Sprawozdanie

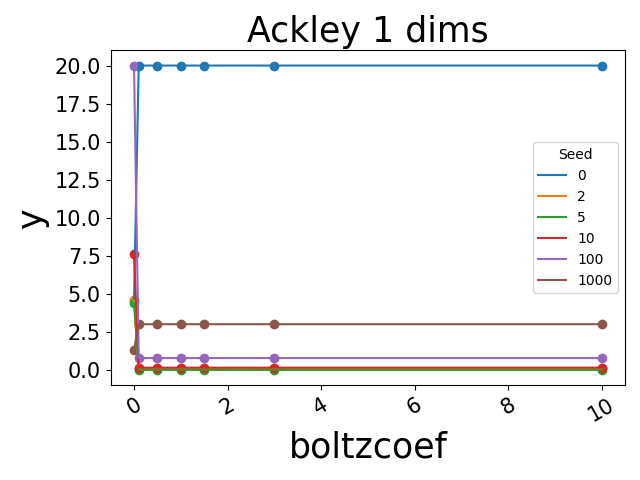
Ćw. 8 – Symulowane wyżarzanie  
Filip Horst 311257

# Badanie implementacji C++

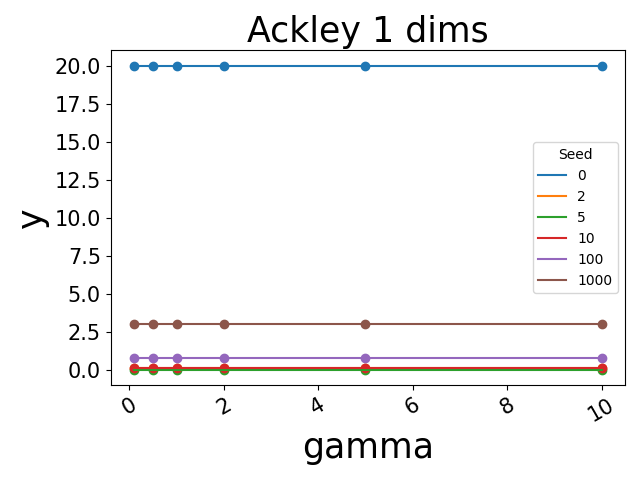
Badania odbywały się poprzez zmianę tylko jednego parametru na raz w celu sprawdzenia, czy występują jakieś zależności. Wnioski i obserwacje były tworzone na podstawie wykresów oraz ręcznego przeglądu plików wyjściowych. Niektóre badania zostały opisane w dużym skrócie, jeśli nie wniosły nic wartościowego 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.

