych iteracjach rozwiązanie nie ulegało znaczącej poprawie.

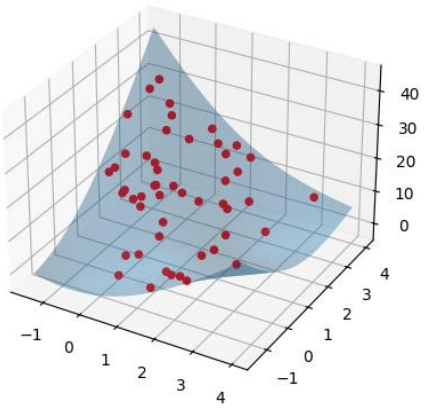
Ostateczne wyniki:

- Najlepsza adaptacja: 44.098472

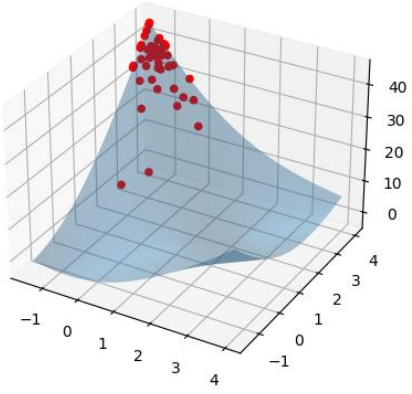
7 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała(komponent) poznawcza: 1.8
- Stała społeczna: 0.6
- Badana dziedzina: -1.5 <= x <=4, -3 <= y <= 4

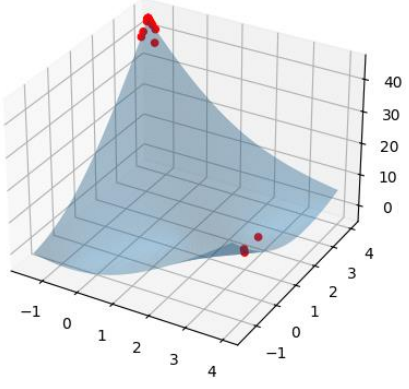
1 iteracja:



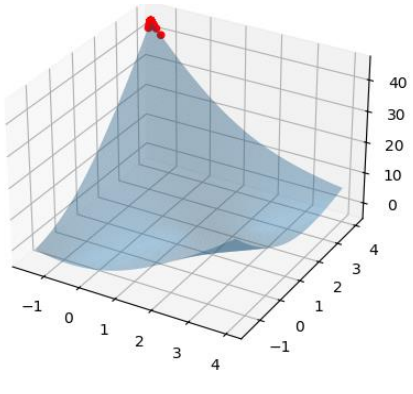
4 iteracja:



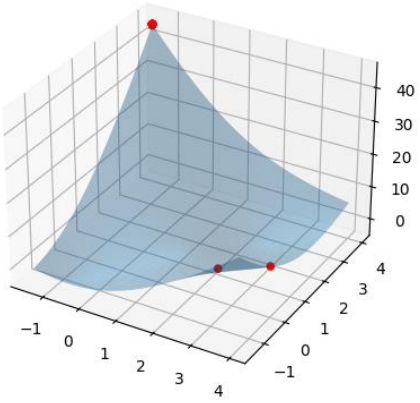
8 iteracja:



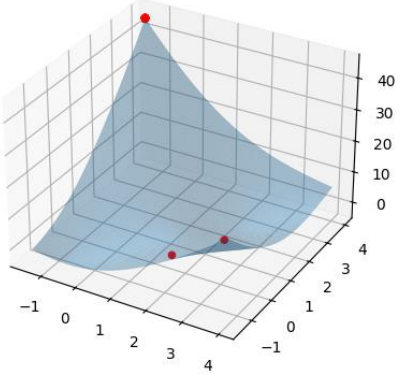
12 iteracja:



16 iteracja:



20 iteracja:



Ostateczne wyniki:

- Najlepsza adaptacja: 44.098472

Wnioski:

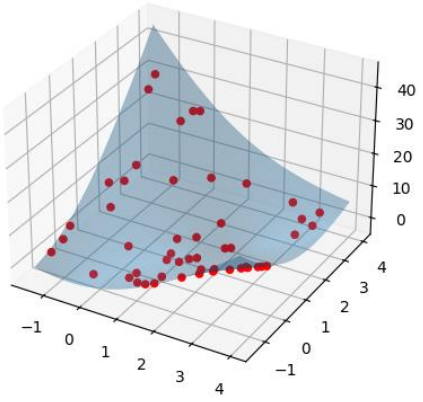
Zauważamy, że przy bardzo wysokiej wartości komponentu poznawczego (równego np. 1.8), niektóre cząstki nawet po skończonej liczbie iteracji są bardzo oddalone od optymalnego wyniku. Przy niskiej wartości tego współczynnika cząstki szybciej „docierają” do optymalnego rozwiązana (ponieważ już od około 12 iteracji wynik nie ulegał znaczącej poprawie). Wyższa wartość tego współczynnika powoduje również wyższą eksploatacje. Są to ciekawe w wnioski, ponieważ są zgoła odwrotne do tych, które otrzymaliśmy w przypadku pierwszej funkcji.

