

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

KIERUNEK: Telekomunikacja (TEL)
SPECJALNOŚĆ: Multimedia w telekomunikacji (TMU)

**PRACA DYPLOMOWA
INŻYNIERSKA**

Projektowanie cyfrowych filtrów z
wykorzystaniem metod inteligentnych
systemów przetwarzania sygnałów na
przykładzie algorytmów ewolucyjnych

The design approaches of digital filters based on
evolutionary optimization techniques

AUTOR:
Kowalczyk Konrad

PROWADZĄCY PRACĘ:
Dr inż. Agnieszka Wielgus, W12n

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszy projekt inżynierski wykonałem(-am) osobiście i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Wrocław, dnia Podpis dyplomanta

Spis treści

1	Wstęp	4
1.1	Cel pracy dyplomowej	4
1.2	Założenia pracy dyplomowej	4
1.3	Wykorzystane narzędzia	5
2	Algorytm	7
2.1	Opis słowny	8
2.2	Lista kroków	8
2.3	Schemat blokowy	9
2.4	Pseudokod	11
3	Filtr cyfrowy	12
3.1	Filtr o skończonej odpowiedzi impulsowej (SOI)	13
4	Algorytm oparty na strategiach ewolucyjnych lub hybrydowych	16
4.1	Algorytm Ewolucyjny	16
4.1.1	Algorytm PSO	18
4.1.2	Schemat blokowy oraz parametry kontrolne	20
4.2	Algorytm hybrydowy	20
4.2.1	Algorytm CSPSO	20
4.2.2	Schemat blokowy oraz parametry kontrolne	21
4.2.3	Funkcja Gamma (Γ)	21
4.2.4	Funkcja lotu Levy'ego	21
5	Implementacja	23
5.1	Algorytm PSO	23
5.1.1	Generowanie pojedynczego rozwiązania	23
5.1.2	Generowanie prędkości	23
5.1.3	Obliczenie $V(\omega_n)$	23
5.1.4	Generowanie prędkości początkowych	24
5.1.5	Obliczenie funkcji dopasowania	24
5.1.6	Tworzenie nowych rozwiązań z obecnego pokolenia	24
5.1.7	Skrypt	25
5.2	Algorytm CSPSO	27
5.2.1	Generowanie pojedynczego rozwiązania	27
5.2.2	Obliczenie $V(\omega_n)$	27
5.2.3	Generowanie prędkości początkowych	27
5.2.4	Obliczenie funkcji dopasowania	27
5.2.5	Tworzenie nowych rozwiązań z obecnego pokolenia	28

SPIS TREŚCI	3
5.2.6 Funkcja Lotu Levy’ego	29
6 Podsumowanie	31

Rozdział 1

Wstęp

1.1 Cel pracy dyplomowej

Celem pracy dyplomowej jest skonstruowanie oraz przebadanie algorytmów bazujących na strategiach ewolucyjnych dla problemu projektowania filtrów cyfrowych (Finite Impulse Response) o wielu pasmach zaporowych oraz liniowej fazie w pasmach przepustowych przy ograniczeniu na wysokości zafalowań w obu pasmach.

1.2 Założenia pracy dyplomowej

Założeniem projektu inżynierskiego jest skorzystanie z dostępnych narzędzi wymienionych w rozdziale pt. "Wykorzystane narzędzia". Następnie znalezienie dostępnych materiałów na temat implementacji algorytmów ewolucyjnych oraz filtrów cyfrowych. Kolejnym elementem pracy dyplomowej będzie przeanalizowanie dostępnych materiałów i na ich podstawie implementacja tytułowych algorytmów. W zamyśle autora pracy będzie implementacja dwóch lub więcej takich algorytmów z wykorzystaniem metod strategii ewolucyjnych. Po zakończeniu fazy implementacji należy zweryfikować czy rozwiązania są prawidłowo zrealizowane oraz w razie negatywnego wyniku, wprowadzenie poprawek. Kolejnym etapem będzie sprawdzenie szybkości wykonywania badanych algorytmów. Ostatecznym etapem wieńczącym pracę będzie zredagowanie podsumowania z odpowiedzią na pytanie, czy tytułowe algorytmy są w stanie zaprojektować filtr w sposób optymalny.

Streszczenie w języku angielsku

The main target of the thesis is to verify how evolutionary algorithms will cope with the problem of designing and producing a digital filter with a finite impulse response. The problem will be solved for the general case, but for the thesis considerations, the implementation will be supported by an example of a digital filter of $n = 99$ order. This means that the filter will consist of 240 samples. The next necessary step after creating the filter will be the necessity to verify the results in terms of correctness and effectiveness against the background of other algorithms that will be presented in this paper. The next step will include checking the correctness of the algorithm against the background of a filter designed in the Matlab environment or the Sci-Py package. The culmination of the thesis will be the specification of the approximate computational complexity of algorithms, efficiency (correlation coefficient in relation to a filter designed in the Matlab / SciPy

environment) and a summary assessment of whether the tested solution is beneficial for the business.

1.3 Wykorzystane narzędzia

- Python - Język wysokopoziomowy dynamicznie typowany ogólnego przeznaczenia, stworzony przez Guido van Rossum'a w roku 1989. Dla celów pracy dyplomowej Python będzie wykorzystywany w wersji 3.8 lub kolejnych wersji. Dokumentacja dla języka Python znajduje się pod adresem : <https://docs.python.org/3.8/>
- NumPy - Pakiet narzędzi dla języka Python obsługujący obliczenia inżynierskie m.in. obliczenia na macierzach, operacje numeryczne, funkcje błędów itp. Dokumentacja do biblioteki znajduje się pod adresem : <https://numpy.org/doc/>
- Pandas - Pakiet narzędzi dla języka Python umożliwiający wczytywanie, analizę oraz wiele operacji przetwarzania informacji lub obrabiania informacji do postaci niezbędnej do prezentacji lub dalszej pracy Dokumentacja do biblioteki znajduje się pod adresem : <https://pandas.pydata.org/docs/>
- Matplotlib - Pakiet dla języka Python stworzony na podstawie środowiska Matlab oraz pakietu NumPy umożliwiający tworzenie wizualizacji informacji w formie wykresów, dokumentacja do biblioteki znajduje się pod adresem: <https://matplotlib.org/stable/contents.html>
- Matlab - Środowisko umożliwiające wykonywanie wszelkich obliczeń inżynierskich wraz z ich wizualizacją z dużą ilością dodatkowych pakietów m.in. dsptoolbox, 5G toolbox itp.
- App.diagrams.net - Narzędzie umożliwiające tworzenie wszelkiego rodzaju schematów blokowych, schematów Gantta pozwalający na wizualizację działania pracy/ algorytmów oraz uproszczenie informacji do prostej formy wizualnej

Wyróżnia się szereg technik heurystycznych należą do nich m.in.:

- Metoda wspinaczki
- Symulowane wyżarzanie
- Przeszukiwanie losowe
- Algorytmy zachłanne
- Przeszukiwanie iteracyjne

Rozdział 2

Algorytm

Algorytm jest to ciąg ściśle zdefiniowanych czynności, które wykonane w odpowiedniej kolejności oraz czasie prowadzą do rozwiązania problemu z określonymi parametrami wejściowymi i odpowiedzią w postaci parametrów wyjściowych. Algorytm w dużym uproszczeniu można interpretować jako tzw. „Czarna skrzynka”



Rysunek 2.1 Uproszczony schemat blokowy każdego algorytmu

Każdy algorytm można przedstawić jako:

- Opis słowny
- Lista kroków
- Schemat blokowy
- Pseudokod

W kolejnych częściach pracy dyplomowej zostaną przedstawione metody prezentacji algorytmów na przykładzie algorytmu NWD(Największy wspólny dzielnik)tzw. algorytm Euklidesa

2.1 Opis słowny

Opisanie procesu krótkimi zdaniami, które jednoznacznie sugerują wykonywaną czynność. Wczytaj liczby a i b , które są liczbami naturalnymi. Dopóki liczby a i b są różne od siebie, odejmujemy liczbę mniejszą od większej. Gdy liczby a i b staną się sobie równe to $\text{NWD}(a, b)$ jest dowolną liczbą, która została uzyskana w ostatnim kroku.

