02.01.2023

Algorytmy Genetyczne

SPRAWOZDANIE Z PROJEKTU

1. Treść polecenia.

Zaimplementować algorytm z podejściem ewolucyjnym, który będzie służył do rozwiązania problemu pchania wózka. Celem tego zadania jest maksymalizacja całkowitej drogi przebytej przez wózek w określonym czasie, pomniejszonej o całkowity wysiłek włożony w pchanie. Model stanu dyskretnego opisujący ten problem przedstawia się następująco:

$$x_1(k+1) = x_2(k),$$

$$x_2(k+1) = 2x_2(k) - x_1(k) + \frac{1}{N^2}u_k(k).$$

Jako kryterium poszukiwań przyjmuje się następujący wskaźnik jakości sterowania:

$$J = x_1(N) - \frac{1}{2N}\sum_{k=0}^{N-1} \mathrm{u}^2(k)$$

Zadanie rozwiązać dla następujących parametrów:

N = 5, 10, 15, 20, 25, 30, 35, 40, 45

2. Założenia projektu.

Do rozwiązania użyto następujących rozwiązań:

- kodowanie binarne,

- algorytm genetyczny,

- selekcję metodą proporcjonalną,

- krzyżowanie jednopunktowe z prawdopodobieństwem krzyżowania p_c = 0.7,

- mutację genotypową dla kodowania binarnego z prawdopodobieństwem mutacji p_m = 0.02,

- reprodukcję pełną,

- sterowanie *u* z zakresu [0, 1] z dokładnością sześciu cyfr po przecinku, taki sposób kodowania wymaga użycia 20 bitów dla zakodowania każdego parametru

- liczebność populacji: 100 osobników,

- implementację: język Python (3.9) z wykorzystaniem bibliotek Matplotlib, Numpy.

3. Realizacja rozwiązania.

Osobniki w populacji zostały zaimplementowane jako obiekty klasy Specimen (*ang. osobnik*). Ta klasa zawiera następujące pola:

res - liczba bitów potrzebnych na zakodowanie jednego parametru wektora sterowania u,

n - liczba parametrów zawartych w wektorze sterowania *u*,

x_1 i x_2 - parametry związane z modelem stanowym (odległość i prędkość wózka),

J - wskaźnik jakości sterowania (odległość pomniejszona o siłę),

u_bin - tablica poszukiwanych sterowań zapisanych w postaci binarnej (wymiar: *n* x *res*),

u - tablica poszukiwanych sterowań zapisanych w postaci dziesiętnej (wymiar: 1 x *n*); jest potrzebna do obliczenia wartości *J*.

Poniżej przedstawiono inicjalizację instancji klasy *Specimen*.

```
class Specimen:
   def __init__(self, n: int = 5, res: int = 20):
      self.res = res
      self.n = n
      self.x1 = 0.
      self.x2 = 0.
      self.J = 0.0
      self.u_bin = np.random.randint(2, size=(n, res))     # inicjalizacja losowych binarnych parametrów
      self.u = np.zeros(n)
      self.bin_to_dec()                          # zamiana na reprezentację dziesiętną
```

Klasa zawiera trzy metody:

bin_to_dec - metoda konwertująca wartości zawarte w tablicy binarnej *u_bin* na odpowiadające im wartości dziesiętne z obranego przedziału. Obliczone wartości są zapisywane w tablicy *u*.

```
def bin_to_dec(self):
    """
    Method that converts binary to decimal
    :return:
    """
    g1 = 0.                          # dolna granica
    g2 = 1.                          # górna granica
    for i in range(self.n):  # convert u_bin to u
        # konwersja z binarnej reprezentacji na dziesiętną
        u_bin_val = sum(b * 2**(self.res - 1 - j) for j, b in enumerate(self.u_bin[i]))
        # zapisanie wartości zaokrąglonej do 6 cyfr po przecinku
        self.u[i] = round(g1 + (u_bin_val * g2 / ((2**self.res) - 1)), 6)
    self.count_J()
```

Metoda *count_J* oblicza wskaźnik jakości sterowania, który jest następnie wykorzystywany do obliczenia funkcji przystosowania.

```
def count_J(self):
    """
    Method to count J value
    :return:
    """
    self.x1 = 0.
    self.x2 = 0.
    for i in range(1, self.n):
        # oblicz x1, x2 dla zadanego u
        self.x1, self.x2 = self.x2, 2 * self.x2 - self.x1 + (1 / (self.n**2)) * self.u[i - 1]
    # oblicz J
    self.J = self.x2 - (1 / (2 * self.n)) * sum(u**2 for u in self.u)
```

Metoda *mutate* wykonuje losową mutację z uprzednio określonym prawdopodobieństwem.

```
def mutate(self, probability:int = 0.02):
    """
    Method that performs mutation by flipping (0 -> 1 and 1 -> 0)
    :probability: probabilty of mutation (default value 0.01) = 1%
    :return:
    """
    if random.random() <= probability:
        k = random.randint(0, self.n - 1)
        k_id = random.randint(0, self.res - 1)
        self.u_bin[k][k_id] = 1 if self.u_bin[k][k_id] == 0 else 0
        self.bin_to_dec()
```

W projekcie została także zdefiniowana funkcja *selection*, która modyfikuje całą populację osobników. Działanie tej funkcji polega na posortowaniu osobników według wskaźnika jakości od najmniejszego do największego, a następnie konwersji ich wartości na odpowiadające im wartości funkcji przystosowania. Dzięki temu możliwe jest wykonanie metody proporcjonalnej. Dla każdego osobnika obliczana jest dystrybuanta. Następnie wykonywana jest n-krotna losowa selekcja liczby z zakresu [0, 1.0]. Osobnik, dla którego dystrybuanta jest liczbą większą od wylosowanej, przechodzi do nowego pokolenia.

```
def selection(population: list, pop_num: int):
    """
    Function that performs selection: 'eliminates weak specimens'

    metoda proporcjonalna
    :param population:
    :param pop_num:
    :return:
    """
    population.sort(key=lambda x: x.J)              # sortowanie po wartości wskaźnika
    min_C = population[0].J
    J_sum=0
    f_przystosowania=np.ndarray(pop_num, dtype=float)   # przekształcenie na fun. przystosowania dla zad. maksymalizacji
    for i, osobnik in enumerate(population):
```

```
        f_przystosowania[i]=osobnik.J-min_C
        J_sum+=f_przystosowania[i]
    f_przystosowania_relative=f_przystosowania/J_sum    # obliczenie proporcji
    dystryb=np.cumsum(f_przystosowania_relative)       # obliczenie dystrybuanty
    list_of_ids=[]
    for i in range(population.__len__()):            # selection proporcjonalna
        rand=random.random()                        # liczba losowana z zakresu 0 do 1
        j=0
        while(rand > dystryb[j]):                   # gdy wylosowana liczba jest mniejsza od dystrybuanty
            j += 1                                  # następuje wyjście z pętli
        list_of_ids.append(j)                       # element o danym id zostanie dodany do nowej
populacji
    new_population=[]
    for el in list_of_ids:
        new_population.append(population[el])
    return new_population
```