Badanie wpływu stałej społecznej:

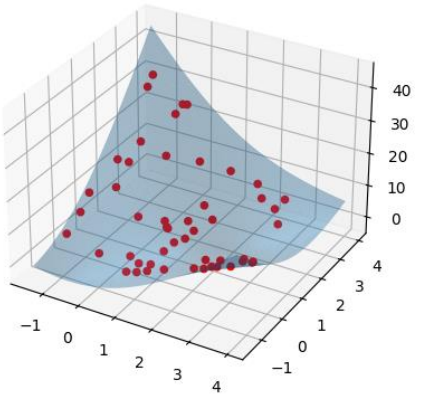
8 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała(komponent) poznawcza: 0.3
- Stała społeczna: 0.1
- Badana dziedzina: : -1.5 <= x <=4, -3 <= y <= 4

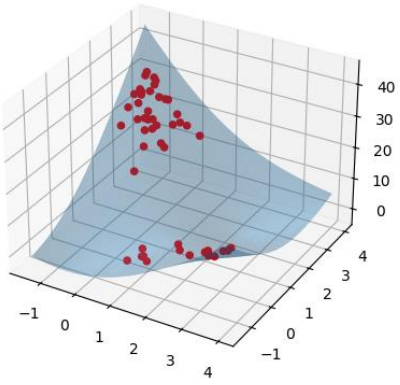
1 iteracja:



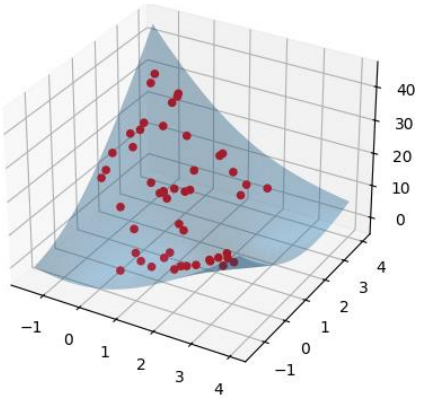
4 iteracja:



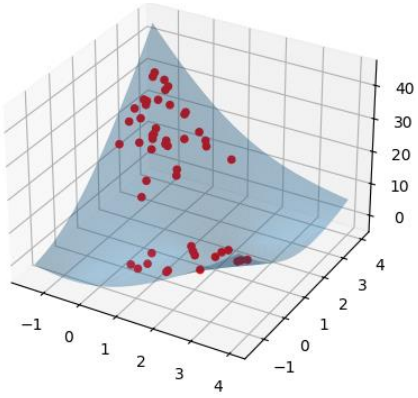
8 iteracja:



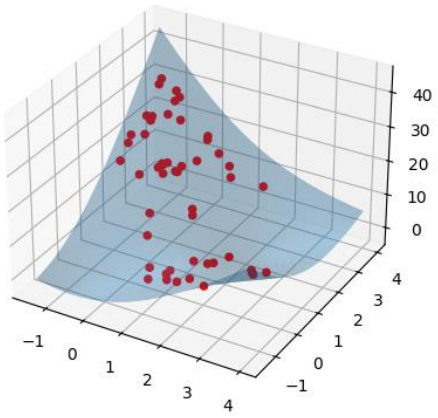
12 iteracja:



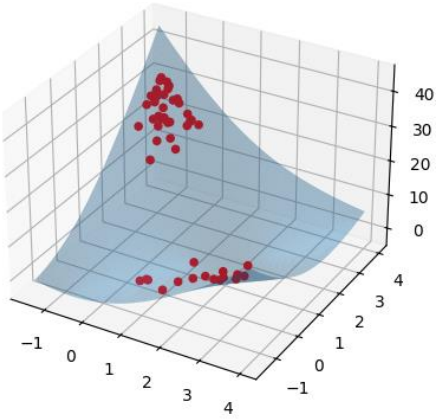
16 iteracja:



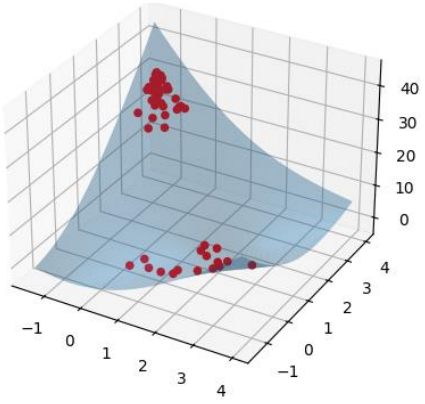
20 iteracja:



