Sprawozdanie

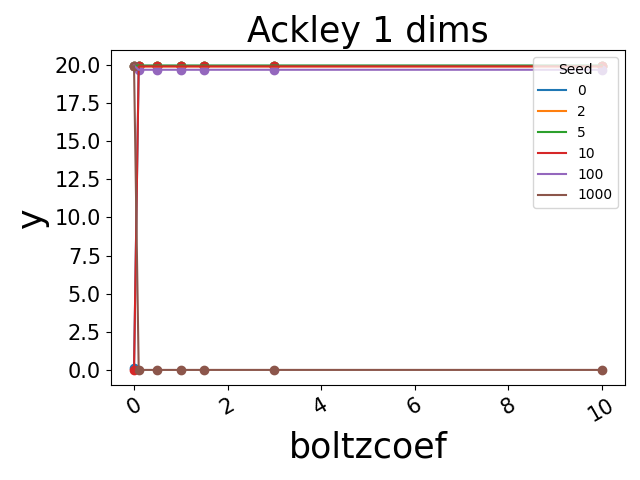
Ćw. 8 – Symulowane wyżarzanie  
Filip Horst 311257

# Badanie implementacji C++

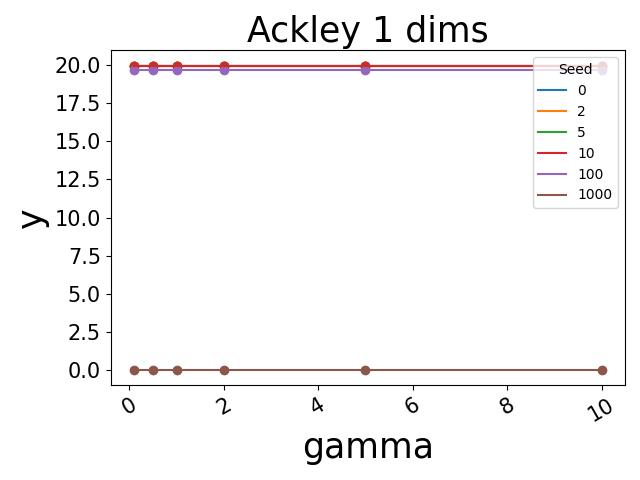
Badania odbywały się poprzez zmianę tylko jednego parametru na raz w celu sprawdzenia, czy występują jakieś zależności. Wnioski i obserwacje były tworzone na podstawie wykresów oraz ręcznego przeglądu plików wyjściowych. Niektóre badania zostały opisane w dużym skrócie, jeśli nie wniosły nic wartościowego i/lub pokrywały się z innym przypadkiem. Badane zakresy wykorzystane do porównań były takie same dla wszystkich funkcji, jednak w niektórych szczegółowych badaniach były zmienianie w celu przeszukania dokładniejszych zakresów i analizy cech algorytmu.