2.2 Lista kroków





Opisanie procesu za pomocą krótkich poleceń ponumerowanych w kolejności ich realizacji

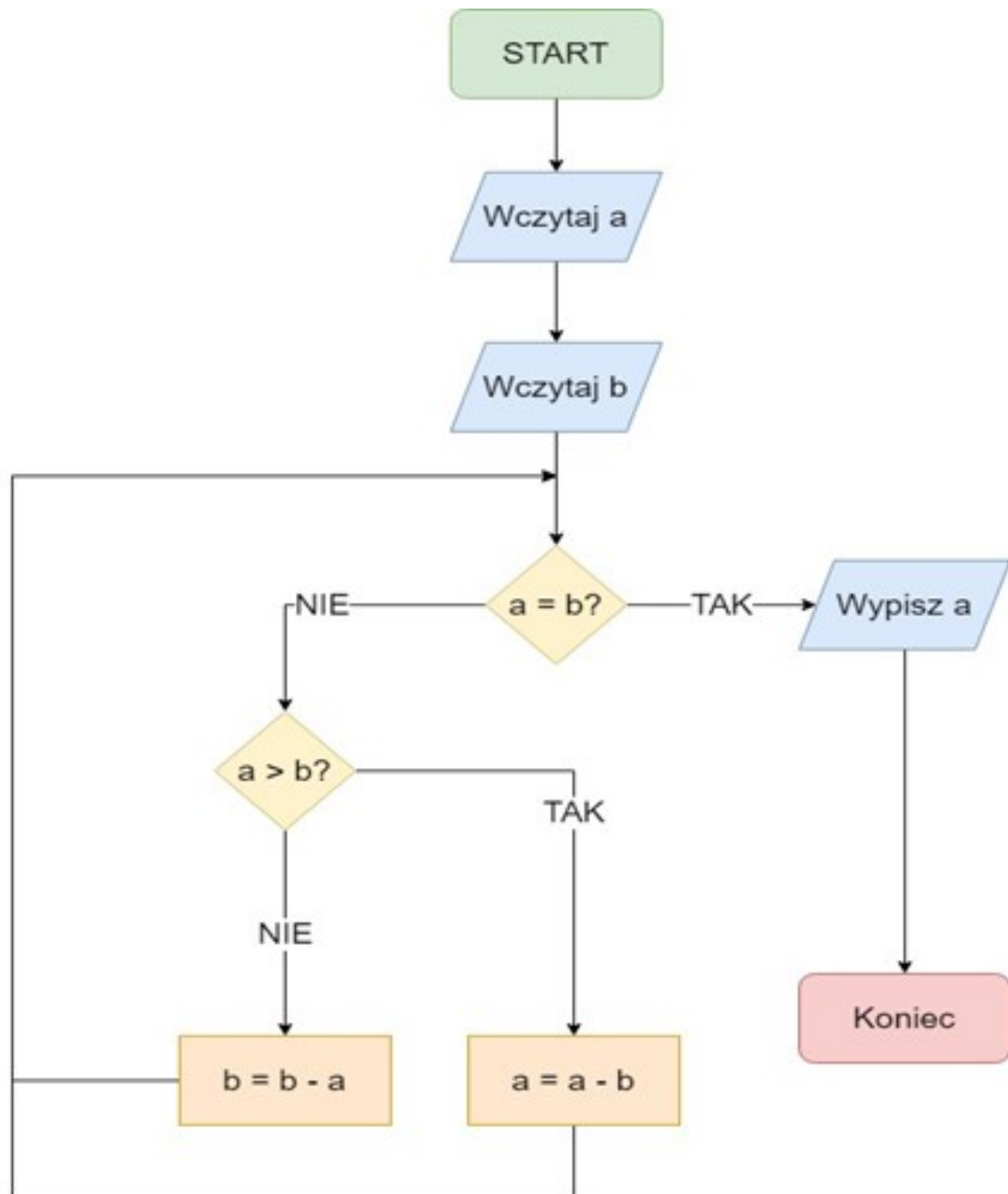
1. Wczytaj liczbę a zgodną z założeniami początkowymi algorytmu
2. Wczytaj liczbę b zgodną z założeniami początkowymi algorytmu
3. Jeżeli liczba $b = 0$ idź do kroku 6 w innym wypadku odejmij mniejszą liczbę od większej
4. Większą liczbę zastąp mniejszą
5. Mniejszą liczbę zastąp resztą z dzielenia (operacja modulo)
6. Wróć do kroku 2
7. Wypisz liczbę a
8. Zakończ działanie programu

2.3 Schemat blokowy

Graficzna prezentacja kolejnych etapów, umożliwiających precyzyjne określenie wykonywanie czynności m.in. Rozpoczęcie algorytmu, zakończenie, wczytanie informacji, wyświetlenie informacji, lub bloku decyzyjnego.

Tabela. 2.1 Przedstawienie podstawowych bloków używanych do tworzenia schematów blokowych

	Blok graniczny - Symbol oznaczający początek, koniec, przerwanie lub wstrzymanie działania programu
	Blok wejścia/ wyjścia -Reprezentuje czynność wprowadzania danych do programu
	Blok procesu - Reprezentuje operację w wyniku której zmienia się wartość lub zapis danych w zmiennej
	Blok decyzyjny - Reprezentuje wybór jednego z dwóch możliwych stanów tzw. True (prawda) lub false (fałsz). Jeżeli badany warunek nie spełnia założenia wykonywany jest blok przypisany do części fałszu logicznego w drugim przypadku wykonywany jest blok przypisany do wartości prawdy logicznej.



Rysunek 2.2 Schemat blokowy algorytmu Euklidesa

2.4 Pseudokod

Prezentacja algorytmu podobnie jak w przypadku opisu słownego jednak w tym przypadku kolejne kroki zawierają kod przypominający składnię języka lub będący częścią składni m.in. średniki, dwukropki, wyrażenia typu for, while if itd.