Funkcja *crossbread* losuje pary osobników - rodziców - oraz dla każdej pary losuje liczbę z zakresu [0, 1.0]. Jeżeli wylosowana liczba jest mniejsza niż określone prawdopodobieństwo krzyżowania, para rodziców jest poddawana krzyżowaniu. Potomkowie są tworzeni poprzez wymianę genotypów rodziców od losowo wybranego punktu krzyżowania.

```
def crossbreed(population: list, pop_num: int):
    """
    Function that breeds speciments

    :param population: list of specimens
    :param pop_num: number of specimens
    :return:
    """
    ids=list(range(0, pop_num))                    # losowanie par
    random.shuffle(ids)
    pairs=[]
    while(ids):
        px=ids.pop()
```

```
        py=ids.pop()
        pair=(px, py)
        pairs.append(pair)
    p_c=np.random.rand(pop_num // 2)
    crossover_ids=np.where(p_c < 0.7)                # wybranie par, dla których wylosowana liczba
                                                     # jest mniejsza od prawdopodobieństwa krzyżowania
                                                     # obrane prawdopodobieństwo krzyżowania = 60%
    for el in crossover_ids[0]:                      # dla każdej pary wybranej do krzyżowania
        selected_pair=pairs[int(el)]
        parent_1=copy(population[selected_pair[0]])
        parent_2=copy(population[selected_pair[1]])
        parent_1.u_bin=parent_1.u_bin.flatten()      # złożenie n wymiarowej tablicy parametrów
w tab. jednowym.
        parent_2.u_bin=parent_2.u_bin.flatten()
        rand=random.randint(0,parent_1.u_bin.size-1)     # losowanie pozycji krzyżowania
        temp1=parent_1.u_bin[:rand]
        parent_1.u_bin[:rand]=parent_2.u_bin[:rand]
        parent_2.u_bin[:rand]=temp1
        parent_1.u_bin=parent_1.u_bin.reshape((parent_1.n,parent_1.res))
        parent_2.u_bin=parent_2.u_bin.reshape((parent_2.n,parent_2.res))
        population[selected_pair[0]]=copy(parent_1)
        population[selected_pair[0]].bin_to_dec()
        population[selected_pair[1]]=copy(parent_2)
        population[selected_pair[1]].bin_to_dec()
```

W projekcie zostały także zdefiniowane funkcje statystyczne:

j_mean_in_population - obliczanie średniej wartości wskaźnika jakości w populacji

u_mean_in_population - obliczanie średniej wartości parametrów u_1 - u_n w populacji

standard_deviation - obliczanie odchylenia standardowego w populacji

4. Przegląd osiągniętych wyników.

Przeprowadzono obliczenia dla różnych wartości N, gdzie N oznacza liczbę parametrów wektora u. Testy zostały wykonane dla N = 5, 10, 15, 20, 25, 30, 35, 40, 45. Dla każdej z tych wartości N znaleziono populację osobników po upływie 1000 pokoleń. Dodatkowo, dla N = 45, obliczenia zostały wykonane dla 10000 pokoleń. Otrzymane w ostatnim pokoleniu wartości wskaźnika jakości J zostały przedstawione w tabeli poniżej.

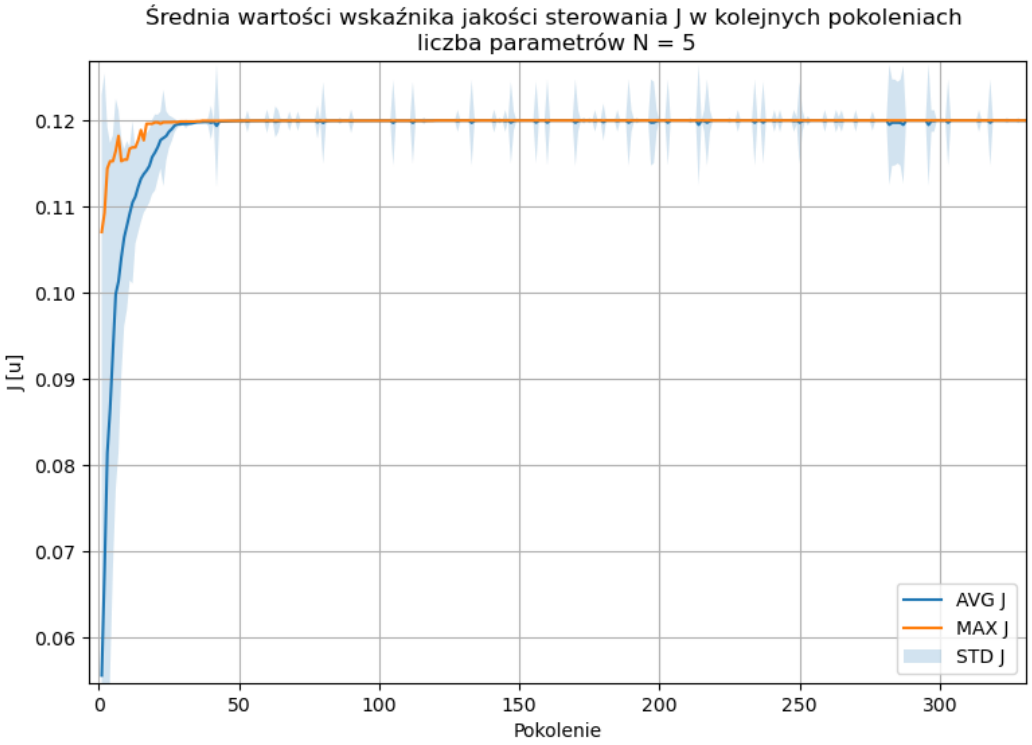
N, 1000 epok	**max J**	**avg J**
5	0.119937476	0.119937476
10	0.141942682	0.141942106
15	0.149961807	0.14996179
20	0.153832283	0.153825386
25	0.155072829	0.155070461
30	0.157380392	0.157380301
35	0.159340383	0.159340378
40	0.158473052	0.158473015
45	0.159241376	0.159240813

