Wojskowa Akademia Techniczna

Wydział Cybernetyki

**METODY OPTYMALIZACJI INSPIROWANE BIOLOGICZNIE**

Laboratorium 2

Strategia 1+1

Skład grupy:

Łukasz CĄKAŁA

Bartosz FIUTKA

Paulina FORMAS

Sebastian ROMANIUK

Spis treści

[1. Zad1 3](#_Toc137805814)

[1.1. Język Programowania 3](#_Toc137805815)

[1.2. Kod Programu 3](#_Toc137805816)

[1.3. Opis Kodu 5](#_Toc137805817)

[1.4. Przykłady 6](#_Toc137805818)

[1.4.1. Nr 1 6](#_Toc137805819)

[1.4.2. Nr2 6](#_Toc137805820)

[2. Zad2 7](#_Toc137805821)

[2.1. Język Programowania 7](#_Toc137805822)

[2.2. Kod Programu 7](#_Toc137805823)

[2.3. Opis Kodu 9](#_Toc137805824)

[2.4. Przykłady 11](#_Toc137805825)

[2.4.1 Nr1 11](#_Toc137805826)

[2.4.2. Nr2 11](#_Toc137805827)

[2.4.3 Nr3 12](#_Toc137805828)

[2.5. Porównanie z Lab1 Zad1 13](#_Toc137805829)

# Zad1

Lider Sprawozdania Zad1 i Zad2 – Bartosz Fiutka

Wykorzystując strategię ewolucyjną (1+1) znajdź maksimum funkcji:

Dla x [-2, 2], i = 1, 2

W celu zobrazowania dynamiki prowadzonych obliczeń proszę dla 100 generacji wykonać wykresy wartości funkcji przystosowania oraz zmian zasięgu mutacji 𝜎w zależności od liczby generacji.

## Język Programowania

Algorytm zaimplementowałem w języku Python 3.11

## Kod Programu

import random  
import matplotlib.pyplot as plt  
  
# PARAMETRY  
c1 = 0.82  
c2 = 1 / c1  
fi = 1  
k = 5  
t = 100 # iteracje  
x = [random.uniform(-2, 2), random.uniform(-2, 2)] # chromosom  
ilosc\_sukcesow = 0  
  
# Wykres  
time = []  
tab\_wyniki = []  
tab\_fi = []  
  
# Debug  
debug = True  
  
def f\_x1\_x2(tab): # funkcja przystosowania  
 return 2 - tab[0] \*\* 2 - tab[1] \*\* 2  
  
  
def generate\_y(value, fii):  
 new = value + fii \* random.gauss(0, 1)  
 if new < -2: new = -2  
 if new > 2: new = 2  
 return new  
  
  
for i in range(t):  
 if i % k == 0 and i != 0:  
 if debug: print(f"Old fi: {fi}")  
 if (ilosc\_sukcesow / k) < 0.2:  
 fi = c1 \* fi  
 elif (ilosc\_sukcesow / k) > 0.2:  
 fi = c2 \* fi  
 if debug: print(f"New fi: {fi}")  
 ilosc\_sukcesow = 0  
  
 wynik\_x = f\_x1\_x2(x)  
 y = [generate\_y(x[0], fi), generate\_y(x[1], fi)]  
 wynik\_y = f\_x1\_x2(y)  
  
 if wynik\_y > wynik\_x:  
 x = y  
 ilosc\_sukcesow += 1  
  
 time.append(i)  
 tab\_wyniki.append(wynik\_x)  
 tab\_fi.append(fi)  
  
 if debug: print(f"Iteracja: {i+1}, Sukces: {ilosc\_sukcesow}, X: {x}, Wynik\_x: {round(wynik\_x, 4)}, Y: {y}, Wynik\_y: {round(wynik\_y, 4)}")  
  
plt.plot(time, tab\_wyniki, label="Wartosc\_funkcji")  
plt.plot(time, tab\_fi, label="Wartosc\_mutacji")  
  
plt.xlabel("Iteracja")  
plt.ylabel("Wartosc")  
  
plt.legend()  
  
plt.show()

Zakładam parametry:

C1 = 0.82

C2 = 1/ C1

fi (wartość startowa) = 1

k (ilość sukcesów) = 5

t (ilość iteracji programu) = 100

## Opis Kodu

Pozostałe zmienne:

Początkowy chromosom i ilość sukcesów startowa:

x = [random.uniform(-2, 2), random.uniform(-2, 2)] # chromosom  
ilosc\_sukcesow = 0

Do narysowania wykresów:

# Wykres  
time = []  
tab\_wyniki = []  
tab\_fi = []

Do Dubugowania wyników:

