

WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

ROZPRAWA DOKTORSKA



mgr inż. Rafał KIESZEK

**WYKORZYSTANIE ALGORYTMU OPARTEGO
NA SZTUCZNYCH SIECIACH NEURONOWYCH
DO OPTIMALIZACJI ELEMENTÓW WIRUJĄCYCH
SILNIKA TURBINOWEGO**

Promotor:

płk dr hab. inż. Adam KOZAKIEWICZ

Promotor pomocniczy:

ppłk dr inż. Robert ROGÓLSKI

Warszawa 2022

SPIS TREŚCI

Spis treści	5
Wykaz oznaczeń	7
Wstęp	10
1. Analiza stanu wiedzy	12
1.1. Sztuczne Sieci Neuronowe w zastosowaniach lotniczych	12
1.2. Zastosowanie Sztucznych Sieci Neuronowych w maszynach przepływowych	16
1.3. Inne zastosowania SSN w lotnictwie	32
1.4. Podsumowanie i teza pracy	34
2. Algorytmy sztucznej inteligencji	35
2.1. Algorytm genetyczny	35
2.2. Sztuczne Sieci Neuronowe	37
2.3. Metody uczenia Sztucznych Sieci Neuronowych	40
2.4. Badania testowe SSN	46
3. Metamodel oparty na SSN do optymalizacji tarczy sprężarki osiowej	59
3.1. Główne elementy algorytmu	60
3.2. Optymalizacja tarczy sprężarki	65
4. Funkcja celu i kary do optymalizacji konstrukcji wirnika tarczowo-bębnowego	69
4.1. Zastosowanie języka APDL do wyznaczania naprężenia w konstrukcji tarczowo-bębnowej	72
4.2. Funkcja celu i funkcja kary	77
5. Dobór struktury i metody uczenia sieci neuronowej	81
5.1. Dobór metody uczenia	81
5.2. Dobór struktury SSN	84
6. Wyniki optymalizacji	93
Wnioski końcowe	106
Elementy nowości	107
Kierunki dalszych prac	108
Załączniki	109
Załącznik 1 - Tarcza_1D.m	109
Załącznik 2 - sigma_gen.m	127

Załącznik 3 - sigma_gen_poprawiacz.m	129
Załącznik 4 - generator_input.m	130
Załącznik 5 - generator_out.m	131
Załącznik 6 – funkcja_masa.m	132
Załącznik 7 – sigmy.m	135
Załącznik 8 – main_program.m	142
Załącznik 9 – NN.m	148
Załącznik 10 – resetAPDL	149
Załącznik 11 – trener_sieci	150
Załącznik 12 – porownanie_sieci.m	151
Załącznik 13 – funkcja.m	155
Załącznik 14 – funkcja2.m	156
Załącznik 15 - funkcja3.m	157
Załącznik 16 – Generator_kodu_grip.m	158
Załącznik 17 – Przykładowa odpowiedź programu po optymalizacji	164
Bibliografia	165
Spis tabel	172
Spis rysunków i wykresów	173

WYKAZ OZNACZEŃ

Litery alfabetu greckiego:

α	– parametr uczenia stosowany w metodzie momentum
β	– współczynnik sprzężenia w metodzie gradientów sprzężonych
β	– współczynnik stromości funkcji sigmoidalnej
δ	– sygnał błędu
δ	– błąd względny
δ_i	– błąd względny i -tej sieci neuronowej
η	– współczynnik uczenia
η	– efektywność uczenia sieci
λ_k	– parametr regulacyjny w metodzie gradientów sprzężonych z regulacją
λ	– liczba potomstwa z wybrednej liczby rodziców
μ	– liczba rodziców
ν	– liczba Poissona
$\pi^{(j)}$	– obecny kierunek poszukiwań minimum funkcji
ρ	– gęstość materiału
σ	– naprężenia (jeżeli występuje indeks dolny)
σ	– odchylenie standardowe
σ_{red}	– naprężenia zredukowane
ω	– prędkość kątowna tarczy

Litery alfabetu łacińskiego:

a	– parametr uczenia stosowany w algorytmie RPROP
b	– parametr uczenia stosowany w algorytmie RPROP
$E_a(\eta p_k)$	– aproksymowana wartość funkcji celu
B	– szerokość pasa dźwigara
b	– szerokość pasa bez ścianki
$b(\hat{\theta})$	– obciążenie estymatora definiowane równaniem
D^2	– wariancja estymatora
E^2	– sumaryczny błąd kwadratowy
F	– siła
F_t	– siła odśrodkowa od części zamkowej tarczy
F_z	– siła odśrodkowa od łopatk
G_k	– przybliżona wartości hesjanu w k -tej iteracji

$g_k = g(W_k)$	– aktualna wartość gradientu w metodzie gradientów sprzężonych
$g(W)$	– wektor gradientu wektora wag
H	– wysokość dźwigara
$H(W)$	– hesjan wektora wag
h	– wysokość ścianki dźwigara, grubość tarczy
h_i	– grubość tarczy dla kolejnego, znormalizowanego promienia
k	– numer cyklu
k	– stała wzmacniająca karę
$O(h^3)$	– błąd obciążenia szeregu wag
p	– kierunek poszukiwania ekstremum
R	– promień zewnętrzny tarczy
R_e	– granica plastyczności
R_m	– wytrzymałość na rozciąganie
R_i	– promień zaokrąglenia
R_0	– promień otworu centralnego
R^2	– współczynnik determinacji
r	– promień
$\bar{r}$	– promień względny
l	– całkowita długość belki
m	– masa
n	– ilość
t	– czas uczenia pierwszej sieci
$Q(W)$	– funkcja błędu
u	– argument funkcji sumacyjnej
W	– wektor wag
W_y	– wskaźnik wytrzymałości na zginanie
w_{ij}	– wartość wybranego współczynnika wagowego,
w_{i0}	– wartość biasu
V	– objętość
Vol	– objętość części zamkowej
X	– wektor wejściowy
x	– współrzędne położenia przekroju obliczeniowego
$x(t)$	– sygnał wejściowy do neuronu,
y	– wartość wyjściowa z neuronu

Indeksy dolne:

AN – parametr obliczony analitycznie

$ANSYS$ – parametr obliczony za pomocą programu ANSYS

$H - M - M$ – parametr obliczony z wykorzystaniem hipotezy Hubera–von Misesa–Hencky'ego

i, j – dotyczy i -tego, j -ego elementu lub połączenia między nimi

li – dotyczy łopatek sprężarki i -tej tarczy

l – dotyczy łopatki

r – dotyczy kierunku promieniowego

sm – środek masy

t – dotyczy kierunku obwodowego

w – wieńcowe

$MAPE$ – średni bezwzględny błąd procentowy

max – parametr maksymalny

MES – parametr obliczony za pomocą autorskiego programu, opartego na metodzie elementów skończonych

min – parametr minimalny

MSE – błąd średniokwadratowy

$RMSE$ – pierwiastek błędu średniokwadratowego

SSN – parametr obliczony za pomocą Sztucznych Sieci Neuronowych

Stosowane skróty:

BP – wsteczna propagacja błędu

GA – algorytm genetyczny

GAM – uogólniony model addytywny

GBT – wariant algorytmu drzewa decyzyjnego

GDM – metoda momentum

MES – metoda elementów skończonych

MOS – metoda objętości skończonych

MLR – regresja nieliniowa

PSO–BP – Particle Swarm Optimization–Back Propagation

SCG – metoda gradientów sprzężonych z regulacją

SSN – Sztuczne Sieci Neuronowe

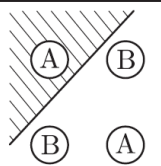
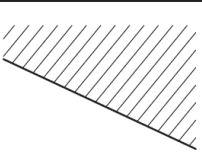
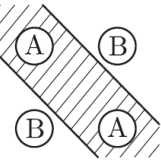
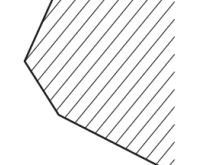
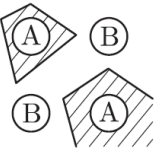
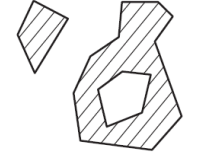
SVR – maszyna wektorów nośnych

WSTĘP

Rozwój metod numerycznych, umożliwiony poprzez zwiększanie mocy obliczeniowej komputerów, pozwala analizować coraz bardziej skomplikowane modele obiektów rzeczywistych, począwszy od obliczeń wytrzymałościowych, zmęczenia czy związanych z mechaniką płynów, a kończąc na analizach sprzężonych kilku wcześniej wymienionych modeli numerycznych. Pomimo wykładniczego wzrostu mocy komputerów ciągle występują zagadnienia, których analiza jest wątpliwa ze względu na potrzebny czas obliczeń. Do zagadnień takich zaliczyć można optymalizację w ujęciu Monte Carlo, zwłaszcza, gdy wartość funkcji celu lub jej ograniczenia oblicza się wielokrotnie z wykorzystaniem wspomnianych metod numerycznych. Proces ten jest niezwykle czasochłonny, nawet przy wstępnym projektowaniu, gdy występuje konieczność przeanalizowania wielu wariantów prototypowych.

Algorytmy genetyczne, przedstawione po raz pierwszy przez Hollanda [1], to grupa inteligentnych algorytmów opartych na naturalnych procesach rozmnażania organizmów żywych takich jak krzyżowanie i mutacje. Algorytmy genetyczne różnią się założeniami wyjściowymi od większości najczęściej stosowanych algorytmów. Zmienne decyzyjne są kodowane w postaci binarnej, a algorytm wykorzystuje bezpośrednio funkcję celu, a nie jej pochodne. Poszukiwanie minimum nie rozpoczyna się w jednym punkcie, jak w przypadku np. metod gradientowych, lecz od pewnego zbioru rozwiązań, zwanego populacją początkową. Ponadto większość procesów zachodzących w programie, takich jak krzyżowanie czy mutacje, ma charakter probabilistyczny, a nie deterministyczny. Istnieje wiele odmian algorytmów genetycznych, zebranych m.in. przez Lee [2] czy Katocha [3]. Krzyżowanie polega na wymianie losowej części bitów między odpowiednio dobranymi rozwiązaniami (osobnikami), które może być realizowane na wiele sposobów [4-8]. Rozwiązania podlegające krzyżowaniu też mogą być wybierane w różny sposób [9-11]. Wszystkie wspomniane metody są stochastyczne, co jest cechą charakterystyczną algorytmów genetycznych. Sposoby regulowania wielkości populacji także mogą być różne. Liczba potomstwa również może być zmienna. Algorytmy typu $(1+1)$, $(\mu+1)$, $(\mu+\lambda)$ i (μ,λ) [12], opisują, jak usuwane są nadmiarowe rozwiązania. Mutacje polegają na negacji pewnej liczby bitów u wybranych osobników. Algorytmy genetyczne są łączone z innymi metodami optymalizacji, takimi jak przeszukiwanie Tabu [13], lub z algorytmami miękkimi, takimi jak logika rozmyta [14] czy sieci neuronowe [15,16].