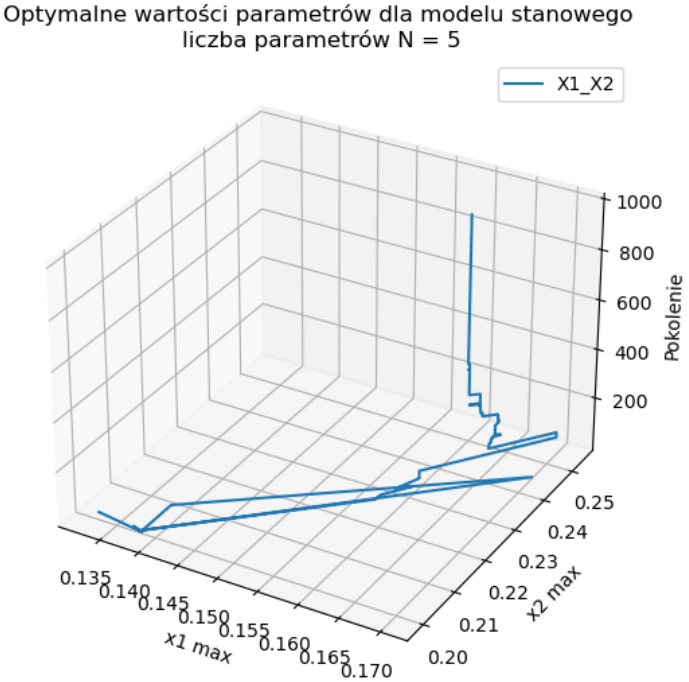
Poniżej przedstawiono porównanie wyników naszego algorytmu z przykładowymi wartościami.

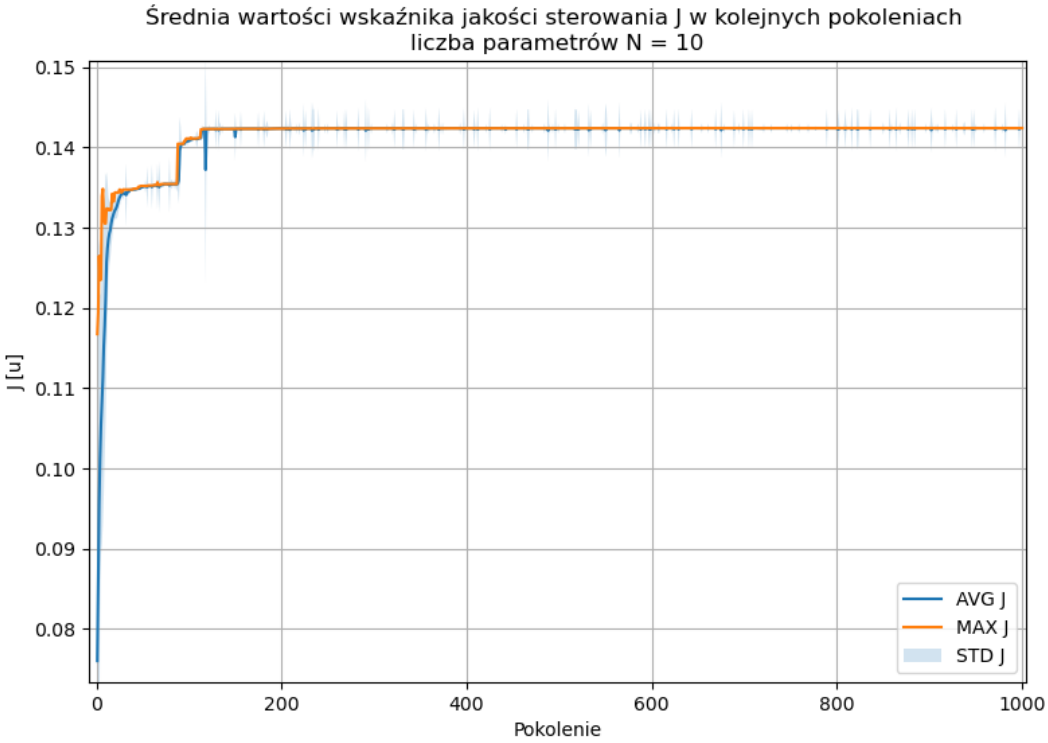
	Pokolenia								
	1			1000			10000		
N	Przykład	Nasze	Błąd względny [%]	Przykład	Nasze	Błąd względny [%]	Przykład	Nasze	Błąd względny [%]
5	-3.00835	0.1068	103.55	0.119979	0.119937	0.03	0.12	0.11999	0.01
10	-5.66829	0.1184	102.09	0.140195	0.141943	1.25	0.142496	0.142341	0.11
15	-6.88524	0.1153	101.67	0.142546	0.149962	5.20	0.150338	0.149975	0.24
20	-7.47787	0.1085	101.45	0.149953	0.153832	2.59	0.15434	0.152788	1.01
25	-8.66893	0.1184	101.37	0.14303	0.155073	8.42	0.156775	0.156092	0.44
30	-112.257	0.1216	100.11	0.123045	0.15738	27.90	0.158241	0.156651	1.00
35	-11.7895	0.1157	100.98	0.110964	0.15934	43.60	0.159307	0.158747	0.35
40	-10.9856	0.1121	101.02	0.072378	0.158473	118.95	0.16025	0.159429	0.51
45	-12.7893	0.1179	100.92	0.072364	0.159241	120.06	0.160913	0.159988	0.57

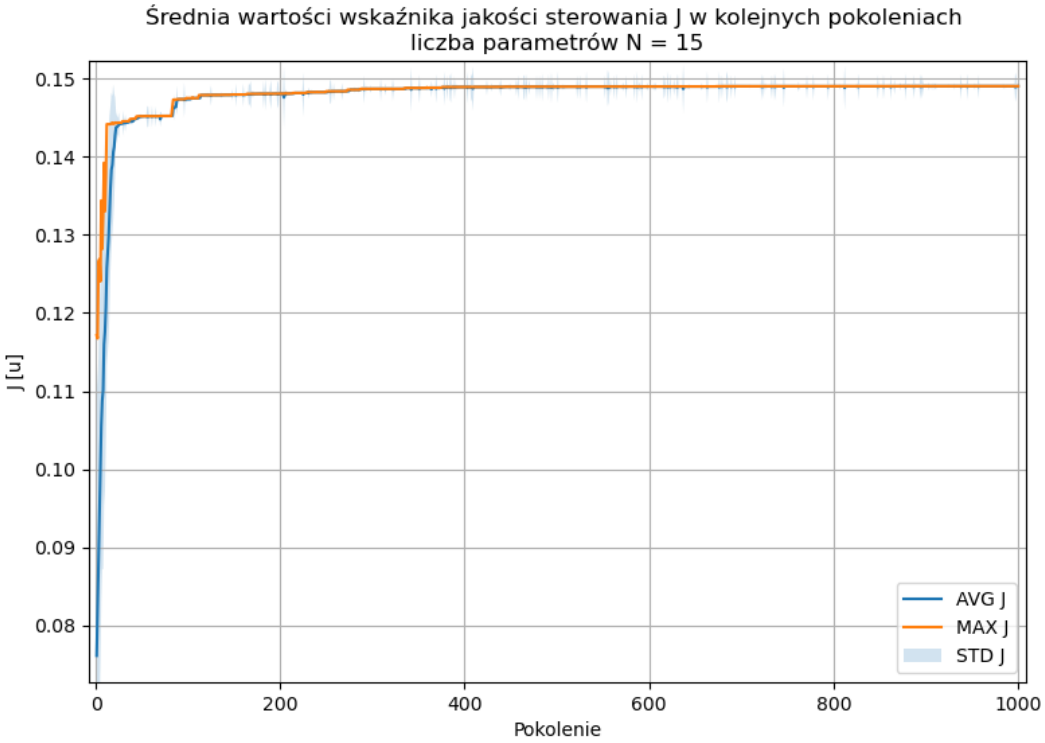
	Pokolenia								
	20000			30000			40000		
N	Przykład	Nasze	Błąd względny [%]	Przykład	Nasze	Błąd względny [%]	Przykład	Nasze	Błąd względny [%]
5	0.12	0.1199	0.06	0.12	0.12	0.00	0.12	0.1197	0.25
10	0.1425	0.1422	0.20	0.1425	0.1423	0.14	0.1425	0.1418	0.49
15	0.15037	0.1491	0.84	0.15037	0.1502	0.11	0.150371	0.1494	0.65
20	0.154375	0.1534	0.63	0.154375	0.1532	0.76	0.154377	0.1526	1.15
25	0.1568	0.1562	0.38	0.1568	0.1563	0.32	0.1568	0.156	0.51
30	0.158421	0.1553	1.98	0.158426	0.1575	0.58	0.158426	0.1572	0.77
35	0.159586	0.1586	0.62	0.159592	0.1583	0.81	0.159592	0.1587	0.56
40	0.160466	0.1598	0.42	0.160469	0.16	0.29	0.160469	0.159791	0.42
45	0.161127	0.1605	0.39	0.161152	0.16033	0.51	0.161152	0.160095	0.66