# Debug  
debug = True

Funkcja celu:

def f\_x1\_x2(tab): # funkcja przystosowania  
 return 2 - tab[0] \*\* 2 - tab[1] \*\* 2

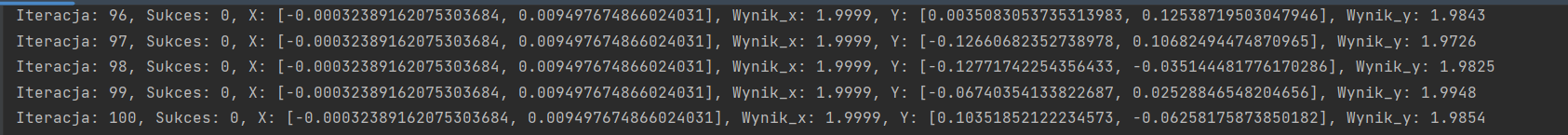
Główna pętla algorytmu:

for i in range(t):   
 if i % k == 0 and i != 0: #zmiana wartości fi(mutacji)  
 if debug: print(f"Old fi: {fi}")  
 if (ilosc\_sukcesow / k) < 0.2:  
 fi = c1 \* fi  
 elif (ilosc\_sukcesow / k) > 0.2:  
 fi = c2 \* fi  
 if debug: print(f"New fi: {fi}")  
 ilosc\_sukcesow = 0  
  
 wynik\_x = f\_x1\_x2(x) # zapisanie wyniku obecnego  
 y = [generate\_y(x[0], fi), generate\_y(x[1], fi)] # nowy chromosom  
 wynik\_y = f\_x1\_x2(y) # zapisanie wyniku nowego  
  
 if wynik\_y > wynik\_x: # jezeli sukces  
 x = y  
 ilosc\_sukcesow += 1  
  
 time.append(i) # dodanie wartosci do tablic do wykresow  
 tab\_wyniki.append(wynik\_x)  
 tab\_fi.append(fi)  
  
 if debug: print(f"Iteracja: {i+1}, Sukces: {ilosc\_sukcesow}, X: {x}, Wynik\_x: {round(wynik\_x, 4)}, Y: {y}, Wynik\_y: {round(wynik\_y, 4)}")

Poza główną pętlą mam rysowanie wykresów na podstawie tablic:  
plt.plot(time, tab\_wyniki, label="Wartosc\_funkcji")  
plt.plot(time, tab\_fi, label="Wartosc\_mutacji")  
  
plt.xlabel("Iteracja")  
plt.ylabel("Wartosc")  
  
plt.legend()  
  
plt.show()

## Przykłady

### Nr 1

Dla wyżej przedstawionych parametrów:  


Przy 100-tnej iteracji program się zakończył i widać że max(f(x)) = 1.9999 dla chromosomu:

X: [-0.00032389162075303684, 0.009497674866024031]

Wykres:

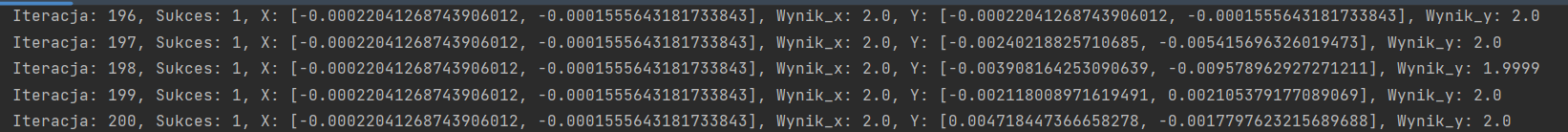
Obraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

Widać że w okolicy 30 iteracji algorytm nie zmieniał swojej obecnej największej wartości (znalazł max) oraz mutacja spada wraz z kolejnymi iteracjami od iteracji 10

### Nr2

Dla iteracji = 200 (t = 200)



W tym przypadku max(f(x)) = 2.0 dla chromosomu:

X: [-0.00022041268743906012, -0.0001555643181733843]

Jak widać udało się osiągnąć dokładny max = 2. Oczywiście dla t = 100 też jest to możliwe, ponieważ algorytm działa na zasadzie losowania genów o pewien zakres mutacji.

Jest to losowe co oznacza że dla t = 200 też moglibyśmy nie otrzymać wyniku 2.0

Wykres:

Obraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

Tutaj widać że max został osiągnięty w okolicy 25 iteracji

# Zad2

Dla danych jak w zadaniu 1 z laboratorium 1 proszę opracować algorytm genetyczny bazując na własnych opracowaniach laboratoryjnych z lab.1 i porównać uzyskane wyniki z wynikami strategii ewolucyjnej (1+1).

