Implementacja i badanie algorytmów Swarm Intelligence

Wybrane zagadnienia sztucznej
inteligencji. Zadanie 3.

Norbert Ropiak, Maksym Telepchuk

Czerwiec 2020

Intelegencja rojowa (ang. Swarm Intelligence) jest to dziedzina algorytmów intelegencji obliczeniowej (ang. Computational Intelligence), algorytmy której są inspirowane biologicznymi wieloagentowymi systemami. SI wykorzystuje rojowe własności tych systemów w celu rozwiązywania danego problemu.

Algorytmy SI są skuteczne w problemie optymalizacji. Z matematycznego punktu widzenia, aby rozwiązać rzeczywisty problem optymalizacji za pomocą algorytmów inteligencji obliczeniowej, potrzebujemy matematycznej reprezentacji naszego problemu. Ta reprezentacja nazywa się funkcją celu (ang. objective function), która jest matematyczną regułą opisującą problem i wszystkie zmienne decyzyjne w ramach tego problemu.

Celem problemu optymalizacji jest znalezienie najlepszego rozwiązania spośród wszystkich możliwych rozwiązań, tzn. celem jest zminimalizowanie lub zmaksymalizowanie funkcji celu.

1 Algorytmy

1.1 PSO - Particle Swarm Optimization

Metaheurystyka optymalizująca problem przez iteracyjną poprawę kandydata na globalne minimum/maksimum. Problem jest rozwiązywany przez posiadanie populacji kandydatów (w tym algorytmie nazywane cząsteczkami) na globalne optimum i poruszanie tych cząsteczek w przestrzenie poszukiwań zgodnie z formułą związaną z pozycją i prędkością cząsteczek. Ruch każdej cząsteczki zależny jest od najlepszej lokalnej znanej pozycji, ale również skierowany jest do aktualnie znalezionego globalnego optimum, które jest aktualizowane przez ruchy innych cząsteczek.

PSO jest metaheurystyką i nie zakłada żadnych ograniczeń lub bardzo mało w problemie, który optymalizuje, przez to może przeszukiwać ogromne przestrzenie możliwych rozwiązań. Jednakże, metaheurystki takie jak PSO nie gwarantują znalezienia globalnego optimum. PSO również nie używa gradientu do optymalizacji problemu, co oznacza że funkcje nie muszą być różniczkowalne tak jak w metodach tj. gradient descent.

1.1.1 Inicjalizacja

Dla każdej cząsteczki:

1. Zainicjalizuj pozycje cząsteczki z rozkładu jednostajnego

2. Przypisz aktualną pozycję do aktualnie najlepszej znalezionej pozycji cząsteczki
3. Jeśli funkcja w tym miejscu osiąga lepszy wynik przypisz do globalnego optimum
4. Zainicjalizuj prędkość cząsteczki z rozkładu jednostajnego

1.1.2 Algorytm

1. Powtarzaj dopóki nie spełnione kryterium stopu lub nie osiągnięto założenj liczby iteracji
2. Dla każdej cząsteczki
   (a) Aktualizacja prędkości cząsteczki według wzoru

$$v = \omega v + \phi_p r_p (p - x) + \phi_g r_g (g - x)$$

   gdzie $r_p, r_g \sim U(0, 1)$
   ϕ_r, ϕ_g to parametry
   x to pozycja cząsteczki
   g globalne optimum
   p lokalne optimum cząsteczki

   (b) Aktualizacja pozycji cząsteczki

$$x = x + v$$

   (c) Jeśli aktualne pozycja cząsteczki jest lepsza od globalnego oraz jej lokalnego optimum, podstaw pod optimum obecną pozycje.

1.2 ABC - Artificial Bee Colony

Artificial Bee Colony jest algorytmem inspirowanym zachowaniem kolonii pszczół. Miejsca w których znajdują się cząsteczki są intuicyjnie traktowane jako źródła jedzenia. W tym przypadku kolonia zawiera 3 typy pszczół : **Employee Bee** cząsteczki, które zbierają jedzenie do ula z określonego źródła; **Onlook Bee** - patrulują wybrane przez cząsteczki pracownicze źródła jedzenia oraz wybierają sobie źródło w sposób losowy, gdzie lepsze źródła mają większą szansę na wylosowanie; **Scout Bee** - cząsteczki, które wybierają nowe źródło jedzenia, jeśli dane źródło jedzenia jest wyczerpane.