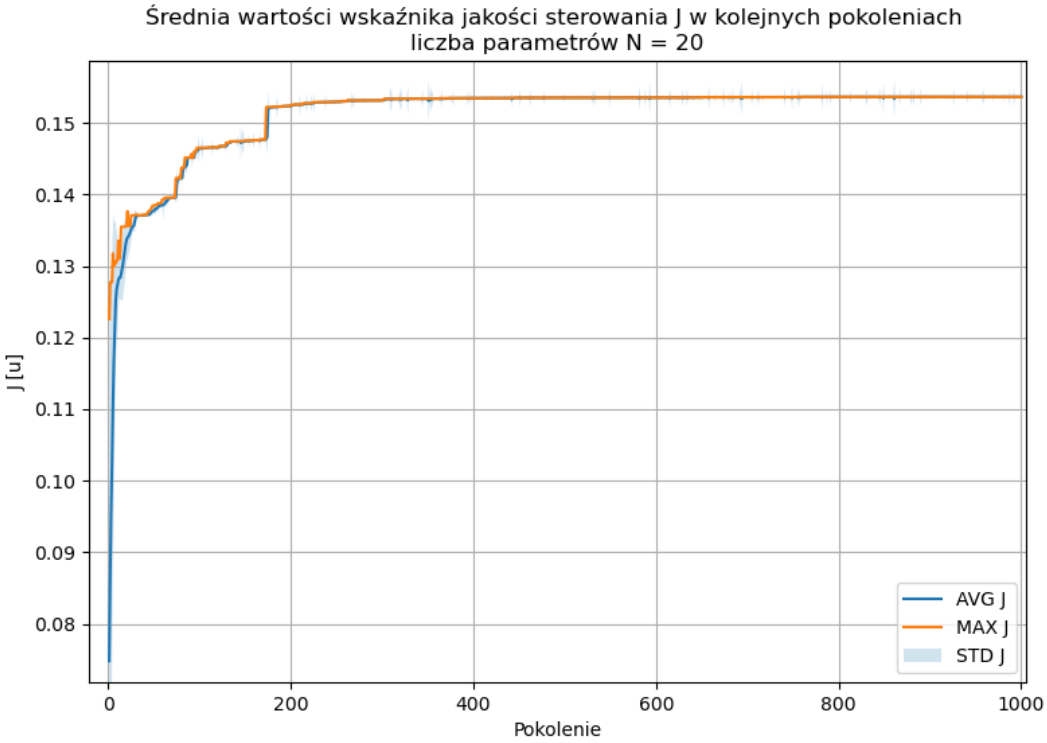
Poniżej zostały przedstawione wykresy przedstawiające maksymalną i średnią wartość wskaźnika jakości J oraz jego odchylenie standardowe w zależności od pokolenia. Ponadto, dla N = 5 i N = 45 zostały narysowane wykresy optymalnych wartości parametrów modelu stanowego.