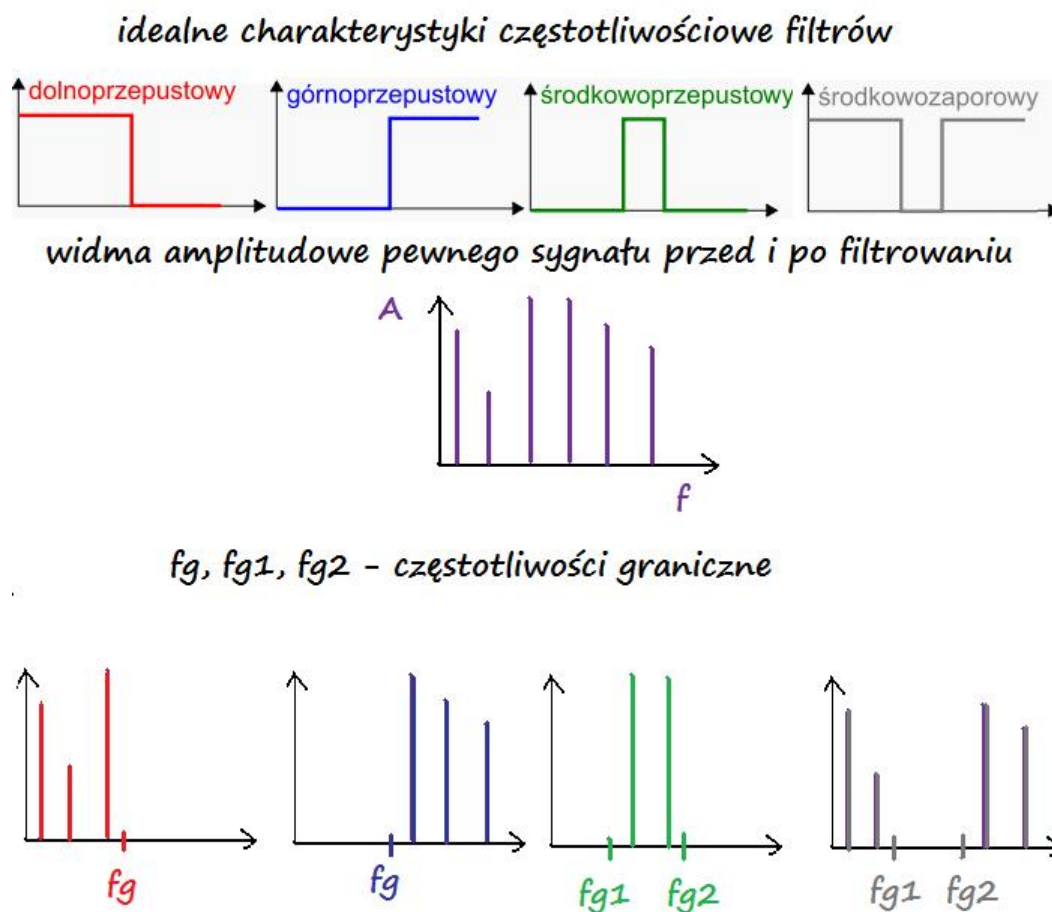
```
1  def NWD(a, b):
2      """
3      Funkcja zwraca NWD(największy wspólny dzielnik liczb a i b)
4      :param a: Liczba naturalna
5      :param b: Liczba naturalna
6      :return: NWD(a, b)
7      """
8      if a == b:
9          return a
10     else:
11         return NWD(b, a % b)
```

Rysunek 2.3 Program wykonujący algorytm Euklidesa w pseudokodzie języka Python

Rozdział 3

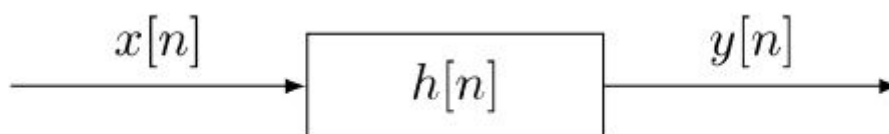
Filtr cyfrowy

Filtr cyfrowy jest programem komputerowym służącym do usunięcia niechcianych pasm częstotliwości lub innych składowych sygnałów. Każdy z filtrów można projektować w dziedzinie czasu lub częstotliwości. Klasyczne filtry można projektować przy użyciu tzw. konwolucji (operacja splotu matematycznego) lub rekursji (rekurencji). Filtry można kategoryzować m.in. ze względu na charakterystyki na, których działa algorytm.



Rysunek 3.1 Podział filtrów ze względu na charakterystyki pracy a) Dolnoprzepustowy, b) Górnoprzepustowy, c) Środkowoprzepustowy, d) Środkowozaporowy

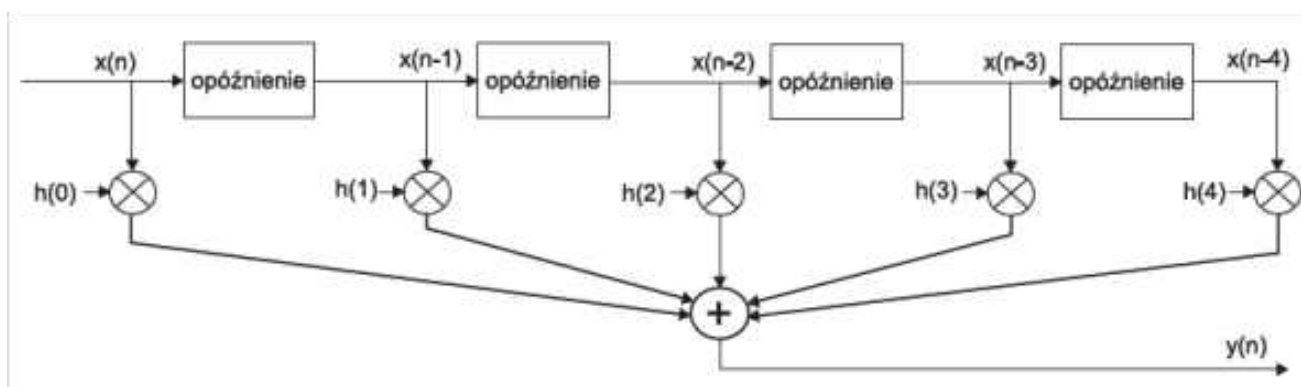
- Charakterystyka dolnoprzepustowa - Filtr "usuwa" częstotliwości powyżej zadanej częstotliwości, podobnie jak czerwona charakterystyka na rysunku 3.1
- Charakterystyka górnoprzepustowa - Filtr "usuwa" częstotliwości poniżej zadanej częstotliwości, podobnie jak niebieska charakterystyka na rysunku 3.1
- Charakterystyka środkowoprzepustowa - Filtr "usuwa" częstotliwości poniżej oraz częstotliwości powyżej zadanego pasma, podobnie jak zielona charakterystyka na rysunku 3.1
- Charakterystyka środkowozaporowa - Filtr "usuwa" częstotliwości w zadanym paśmie, tzn. działa w sposób odwrotny niż filtr środkowoprzepustowy, przykładem filtru jest szara charakterystyka na rysunku 3.1