## Funkcja Ackley’a

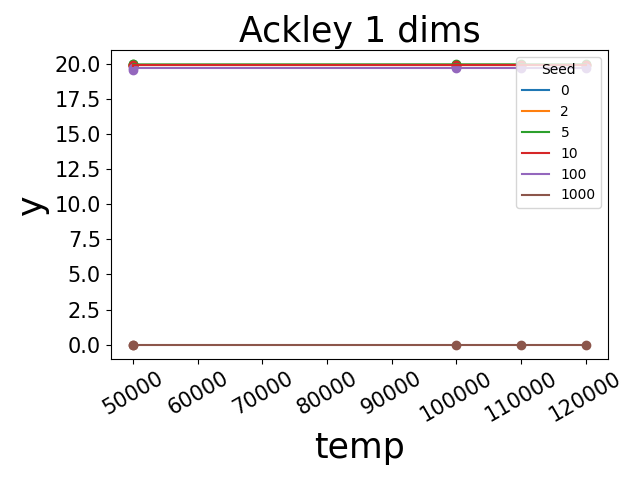
### 1 wymiar

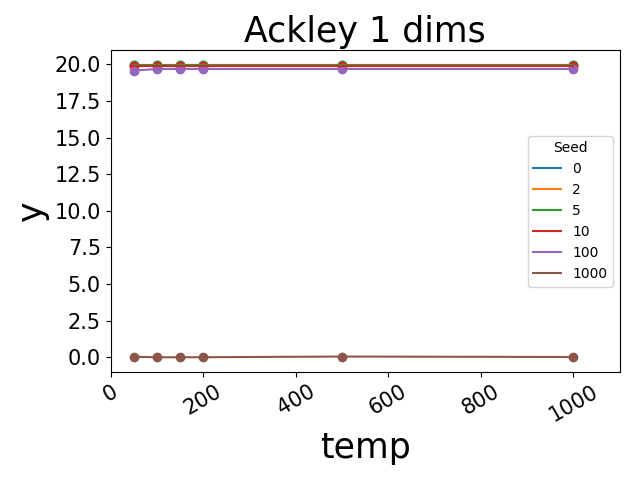
  
Zależność wyniku y od ustawień współczynnika Boltzmana

Wpływ ustawień wsp. Boltzmana na wyniki był niezauważalny – wyniki pozostawały stałe dla każdego ustalonego ziarna. Wyjątkiem było ustawienie kb = 0, gdzie wyniki się zmieniły, jednak nie było w nich wyraźnej zależności. Dla większości kb =0 pogorszyło wyniki, ale np. dla seed = 10 wynik był wtedy lepszy.

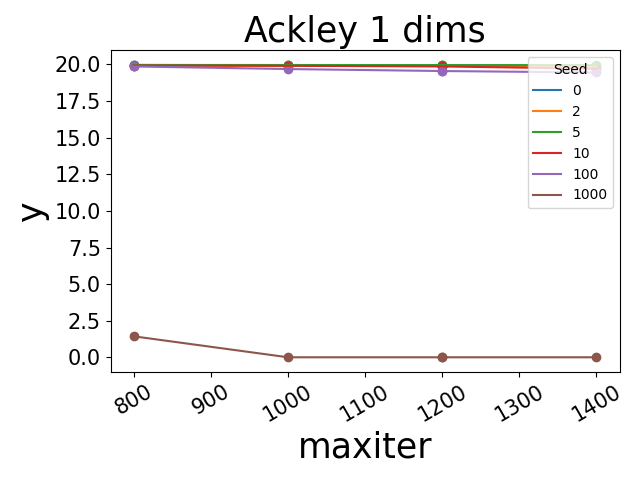
  
Zależność wyniku od ustawień gamma

Gamma w odróżnieniu od stałej Boltzmana nie miała absolutnie żadnego wpływu na wynik niezależnie od testowanych ustawień.



  
Zależność wyniku y od ustawień temperatury – dwa zakresy

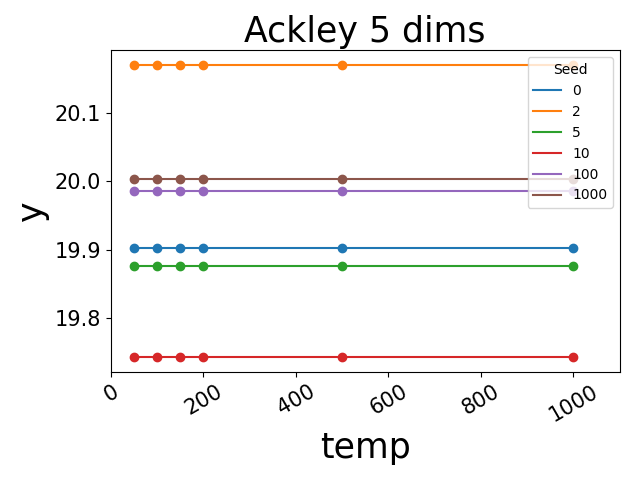
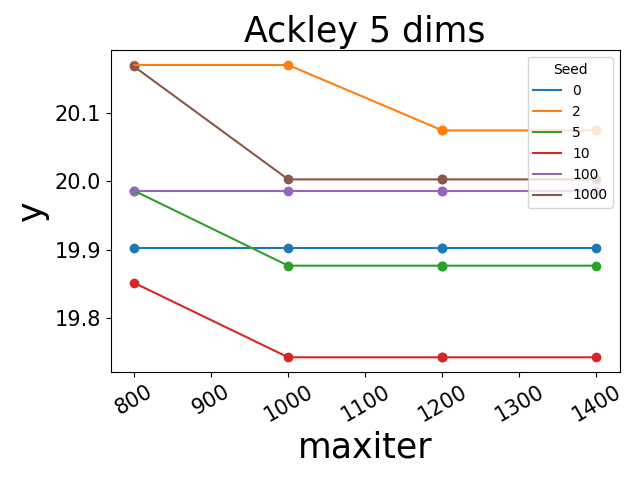
Również temperatura nie ma zauważalnego wpływu niezależnie od badanego zakresu.