## Język Programowania

Algorytm zaimplementowałem w języku Python 3.11

## Kod Programu

Poniżej przedstawiam kod strategii (1+1) dla danych z Lab1 zad1

import random  
import matplotlib.pyplot as plt  
import math  
  
# PARAMETRY  
c1 = 0.82  
c2 = 1 / c1  
fi = 1  
k = 5  
  
x = [random.randint(0, 31)] # chromosom  
ilosc\_sukcesow = 0  
  
iteracja = 0  
podRzad = 0  
ilePodRzad = 30  
  
# Wykres  
time = []  
tab\_wyniki = []  
tab\_fi = []  
  
# Debug  
debug = True  
  
  
def f\_x1(tab): # funkcja przystosowania  
 return (tab[0] / 15) \* (math.sin((math.pi \* tab[0]) / 8) + 1)  
  
  
def generate\_y(value, fii):  
 new = value + fii \* random.gauss(0, 1)  
 if new < 0: new = 0  
 if new > 31: new = 31  
 return new  
  
  
def wykres():  
 plt.plot(time, tab\_wyniki, label="Wartosc\_funkcji")  
 plt.plot(time, tab\_fi, label="Wartosc\_mutacji")  
  
 plt.xlabel("Iteracja")  
 plt.ylabel("Wartosc")  
  
 plt.legend()  
  
 plt.show()  
  
  
while True:  
 if iteracja % k == 0 and iteracja != 0:  
 if (ilosc\_sukcesow / k) < 0.2:  
 fi = c1 \* fi  
 elif (ilosc\_sukcesow / k) > 0.2:  
 fi = c2 \* fi  
 ilosc\_sukcesow = 0  
  
 wynik\_x = f\_x1(x)  
 y = [generate\_y(x[0], fi)]  
 wynik\_y = f\_x1(y)  
  
 if wynik\_y > wynik\_x:  
 x = y  
 ilosc\_sukcesow += 1  
 podRzad = 0  
 else:  
 podRzad += 1  
  
 time.append(iteracja)  
 tab\_wyniki.append(wynik\_x)  
 tab\_fi.append(fi)  
  
 if debug: print(  
 f"Iteracja: {iteracja + 1}, Sukces: {ilosc\_sukcesow}, X: {x}, Wynik\_x: {round(wynik\_x, 4)}, Y: {y}, Wynik\_y: {round(wynik\_y, 4)}, PodRzad: {podRzad}")  
 iteracja += 1  
   
 if podRzad >= ilePodRzad:  
 wykres()  
 break

Zakładam parametry:

Takie jak w Zad1 z pewnymi różnicami:

W tym przypadku nie bierzemy pod uwagę ilości iteracji, ale ilość wystąpienia tego samego chromosomu pod rząd. Tak samo jak zrobiliśmy w Lab1 zad1  
podRzad = 0

ilePodRzad = 30 (jeżeli będzie 30 pod rząd taki sam chromosom to kończymy działanie)

## Opis Kodu

Początkowy chromosom

x = [random.randint(0, 31)] # chromosom liczby calkowite  
ilosc\_sukcesow = 0

wyniki = [0, 0] # tablica ostatnich wynikow

Do wykresów i debug:

# Wykres  
time = []  
tab\_wyniki = []  
tab\_fi = []  
  
# Debug  
debug = True

Funkcja celu:

def f\_x1(tab): # funkcja przystosowania  
 return (tab[0] / 15) \* (math.sin((math.pi \* tab[0]) / 8) + 1)

Generowanie nowego chromosomu:

def generate\_y(value, fii):  
 new = value + fii \* random.gauss(0, 1)  
 if new < 0: new = 0  
 if new > 31: new = 31

new = round(new, 0) # liczby calowite  
 return new

Rysowanie wykresu:

def wykres():  
 plt.plot(time, tab\_wyniki, label="Wartosc\_funkcji")  
 plt.plot(time, tab\_fi, label="Wartosc\_mutacji")  
  
 plt.xlabel("Iteracja")  
 plt.ylabel("Wartosc")  
  
 plt.legend()  
  
 plt.show()