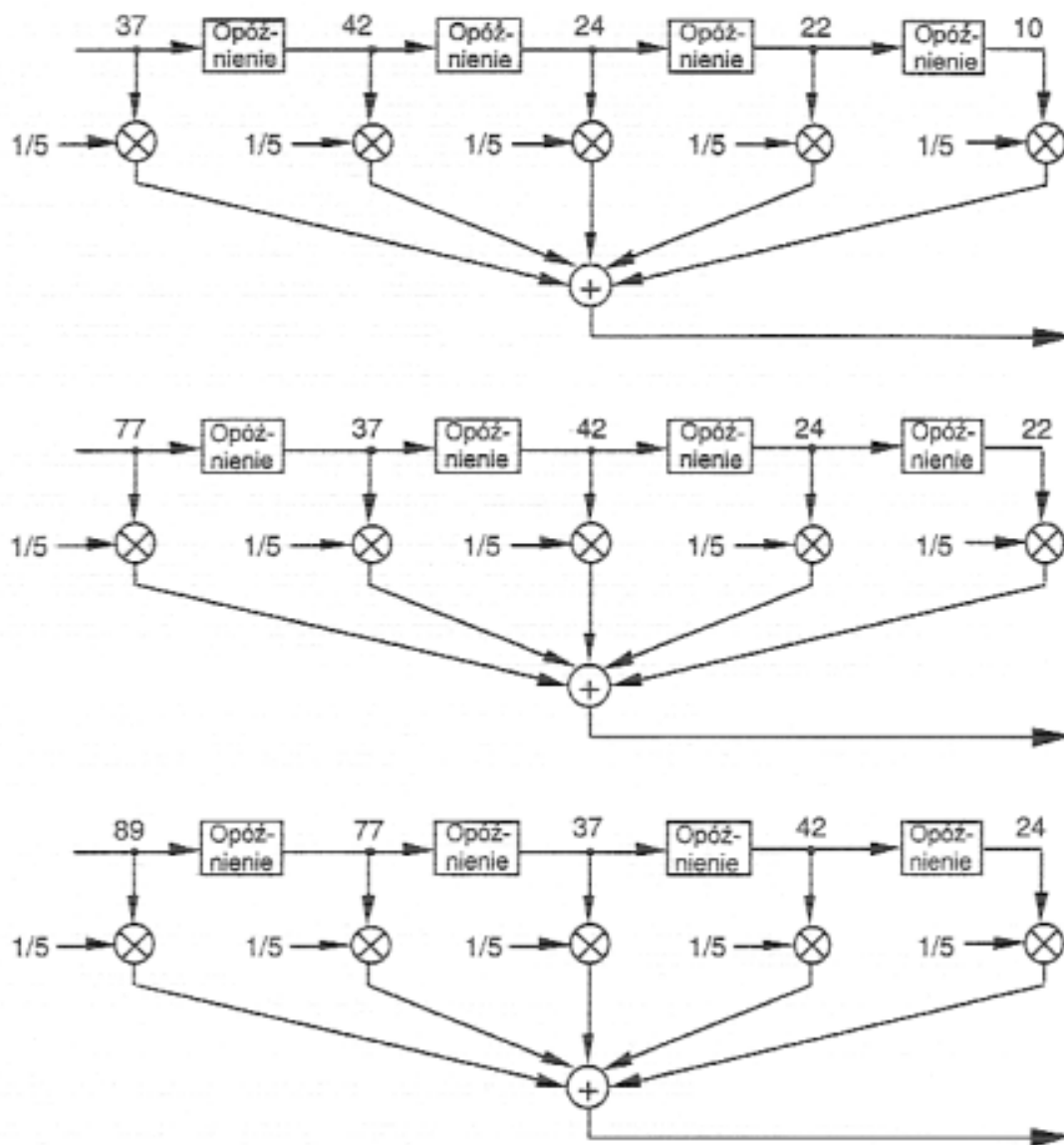
Rysunek 3.2 Ogólny schemat działania filtru cyfrowego, $x(n)$ jest sygnałem wejściowym, $h(n)$ rozumiemy jako operacje zachodzące wewnątrz filtru, natomiast $y(n)$ jest sygnałem wyjściowym

3.1 Filtr o skończonej odpowiedzi impulsowej (SOI)

Filtry o skończonej odpowiedzi impulsowej będące tematem rozważań dla bieżącego podrozdziału do uzyskania aktualnej próbki sygnału wyjściowego korzystają ze zbioru przeszłych próbek wejściowych oraz próbki bieżącej. Filtr nie korzysta z próbek, które znajdują się na jego wyjściu dlatego również ten rodzaj filtru jest nazywany filtrem nierekursywnym. Do obliczenia wartości odpowiednich próbek wyjściowych filtr korzysta z operacji matematycznego dodawania w sposób podobny do procesu uśredniania.



Rysunek 3.3 Schemat filtru cyfrowego typu SOI z rzędem filtru $M = 5$



Rysunek 3.4 Przykładowe obliczenia wartości próbki dla filtra FIR rzędu $M = 5$

$$y_{wyj}(n) = \frac{1}{5}x(k) = \frac{1}{5}x(n-4) + \frac{1}{5}x(n-3) + \frac{1}{5}x(n-2) + \frac{1}{5}x(n-1) + \frac{1}{5}x(n) \quad (3.1)$$

$$\sum_{k=n-4}^n \frac{1}{5} x(k) \quad (3.2)$$

Zarówno równanie numer 3.1 jak również równanie 3.2 są tym samym równaniem w dłuższej i krótszej opcji zapisowej, pozwalającymi obliczyć n-tą próbkę w filtrze o rzędzi złożoności M.

Dla chwil a, b i c obliczono odpowiednio wartości piątej, szóstej oraz siódmej próbki wyjściowej:

$$\begin{aligned} \text{a)} y_{wyj}(5) &= \frac{37+42+24+22+10}{5} = \frac{135}{5} = 27 \\ \text{b)} y_{wyj}(5) &= \frac{77+37+42+24+22}{5} = \frac{202}{5} = 40,4 \\ \text{c)} y_{wyj}(5) &= \frac{89+77+37+42+24}{5} = \frac{269}{5} = 53,8 \end{aligned}$$

W rzeczywistości współczynniki filtrów posiadają różne wartości dlatego dla przykładów a,b,c oraz równań 3.1 i 3.2 odpowiednia będzie ogólna postać.

$$\sum_{k=0}^M h(k)x(n-k) \quad (3.3)$$

Równanie 3.3 jest równaniem splotu matematycznego. Jest to nic innego jak suma iloczynów jednak w tym przypadku przed przystąpieniem do obliczenia sum należy odwrócić kolejność czasową z jaką próbki wchodziły na wejście filtru FIR.

Rozdział 4

Algorytm oparty na strategiach ewolucyjnych lub hybrydowych

4.1 Algorytm Ewolucyjny

Rodzina algorytmów, które do działania podobnie jak sieci neuronowe wykorzystują techniki inspirowane analogiami biologicznymi. Algorytmy ewolucyjne zamiast wyszukiwać rozwiązania np.(metoda Brute force) starają się ”wychodować” najlepsze możliwe rozwiązanie. Wychodowanie optymalnego rozwiązania wymaga szeregu operacji m.in. krzyżowania, rozmnażania, konkurencji itp. Algorytmy ewolucyjne (AE) wykorzystują nomenklaturę zaczerpniętą z genetyki :

- **Gen** - Najmniejsza jednostka algorytmu ewolucyjnego. Jest cechą, częścią rozwiązania
- **Osobnik** - Zbiór parametrów zadania będący potencjalnym rozwiązaniem problemu
- **Chromosom** - Uporządkowany ciąg genów
- **Locum** - Wskazują miejsce położenia każdego z genów w chromosomie
- **Genotyp** - Struktura złożona z chromosomów określonego osobnika
- **Fenotyp** - Zestaw wartości odpowiadających danemu genotypowi, będący jakością konkretnego osobnika
- **Allel** - Określa wartość cechy konkretnego genu
- **Populacja** - Zbiór osobników spełniające odpowiednie wartości danej cechy oraz spełniająca zdefiniowaną ilość osobników