Źródłem jedzenia jest nazywany punkt na przestrzeni wyszukiwania. Jakość źródła jest określona funkcją jakości:

$$fit(\vec{x}) = \begin{array}{ll} \frac{1}{1+f(\vec{x})} & \text{gdy } f(\vec{x}) \geq 0 \\ 1 + |(f(\vec{x}))| & \text{gdy } f(\vec{x}) < 0 \end{array}$$

gdzie f - funkcja celu, $\vec{x}$ - punkt w przestrzeni wyszukiwania.
Algorytm ABC można przedstawić w kilku krokach:

1. Inicjalizacja. Każdej cząsteczce jest przypisywany losowe źródło jedzenia. Domyślnie połowa kolonii jest inicjalizowana jako Employee Bee (EB) oraz druga połowa jako Onlook Bee (OB).

2. Faza EB. Cząsteczki EB zaczynają „wydobywać jedzenie" ze źródła. Oznacza to, że ze źródła jest losowo wybierana wartość jednej ze współrzędnych c, oraz jest sprawdzana sąsiednia pozycja zadana wzorem

$$x_i' = x_i + (x_i - c) * phi$$

, gdzie x jest aktualnym źródłem, $\phi \in [-1, 1]$ - losowy współczynnik.

Jeśli jakość nowego źródła będzie lepsze od aktualnego, cząsteczka przechodzi do nowego źródła.

3. Faza OB. Cząsteczki OB patrzą na źródła EB oraz w sposób losowy wybierają sobie źródła, które badają w sposób podobny do EB. Prawdopodobieństwo wylosowania źródła x określa się wzorem

$$p(x) = \frac{fit(x)}{\sum_{x'} fit(x')}$$

4. Faza SB. W tej fazie jest sprawdzane czy źródła jedzenia są wyczerpane. Jeśli cząsteczka już długo nie zmienia źródła jedzenia (parametr maksymalnej ilości prób `max_trial` jest parametrem wejściowym), to cząsteczce jest losowo przypisywane nowe źródło jedzenia.

5. Kroki 2-4 są powtarzane stałą ilość iteracji, określoną parametrem wejściowym. W każdym kroku cząsteczki zapisują najlepsze dotychczas znalezione źródło.

W celu późniejszej analizy, najlepsza wartość po każdej iteracji jest zapamiętywana.

1.3 BA - Bat Algorithm

Bat algorithm to algorytm inspirowany zachowaniem nietoperzy i ich zdolnością do określenia miejsca w przestrzeni za pomocą echolokacji. Dla uproszczenia, algorytm stosuje 3 reguły do opisu tych zjawisk:

1. Wszystkie nietoperze używają echolokacji do wykrywania dystansu oraz znają różnicę pomiędzy jedzeniem, a ścianą.

2. Nietoperze latają losowo z prędkością v na pozycji x z stała częstotliwością f_{min}, zmieniając długość fali λ oraz głośnością A_0 aby znaleźć pożywienie. Potrafią automatycznie dostosowywać długość fali (lub częstotliwość) emitowanych pulsów oraz ich tempo $r \in [0, 1]$, w zależności od odległości od celu.

3. Mimo że głośność może się zmieniać na wiele sposobów, przyjęte jest, że zawiera się w przedziale od maksimum A_{max} do minimum A_{min}

Dla symulacji nietoperzy określa się reguły w jaki sposób zmieniają się ich pozycje oraz prędkości.

$$f = f_{min} + \beta(f_{max} - f_{min})$$

$$v = v + f(x - p)$$

$$x = x + v$$

gdzie $\beta \in [0, 1]$ to wektor wylosowany z rozkładu jednostajnego, p to globalne optimum.

Dla lokalnego przeszukiwania, kiedy lokalne minimum wyszukiwane jest pośród aktualnie najlepszych globalnych optimum, nowe rozwiązanie dla każdego nietoperza generowanie jest przez losowy spacer:

$$x = x + \epsilon A_{avg}$$

gdzie $\epsilon \in [-1, 1]$, a A_{avg} to średnia głośność wszystkich nietoperzy.

Ponadto, głośność i tempo emisji musi zmieniać się zgodnie z iteracjami symulacji. Głośność zazwyczaj zmniejsza się, jeśli nietoperz znalazł ofiarę, tak tempo emisji rośnie.

$$A = \alpha A$$

$$r = r[1 - \exp(-\gamma t))]$$

gdzie α, γ to parametry.