## Funkcja Ackley’a

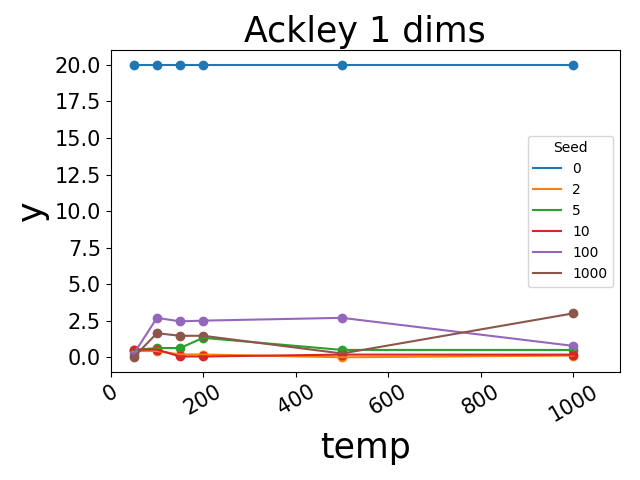
### 1 wymiar

  
Zależność wyniku y od ustawień współczynnika Boltzmana

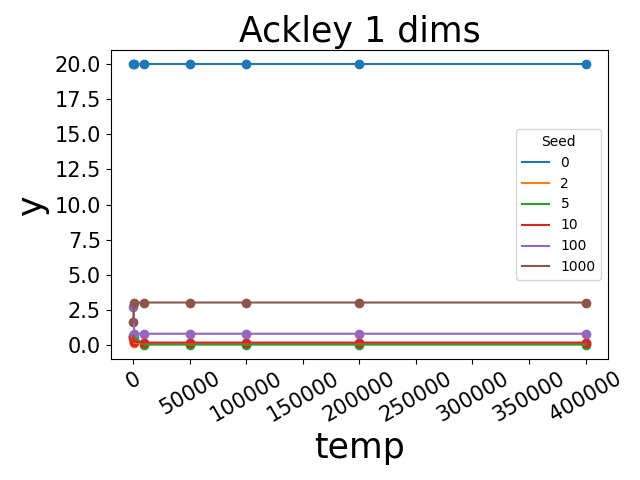
Wpływ ustawień wsp. Boltzmana na wyniki był niezauważalny – wyniki pozostawały stałe dla każdego ustalonego ziarna. Wyjątkiem było ustawienie kb = 0, gdzie wyniki się zmieniły, jednak nie było w nich wyraźnej zależności. Dla większości kb =0 pogorszyło wyniki, ale np. dla seed = 1000 wynik był wtedy lepszy.

  
Zależność wyniku od ustawień gamma

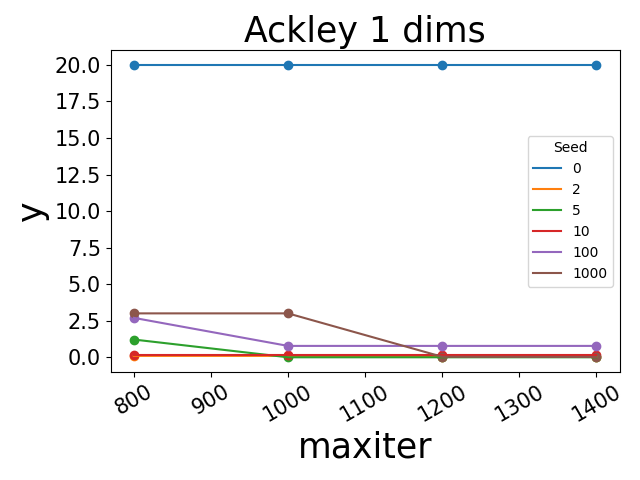
Gamma w odróżnieniu od stałej Boltzmana nie miała absolutnie żadnego wpływu na wynik niezależnie od testowanych ustawień.

  
Zależność wyniku y od ustawień temperatury – dokładne badanie

Domyślne ustawienie temperatury wynosiło 100 000, jednak jakiekolwiek zmiany były możliwe do zaobserwowania dopiero po zejściu poniżej 1000. Nie było żadnej jednoznacznej zależności, jednak dla większości przypadków zwiększenie temperatury poprawiało wyniki do pewnego momentu.

  
Zależność wyniku y od ustawień temperatury – ogólne badanie

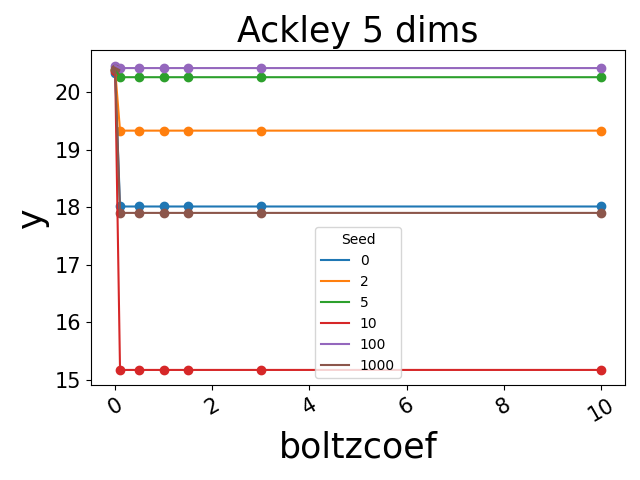
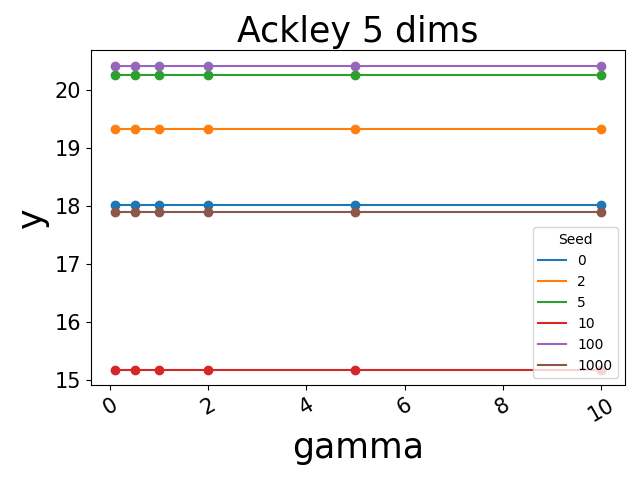
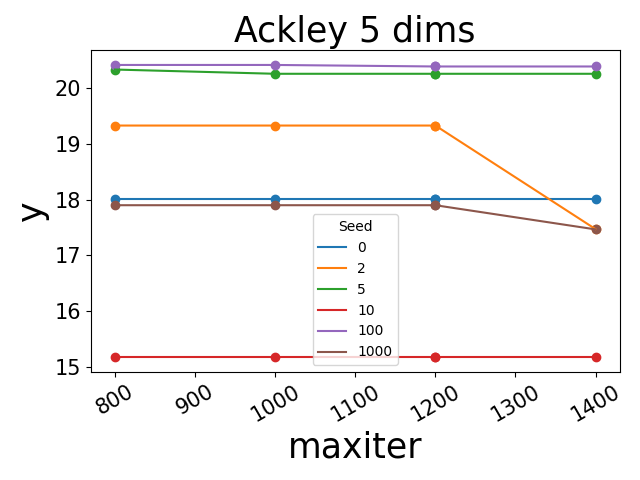
Patrząc na większy zakres wartości, temperatura po pewnym progu przestaje mieć wpływ na wynik.