Sztuczne Sieci Neuronowe (SSN) [17,18] są algorytmem uczenia maszynowego. Jest to algorytm, którego zasada działania jest wzorowana na strukturze i sposobie działania komórki nerwowej. Matematycznie sztuczny neuron składa się z bloku sumacyjnego i bloku funkcji aktywacyjnej. Wszystkie wartości liczbowe wchodzące do sumatora przemnażane są przez odpowiednie wartości wagowe, których zmiana odpowiedzialna jest za proces uczenia sieci. Pojedynczy sztuczny neuron (perceptron) może realizować ograniczone zadanie takie jak dzielenie przestrzeni za pomocą prostej. Połączenie neuronów w sieć neuronową pozwala rozwiązywać bardziej złożone zadania, takie jak: skomplikowane grupowania obiektów, aproksymację niemal dowolnej funkcji, rozpoznawanie dźwięków czy obrazów. Na rys. 0.1 przedstawiono porównanie struktury sieci neuronowej z możliwym podziałem przestrzeni danych.

Struktura	Typ obszarów decyzyjnych	Problem XOR	Najbardziej ogólne kształty obszarów
Jedno-warstwowa 	Półpłaszczyzna ograniczona przez hiperpłaszczyznę		
Dwu-warstwowa 	Wypukłe obszary otwarte lub zamknięte		
Trój-warstwowa 	Dowolne (złożoność ograniczona liczbą neuronów)		

Rys. 0.1. Rodzaje odwzorowań realizowane przez perceptron [19], gdzie XOR – alternatywa rozłączna

Istnieje wiele typów sieci neuronowych jak i wiele algorytmów ich uczenia [20-25]. W przeanalizowanej literaturze najczęściej stosowanymi sieciami neuronowym są przedstawione na rysunku 0.1, wielowarstwowe sieci typu Feed Forward.

W celu zmniejszenia czasu pracy, w ostatnich latach bada się możliwość łączenia Sztucznych Sieci Neuronowych z algorytmami twardymi do obliczeń numerycznych. Wiąże się to bezpośrednio z kosztami projektowania. W lotnictwie SSN mają coraz szersze zastosowanie [26-28]. Wykorzystuje się je w analizie niezawodności [29,30], detekcji usterek [31-34], identyfikacji [35], sterowaniu [36,37] oraz w projektowaniu i optymalizacji konstrukcji [38-40]. Prace te skupiają się na minimalizacji czasu pracy algorytmu w opracowaniu [41], gdzie zasymulowano 7700 sekundową misję w czasie 15 sekund z 99,75% dokładnością. W pracy [40] udowodniono, że możliwa jest symulacja parametrów sprężarki, z użyciem sieci, z dokładnością do 0,1% w czasie około pół sekundy. Innym obszarem jest identyfikacja skomplikowanego matematycznie modelu dynamiki przepływu przez maszyny przepływowe i sterowania nimi. W opracowaniu [42] podjęto próbę zwiększenia skuteczności symulacji pracy turbiny gazowej. Błąd zastosowanego modelu neuronowego nie przekroczył 1%, a dokładność ta została potwierdzona zarówno przez zbiór kontrolny, jak i dla rzeczywistych cyfrowych układów sterowania silnikami. W artykule [43] wykorzystano logikę rozmytą i Sztuczne Sieci Neuronowe do symulacji jednorprzepływowego silnika turbinowego. W publikacji [44] przy wysoko nieliniowej dynamice pracy silnika użyto SSN do detekcji przyczyn uszkodzeń. W pracy [45] użyto SSN do oceny stanu technicznego łopatek na podstawie ich zdjęć. Zastosowano podział na trzy grupy (zdalny, częściowo zdalny i niezdalny), co pozwala wydłużyć ich żywotność pod warunkiem dodatkowego nadzoru. W tej samej pracy metoda ta została wykorzystana do oceny procesu wygrzewania nowych łopatek.

Celem niniejszej pracy było opracowanie algorytmu opartego na SSN do optymalizacji wybranych elementów lotniczego turbinowego silnika odrzutowego z wykorzystaniem algorytmów uczenia maszynowego w celu optymalizacji zespołu sprężarkowego. Struktura programu miała zapewnić zmniejszenie wymaganej ilości iteracji poprzez modyfikację algorytmu genetycznego, a także zastąpienie części obliczeń twardych algorytmem miękkim (Sztucznymi Sieciami Neuronowymi).

1. ANALIZA STANU WIEDZY

Przeanalizowana literatura w dziedzinie napędów lotniczych skupia się na kilku charakterystycznych problemach badawczych. Do jednej grupy problemowej można zaliczyć zagadnienia projektowania i optymalizacji [46-48], drugą z kolei grupę problemów stanowią tematy związane z diagnozowaniem i detekcją usterek [49-51].

Sztuczne Sieci Neuronowe (SSN) są obecnie narzędziem cieszącym się dużym zainteresowaniem zarówno wśród inżynierów, biologów czy przedstawicieli nauk medycznych. Poniżej przedstawione zostaną przykłady implementacji Sztucznych Sieci Neuronowych w zagadnieniach związanych z inżynierią lotniczą ze szczególnym uwzględnieniem analizy maszyn przepływowych w zadaniach z obszaru projektowania i optymalizacji konstrukcji oraz z dziedziny diagnozowania i detekcji usterek.

1.1. SZTUCZNE SIECI NEURONOWE W ZASTOSOWANIACH LOTNICZYCH

Inżynieria lotnicza jest obecnie jednym z bardziej zaawansowanych obszarów techniki, a jej dalszy rozwój wymaga wprowadzania nowych technologii oraz skutecznych metod naukowych, w tym zastosowania zaawansowanych analiz numerycznych. Symulacje te wymagają dużych mocy obliczeniowych i dłuższych czasów obliczeniowych. Metodą wspomagającą w tym obszarze mogą być Sztuczne Sieci Neuronowe, które mają coraz szersze zastosowanie w lotnictwie [26]. Ich implikacja pozwala zwiększyć ilość analizowanych wariantów rozwiązań technicznych i jednocześnie poprawiać ich dokładność. Pozwala także zastosować sterowniki czasu rzeczywistego, których standardowy model matematyczny jest zbyt skomplikowany, aby działać w czasie rzeczywistym. Poniżej przedstawione zostały wybrane prace naukowe, gdzie z wyżej wymienionych powodów zastosowano Sztuczne Sieci Neuronowe.

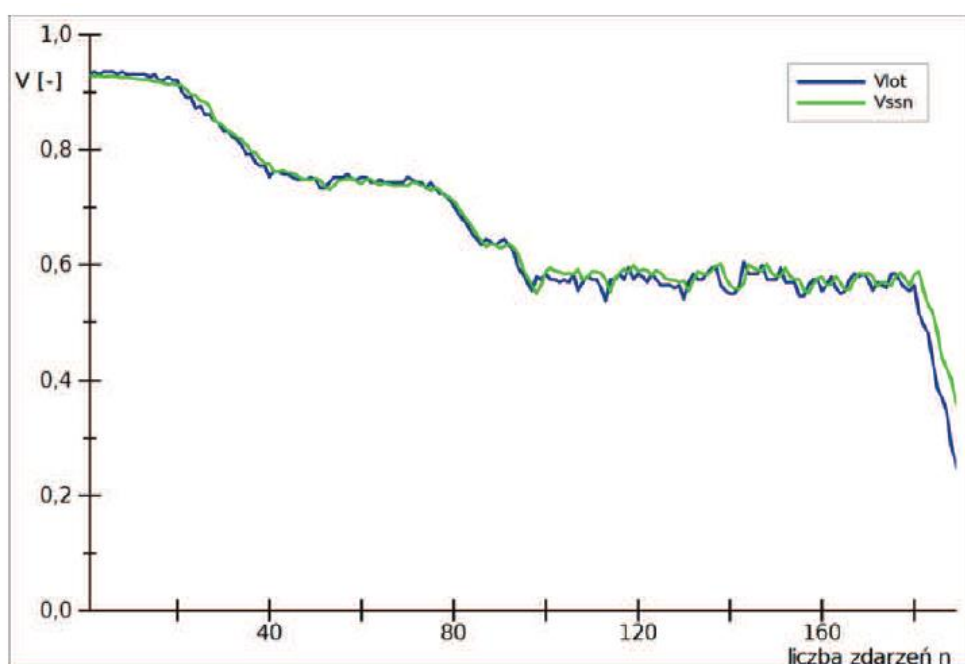
Sieci neuronowe wykorzystuje się w analizie niezawodności, detekcji usterek [33,34,52,53], identyfikacji np. samolotu na podstawie hałasu [35], sterowaniu [36,54] oraz w projektowaniu i optymalizacji.

W pracy [55] autorzy używają Sztucznych Sieci Neuronowych do modelowania fazy zniżania i lądowania samolotów Boeing 767-300ER. Zauważają, że SSN są doskonałym narzędziem do analizy, odzwierciedlania i uogólniania dużej ilości nieliniowych danych. W pracy podzielono fazę zniżania i lądowania na osiem segmentów, których parametry przedstawiono w tabeli 1.1.

Tabela 1.1. Podział faz lotu podczas zniżania i lądowania [55].