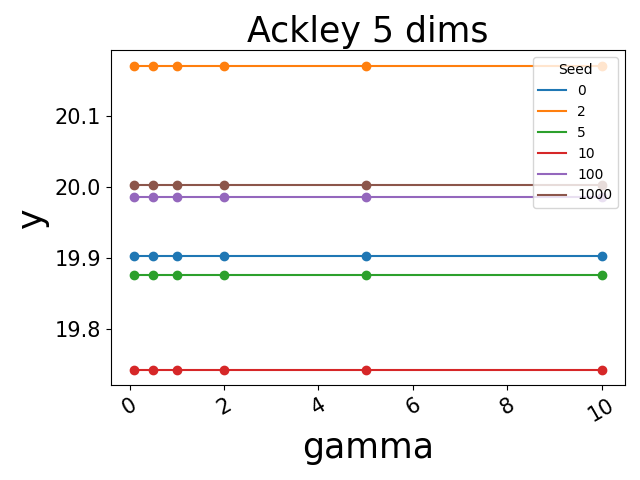
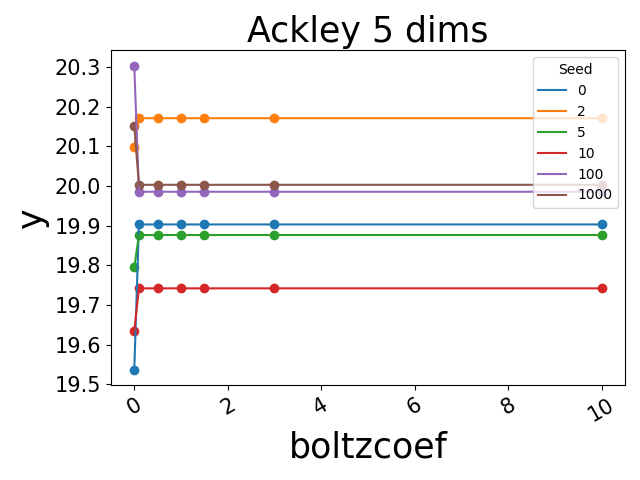
  
Zależność wyniku od ustawień maxiter

Najbardziej wyraźną zależność miał parametr maxiter, ponieważ wynik końcowy poprawiał się wraz ze zwiększaniem iteracji. Dla części seed widać jednak, że po pewnym momencie wynik się zatrzymywał i większa liczba iteracji nic nie dała. Można to jednak porównać do zwykłego zgadywania – oczywiście im więcej „strzałów” wykonamy tym większa szansa, że coś się uda. Istnieje również szansa, że w badanym przypadku wynik wpadał w minimum lokalne, a ze względu na zaimplementowaną metodę tworzenia kolejnego rozwiązanie przez dodanie szumu nie był on w stanie się z niego wybić, więc niezależnie od liczby iteracji algorytm tkwił w miejscu.

Ogólnie należy również wspomnieć, że tylko jedno rozwiązanie było bardzo dobre, bo trafiło już na samym początku w okolice optimum.