  
Zależność wyniku od ustawień maxiter

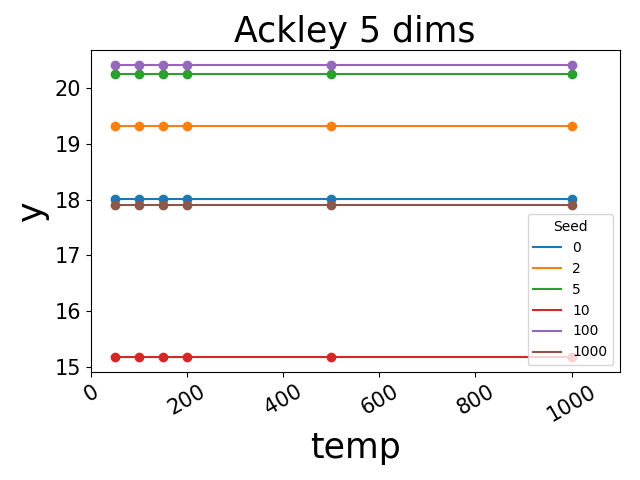
Najbardziej wyraźną zależność miał parametr maxiter, ponieważ wynik końcowy poprawiał się wraz ze zwiększaniem iteracji. Dla części seed widać jednak, że po pewnym momencie wynik się zatrzymywał i większa liczba iteracji nic nie dała. Można to jednak porównać do zwykłego zgadywania – oczywiście im więcej „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 rozwiązania były bardzo dobre i często przyjmowały wartości bardzo bliskie 0.

### 5 wymiarów



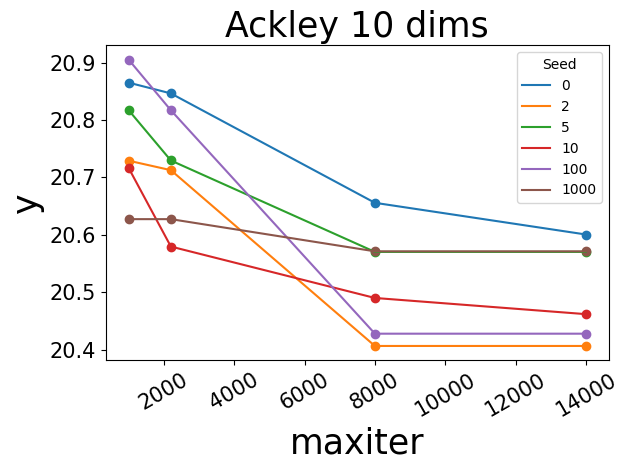
Dla 5 wymiarów większość zależności (a raczej ich brak) był analogiczny. Wyjątkiem była temperatura przedstawiona poniżej – w tym przypadku nawet dla niskich wartości nic się nie działo.