Nr fazy lotu	Początek fazy	Koniec fazy
I	Opuszczenie przelotowego poziomu lotu	FL 290
II	Prędkość CAS 144-159 m/s	129 m/s
III	129 m/s	628 m/s; klapy w pozycji 0
IV	628 m/s; klapy 1	103 m/s; klapy 5
V	103 m/s; klapy 5	93 m/s; klapy 20; wypuszczenie podwozia
VI	93 m/s	77 m/s
VII	75 m/s	75 m/s
VII	72 m/s	15 m/s; opuszczenie pasa

Zastosowano sztuczną sieć neuronową o czterech wejściach (czas, fazy lotu, wartość prędkości przyrządowej w chwili czasu $i - 2$ i w chwili czasu $i - 1$, gdzie i to obliczana chwila czasu) i jednym wyjściu (prędkość przyrządowa w chwili czasu v_i). Wewnątrz sieci znajdowały się dwie warsty ukryte do sześciu neuronów w każdej. Do obliczeń zastosowano program JETNET 2, który do nauki sieci używa metody momentum. Na rysunku 1.1 przedstawiono przebieg prędkości lotu wg wskazań przyrządu v_{lot} oraz prędkości wygenerowanej przez sieć neuronową v_{SSN} podczas lądowania samolotu.



Rys. 1.1. Porównanie przebiegu prędkości przyrządowej i wygenerowanej przez SSN podczas lądowania samolotu [55]

Przy zastosowanej przez autorów artykułu [55] sieci o architekturze 4-4-1 średni błąd średniokwadratowy wyniósł $\chi_{sr}^2 = 0,56618 \cdot 10^{-3}$. Pozwoliło to wysnuć wniosek, że sztuczna sieć neuronowa z dość dużym powodzeniem odwzorowała prędkość lotu samolotu w kolejnych jego fazach. Natomiast nieliniowy przebieg zmiany prędkości najlepiej odwzorowują proste i nieduże sieci.

W kolejnej publikacji [56] opisano koncepcję nowego systemu PREDYKTOR wspomagania zarządzaniem remontami silników lotniczych, opartą na przewidywaniach sieci neuronowych. System ten miałby poprawić trafność prognoz zapotrzebowania na części zamienne, co zwiększyłoby płynność finansową zakładu. Miałby on być uzupełnieniem systemu ERP SAP R/3, w którego zakresie mieści się między innymi: planowanie remontów i napraw silników lotniczych, prognozowanie zapotrzebowania na części i podzespoły wymieniane w trakcie remontu, monitoring realizacji weryfikacji zdemontowanych części czy wycena części i procesów naprawczych. System PREDYKTOR miałby dodatkowo przewidywać przyszłe usterki i dostosowywać do nich zapotrzebowanie na części, modyfikując główny plan remontów. Autorzy szacują, że doposażenie systemu ERP SAP R/3 w oparciu o SSN system PREDYKTOR spowoduje zmniejszenie średniego błędu predykcji o około 30%.

Innym obszarem wykorzystywania SSN w lotnictwie jest łączenie pojedynczych zdjęć lotniczych, co zostało omówione w pracy [57]. Zdjęcia lotnicze wykonywane są przez samolot, który stara się utrzymać stałą wysokość i stały kąt względem ziemi. Kolejne zdjęcia robione są w taki sposób, aby sąsiadujące ze sobą pokrywały się w około 60%. Łączenie zdjęć lotniczych wykonywane jest w oparciu o dwie metody. Pierwsza korzysta z mocy obliczeniowej komputerów, gdzie tworzone są siatki, które następnie łączą się z siatkami sąsiednich fotografii. Druga metoda to łączenie zdjęć ręcznie przez człowieka. Człowiek działa wolno, a komputer stosunkowo często popełnia błędy dopasowania. Dlatego celem autora [57] było sprawdzenie, czy za pomocą SNN możliwe jest stwierdzenie, iż badany fragment obrazu zawiera dostatecznie dużo informacji, które kwalifikują go do dalszego dopasowywania. Zadanie to składa się z dwóch części. Po pierwsze należy zdefiniować w sposób zrozumiały i zunifikowany fragment obrazu, a następnie zastosować odpowiedni klasyfikator. Autor do reprezentacji obrazów używa metody sygnatur, tworząc je za pomocą Sztucznych Sieci Neuronowych. Do uczenia i testowania sieci przetworzono 900 zdjęć, z czego mniej więcej połowa została zakwalifikowana jako nadające się do fotogrametrii. Tworząc sygnatury 25- i 50-elementowe, autor nauczył sieć podejmować trafną decyzję w około 80% przypadków. W dalszych badaniach zasugerowano zastosowanie sieci Kohonena do weryfikacji wyników. Podobne problemy poruszono w pracach [58,59].

W literaturze opisywane jest zagadnienie zastosowania SSN do sterowania obiektami latającymi. Artykuł [60] analizuje ten problem w przypadku sterowania pociskiem mózdzierzowym RAD. W lotniczych pociskach sterowanie opiera się na obrocie obiektu wokół jego osi bezwładności przez odpowiednie działanie powierzchni sterowych, co w następstwie powoduje zmianę kierunku wektora prędkości lotu. Autorzy publikacji chcą za pomocą silników rakietowych działać bezpośrednio na środek masy, co zainicjuje obrót obiektu. Zakładają, że zwiększy to efektywność oddziaływania na wektor prędkości pocisku. Autorzy uproszcili konstrukcję pocisku, rezygnując z większości elementów ruchomych, w tym z ruchomych lotek i żyroskopu. Wprowadza to problem w algorytmie sterowania w postaci określenia, czy cel znajduje się poniżej czy powyżej toru lotu pocisku. Do rozwiązania tego problemu użyto tzw. perceptronowej sieci dwuwarstwowej, która to na podstawie zmiany kąta uchybu w czasie określała położenie celu względem osi pocisku. Drugim badanym rozwiązaniem jest zastosowanie SNN zamiast regulatora

proporcjonalnego określającego względne położenie celu. W przypadku doprowadzenia do sieci neuronowej informacji, czy cel znajduje się powyżej czy poniżej osi pocisku, zwiększono celność od 30% do 50%, przy jednoczesnym zmniejszeniu liczby impulsów generowanych przez silniki rakietowe.

Inny przykład poruszający problem sterowania i SSN został opisany w źródle [61], gdzie przedstawiono metodykę i algorytm estymacji zmiennych stanu BSP w przypadku niekatastroficznej awarii systemu pomiarowego. Autorzy swoje badania przeprowadzają zarówno w oparciu o symulacje w systemie software-in-the-loop, jak i dane zarejestrowane w trakcie lotu. Opracowano kilka algorytmów estymacji niemierzalnych zmiennych stanu samolotu pozwalających na realizację redundancji analitycznej, czyli obliczenia wartości wybranych wielkości na podstawie dostępnych sygnałów. Jakość działania przedstawionych algorytmów jest różna, jednak wystarczająca do dalszego poprawnego sterowania statkiem powietrznym. Według autorów zadowalające jest działanie algorytmów estymacji opartych na sieciach neuronowych.

W kolejnych przeanalizowanych pracach dotyczących lotnictwa Sztuczne Sieci Neuronowe używane są w większości do dwóch celów:

- diagnozowania silników lotniczych, np. [29,30];
- symulowania zachowań statku powietrznego lub silnika turbinowego i/lub sterowania nim, np. [42,43,62].

Praca [30] porusza problem diagnostyki i planowania napraw np. lotniczych silników turbinowych. Przedstawia osiągnięcia z zakresu technologii i strategii monitorowania stanu silników. Poruszane w niej problemy to:

- stosowanie analizy statystycznej i SNN w celu filtracji danych;
- zastosowanie SNN do wykrywania zmian trendów i klasyfikacji;
- programy eksperckie mające służyć do diagnozowania, podawania ostrzeżeń i oceniania zaleceń dotyczących działań związanych z ciągłą zdolnością do lotu.

W pracy wykorzystano lata doświadczeń autorów, będących pracownikami firmy Pratt&Whitney, oraz zgromadzone przez firmę dane o uszkodzeniach oraz ich potencjalnych przyczynach i skutkach. Zgromadzono dostatecznie dużo danych, aby zastosować systemy sztucznej inteligencji. W wyniku pracy przedstawiono modułowy, inteligentny i adaptacyjny system do diagnostyki i prognozyki turbin gazowych. Zaproponowany system wyposażony jest w inteligentną diagnostykę do przewidywania zmian trendów. Poprawia on jakość danych (oddzielając szumy) oraz ich interpretację i prognozę możliwych stanów silnika. Zapewnia to poprawę planowania i przewidywanie napraw, umożliwiając obliczenie ich kosztów, a także predykcję przyszłych uszkodzeń. Przykładowo system klasyfikuje spadek sprawności turbiny o 2,3% jako uszkodzenie TCC (system chłodzenia korpusu turbiny) z prawdopodobieństwem 100% lub spadek ciśnienia spalania z prawdopodobieństwem 2%. Dzięki zastosowaniu połączonych metod takich jak: sieci neuronowe, filtry Kalmana, analiza statystyczna i systemy eksperckie uzyskano znaczną poprawę dokładności, jakości analizy, terminowości i użyteczności raportowania. W kolejnym etapie pracy zaproponowano rozszerzenie systemu tak, aby obejmował drgania, ich skutki i przyczyny, funkcjonowanie układu olejowego, poprawę telemetrii itd., co zapewni dalszą redukcję kosztów.

Podobnie w artykule [29] autorzy, opierając się na swoich wcześniejszych pracach, analizują wydajność i szukają anomalii w niewielkich turbinach gazowych przy wykorzystaniu Sztucznych Sieci Neuronowych.

Przykładem drugiego obszaru badań może być praca [62]. Już pod koniec ubiegłego wieku istniały próby użycia SSN do sterowania statkiem powietrznym. W opracowaniu [62] przedstawiono udane próby, przeprowadzone przez NASA w 1999 roku, sterowania samolotem NF-15B z wykorzystaniem Sztucznej Sieci Neuronowej. W trakcie lotu wstępnie nauczona sieć neuronowa potrafiła bardzo dobrze oszacować stabilność i kontrolować charakterystykę aerodynamiczną obiektu, bazując na informacjach z tuneli aerodynamicznych. Algorytm sterowania wykorzystywał „nabytą wiedzę” do wychylania powierzchni sterowych, stabilizując system i zapewniając ustalone warunki lotu. Udowodniono, że wstępnie wyszkolona sieć neuronowa jest w stanie dostarczać prawidłowe dane niezbędne do kontroli lotu. Dodatkowo wykazano, że model samolotu można zdefiniować w sieci neuronowej i może on pracować poprawnie w dowolnym trybie misji.