Główna pętla algorytmu:

while True:  
 if iteracja % k == 0 and iteracja != 0:  
 if (ilosc\_sukcesow / k) < 0.2:  
 fi = c1 \* fi  
 elif (ilosc\_sukcesow / k) > 0.2:  
 fi = c2 \* fi  
 ilosc\_sukcesow = 0  
  
 wynik\_x = f\_x1(x)  
 y = [generate\_y(x[0], fi)]  
 wynik\_y = f\_x1(y)  
  
 if wynik\_y > wynik\_x:  
 x = y  
 ilosc\_sukcesow += 1  
 wyniki.append(wynik\_y)  
 else:  
 wyniki.append(wynik\_x)  
  
 if (iteracja + 1) % 2 == 0: # co 2 iteracja sprawdzamy wyniki   
 if wyniki[0] == wyniki[1]: # jeżeli takie same  
 podRzad += 1 # dodajemy podRzad  
 wyniki.clear() # czyścimy tablice  
 else: # jeżeli rozne  
 podRzad = 0  
 wyniki.clear()  
  
 time.append(iteracja)  
 tab\_wyniki.append(wynik\_x)  
 tab\_fi.append(fi)  
  
 if debug: print(  
 f"Iteracja: {iteracja + 1}, Sukces: {ilosc\_sukcesow}, X: {x}, Wynik\_x: {round(wynik\_x, 4)}, Y: {y}, Wynik\_y: {round(wynik\_y, 4)}, PodRzad: {podRzad}")  
 iteracja += 1  
   
 if podRzad >= ilePodRzad: # zakończenie programu  
 wykres()  
 break

## Przykłady

### 2.4.1 Nr1

Obraz zawierający tekst, zrzut ekranu, Czcionka, informacja

Opis wygenerowany automatycznie

Widać że po 82 iteracjach udało się uzyskać wartość ta sama pod rząd 30 razy dla wyniku max(f(x)) = 2.69 dla x = 21

Wykres:

Obraz zawierający tekst, diagram, linia, zrzut ekranu

Opis wygenerowany automatycznie

Widać że w okolicy 20 iteracji wynik się ustabilizował

### 2.4.2. Nr2

Zdarza się że program na podanych wyżej parametrach zatrzymuje się w ekstremum lokalnym

Obraz zawierający tekst, zrzut ekranu, Czcionka, informacja

Opis wygenerowany automatycznie

A konkretnie w x = 6

Jest to spowodowane dobraniem złych parametrów do problemu. Wcześniej zakres danych był [-2, 2]. Teraz jest [0, 31] co powoduje że powinniśmy mieć dużo większą mutacje bo liczby powinny się bardziej zmieniać

Wykres:

Obraz zawierający tekst, diagram, zrzut ekranu, linia

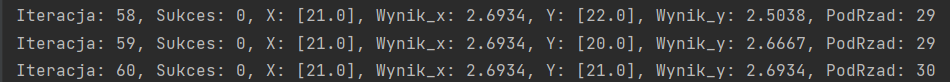
Opis wygenerowany automatycznie

### 2.4.3 Nr3

Przy innych parametrach:

fi = 5

Już bardzo rzadko występuje x = 6. Przykład:



Wykres:

Obraz zawierający tekst, diagram, linia, zrzut ekranu

Opis wygenerowany automatycznie

Widać że bardzo ważne jest dobranie odpowiednich parametrów do problemu aby algorytm zadziałał poprawnie.

## Porównanie z Lab1 Zad1

|  |  |
| --- | --- |
| **(1+1)** | **Algorytm Genetyczny Lab1 Zad1** |
| X = 21 f(x) = 2.69 Iteracje = 60 | X = 22 f(x) = 2.50 Iteracje = 34 |
| X = 21 f(x) = 2.69 Iteracje = 72 | X = 21 f(x) = 2.69 Iteracje = 30 |
| X = 21 f(x) = 2.69 Iteracje = 70 | X = 21 f(x) = 2.69 Iteracje = 30 |
| X = 21 f(x) = 2.69 Iteracje = 78 | X = 19 f(x) = 2.43 Iteracje = 44 |
| X = 21 f(x) = 2.69 Iteracje = 70 | X = 19 f(x) = 2.43 Iteracje = 32 |

Jak widać Algorytm Genetyczny jest dużo szybszy, ale mniej dokładny. Stragia 1+1 za każdym razem dała odpowiednie rozwiązanie, ale dużo wolniej.

Jest to spowodowane złym doborem ilości populacji w Algorytmie Genetycznym. Jeżeli ustawimy populacje na 100 (wcześniej 10) to za każdym razem dostaniemy wynik maksymalny w max 34 iteracjach.

Nie da się jednoznacznie powiedzieć który algorytm jest lepszy bo wszystko zależy od wylosowanych wartości i parametrów, ale jeżeli mielibyśmy nieograniczone wielkości tablic, populacji i pełną dowolność w parametrach to algorytm genetyczny byłby szybszy bo od razu znajdował by rozwiązanie opymalne.