2 Analiza

2.1 Funkcje celu

Dla analizy skuteczności optymalizacji zostały wybrane następujące funkcje :

1. **Ackley function** jest szeroko stosowana do testowania algorytmów optymalizacji. Charakteryzuje się prawie płaskim obszarem zewnętrznym i dużym pogłębieniem w środku. Ta funkcja stwarza ryzyko uwięzienia algorytmów optymalizacji w jednym z wielu lokalnych minimów.

$$f(x) = f(x_1, \ldots, x_n) = -a \cdot \exp(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)) + a + \exp(1)$$

   Domyślne wartości parametrów : $a = 20$, $b = 0.2$, $c = 2\pi$.

   Globalne minimum : $f(x) = 0, x = (0, \ldots, 0)$

2. **Sphere function** ma d lokalnych minimum oprócz globalnego. Jest ciągła, wypukła i jednomodalna.

$$f(x) = \sum_{i=1}^{d} x_i^2$$

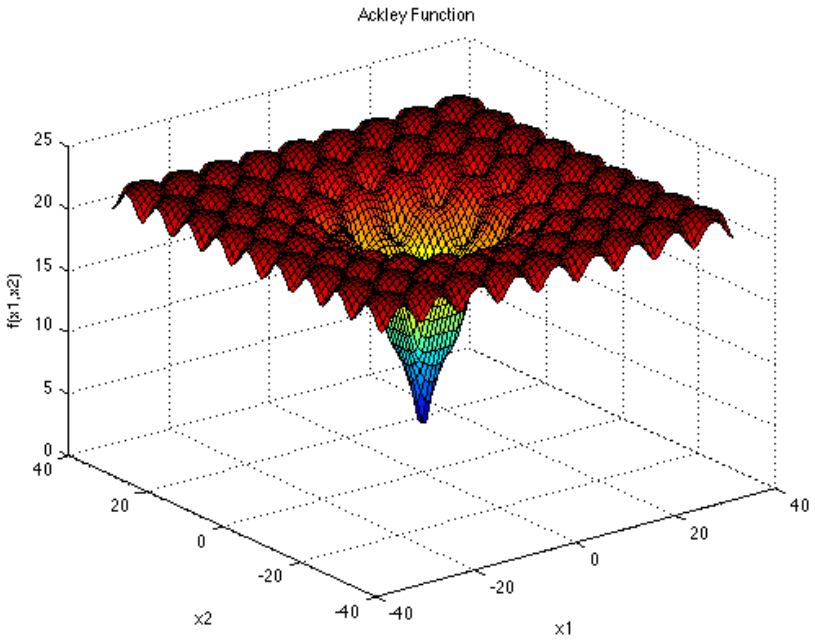
   Globalne minimum: $f(x) = 0, x = (0, \ldots, 0)$

3. **Rosenbrock function** ma kształt doliny, jest unimodalna oraz ma jedno globalne minimum znajdujące się na jej wąskim dnie.

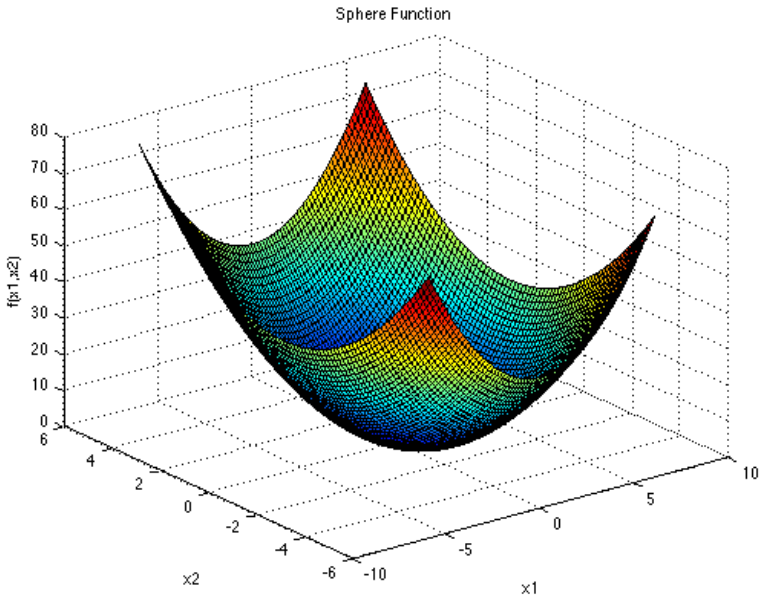
$$\sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