1.2. ZASTOSOWANIE SZTUCZNYCH SIECI NEURONOWYCH W MASZYNACH PRZEPŁYWOWYCH

SSN wykorzystuje się również przy pracach związanych z maszynami przepływowymi, w szczególności w silnikach odrzutowych i ich podzespołach. Obszary badań nad Sztucznymi Sieciami Neuronowymi w tym zakresie autor podzielił na dwie grupy:

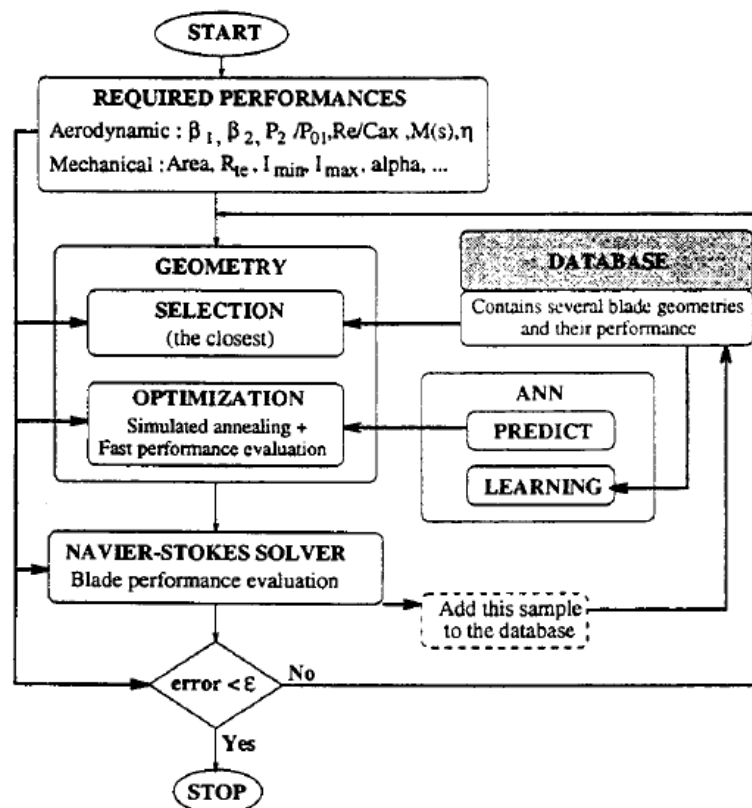
- projektowanie i optymalizacja;
- sterowanie i diagnozowanie lotniczych zespołów napędowych lub ich podzespołów.

PROJEKTOWANIE I OPTYMALIZACJA

W ostatnich latach bada się zastosowanie SSN do standardowych obliczeń numerycznych w połączeniu z MES, przedstawionych w publikacjach [27-28, 63-66] lub w połączeniu z MOS w pracach [67-70].

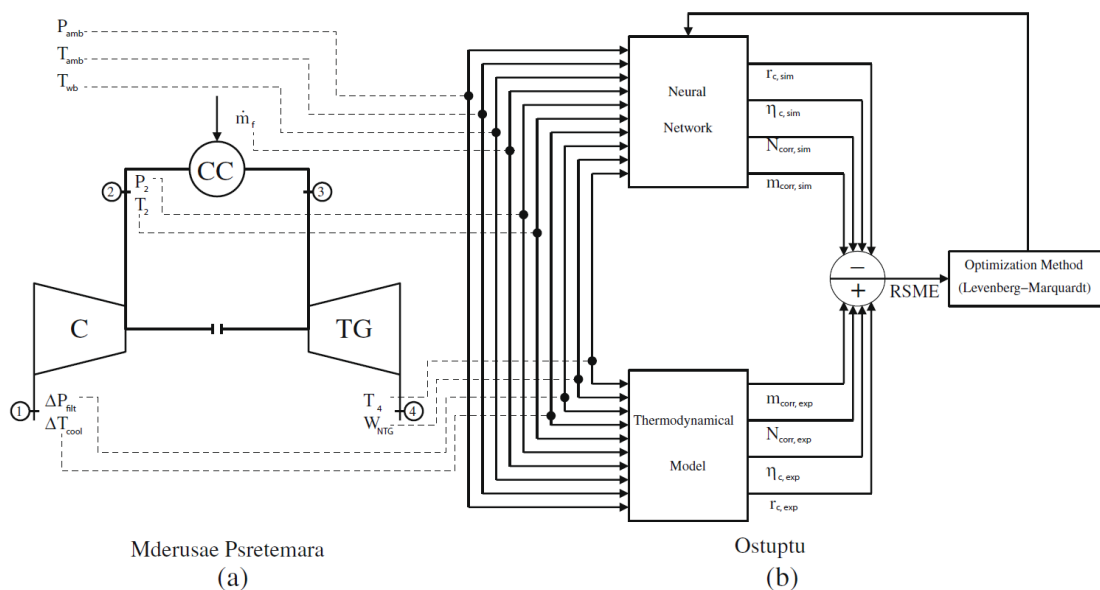
Problem optymalizacji elementów lotniczych turbinowych silników odrzutowych skupia się głównie wokół dwóch zagadnień: optymalizacji kształtu łopatki, np. [71-73], gdzie zazwyczaj funkcją celu jest sprawność stopnia lub optymalizacji kształtu kanału przepływowego silnika, w którym sprężarka jest tylko jednym z podzespołów [74]. Przykładowy problem optymalizacji masy tarczy sprężarki przedstawiono w [75, 76], gdzie autorzy z wykorzystaniem MES (bez użycia SSN) analizują możliwości optymalizacji tarczy, pracującej w podwyższonych temperaturach. W większości tego typu przypadków dopiero zaczyna się stosować SSN jako metamodel aproksymujący dokładne rozwiązanie.

Jednym ze starszych opracowań opisujących wykorzystanie Sztucznych Sieci Neuronowych przy projektowaniu jest praca [67]. Jej autorzy używają SSN do projektowania i optymalizacji kształtu łopatek turbiny za pomocą algorytmu przedstawionego na rysunku 1.2. Funkcją celu jest maksymalizacja sprawności turbiny z uwzględnieniem funkcji kary, wiążącej dopuszczalny zakres liczby Macha i oderwanie przepływu. Zaprezentowany algorytm pozwala skrócić czas optymalizacji, minimalizując ilość pracy człowieka przy jednoczesnym przyspieszeniu obliczeń poprzez zastąpienie części z nich szybszym algorytmem neuronowym.



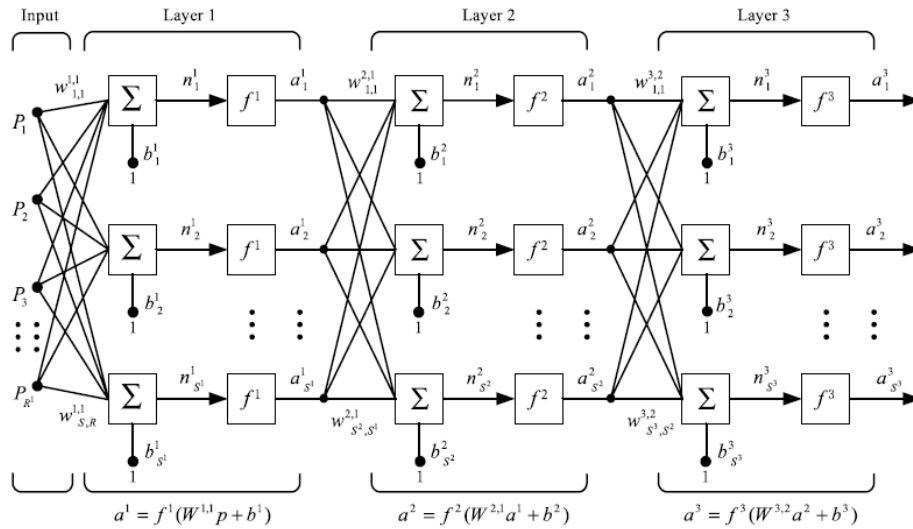
Rys. 1.2. Ogólna struktura algorytmu hybrydowego przedstawionego w pracy [67]

Innym przykładem zastosowania sieci neuronowej do projektowania i optymalizacji może być praca [40]. W tym artykule autorzy wykorzystują odwrócone SSN i sympleksową metodę spadku do przewidywania wartości podstawowych parametrów sprężarki (sprężu, sprawności, prędkości kryterialnej i masowego natężenia przepływu), na podstawie danych wejściowych takich jak ciśnienie i temperatura otoczenia, spadek ciśnienia na filtrze, temperatura gazów wylotowych oraz przed turbiną, ciśnienie za sprężarką, moc turbiny i zużycie paliwa (rys 1.3).



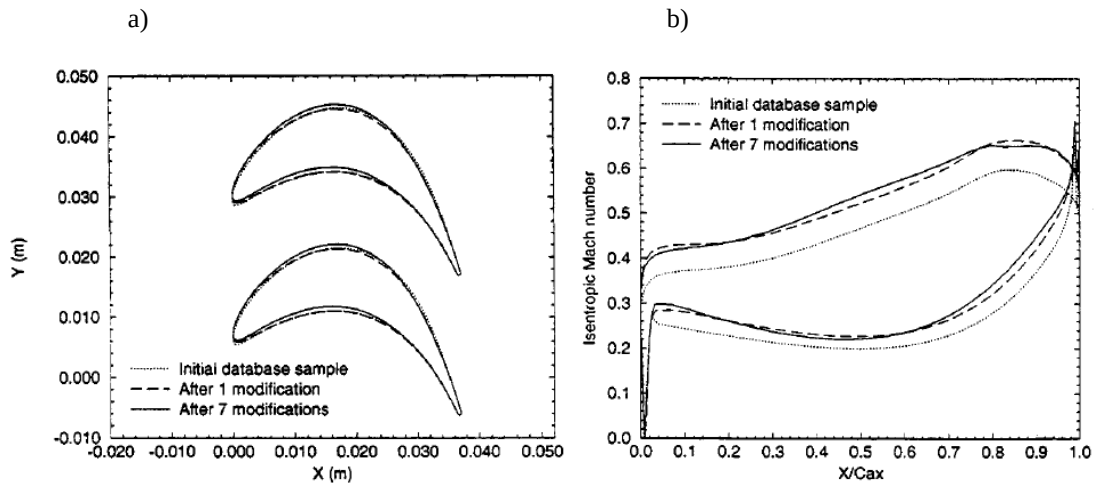
Rys. 1.3. Schemat danych wejściowych do uczenia SSN wraz z zaznaczoną pętlą sprzężenia zwrotnego błędu średniokwadratowego [40]; (a) schemat turbiny gazowej, (b) schemat uczenia Sztucznej Sieci Neuronowej

W tej pracy udowodniono, że możliwa jest symulacja parametrów sprężarki, z użyciem sieci, z dokładnością do 0,1% w czasie około pół sekundy. Podobny problem wyznaczania charakterystyki sprężarki został rozwiązany także w publikacjach [38, 39]. Wykorzystaną w pracy [38] sieć neuronową przedstawiono na rysunku 1.4.