### 5 wymiarów



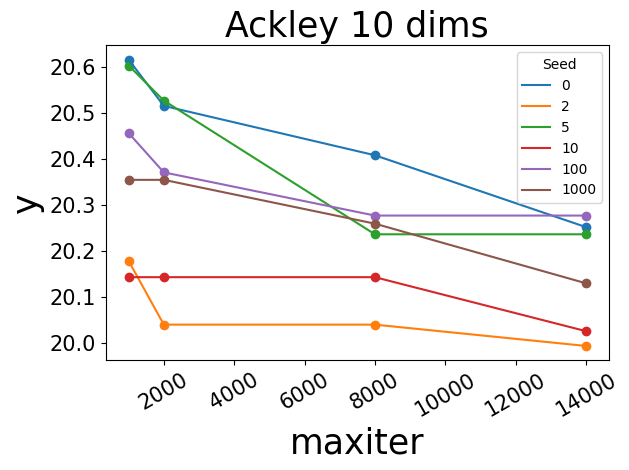


Dla 5 wymiarów większość zależności (a raczej ich brak) był analogiczny.

Jak można zaobserwować po samych wartościach osi Y, algorytm nie poradził sobie zbyt dobrze z rozwiązaniem problemu dla 5 wymiarów.

### 10 wymiarów

Dla 10 wymiarów podobnie jak w przypadku 5 brakowało jakichkolwiek zależności. Jednak z powodu wyglądu wykresu Y od maxiter zdecydowano się na zbadanie dużo większej liczby iteracji niż poprzednio, by sprawdzić czy ich zbyt mała ilość może być powodem słabych wyników.

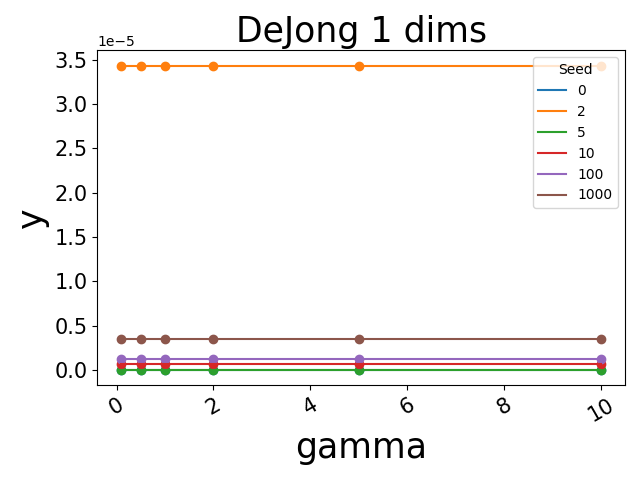
  
Zależność Y od maxiter

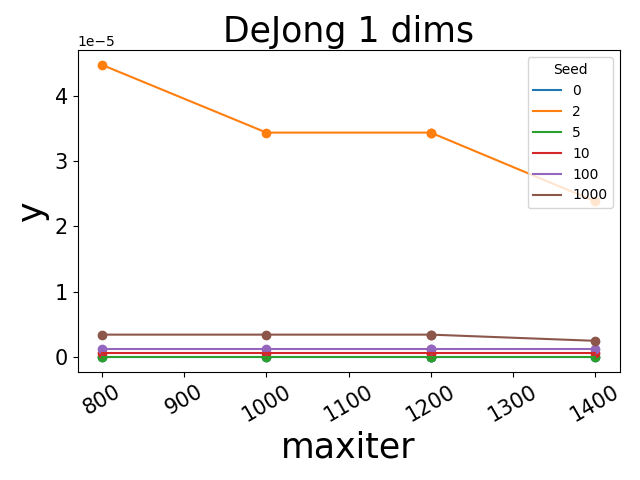
Jako przykład jednego parametru wstawiony został wykres Y od maxiter. Można z niego wywnioskować, że algorytm nie poradził sobie z zadaniem minimalizacji, ponieważ nawet przy 14000 iteracji nie jest w stanie nawet przybliżyć się do 0. W tym badaniu ymin = 19.99356, ale do porównania wybrany został przypadek dla 1400 iteracji gdzie ymin = 20.14332. Widać jednak, że dodatkowe iteracje trochę pomogły.