   Globalne minimum : $f(x) = 0, x = (0, \ldots, 0)$

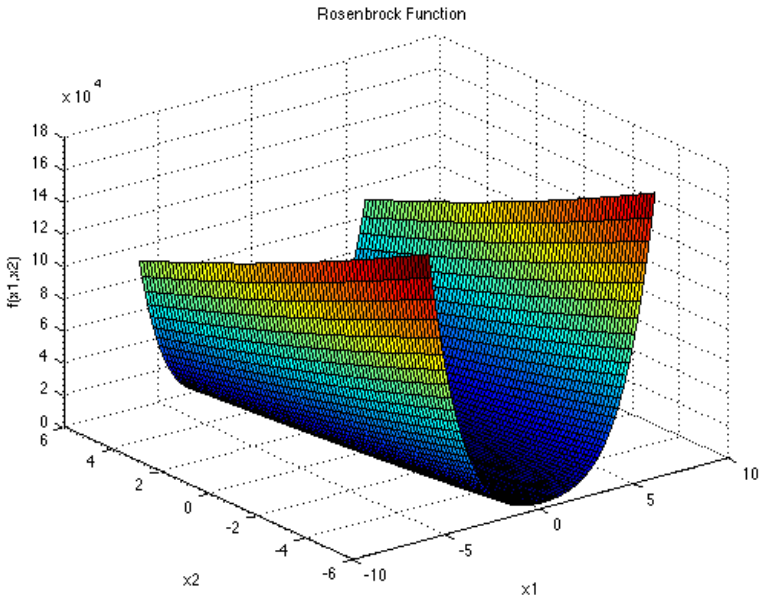
Rysunek 1: Ackley function



Rysunek 2: Sphere function



Rysunek 3: Rosenbrock function



4. **Rastrigin function** posiada dużo lokalnych minimum. Jest multimodalna, ale minima są rozłożone regularnie w przestrzeni.

$$f(x) = 10d + \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i)]$$

Globalne minimum: $f(x) = 0, x = (0, \ldots, 0)$

2.2 Wyniki

Poniższe wyniki są uśrednieniem z 10 różnych symulacji, aby ograniczyć wpływ losowości na badanie algorytmów.

Liczba iteracji: 100

Wielkość populacji: 30

Parametry ABC: `max_trials` = 100

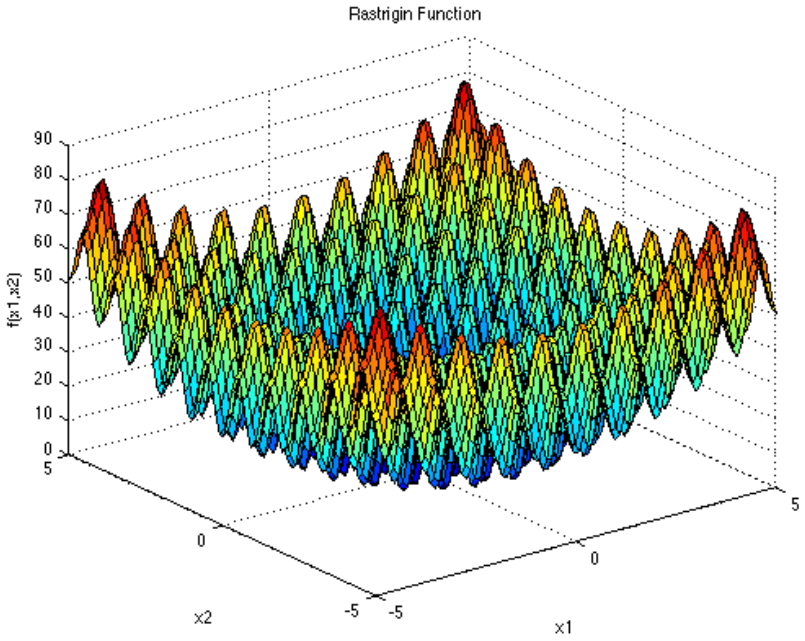
Parametry PSO: `w` = 0.5, ϕ_p=0.4, $\phi_g = 0.9$

Parametry BA: $f_{min} = 0$, $f_{max} = 2$, $A_{min} = 0$, $A_{max} = 100$, $\alpha = 0.9$, $\gamma = 0.9$