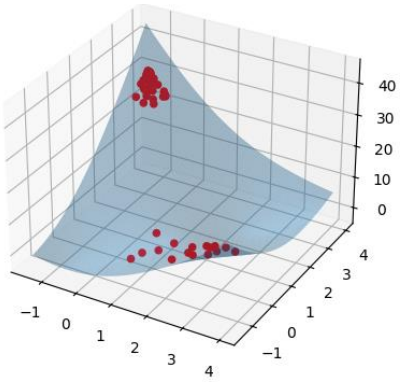
24 iteracja:



28 iteracja:



30 iteracja:



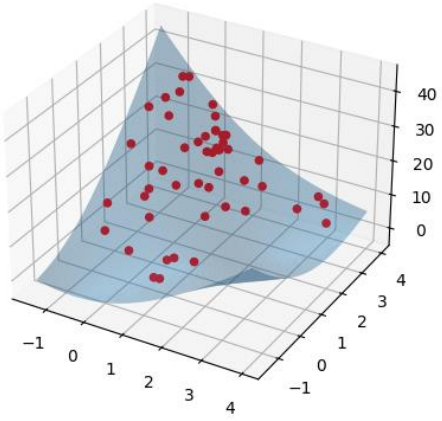
Ostateczne wyniki:

- Najlepsza adaptacja: 32.71878

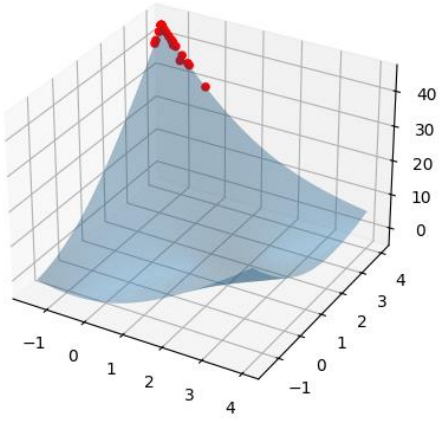
9 uruchomienie algorytmu:

- Liczba iteracji: 30
- Rozmiar populacji: 50
- Inercja (bezwładność): 0.4
- Stała(komponent) poznawcza: 0.3
- Stała społeczna: 0.9
- Badana dziedzina: -1.5 <= x <=4, -3 <= y <= 4

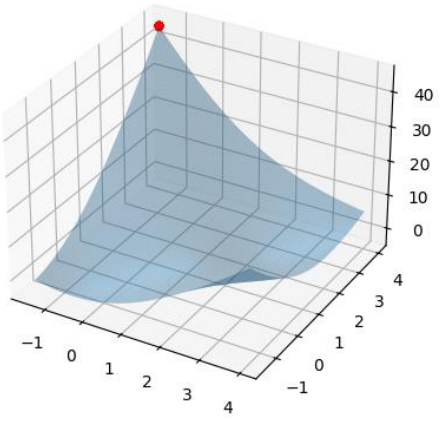
1 iteracja:



4 iteracja:



8 iteracja:



Ostateczne wyniki:

- Najlepsza adaptacja: 44.09847

Wnioski

Zauważamy, że w przypadku bardzo niskiej wartości komponentu społecznego (przykładowo wartość = 0.1) to nawet po skończonej liczbie iteracji, niektóre cząstki nie zbliżyły się do optymalnego rozwiązania. Przy zwiększeniu tej wartości obserwujemy, że wszystkie cząstki bardzo szybko znalazły zbliżone rozwiązanie (już po 8 iteracjach). Komponent społeczny określa jak bardzo dana cząstka powinna być „zainspirowana” przez sąsiadów. Rezultat znacząco lepszy uzyskaliśmy w przypadku wyższej wartości tego współczynnika.

4. Wnioski ogólne

- W większości uruchomień programu nie było konieczne zastosowanie 30 iteracji. Liczba ta powinna być mniejsza, albo powinien zostać zaimplementowany mechanizm zatrzymywania (w momencie gdy rozwiązanie zmienia się nieznacznie przez dłużysz okres czasu)
- Optymalna ilość cząstek użytych w algorytmie mieści się w zakresie od 50 do 100
- Zbyt duża wartość współczynnika inercji powoduje, że niektóre cząstki potrafią się „zagubić”. Uzyskana przez nie adaptacja znacząco odbiega od optymalnego rozwiązania

- Zbyt duża wartość inercji powodowała znaczące pogorszenie się rozwiązania
- Mała wartość stałej poznawczej powodowała, że cząstki wolniej „zbliżały się” do optymalnego rozwiązania, natomiast wyższa pozwalała na szybsze odnalezienie poprawnej adaptacji.
- Mała wartość stałej społecznej powodowała, że uzyskana adaptacja nie była zbliżona do optymalnego rozwiązania, a wiele cząstek utknęło w optimum lokalnych. Duża wartość natomiast pozwalała na szybkie znalezienie ekstremum globalnego