Każdy z algorytmów ewolucyjnych posiada szereg operacji charakterystycznych dla całej rodziny algorytmów czerpiących z teorii ewolucji:

1. Wytworzenie populacji złożonej ze zdefiniowanej ilości osobników, o wartościach spełniającymi ściśle zdefiniowane wartości.

$$x_n \in \mathbb{Z} \tag{4.1}$$

x_n - Liczba osobników

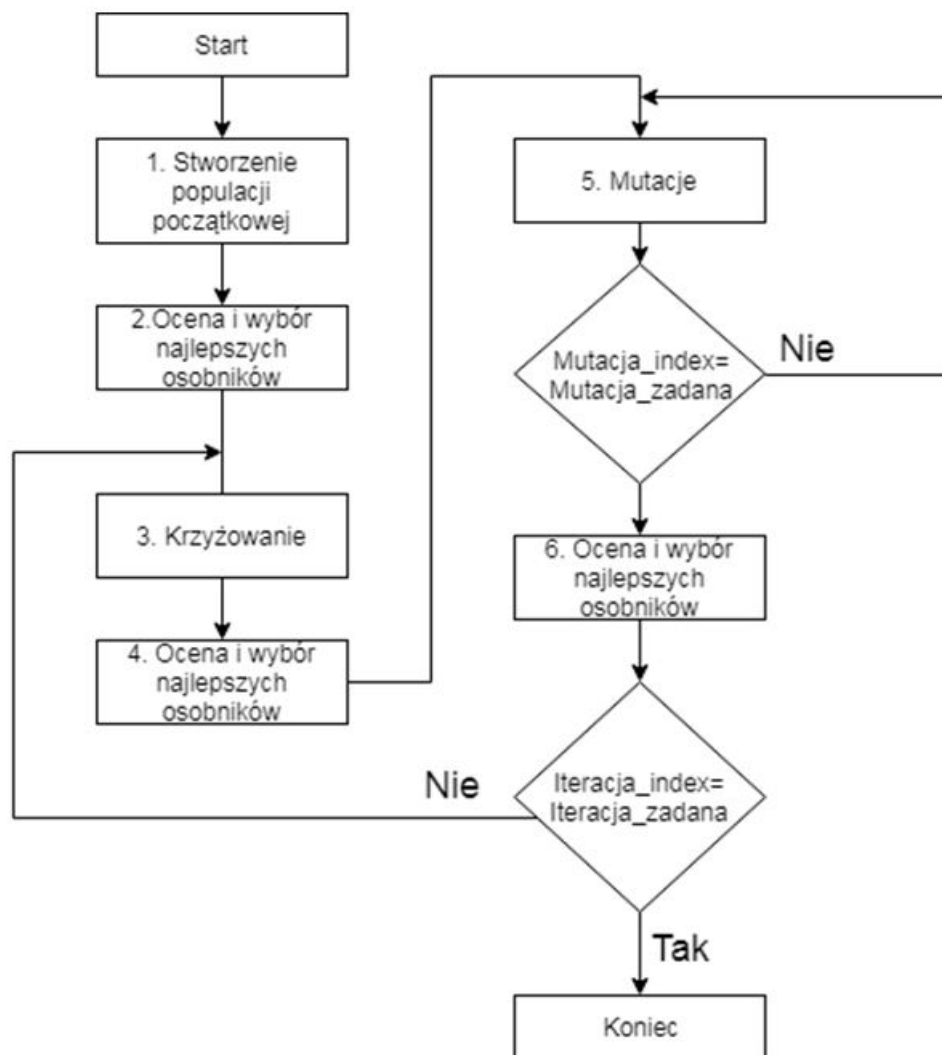
2. Każdy osobnik będący w zbiorze populacji powinien zostać zaklasyfikowany przez specjalną funkcję klasyfikującą (fitness function), zdefiniowaną na etapie projektowania algorytmu. Wartość nadawana przez **fitness function** określa na ile rozwiązanie odbiega od rozwiązania optymalnego. Z tego wynika, że wartość idealnego rozwiązania powinna być bliska lub równa 0

$$\lim_{x_n \rightarrow 0} fitness(x_n) = 0 \quad (4.2)$$

3. Kolejnym krokiem jest zostać spełniony warunek dotyczący maksymalnej ilości cykli(iteracji) lub przystosowania najlepszego osobnika w obrębie całego dostępnego zbioru populacji
4. Selekcja osobników na podstawie ocen jakie zostały do nich przypisany przez funkcję przystosowania. Jedną z najpopularniejszych metod wykorzystywanych do selekcji jest tzw. metoda koła ruletki. Polega
5. Wybrane osobniki podlegają operatorom genetycznym (korzyżowanie oraz mutacja)
6. Wytworzenie nowej populacji osobników będące skutkiem wykonania operacji genetycznych
7. Powrót do punktu numer 2

Metody selekcji osobników:

- **Metoda koła ruletki** - Każdemu z osobników zostaje przydzielony wycinek koła odpowiadający wartości funkcji przystosowania badanego osobnika.
- **Metoda turniejowa** - Osobniki zostają podzielone na podgupy, następnie każda z podgrup dokonuje selekcji najlepszego osobnika z punktu widzenia funkcji przystosowania
- **Strategia elitarna** - Osobniki o najlepszej wartości funkcji przystosowania z punktu widzenia algorytmu jest 'chroniony'. Zapobiega się aby osobnik o najlepsze wartości został utracony
- **Selekcja rankingowa** - Każdemu z osobników będących w zbiorze populacji zostaje przypisana ranga skorelowana z wartością funkcji przystosowania dla konkretnego osobnika



Rysunek 4.1 Ogólny schemat blokowy algorytmu ewolucyjnego

4.1.1 Algorytm PSO

Algorytm Particle Swarm Optimization został opublikowany w roku 1995 przez Panów James'a Kennedy'iego oraz Russel'a Eberhart'a. Metoda optymalizacji bazuje na dwóch innych metodach

- **Ewolucja obliczeniowa** - zastosowanie strategii ewolucyjnych, algorytmów genetycznych
- Tzw. **Sztuczne życie** teoria roju, stada ptaków (powrót do miejsc, których w poprzednich okresach czasu(pobytu) lokalizowano pożywienie)

Zauważono, że stada lub ławice zwierząt wracają do miejsc, w których odkryto pożywienie, w miejsca te docierały również osobniki, których wcześniej z racji na ich wiek nie było możliwe przemieszczenie. Dlatego obserwując skupienie osobników sformułowano cechy takie jak:

- **Bezpieczeństwo ruchu** - Zdolność do zapobiegania kolizji z obiektami będącymi przeszkodami lub osobnikami wewnątrz stada
- **Rozprzestrzenianie się** - Zdolność do rozproszenia stada, celem zarządzania odległościami pomiędzy osobnikami

- **Orientacja** - Możliwość znajdowania ustalonych miejsc, regionów punktów w przestrzeni poszukiwań
- **Skupianie** - Umiejętność gromadzenia się stada celem ustalenia maksymalnych odległości pomiędzy poszczególnymi osobnikami