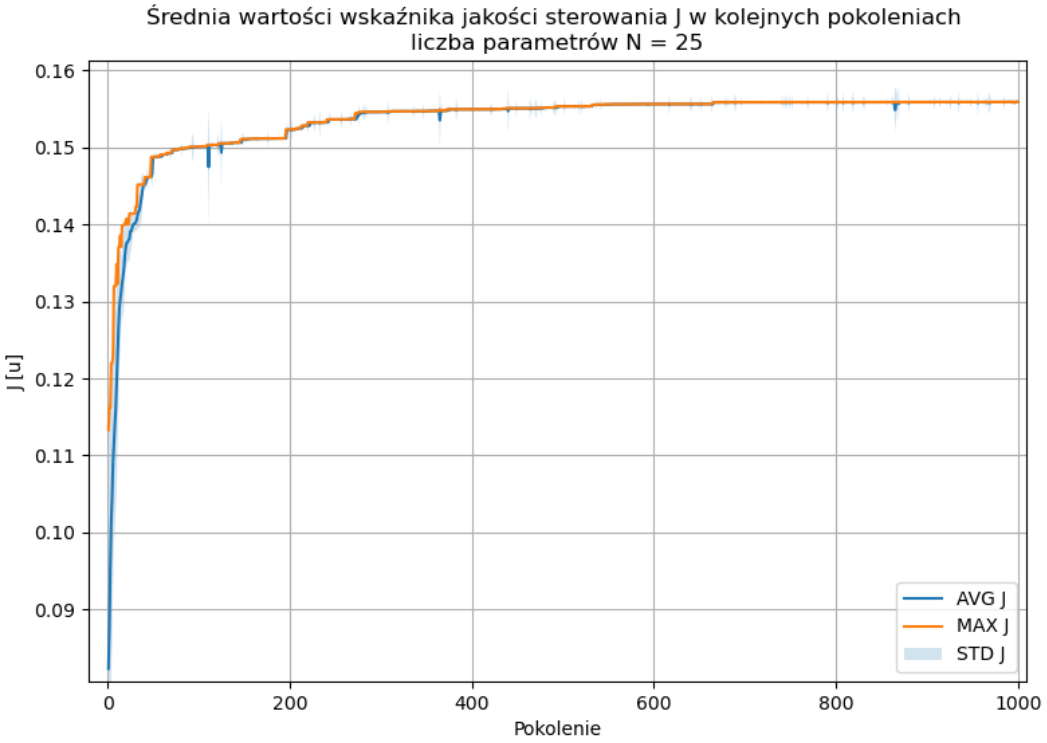


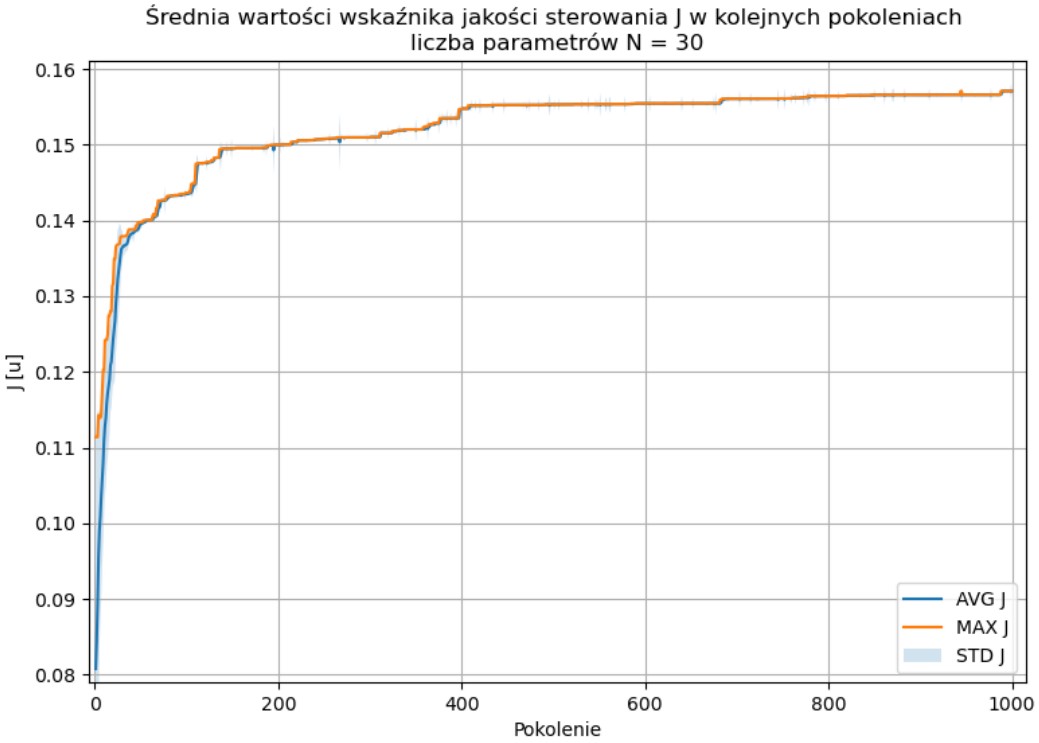


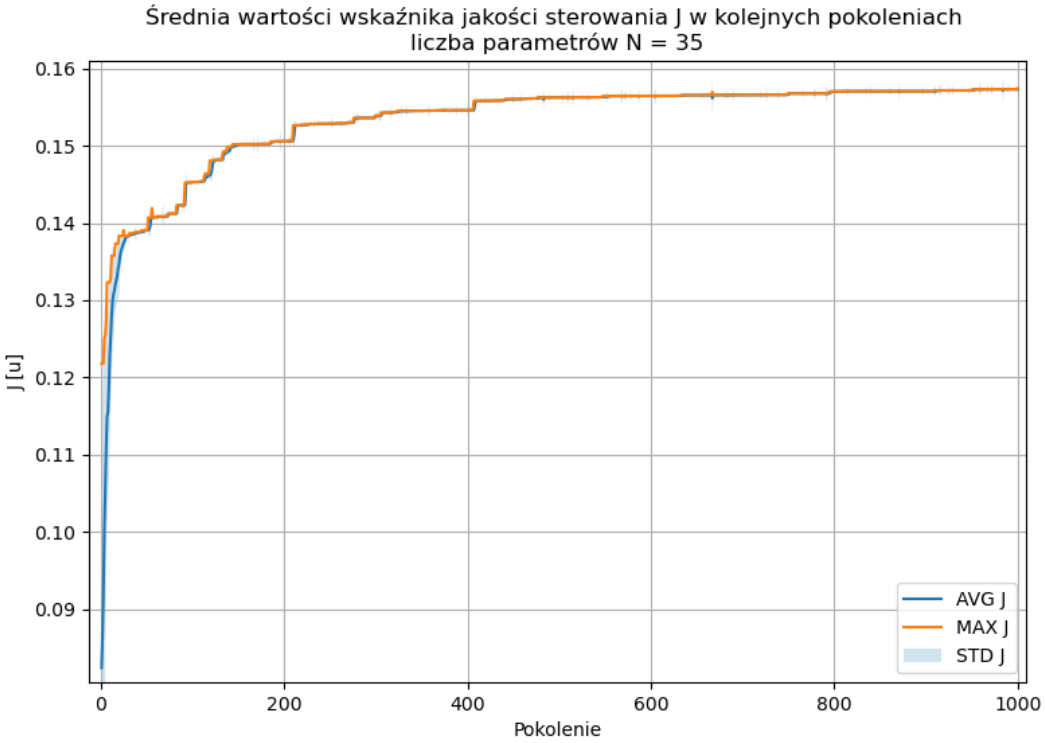

Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 10
J [u]
Pokolenie
AVG J
MAX J
STD J