  
Zależność wyniku od temperatury – dokładne porównanie

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.

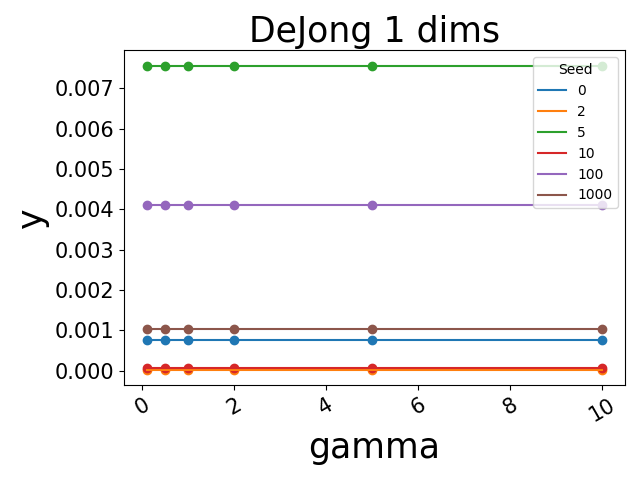
  
Zależność Y od maxiter

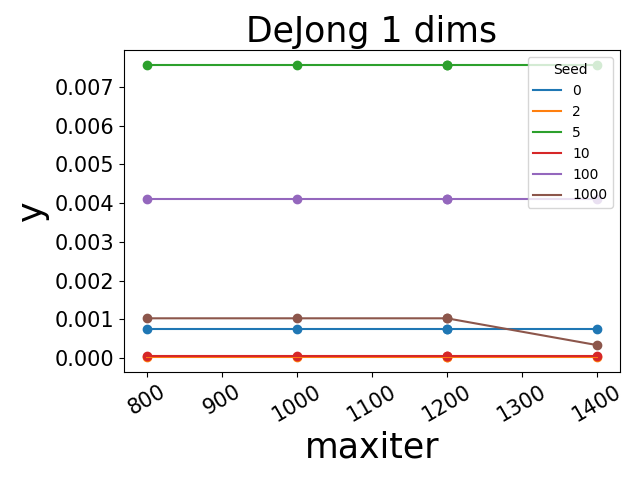
Jako przykład jednego parametru wstawiony został wykres Y od maxiter. Można z niego wywnioskować, że algorytm nie poradził sobie z zadaniem minimalizacji, ponieważ nawet przy 14000 iteracji nie jest w stanie nawet przybliżyć się do 0.

## 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

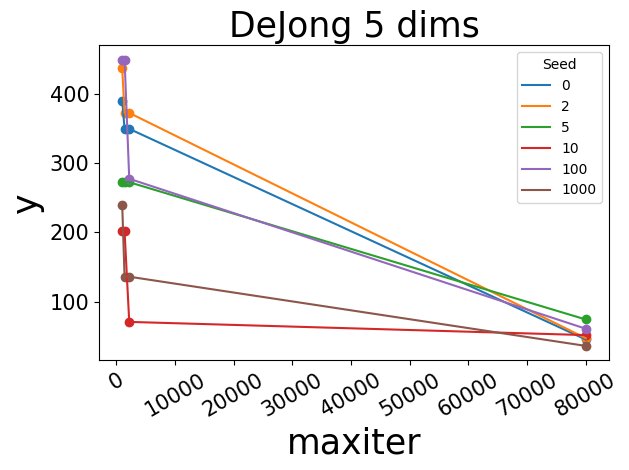
  
Wynik w zależności od iteracji

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 = 136.2839, co jest daleko od oczekiwanego 0.