## Funkcja DeJonga

### 1 wymiar

Podobnie jak w funkcji Ackley’a nie udało się znaleźć żadnej zależności wyników od parametrów poza maxiter. Załączony przykładowy parametr gamma, który przedstawia prostą poziomą:

  
Wynik w zależności od Gamma

  
Wynik w zależności od iteracji

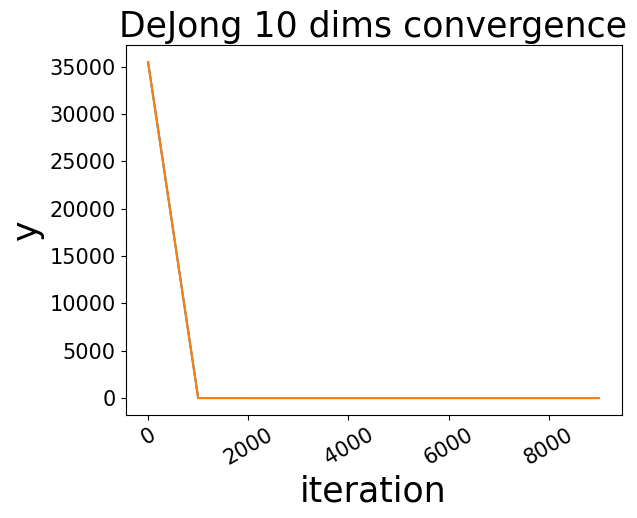
Można jednak zauważyć, że wyniki są bliskie oczekiwanemu, a w niektórych przypadkach prawie równe. Można podejrzewać, że wyniki dla gorszych seed można by było poprawiać dalej zwiększając liczbę iteracji.

### 5 wymiarów

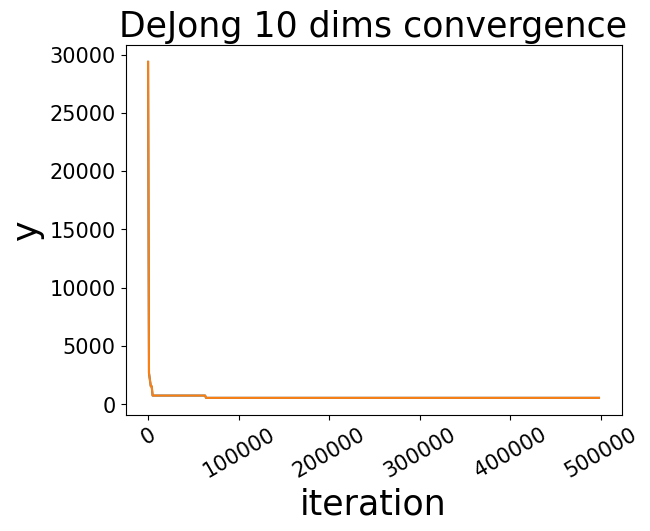
Podobnie jak w przypadku funkcji Ackley’a dla większej liczby wymiarów nie udało się osiągnąć poprawnych wyników. Najlepszy okazał się y = 0.04234984, co jest dość dobrym wynikiem, ale w zależności od problemu może być uznany za daleki od optimum.

### 10 wymiarów

Dla 10 wymiarów przy ustalonych 1400 iteracjach udało się uzyskać wynik zaledwie y = 0.5510613. W tym przypadku przeprowadzono dodatkowe doświadczenie polegające na sprawdzeniu zbieżności dla 10 000 iteracji. Wybrane zostało inne jądro generatora, by sprawdzić czy dzięki większej liczbie zdoła „dogonić” to najlepsze. Nie udało się to i wynik ostateczny wyniósł y = 0.629611.

  
Wykres nauki dla 10k iteracji

Wykres pokazuje, że zbyt duża liczba iteracji jest bez sensu. Już w okolicach 1000 iteracji znalezione zostało rozwiązane i od tamtej pory nic się nie zmieniło. Co do wcześniejszych iteracji nie widać żadnych stabilnych i konsekwentnych popraw tylko nagły skok do lepszej wartości. Może to oznaczać zbyt duże zmiany między rozwiązaniami. Żeby to sprawdzić, zmieniono operację tworzenia nowych rozwiązań na dodawanie szumu losowego z przedziału -50, 50 na -1,1.