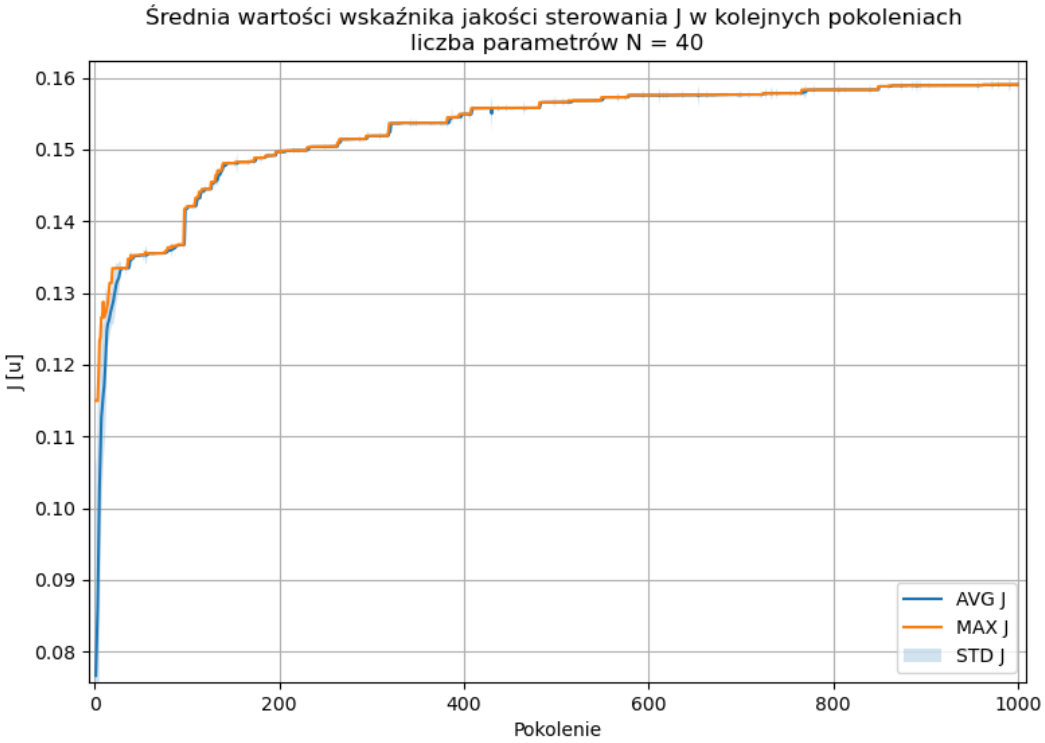

Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 15
J [u]
Pokolenie
AVG J
MAX J
STD J

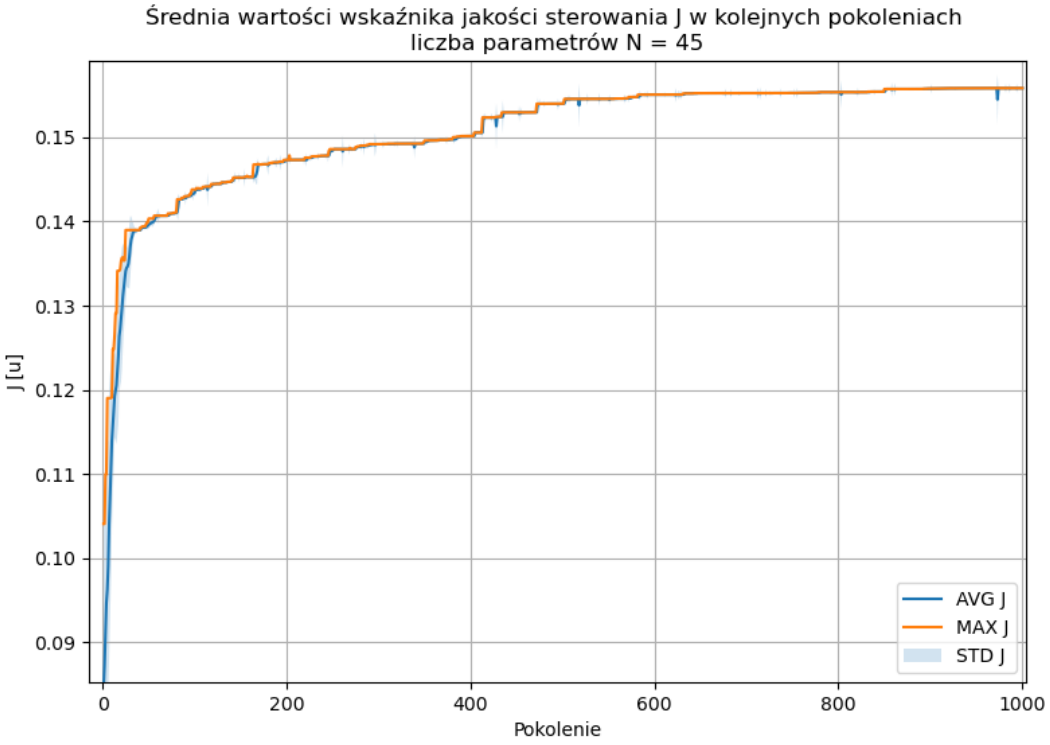

Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 20
J [u]
Pokolenie
AVG J
MAX J
STD J

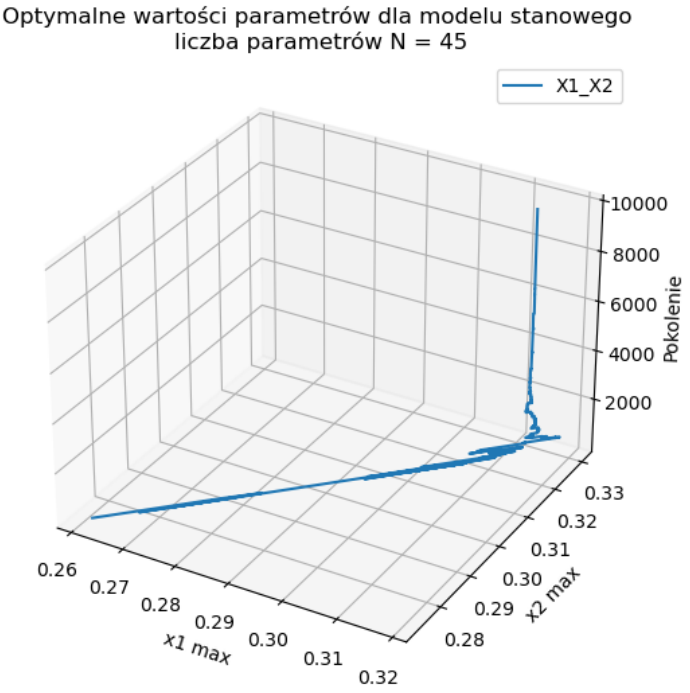

Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 25
J [u]
Pokolenie
AVG J
MAX J
STD J