W tym przypadku przeprowadzone zostało również badanie, czy przeprowadzenie ogromnej liczby iteracji pozwoli poprawić wyniki końcowe

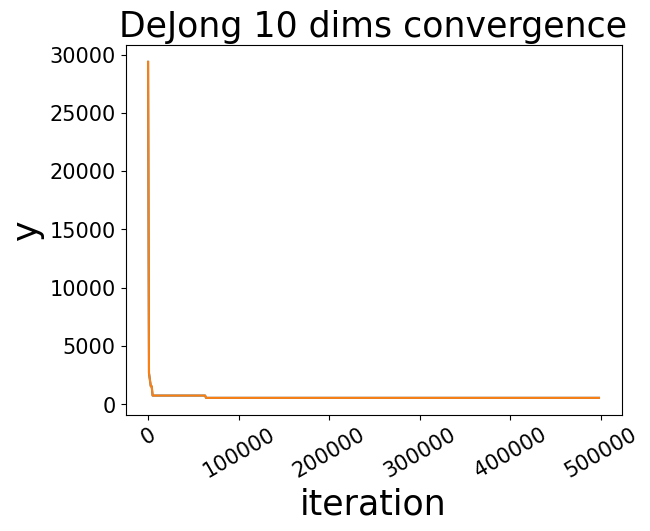
  
Wpływ dużej liczby iteracji na wyniki

Można zaobserwować, że ogromna liczba iteracji pomogła uzyskać lepszy wynik końcowy, jednak jest on dalej równy y = 36,63467, co jest bardzo słabym osiągnięciem. Dalsze zwiększanie iteracji było by już nieopłacalne ze względu na czas obliczeń oraz fakt, że to praktycznie losowa „zgadywanka” (choć z ciekawości wykonano 500 000 iteracji dla najlepszego seed, co dało wynik y = 10.45989).

Innym wnioskiem z tego badania jest to, że ten algorytm/ta implementacja jest bardzo wolno zbieżna.

### 10 wymiarów

Dla 10 wymiarów przy ustalonych 1400 iteracjach udało się uzyskać wynik zaledwie y = 1543.112. Dla innego ustawienia - 500 000 iteracji dodatkowo została sprawdzona zbieżność do rozwiązania:

  
Zależność najlepszego Y od iteracji algorytmu

Jak można zaobserwować poprawa występuje tylko raz na kilkadziesiąt tysięcy iteracji, a ponadto od ok. 80 tysięcy już się nie zmienia. Oznacza to, że metoda nie tylko jest wolno zbieżna, ale wręcz działa losowo i przechodzi krok dalej tylko kiedy „akurat się trafi lepiej” – nie ma stabilnego, konsekwentnego postępu.

Przeprowadzono również prosty test polegający na połączeniu 10-krotnie wyższej temperatury początkowej T = 1 000 000. Wynik w obu wersjach wyniósł y = 557.4477.

## 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 = 61.84226.

### 10 wymiarów

Nic się nie zmieniło. W celach porównawczych najlepszy wynik y = 171.7201 został zapisany przy ustawieniu 1400 iteracji. W porównaniu do funkcji DeJonga wynik był zaskakująco niski, jednak wciąż daleki od ideału.

## Porównanie

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 | 15.17185 |
| Ackley | 10 | 20.57954 |
| DeJong | 1 | 2.095604e-05 |
| DeJong | 5 | 136.2839 |
| DeJong | 10 | 1543.112 |
| Rastragin | 1 | 0.002085363 |
| Rastragin | 5 | 61.84226 |
| Rastragin | 10 | 171.7201 |

Według osiąganych wyników można powiedzieć, że najtrudniejszą funkcją dla tego algorytmu jest f. DeJong’a, średnia jest f. Rastragina, a najprostsza jest funkcja Ackley’a. Nawet dla najprostszej z nich, wyniki dla wielu wymiarów były bardzo słabe, ale we wszystkich przypadkach jednowymiarowych algorytm poradził sobie całkiem dobrze.

## 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
* Nawet przy ogromnej liczbie iteracji algorytm ma problem z dojściem do choć dobrych wyników przy wielu wymiarach
* Algorytm radzi sobie dobrze tylko dla funkcji o bardzo niskiej liczbie wymiarów (łatwych)
* Zbieżność algorytmu jest niska i bardzo 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.
* Przy braku popraw wraz z iteracjami, nie ma pewności, że algorytm kiedykolwiek znajdzie lepsze rozwiązanie – ciężko zdecydować kiedy zatrzymać
* Trudność funkcji ma wpływ na osiągane wyniki

Podsumowując, prymitywność i prostota algorytmu sprawia, że jest **skuteczny tylko dla bardzo prostych zadań**. Dla bardziej skomplikowanych problemów banalne **podejście losowego przeszukiwania wokół znanego minimum jest niewystarczające**, nawet przy wielu tysiącach iteracji.

# Bardziej odporna implementacja w Python