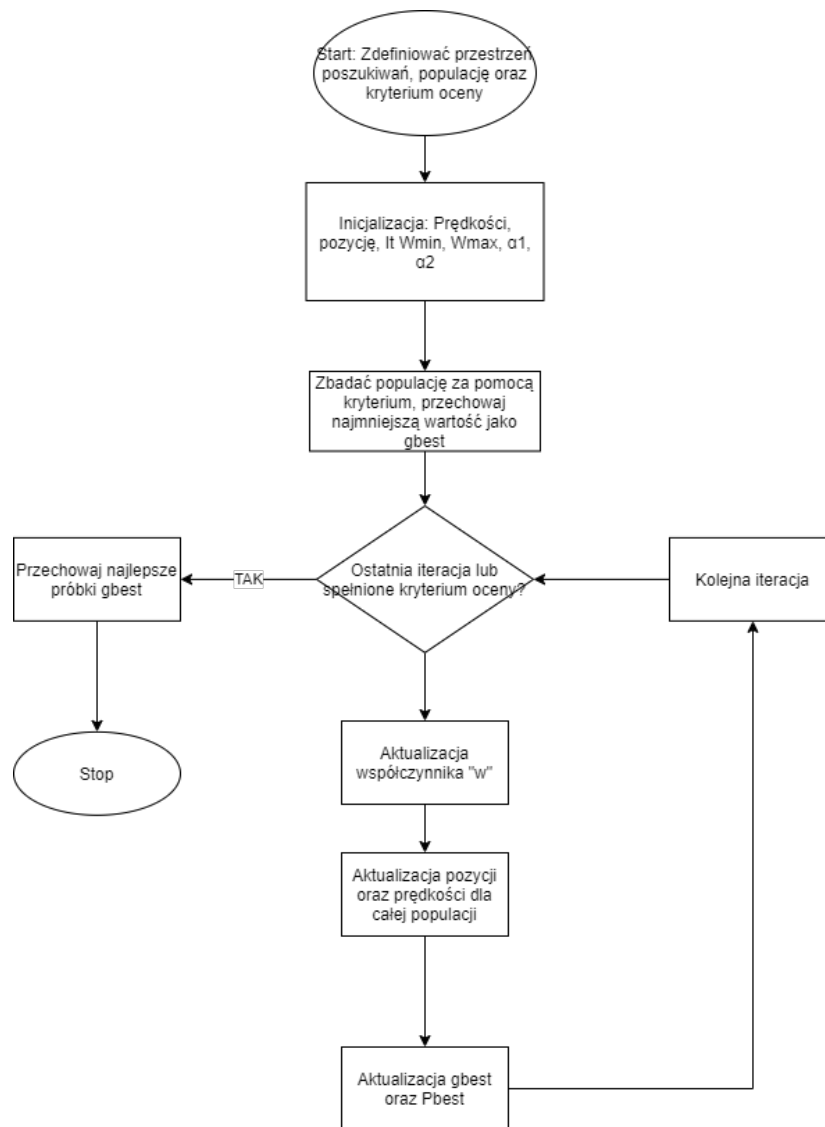
Z punktu widzenia algorytmu PSO takie cechy jak rozprzestrzenianie się oraz bezpieczeństwo ruchu nie mają większego sensu dlatego zostały one wyeliminowane, okazało się, że zachowanie algorytmu koreluje w bliższy sposób z rojem aniżeli stadem/lawicą. Na podstawie własnych obserwacji oraz pracy Pana Milonas's zdefiniowano cechy roju takie jak:

- **Jakość** - Jakość osobników jest zależna od zdefiniowanej funkcji klasyfikacyjnej
- **Stabilność** - Populacja powinna być odporna na pewne bodźce (nie zmieniać zachowania mało znaczącej zmianie środowiska)
- **Adaptacyjność** - Rój powinien ocenić czy zmiana zachowania jest opłacalna
- **Różnorodność** - Rój powinien posiadać szeroki zbiór zachowań
- **Bliskość** - Rój powinien być przechowywany w strukturze danych, która nie wymaga dużych nakładów złożoności obliczeniowej

Każdy z osobników powinien mieć informacje na temat:

- Własnej prędkości
- Wartości funkcji klasyfikacyjnej dla obecnego oraz najlepszego swojego położenia
- Swoje położenie w przestrzeni poszukiwań
- Zbiór sąsiadów, położenie oraz wartość funkcji klasyfikacyjnej dla teego położenia

4.1.2 Schemat blokowy oraz parametry kontrolne



Rysunek 4.2 Schemat blokowy algorytmu PSO(Particle Swarm Optimization)

4.2 Algorytm hybrydowy

4.2.1 Algorytm CSPSO

Tabela. 4.1 Parametry kontrolne algorytmu PSO

Parametry	PSO
Wielkość populacji	150
Ilość iteracji	400
α_1, α_2	1.9, 1.8
u_{min}, u_{max}	0.01, 1.0
w_{min}, w_{max}	0.4, 0.9
Wartości brzegowe współczynników	[-1, 1]

4.2.2 Schemat blokowy oraz parametry kontrolne

Tabela. 4.2 Parametry kontrolne algorytmu CSPSO

Parametry	CSPSO
Wielkość populacji	150
Ilość iteracji	400
α_1, α_2	1.9, 1.8
u_{min}, u_{max}	0.01, 1.0
w_{min}, w_{max}	0.4, 0.9
λ, β	1.1, 1.8
Wartości brzegowe współczynników	[-1, 1]