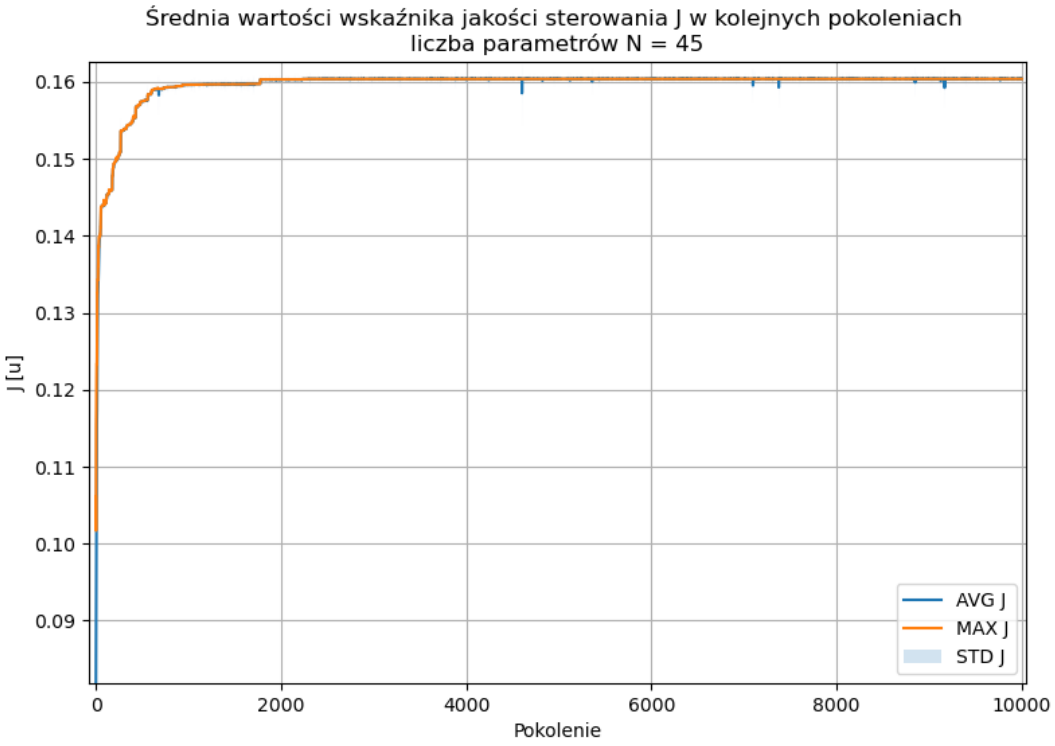

Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 30
J [u]
Pokolenie
AVG J
MAX J
STD J
0.08
0.09
0.10
0.11
0.12
0.13
0.14
0.15
0.16
0
200
400
600
800
1000


Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 35
J [u]
Pokolenie
AVG J
MAX J
STD J
0.09
0.10
0.11
0.12
0.13
0.14
0.15
0.16
0
200
400
600
800
1000


Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 40
J [u]
Pokolenie
AVG J
MAX J
STD J

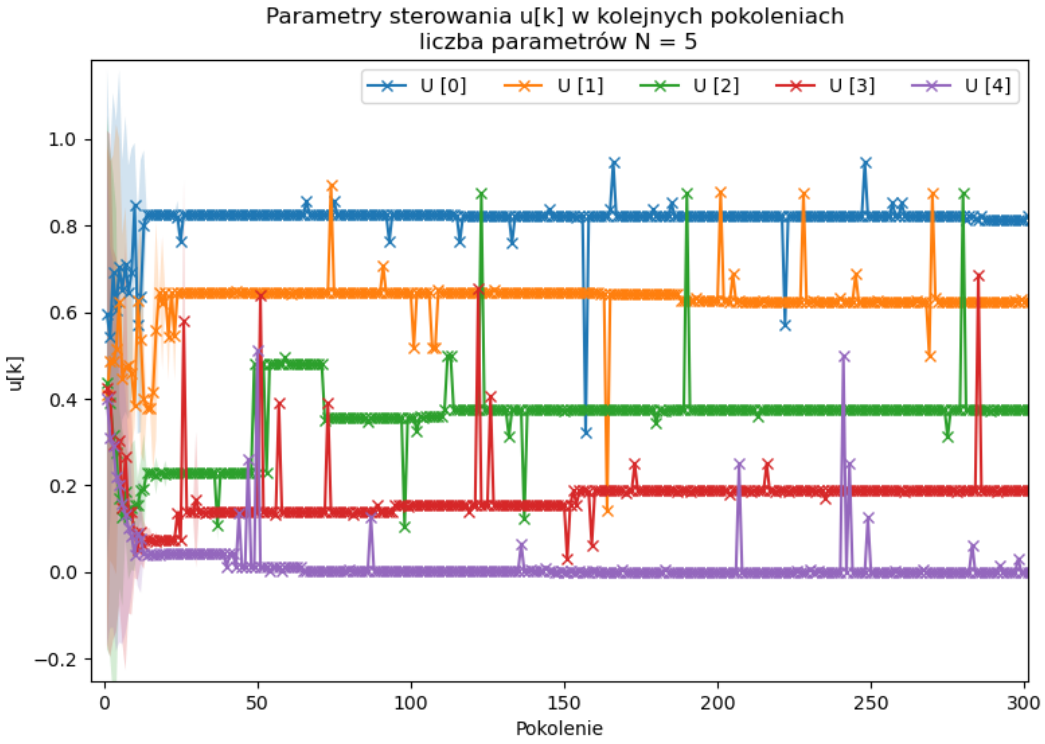

Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 45
J [u]
Pokolenie
AVG J
MAX J
STD J


Optymalne wartości parametrów dla modelu stanowego
liczba parametrów N = 45
X1_X2
10000
8000
6000
4000
2000
Pokolenie
0.26
0.27
0.28
0.29
0.30
0.31
0.32
x1 max
0.33
0.32
0.31
0.30
0.29
0.28
x2 max


Średnia wartości wskaźnika jakości sterowania J w kolejnych pokoleniach
liczba parametrów N = 45
0.16
0.15
0.14
0.13
0.12
0.11
0.10
0.09
J [u]
0
2000
4000
6000
8000
10000
Pokolenie
AVG J
MAX J
STD J

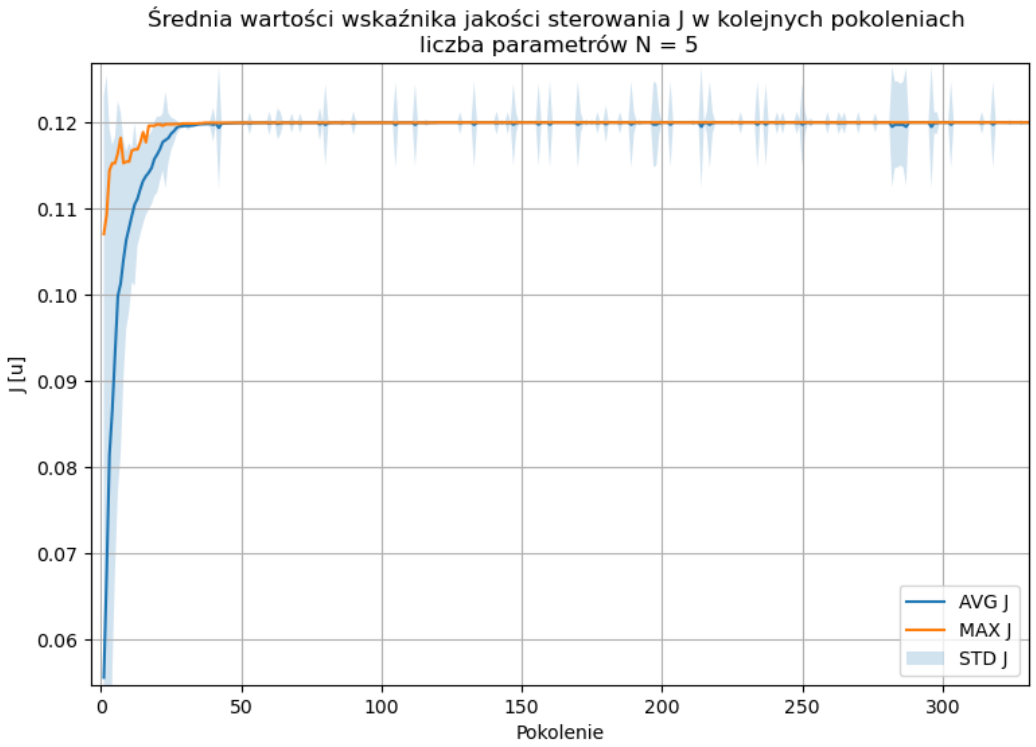
5. Wnioski.