Rys. 1.4. Struktura trójwarstwowej sieci neuronowej z wsteczną propagacją błędów do wyznaczania parametrów turbiny gazowej [38]

Innym zbliżonym tematycznie przykładem zastosowania SSN w optymalizacji może być publikacja [67]. W wyniku przeprowadzonej optymalizacji, z użyciem algorytmu zawierającego SSN, autorom udało się zmniejszyć współczynnik strat o około 4% względem wyjściowego profilu. Rysunek 1.5 przedstawia wyniki uzyskane z przeprowadzanych optymalizacji.



Rys. 1.5. Wyniki optymalizacji płaskiej palisady turbiny; a) geometria łopatki turbiny, b) rozkład liczby Macha na powierzchni łopatki turbiny [67]

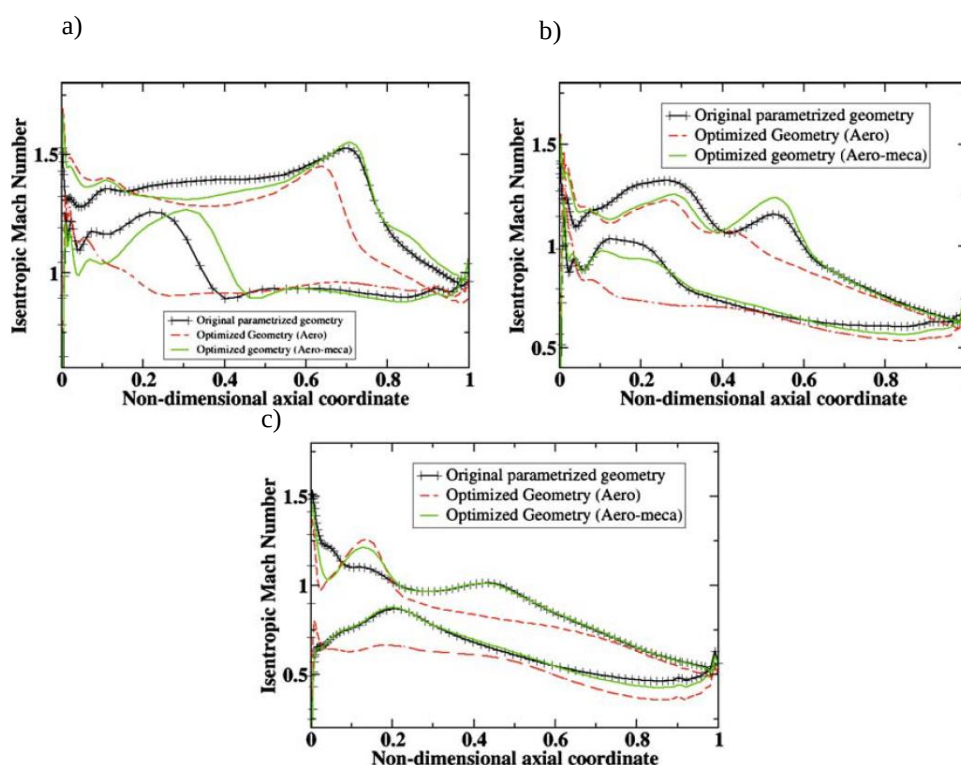
W pracy [67] zauważono, że zaproponowany algorytm może z powodzeniem służyć do optymalizacji łopatek turbin. Wyklucza on powstawanie i sprawdzanie nierealistycznych łopatek, a zastosowanie SSN znacznie zwiększa szybkość obliczeń poprzez nabywane przez sieci doświadczenie. Dodatkowo algorytm ten pozwala w pełni zautomatyzować proces optymalizacji, gwarantując, że algorytm nie utknie w lokalnym minimum. Zaproponowana w artykule [67] metodyka może być zastosowana do

optymalizacji łopatek sprężarek, a także do optymalizacji łopatek w 3D. Podobne zagadnienia poruszone zostały również w artykułach [69, 70], jednak tam ograniczono się do szczegółowych obliczeń gazodynamicznych.

W publikacji [77] przedstawiono nowatorską metodę prognozowania pola temperatur w trójwymiarowym modelu wieńca wirnikowego turbiny. W tym celu wykorzystano algorytm łączący MOS i SSN. Za pomocą programu ANSYS wyznaczono rozkład temperatur służący jako dane wejściowe do uczenia SSN. Dało to możliwość predykcji rozkładu temperatury dla wybranych węzłów modelu turbiny, przy zmianie prędkości obrotowej wirnika, temperatury i ciśnienia na wejściu do niej. Maksymalny błąd bezwzględny temperatury wynosi 44,7 [K], a średni bezwzględny błąd (MAPE) to 0,5%.

W artykule Keshtegar [78] SSN zostały użyte do przewidywania naprężeń, przemieszczeń i odkształceń w silnikach turbinowych. Autorzy analizowali wybrane parametry tarczy turbiny o konstrukcji typu blisk w czasie zadanej misji.

W publikacji [71] przeprowadzono trójwymiarową optymalizację stopnia sprężarki z wykorzystaniem SSN. Optymalizację, której funkcją celu była sprawność sprężarki, wykonano w dwojaki sposób. Najpierw przeprowadzono optymalizację aerodynamiczną a następnie aeromechaniczną. Wyniki analizy numerycznej przedstawiono na rysunku 1.6.

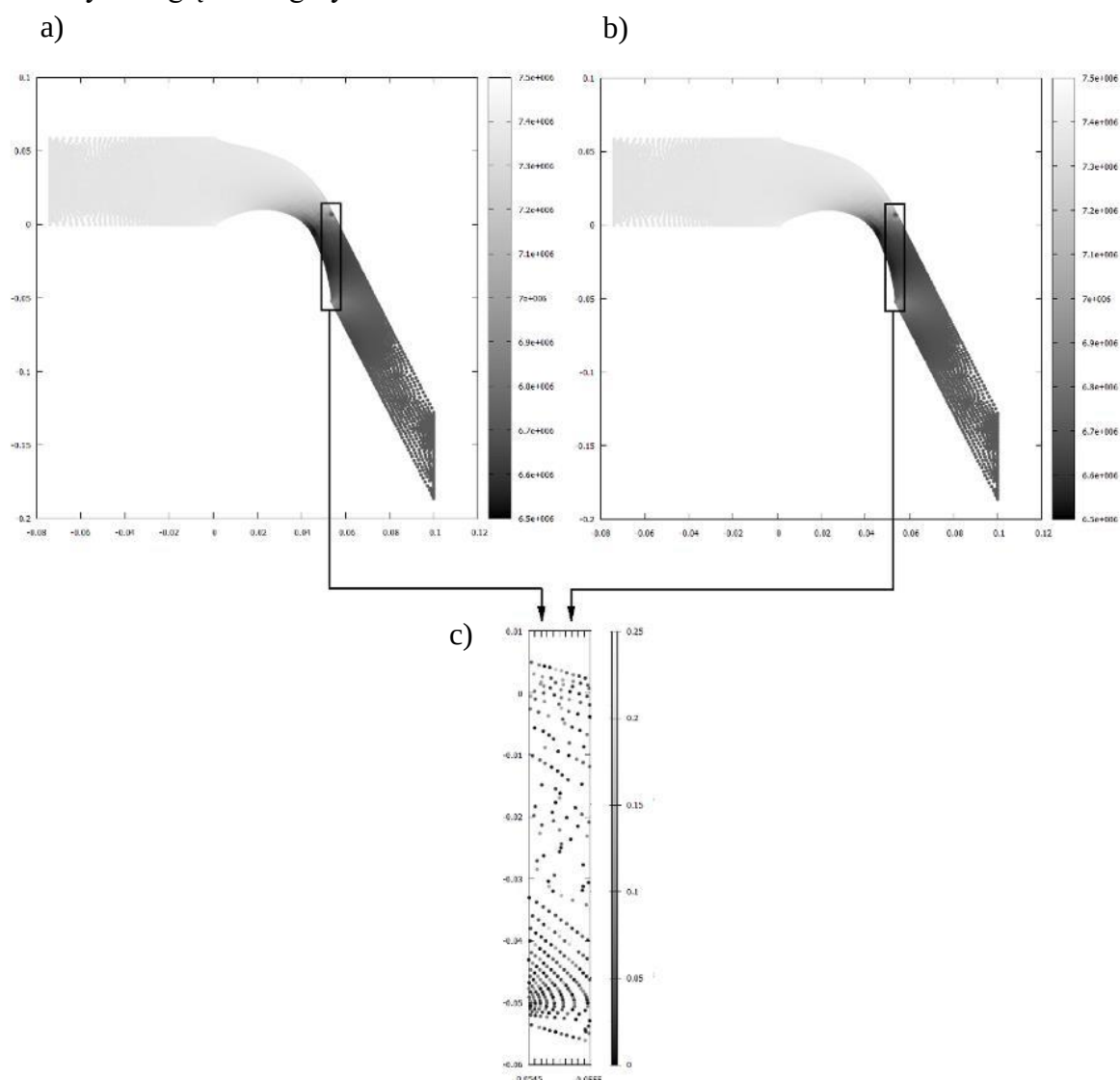


Rys. 1.6 Rozkład liczby Macha na promieniu zewnętrznym (a), promieniu średnim (b), promieniu wewnętrznym (c) kanału sprężarki [71]

W wyniku optymalizacji uzyskano profile sprężarki na trzech podstawowych promieniach (wewnętrznym, średnim i zewnętrznym), a także zwiększono sprawność sprężarki o 0,5% . Dodatkowo wyznaczono zmienne częstotliwości drgań własnych, które znalazły się poza obszarem częstotliwości potencjalnie powodujących rezonans.