4.2.3 Funkcja Gamma (Γ)

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt \quad (4.3)$$

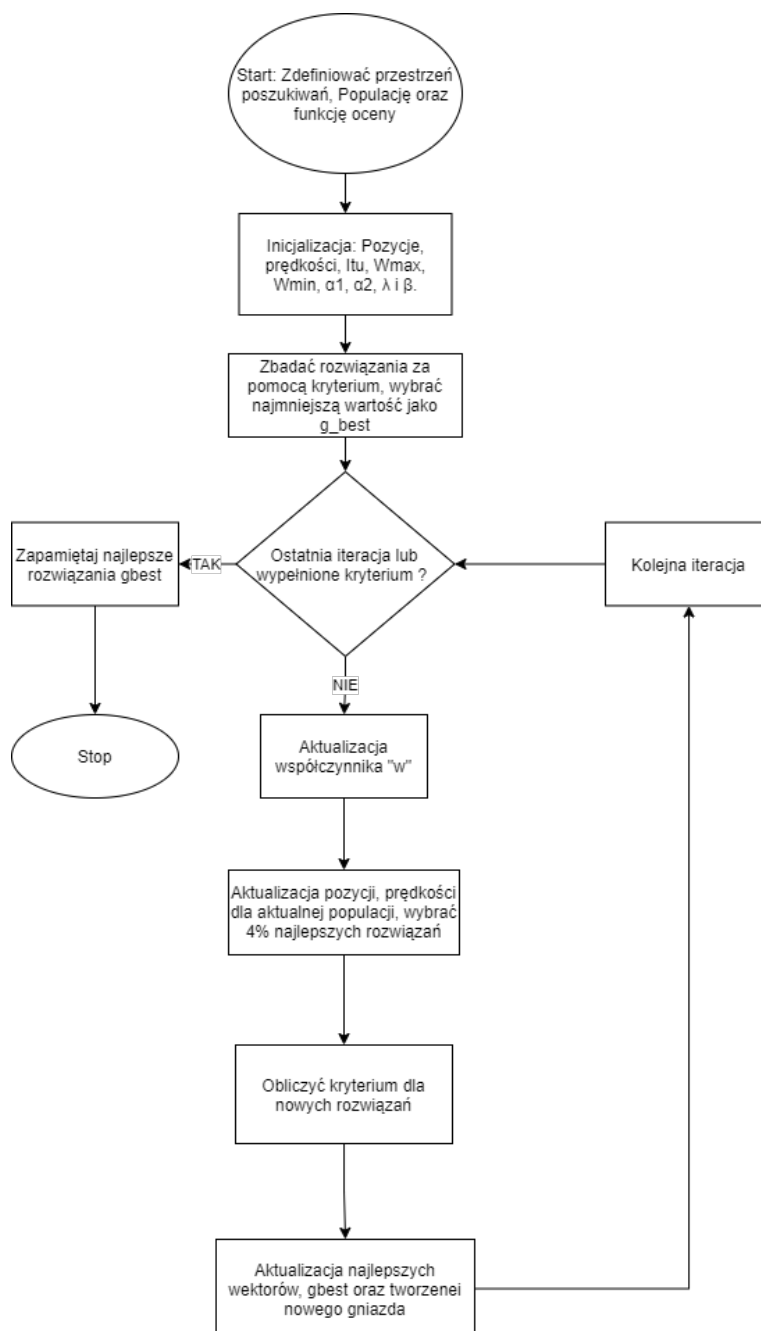
4.2.4 Funckja lotu Levy'ego

$$s = \frac{u}{|v|^{1/\beta}} \quad (4.4)$$

$$u \sim N(0, \sigma_n^2) \quad (4.5)$$

$$s \sim N(0, \sigma_n^2) \quad (4.6)$$

$$\sigma_u = \left\{ \frac{\Gamma(1 + \beta) \sin(\pi\beta/2)}{\Gamma[(1 + \beta)/2] \beta 2^{(\beta-1)/2}} \right\}^{1/\beta} \quad (4.7)$$



Rysunek 4.3 Schemat blokowy algorytmu CSPSO(Cuckoo Search Particle Swarm Optimization)

Rozdział 5

Implementacja

5.1 Algorytm PSO

5.1.1 Generowanie pojedynczego rozwiązania

Listing 5.1 Implementacja stworzenia pojedynczego rozwiązania

```
def generate_solution(filter_order: int):  
    solution = []  
    for gene in range(0, filter_order):  
        solution.append(random.uniform(-1, 1))  
    return solution
```

5.1.2 Generowanie prędkości

Listing 5.2 Generowanie prędkości

```
def generate_velocities(v_min, v_max, order: int):  
    velocities = []  
    for i in range(0, order):  
        velocities.append(random.uniform(v_min, v_max))  
    return velocities
```

5.1.3 Obliczenie $V(\omega_n)$

Listing 5.3 Implementacja równania Obliczenie $V(\omega_n)$

```
def v_omega_n(order: int, solution):  
    final = 0  
    for m in range(0, order):  
        omega = ((2 * np.pi * m) / (order + 1))  
        final += solution[m] * np.exp(-1j * m * omega)  
    return final
```


5.1.4 Generowanie prędkości początkowych

Listing 5.4 Implementacja funkcji generującej prędkości dla każdej cząsteczki

```
def generate_velocities(v_min, v_max, order: int):
    velocities = []
    for i in range(0, order):
        velocities.append(random.uniform(v_min, v_max))
    return velocities
```

5.1.5 Obliczenie funkcji dopasowania

Listing 5.5 Implementacja funkcji klasyfikującej rozwiązanie

```
def fitness_function(filter_order: int, v_a_omega, band_pass, band_stop):
    fitness = 0
    first_sum = 0
    second_sum = 0
    for gene in range(0, filter_order):
        first_sum += (np.abs(np.abs(v_a_omega) - 1) - band_pass) ** 2
        second_sum += (np.abs(v_a_omega) - band_stop) ** 2
        fitness += first_sum + second_sum
    return fitness
```

5.1.6 Tworzenie nowych rozwiązań z obecnego pokolenia

- Aktualizacja współczynnika ω

Listing 5.6 Implementacja aktualizacji współczynnika ω

```
def w_current(w_maximum, w_minimum, current_iteration: int, number_of_
    return w_maximum - (current_iteration - number_of_
```

- Aktualizacja prędkości U_i^{k+1}

Listing 5.7 Implementacja aktualizacji współczynnika prędkości U_i^{k+1}

```
def u_ik_plus_one(w_k: float, u_k, alpha1, alpha2, r1, r2, p_best, p_current,
    u_k_plus_one = [[0] * order] * size
    for i in range(0, size):
        for j in range(0, order):
            u_k_plus_one[i][j] = w_k * u_k[i][j] + alpha1 * r1 * (p_best[j] - p_current[j])
            u_k_plus_one[i][j] = w_k * u_k[i][j] + alpha2 * r2 * (g_best[j] - p_current[i][j])
    return u_k_plus_one
```

- Aktualizacja populacji P_i^{k+1}

Listing 5.8 Implementacja aktualizacji współczynnika prędkości P_i^{k+1}

```
def pi_k_plus_one(p_current, u_k_plus_one, size, order):
    p_current_plus_one = [[0]*order]*size
    for i in range(0, size):
        for j in range(0, order):
            p_current_plus_one[i][j] = p_current[i][j] + u_k_plus_one[i]
    return p_current_plus_one
```

5.1.7 Skrypt

Listing 5.9 Implementacja skryptu z algorytmem PSO

```
import numpy as np
import random

if __name__ == "__main__":
    order = 41
    size = 150
    search_space = int(order / (2 + 1))
    cycles = 400
    alpha = 0.5
    alpha_one = 1.9
    alpha_two = 1.8
    band_pass = 0.6
    band_stop = 0.2
    u_min = 0.01
    u_max = 1.00
    w_min = 0.4
    w_max = 0.9
    _lambda = 1.1
    beta = 1.8
    velocity_vec = []
    population = []
    fitness = []

    for i in range(0, size):
        solution = generate_solution(order)
        solution_omega = v_omega_n(order, solution)
        fitness_res = fitness_function(order, solution_omega, band_pass, band_stop)
        population.append(solution)
        fitness.append(fitness_res)
        velocity_vec.append(generate_velocities(u_min, u_max, order))
    print(f"Fitness{i}_is_{fitness_res}")
    g_best_fitt = min(fitness)
    g_best = population[fitness.index(min(fitness))]
    p_best = g_best
    for iteration in range(0, cycles):
        local_fitness = 0
        r1 = r_generator()
        r2 = r_generator()
```

```
w_k = w_current(w_max,w_min,iteration ,cycles)
u_ki=u_ik_plus_one(w_k,velocity_vec ,alpha_one ,/
alpha_two ,r1 ,r2 ,p_best ,population ,g_best ,size ,order)
new_pop = pi_k_plus_one(population ,u_ki ,size ,order)
for solution in range(0,size):
    local_omega = v_omega_n(order ,new_pop[solution])
    fitness = fitness_function(order ,local_omega , band_pass ,band_stop)
    p_best_fitenss = np.min(fitness)
    p_best = new_pop[fitness.index(p_best_fitenss)]
    if p_best_fitenss < g_best_fitt:
        g_best_fitt = p_best_fitenss
print(g_best)
```