Jak widać na wykresach z poprzedniego rozdziału, im mniejsza liczba parametrów optymalizowanych N, tym szybciej algorytm osiąga optymalną wartość wskaźnika J. Można zauważyć znaczącą różnicę pomiędzy "czasem ustalania" dla N = 45. Dla ostatnich pokoleń, gdy ich liczba wynosi 1000, widać jeszcze znaczące poprawy w wskaźniku J charakteryzujące się "schodkową" zmianą wynikającą z mutacji, krzyżowania i selekcji. Natomiast przy liczbie epok równej 10 000 można zauważyć, że ostatnia wyraźnie zauważalna poprawa występuje w okolicy pokolenia nr 3000 - w kolejnych pokoleniach wykres jest stabilny z pojedynczymi odchyleniami wynikającymi z mutacji. Parametry x1 i x2, poprzez swoją zależność od u, zmieniają się w kolejnych pokoleniach. Można to zauważyć na ostatnim wykresie przedstawiającym przebieg algorytmu dla optymalnych parametrów modelu stanowego. Widać, że dla pierwszych pokoleń wartości tych parametrów ulegają nagłym zmianom, ponieważ początkowo są one losowane. Z każdym kolejnym pokoleniem, dzięki selekcji, krzyżowaniu i mutacji, wartości x1 i x2 zmierzają do ustalonej wartości. Na poniższym wykresie przedstawiono średnie wartości, jakie przyjmują poszczególne parametry u w początkowych pokoleniach, a półprzezroczyste obszary prezentują odchylenie standardowe tych parametrów.

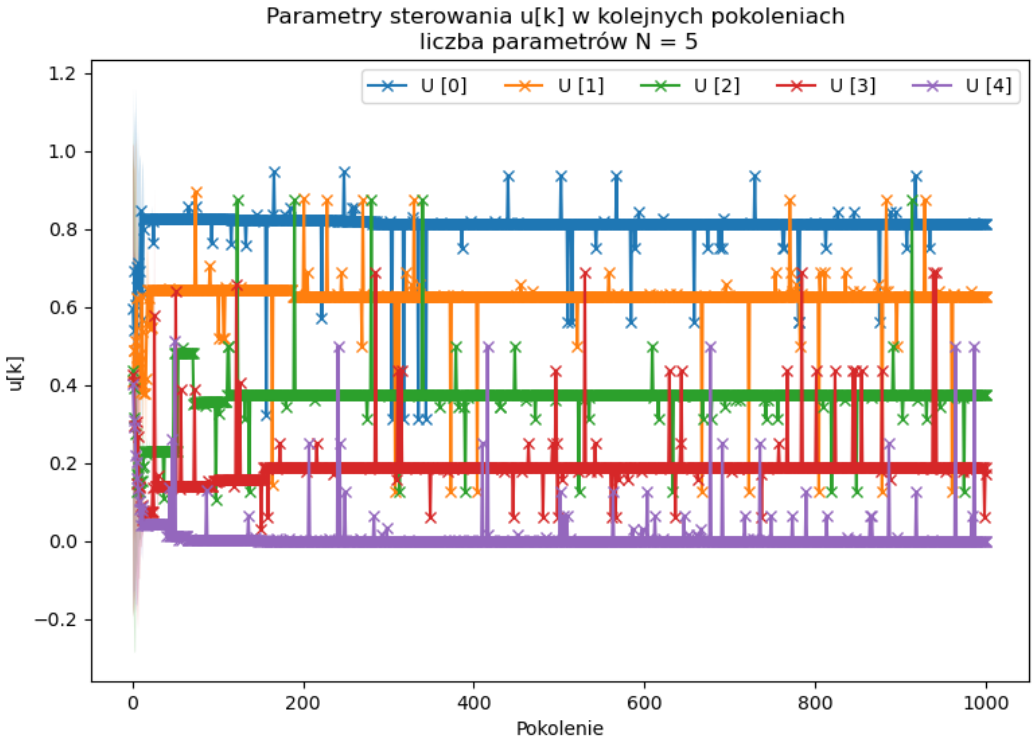


Podczas pierwszych 20 pokoleń średnie wartości parametrów często zmieniają się znacząco. Jest to spowodowane losowością początkowych wartości parametrów. Algorytm używa selekcji, krzyżowania i mutacji, by znaleźć optymalny zestaw parametrów, tak by wskaźnik jakości był maksymalizowany. Proporcjonalna selekcja zwiększa prawdopodobieństwo wybrania najlepszych parametrów. Po około 20 pokoleniach następuje pierwsza stabilizacja parametrów. Odchylenia standardowe są prawie niezauważalne, co oznacza, że wartości parametrów są bardzo podobne do siebie. W kolejnych pokoleniach, od czasu do czasu pojawiają się różne wyniki. Są one spowodowane mutacjami i krzyżowaniem. Jeśli jedna z tych operacji prowadzi do odkrycia lepszego rozwiązania,

dochodzi do zauważalnej, gwałtownej zmiany wartości parametru. Jeśli jednak taka zmiana nie poprawia sytuacji w następnym pokoleniu, parametry wracają do swojej poprzedniej średniej wartości. Wartości wskaźnika jakości przedstawione na wykresie są ściśle związane z wartościami parametru u. Dlatego też istnieje duże początkowe rozproszenie uzyskanych wyników, tak aby wartość wskaźnika jakości stała się coraz bardziej stabilna w kolejnych pokoleniach.



W końcowych pokoleniach, wartości parametrów są już ustabilizowane, jak pokazano na wykresie na kolejnej stronie. Co jakiś czas pojawiają się skoki w średnich wartościach spowodowane mutacjami. Jednak w kolejnym pokoleniu, zmutowane elementy są odrzucane, a średnia wraca do swojego poprzednio ustalonego poziomu.


Parametry sterowania u[k] w kolejnych pokoleniach
liczba parametrów N = 5
U [0]
U [1]
U [2]
U [3]
U [4]
u[k]
Pokolenie
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
400
600
800
1000