W pracy [68] poruszany jest problem czasu obliczeń numerycznych w połączeniu z metodą objętości skończonych. W swoim artykule autorzy używają hybrydowego

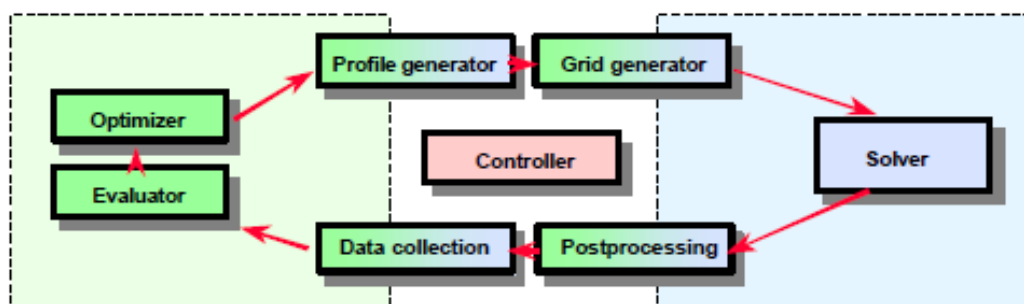
programu łączącego MOS i SSN do analizy stopnia zużycia wieńców turbin parowych. Na podstawie zgromadzonej bazy danych nauczono sieć neuronową obliczać wybrane parametry przepływu z dokładnością do 0,25% (rys. 1.7), w czasie około 90 razy krótszym względem algorytmu MOS.



Rys. 1.7. Rozkład ciśnienia w kanale przepływowym turbiny: a) obliczenia za pomocą SSN, b) obliczenia CFD, c) wartości błęd [68]

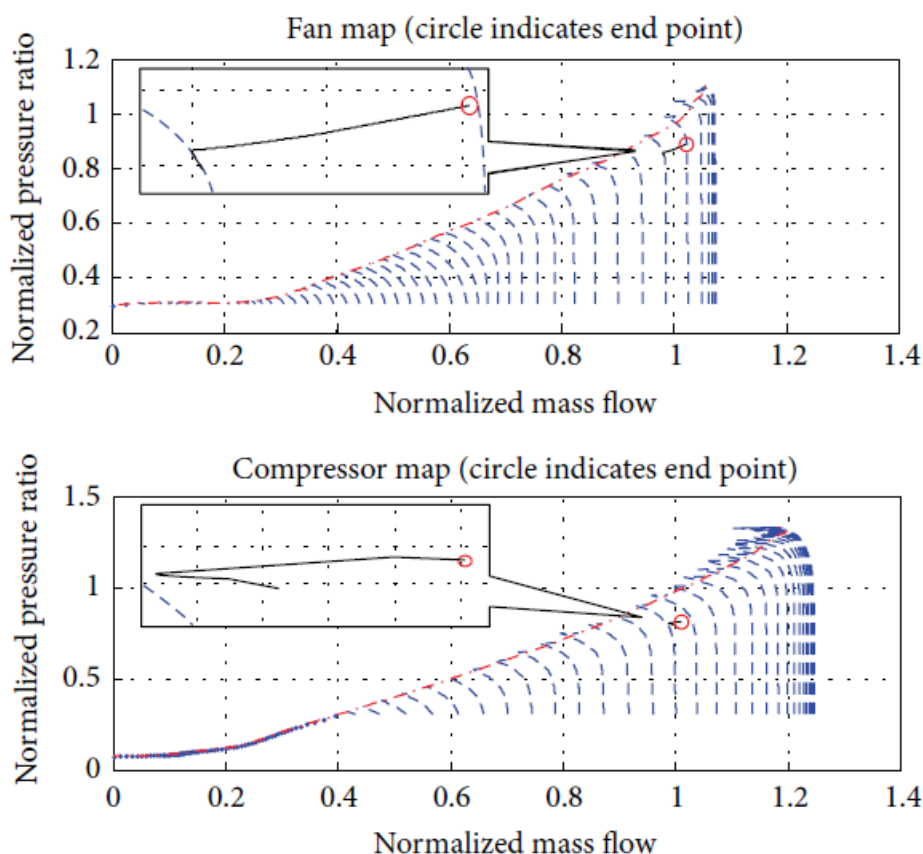
W materiałach konferencyjnych [72] udowodniono, że Sztuczne Sieci Neuronowe można z powodzeniem zastosować do obliczeń optymalizacyjnych, a dokładnie do identyfikacji funkcji celu, której przebieg uniemożliwia poszukiwanie minimum metodami innymi niż Monte Carlo. Autorzy, na przykładzie optymalizacji 3D łopatek stopnia turbiny, wykazali, że czas potrzebny do uczenia SSN przy zachowaniu dokładności rzędu 4% jest około 10-20 razy dłuższy niż aproksymacja funkcji celu równaniem wielomianowym, przy dodatkowo pięciokrotnie dłuższym czasie symulacji. Całkowity czas pracy algorytmu pozostaje pięciokrotnie dłuższy. Wysunięty został wniosek, że w przypadku funkcji celu, których nie można aproksymować za pomocą funkcji wielomianowych z dostateczną dokładnością, dobrze sprawdzają się Sztuczne Sieci Neuronowe. W przeciwnym razie lepiej zastosować aproksymację. Schemat

blokowy wykorzystanego do badań numerycznych algorytmu został przedstawiony na rysunku 1.8.



Rys. 1.8. Algorytm optymalizujący łopatkę 3D turbiny [72]

Przy projektowaniu silnika odrzutowego również stosuje się SSN, czego przykładem może być artykuł [41]. W pracy tej autorzy wykorzystują sieci neuronowe do obliczeń podstawowych parametrów silnika odrzutowego w przypadku pracy na zakresach nieustalonych. Uzyskano tu między innymi charakterystykę sprężarki i wentylatora (rys. 1.9) wraz z linią pracy (wyszczególniony fragment charakterystyki na rysunki 1.9) dla zakresów nieustalonych zgodnych z założonym planem misji.



Rys. 1.9. Charakterystyka sprężarki i wentylatora wraz z linią pracy[41]

W pracy udowodniono, że możliwa jest symulacja 7700-sekundowej misji w czasie 15 sekund z 99,75% dokładnością. Pozwala to przyspieszyć proces projektowania

i zwiększyć sprawność silnika poprzez wykonanie większej ilości analiz w tym samym czasie.

W literaturze nie natrafiono na opis procesu optymalizacji z użyciem Sztucznych Sieci Neuronowych do wyznaczania rozkładu naprężeń elementów konstrukcyjnych maszyn wirnikowych, w tym tarczy sprężarki osiowej, gdzie funkcją celu byłaby minimalizacja czasu obliczeń. Jedynie w publikacji [27] autorzy wykorzystują SSN do wyznaczania rozkładu naprężeń na wybranych promieniach bieżących tarczy sprężarki, jednak bez przeprowadzenia procesu optymalizacji.

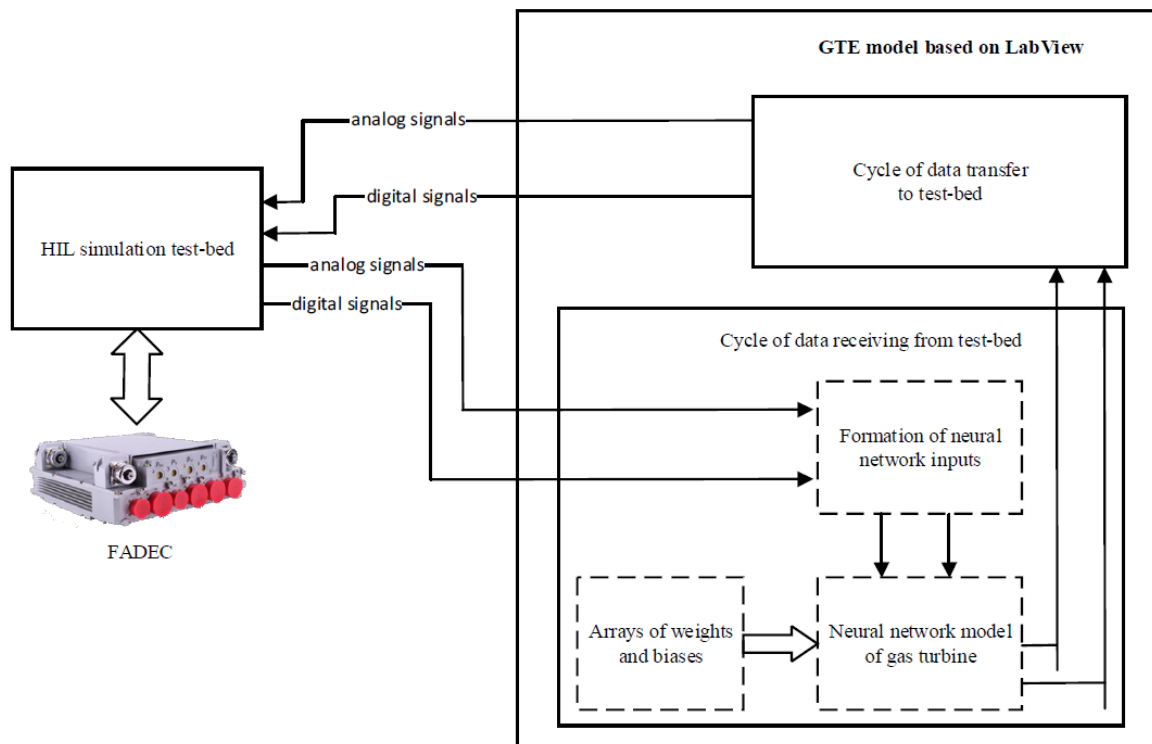
DIAGNOZOWANIE I STEROWANIE

Sztuczne Sieci Neuronowe mogą być również użyte do: diagnozowania silników lotniczych [29, 30, 56, 70] oraz do symulowania stanów i sterowania statkiem powietrznym lub silnikiem turbinowym [42, 43, 79]. Autorzy prac [80-82] stosują SSN do identyfikacji, wykrywania trybów pracy silnika, analizy i przewidywania tendencji, klasyfikacji uszkodzeń i przewidywania stanu silnika, co może być użyteczne zarówno przy diagnozowaniu, jak i sterowaniu.