5.2 Algorytm CSPSO

5.2.1 Generowanie pojedynczego rozwiązania

Listing 5.10 Implementacja stworzenia pojedynczego rozwiązania

```
def generate_solution(filter_order: int):
    solution = []
    for gene in range(0, filter_order):
        solution.append(random.uniform(-1, 1))
    return solution
```

5.2.2 Obliczenie $V(\omega_n)$

Listing 5.11 Implementacja równania Obliczenie $V(\omega_n)$

```
def v_omega_n(order: int, solution):
    final = 0
    for m in range(0, order):
        omega = ((2 * np.pi * m) / (order + 1))
        final += solution[m] * np.exp(-1j * m * omega)
    return final
```

5.2.3 Generowanie prędkości początkowych

Listing 5.12 Implementacja funkcji generującej prędkości dla każdej cząsteczki

```
def generate_velocities(v_min, v_max, order: int):
    velocities = []
    for i in range(0, order):
        velocities.append(random.uniform(v_min, v_max))
    return velocities
```

5.2.4 Obliczenie funkcji dopasowania

Listing 5.13 Implementacja funkcji klasyfikującej rozwiązanie

```
def fitness_function(filter_order: int, v_a_omega, band_pass, band_stop):
    fitness = 0
    first_sum = 0
    second_sum = 0
    for gene in range(0, filter_order):
        first_sum += (np.abs(np.abs(v_a_omega) - 1) - band_pass) ** 2
        second_sum += (np.abs(v_a_omega) - band_stop) ** 2
```

```

    fitness += first_sume + second_sume
    return fitness

```

5.2.5 Tworzenie nowych rozwiązań z obecnego pokolenia

- Aktualizacja współczynnika ω

Listing 5.14 Implementacja aktualizacji współczynnika ω

```

def w_current(w_maximum, w_minimum, current_iteration: int, number_of_
    return w_maximum - (current_iteration - number_of_iterations) * (w_

```

- Aktualizacja prędkości U_i^{k+1}

Listing 5.15 Implementacja aktualizacji współczynnika prędkości U_i^{k+1}

```

def u_k_plus_one(w_k: float, u_k, alpha1, alpha2, r1, r2, p_best, p_current,
    u_k_plus_one = [[0] * order] * size
    for i in range(0, size):
        for j in range(0, order):
            u_k_plus_one[i][j] = w_k * u_k[i][j] + alpha1 * r1 * (p_best[j] - p_
                g_best[j] - p_current[i][j])
    return u_k_plus_one

```

- Aktualizacja populacji P_i^{k+1}

Listing 5.16 Implementacja aktualizacji współczynnika prędkości P_i^{k+1}

```

def pi_k_plus_one(p_current, u_k_plus_one, size, order):
    p_current_plus_one = [[0] * order] * size
    for i in range(0, size):
        for j in range(0, order):
            p_current_plus_one[i][j] = p_current[i][j] + u_k_plus_one[i]
    return p_current_plus_one

```

5.2.6 Funkcja Lotu Levy'ego

Listing 5.17 Implementacja funkcji Levy'ego

```

function nest=get_cuckoos(nest,best,Lb,Ub)
% Levy flights
n=size(nest,1);
% Levy exponent and coefficient
% For details, see equation (2.21), Page 16 (chapter 2) of the book
% X. S. Yang, Nature-Inspired Metaheuristic Algorithms, 2nd Edition, Luniver
beta=3/2;
sigma=(gamma(1+beta)*sin(pi*beta/2))/(gamma((1+beta)/2)*beta*2^((beta-1)/2));
for j=1:n,
    s=squeeze(nest(j,:,:));

    % This is a simple way of implementing Levy flights
    % For standard random walks, use step=1;
    %% Levy flights by Mantegnas algorithm
    u=randn(size(s))*sigma;
    v=randn(size(s));
    step=u./abs(v).^(1/beta);

    % In the next equation, the difference factor (s-best) means that
    % when the solution is the best solution, it remains unchanged.
    stepsize=0.01*step.*(s-best);
    % Here the factor 0.01 comes from the fact that L/100 should be the typical
    % step size of walks/flights where L is the typical lengthscale;
    % otherwise, Levy flights may become too aggressive/efficient,
    % which makes new solutions (even) jump outside of the design domain
    % (and thus wasting evaluations).
    % Now the actual random walks or flights
    s=s+stepsize.*randn(size(s));
    % Apply simple bounds/limits
    %nest(j,:,:)=simplebounds(s,Lb,Ub);
end

```

$$w^k = w_{max} - \frac{k}{It_u}(w_{max} - w_{min}) \quad (5.1)$$

5.1 Aktualizacja współczynnika w^k

```
def w_current(w_maximum, w_minimum, current_iteration: int, number_of_ite
    return w_maximum - (current_iteration - number_of_iterations) * (w_maxim
\label{lst:wk}
```

$$U_i^{(k+1)} = w^k U_i^k + \alpha_1 r_1 (P_{best_i}^k - P_i^k) + \alpha_2 r_2 (g_{best}^k - P_i^k) \quad (5.2)$$

?? Obliczenie prędkości dla kolejnego pokolenia rozwiązań

```
def u_ik_plus_one(w_k, u_k, alpha1, alpha2, r1, r2, pbest, pcurrent, g_best
    return w_k * u_k + alpha1 * r1(pbest - pcurrent) + alpha2 * r2 * (g_bes
```

$$P_i^{(k+1)} = P_i^k + U_i^{(k+1)} \quad (5.3)$$

5.3 Obliczenie pozycji dla kolejnego pokolenia rozwiązań

```
def pi_k_plus_one(p_current, u_k_plus_one):

    return p_current + u_k_plus_one
```

Rozdział 6

Podsumowanie

Literatura

Literatura

$\text{bib}_e\textit{lement1}$

Spis tabel

2.1	Przedstawienie podstawowych bloków używanych do tworzenia schematów blokowych	9
4.1	Parametry kontrolne algorytmu PSO	21
4.2	Parametry kontrolne algorytmu CSPSO	21

Spis rysunków

2.1	Uproszczony schemat blokowy każdego algorytmu	7
2.2	Schemat blokowy algorytmu Euklidesa	10
2.3	Program wykonujący algorytm Euklidesa w pseudokodzie języka Python .	11
3.1	Podział filtrów ze względu na charakterystyki pracy a) Dolnoprzepustowy, b) Górnoprzepustowy, c) Środkowoprzepustowy, d) Środkowozaporowy . .	12
3.2	Ogólny schemat działania filtru cyfrowego, $x(n)$ jest sygnałem wejściowym, $h(n)$ rozumiemy jako operacje zachodzące wewnątrz filtru, natomiast $y(n)$ jest sygnałem wyjściowym	13
3.3	Schemat filtru cyfrowego typu SOI z rzędem filtru $M = 5$	13
3.4	Przykładowe obliczenia wartości próbki dla filtru FIR rzędu $M = 5$	14
4.1	Ogólny schemat blokowy algorytmu ewolucyjnego	18
4.2	Schemat blokowy algorytmu PSO(Particle Swarm Optimization)	20
4.3	Schemat blokowy algorytmu CSPSO(Cuckoo Search Particle Swarm Optimization)	22