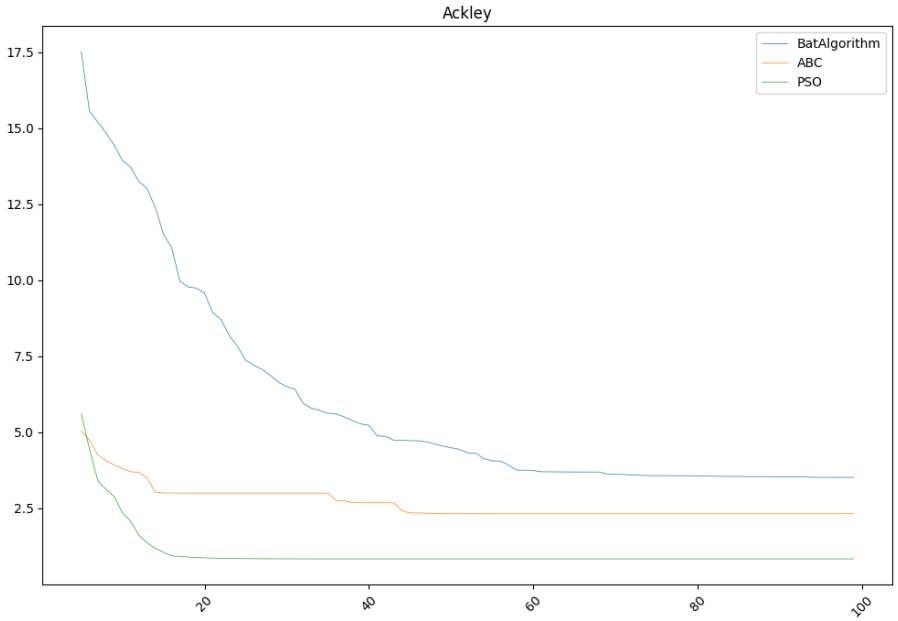
3 Wnioski

Wszystkie algorytmy poradziły sobie świetnie z prostymi funkcjami takim jak Rosenbrok czy Sphere. Otrzymane wyniki były bardzo blisko globalnego minimum jakim było $(0, \ldots, 0)$. Pomimo