Sztuczne Sieci Neuronowe są narzędziem na tyle użytecznym, że coraz częściej uczy się ich stosowania w ramach zajęć akademickich, czego przykładem jest praca [83]. Autor opisuje tu zastosowanie SSN do przewidywania parametrów silnika turbinowego.

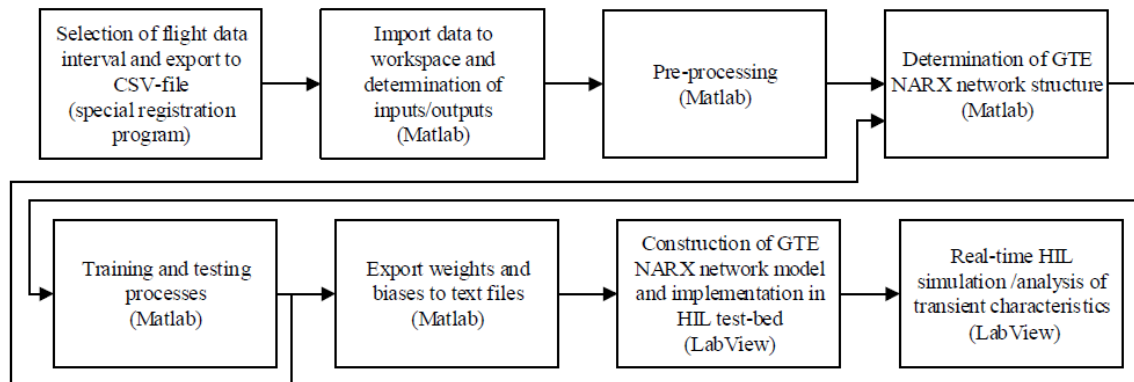
W rozprawie [84] zbadano możliwości stosowania systemów sztucznej inteligencji do sterowania maszynami. W publikacji [85] przedstawiono sposoby realizacji sterowania z zastosowaniem sieci neuronowych przy wykorzystaniu tak zwanego „white-box” i „black-box”, gdzie przez „white-box” należy rozumieć podsystem z jawnie ukazanymi zależnościami, a „black-box” – system, którego wewnętrzne operacje są serią procesów nierozpoznanych przez operatora.

W artykule [42] podjęto próbę zwiększenia skuteczności symulacji pracy turbiny gazowej z wykorzystaniem nieliniowych, dynamicznych modeli matematycznych i rekurencyjnych sieci neuronowych. Opisano metodę budowy sieci neuronowej, strukturę warstwy ukrytej i sposób uczenia sieci, co miało na celu identyfikację modelu turbiny gazowej na podstawie danych z pracy silnika. Metamodel neuronowy, w połączeniu z systemem FADEC, przebadano na stanowisku kontrolnym w zamkniętej pętli czasu rzeczywistego (rys. 1.10). Układ przebadano dla rozruchu na ziemi i podczas lotu.



Rys. 1.10. Struktura modelu w LabView wraz ze schematem jego integracji z układem FADEC [42]

W badaniach turbinowego silnika odrzutowego wykorzystano superkomputer i rekurencyjne Sztuczne Sieci Neuronowe do symulacji w czasie rzeczywistym. Metoda identyfikacji układu przez SSN została przedstawiona na rysunku 1.11.



Rys. 1.11. Metoda identyfikacji modelu przez SSN w oparciu o rzeczywiste dane [42].

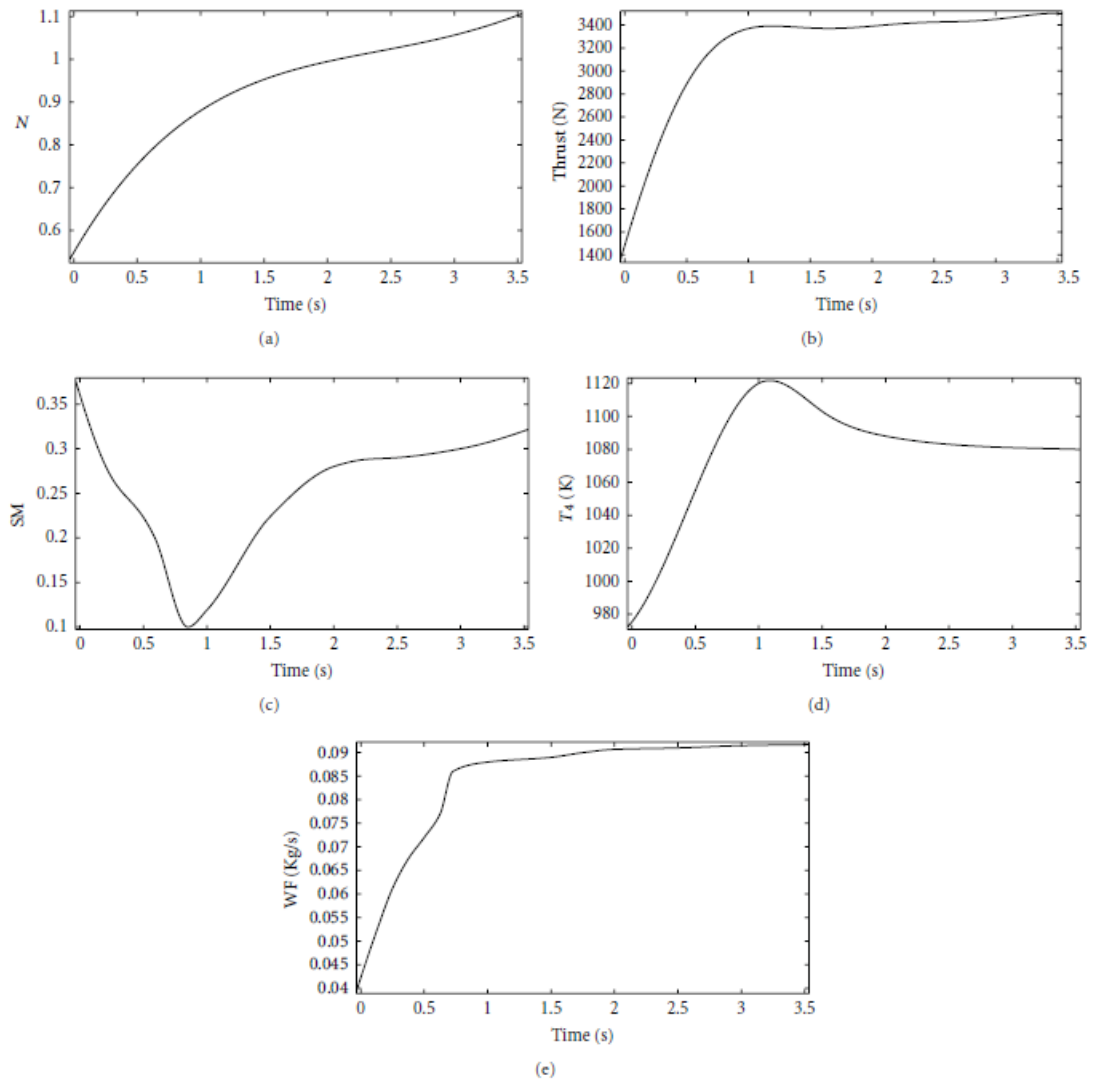
Pierwszy blok to selekcja danych z lotów testowych, następnie dane są importowane do programu MATLAB, gdzie następuje ich grupowanie na dane wejściowe i wyjściowe. Kolejny krok to przygotowanie struktury SSN symulującej silnik odrzutowy. Następnie ma miejsce uczenie i sprawdzanie SSN. Użyto sieci typu NARX (*nonlinear autoregressive neural network with external input*) z jedną warstwą ukrytą, dla której dane wejściowe to parametry lotu i pracy silnika. Po treningu SSN porównano parametry modelu takie jak: prędkość obrotowa wirnika niskiego i wysokiego ciśnienia, temperatura gazów wylotowych, kąt nastawy wieńców kierownic i spręż z parametrami obiektu rzeczywistego. Błąd nie przekroczył 1%. Kolejny krok to eksport metamodelu

neuronowego (wag i wartości biasów) do programu LabView, gdzie ma miejsce symulacja silnika w czasie rzeczywistym i analiza charakterystyk przejściowych.

W wyniku przeprowadzonej pracy stwierdzono, że opracowana metoda analizy i identyfikacji wykorzystująca rekurencyjne sieci neuronowe uwzględnia dynamikę silnika i umożliwia uzyskanie odpowiedniego modelu silnika niezależnie od trybu jego pracy. Dokładność ta została potwierdzona zarówno przez zbiór kontrolny, jak i dla rzeczywistych cyfrowych układów sterowania silnikami. Zastosowana metoda „szarego pudełka” umożliwia uzyskanie i dopracowywanie charakterystyk i cech systemów FADEC. Autorzy publikacji stwierdzają, że najbardziej perspektywicznym zastosowaniem zaprezentowanego podejścia jest wykorzystanie go do oszacowania zużycia i przewidywania pozostałego czasu eksploatacji silnika.

Podobny problem poruszono w artykule [43], gdzie z powodzeniem wykorzystano logikę rozmytą i Sztuczne Sieci Neuronowe do utworzenia algorytmu symulującego pracę jednoprzepływowego, jednowirnikowego silnika turbinowego. Za pomocą programu SIMULINK, w pierwszej kolejności, przygotowano generator danych do uczenia SNN. Generator ten to matematyczny model TSO, który na podstawie odpowiednich danych wejściowych oblicza wybrane parametry pracy silnika. Utworzono wielowarstwową sieć perceptronową i poddano ją uczeniu. Pozwoliło to na określenie rozmytych funkcji logicznych zużycia paliwa. Następnie, używając SNN i sterownika logiki rozmytej, zastąpiono opracowany w pierwszym etapie prac model silnika. Udowodniono, iż zaproponowany sterownik osiąga pożądaną wydajność i stabilność.