  
Przebieg poszukiwań dla szumu -1,1 500k iteracji

Przy takiej wersji algorytmu wymagana liczba iteracji znacznie wzrosła, a wynik się pogorszył, bo wyniósł aż y = 557.4477. Istnieje jednak szansa, że zmiana z szumu +-50 na +-1 była zbyt duża. W ramach eksperymentu zmieniona została również temperatura startowa na T = 1 000 000, ale wynik się nie zmienił. Dla porównania dla szumu +- 50 i 500 000 iteracji y wyniósł y = 0.0007154978, co jest znaczną poprawą.

## Funkcja Rastragina

### 1 wymiar

Wszystkie obserwacje są analogiczne do poprzednich funkcji. Najlepszy wynik przy ustalonych 1400 iteracjach max y = 0.002085363.

### 5 wymiarów

Najlepszy wynik wyniósł zaledwie y = 9.256569.

### 10 wymiarów

Nic się nie zmieniło. W celach porównawczych najlepszy wynik y = 64.21758 został zapisany przy ustawieniu 1400 iteracji.

# Porównanie

Wykonane dla ustawień, gdzie maksymalna liczba iteracji wynosiła 1400. Reszta parametrów pozostała w ustawieniach domyślnych T = 100 000, kB = 1, gamma = 1, ponieważ nic nie wskazywało na to, że mają jakieś poważne znaczenie. Tabela ma na celu porównanie jakie funkcje są najtrudniejsze dla algorytmu.

|  |  |  |
| --- | --- | --- |
| Funkcja | Wymiary | Najlepszy wynik |
| Ackley | 1 | 0.002481523 |
| Ackley | 5 | 19.53637 |
| Ackley | 10 | 20.14332 |
| DeJong | 1 | 3.725518e-09 |
| DeJong | 5 | 0.04234984 |
| DeJong | 10 | 0.5510613 |
| Rastragin | 1 | 1.667147e-06 |
| Rastragin | 5 | 9.256569 |
| Rastragin | 10 | 64.21758 |

Według osiąganych wyników można powiedzieć, że najtrudniejszą funkcją dla tego algorytmu jest f. Rastragina, średnia jest f. Ackley’a, a najprostsza jest funkcja DeJong’a.

## Wnioski

Na podstawie badań powstają następujące wnioski:

* Parametry mają niewielki wpływ na ostateczne wyniki algorytmu
* Największy wpływ ma czynnik losowy, czyli seed oraz liczba iteracji
* Algorytm radzi sobie bardzo dobrze dla funkcji o bardzo niskiej liczbie wymiarów (łatwych)
* Zbieżność algorytmu jest nieprzewidywalna – brakuje stabilnych i konsekwentnych popraw wraz z iteracjami. Zamiast tego występuje „pauza”, a po kilkuset/kilku tysiącach iteracji nagła zmiana – wskazuje to na losowość w działaniu.
* Optymalizacja trudniejszych funkcji daje się wykonać tylko częściowo dla bardzo dużej liczby iteracji – nie da się ocenić, czy dalsze ich zwiększanie doprowadziło by do optimum
* Trudność funkcji ma wpływ na osiągane wyniki
* Algorytm działa dość szybko

Podsumowując, prymitywność i prostota algorytmu sprawia, że działa na zasadzie „albo da dobry wynik albo nie”. Zauważalny jest brak wpływu innych parametrów niż liczba iteracji, co sprawia że algorytm można uznać bardziej za ulepszoną zgadywankę. Nie można mu jednak odmówić tego, że przy odpowiedniej liczbie prób jest w stanie znajdywać rozwiązania stosunkowo bliskie optimum. Trzeba jednak pamiętać, że „stosunkowo bliskie” według osobistej oceny w prostym badaniu projektowym, a rozwiązanie, które było by wystarczające by rozwiązać nim realny problem to dwie odległe rzeczy. Tak zaimplementowany algorytm jest **użyteczny tylko dla prostych problemów**.