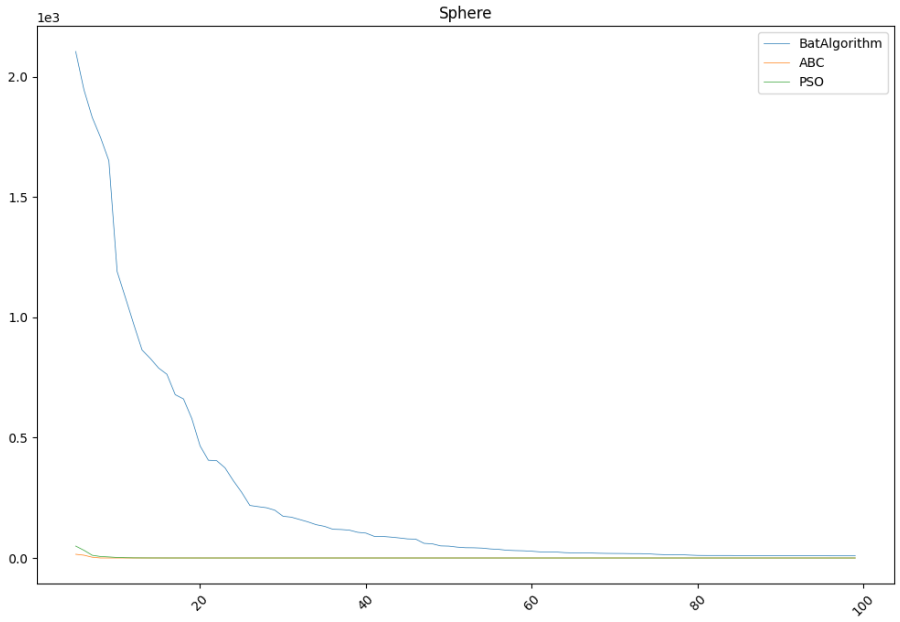
Rysunek 4: Rastrigin function



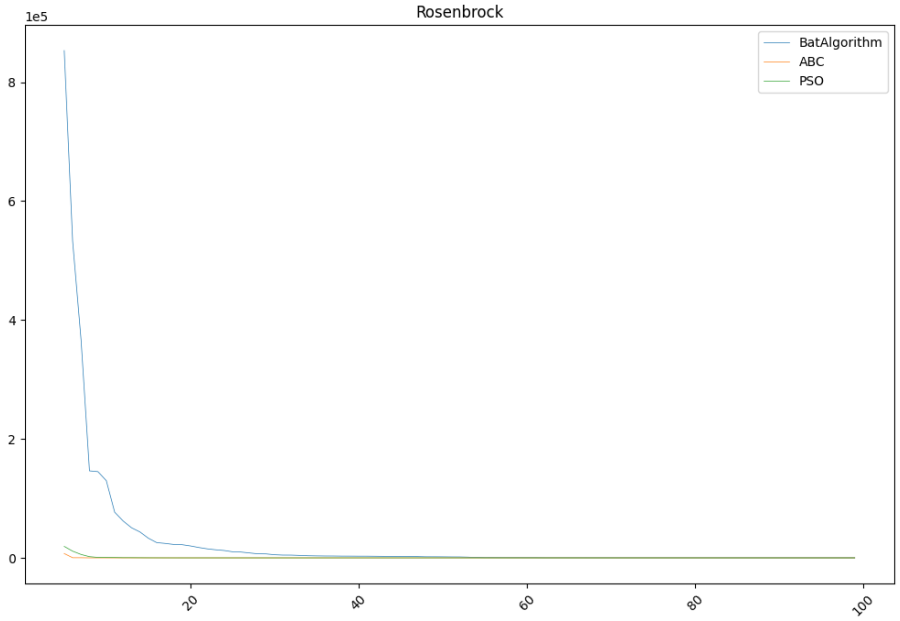
Rysunek 5: Symulacje dla Ackley funtion



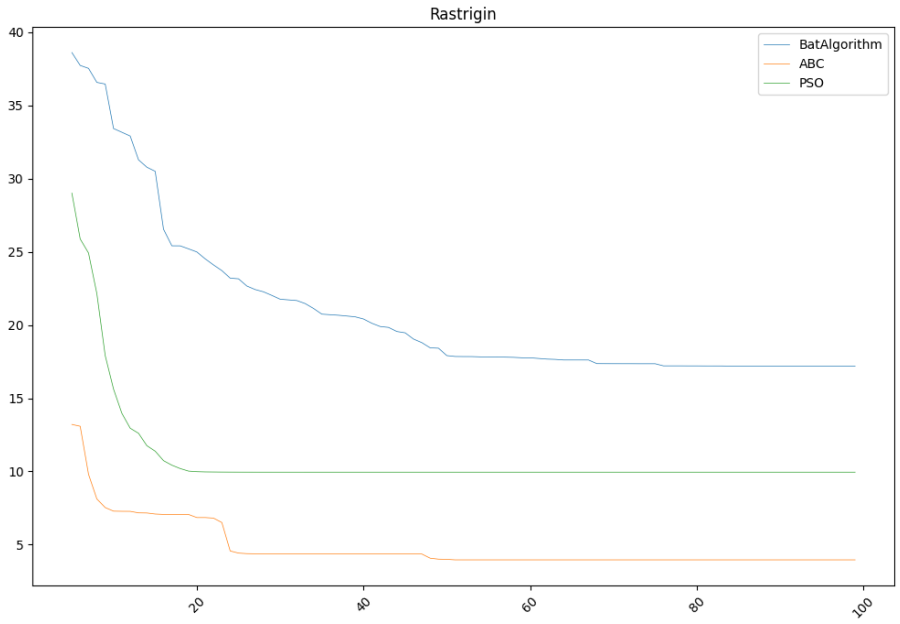
Rysunek 6: Symulacje dla Sphere funtion



Rysunek 7: Symulacje dla Rosenbrock funtion



Rysunek 8: Symulacje dla Rastrigin funtion



dobrych wyników dla wszystkich algorytmów, można zauważyć, że dla tych dwóch funkcji najwolniej zbiegał Bat Algorithm.

W przypadku algorytmów z wieloma lokalnymi minimum, nie wszystkie algorytmy były w stanie zbiec do globalnego minimum. Tak, na przykład, dla funkcji Ackley najlepiej zachowywał się algorytm PSO, który zbiegł do globalnego minimum. ABC zbiegł do punktu, w którym wartość funkcji równa się ok. 2.5. BA wykazał się najgorszą skutecznością, ponieważ minimalna wartość była ok. 4.

Dla funkcji Rastrigin najlepsze zachowanie wykazał ABC, chociaż nie udało mu się osiągnąć globalnego minimum, uzyskał on wartość ok. 5. Dla PSO ta wartość wyniosła ok. 10 a dla BA ok. 20.