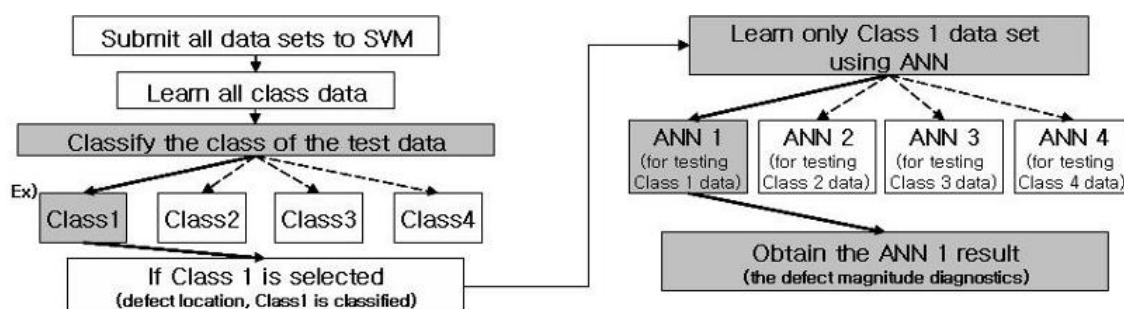
Prace nad problemem inteligentnego sterowania statkiem powietrznym prowadzone są przez NASA [79] oraz inne ośrodki naukowe. Praca [43], podobnie jak [30], używa połączenia SSN i logiki rozmytej. W tym przypadku jednak autorzy wykorzystują zaproponowany algorytm do sterowania skomplikowanym silnikiem turbinowym wojskowego statku powietrznego. Wyniki przedstawione na rysunku 1.12 prezentują przypadek sterowania przy pełnym, skokowym otwarciu przepustnicy. Zarzut temperatury mieści się w granicy dopuszczalnej wartości, a praca sprężarki jest stateczna. Błąd wyznaczania prędkości obrotowej wału wykorzystany został jako rozmyty sterownik, gdzie głównym parametrem sterującym było masowe natężenie przepływu paliwa. Wykorzystanie SSN i rozmytej logiki pozwoliło autorom zapewnić stateczną pracę sprężarki, wysoką sprawność silnika i mieszczącą się w dopuszczalnych granicach temperaturę przed turbiną, przy jednocześnie optymalnym sterowaniu ciągiem.



Rys. 1.12. Odpowiedź modelu silnika na skokową zmianę położenia dźwigni sterowania silnikiem w funkcji czasu, na poziomie morza; a) zredukowana prędkość obrotowa, b) ciąg, c) zapas pracy statecznej, d) temperatura przed turbiną, e) masowe natężenie przepływu paliwa [43]

Kolejnym przykładem zastosowania sieci neuronowych do diagnozowania silnika może być artykuł [33], którego celem było wykorzystanie sztucznej inteligencji (SSN) do przewidywania zużycia silnika VIPER 632-43. W publikacji tej przedstawiono sposób postępowania i doboru Sztucznych Sieci Neuronowych i wykazano, że SSN to dobre narzędzie do przewidywania trendów technicznej degradacji silnika odrzutowego.

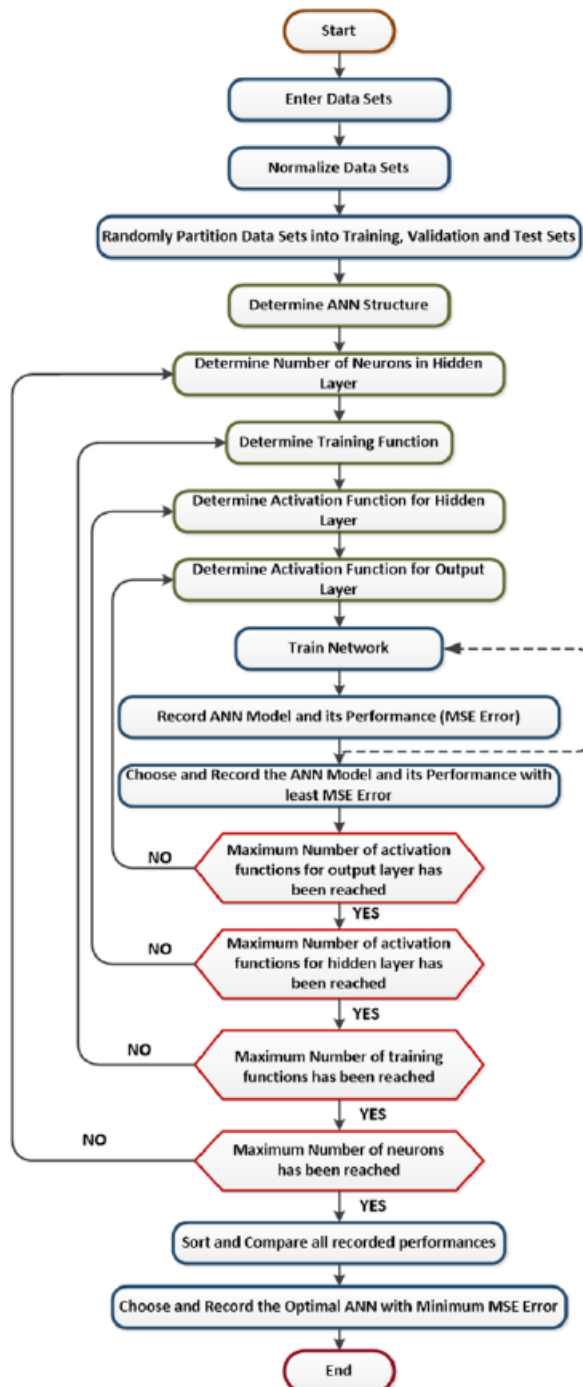
Problem diagnozowania stanu technicznego miniaturowego silnika, bezzałogowego statku powietrznego, został poruszony w pracy [86]. W artykule opisana została metoda połączenia Maszyny Wektorów Nośnych (*Support Vector Machine, SVM*) i Sztucznych Sieci Neuronowych (rys. 1.13) w celu zniwelowania wad SSN zastosowanych do klasyfikacji danych nieliniowych. Połączenie to zmniejsza czas uczenia sieci i zwiększa dokładność diagnozowania usterek silnika turbinowego.



Rys. 1.13. Schemat blokowy hybrydowej metody wektorów nośnych i Sztucznych Sieci Neuronowych [86].

W pracy [70] sieć miała za zadanie zidentyfikować zależność pomiędzy ciśnieniem na wejściu a parametrami przepływu i współrzędnymi poszczególnych punktów przestrzeni obliczeniowej. Przeprowadzane badania numeryczne świadczą o możliwości zastosowania modeli neuronowych do wyznaczania stanów referencyjnych dla potrzeb cieplno-przepływowej diagnostyki turbin parowych.

W rozprawie [84] autor zbadał możliwości stosowania systemów sztucznej inteligencji do sterowania dużymi turbinami takimi jak w turbinach elektrowni i silnikach turbinowych. W rozprawie zbadano możliwości wykorzystania wielowarstwowych sieci neuronowych MLP (*Multilayer Perceptron*) i sieci typu NARX (*Nonlinear Autoregressive Exogenous Model*) do symulowania modelu turbiny umieszczonej w środowisku LabView. Przygotowano 18 720 sieci o różnej strukturze i wykazano, że sieci MLP mogą być stosowane do symulacji silnika z dużą dokładnością i niezawodnością. Na rysunku 1.14 przedstawiono schemat blokowy algorytmu określającego optymalną strukturę SSN do identyfikacji stanów turbiny gazowej.



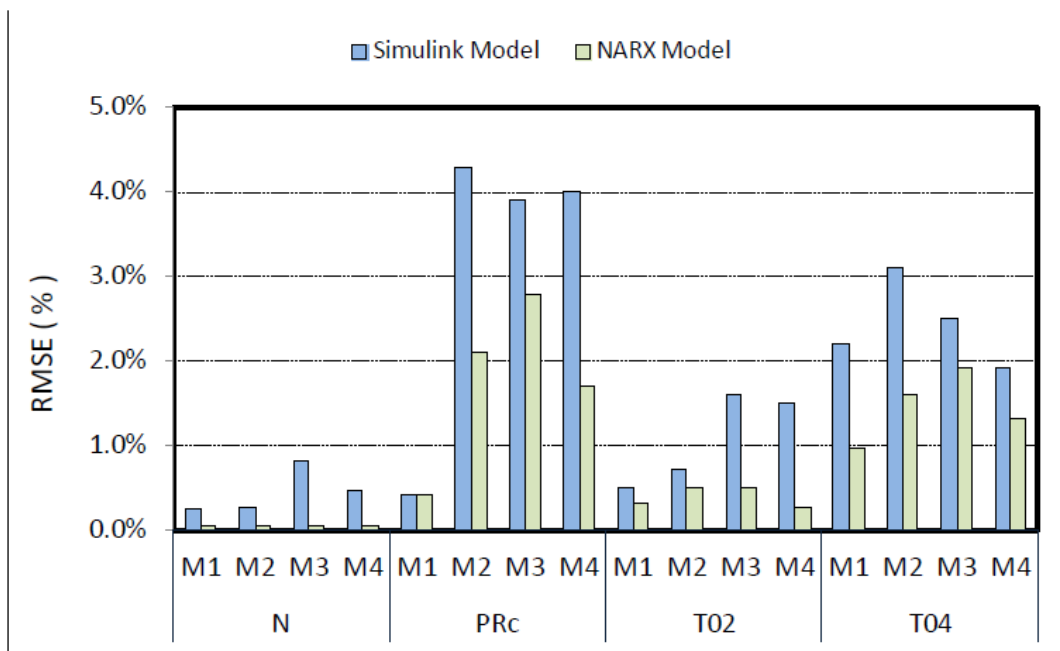
Rys. 1.14. Schemat blokowy algorytmu określającego optymalną strukturę SSN identyfikującej stan turbiny gazowej [84]

Autor pracy dokonał oceny modelu opartego o sieci typu NARX w programie Simulink, wykorzystując rzeczywiste dane eksperymentalne. Modele były konfigurowane i weryfikowane na podstawie zmierzonych zestawów danych (szeregów czasowych). W pracy zauważono, że modele NARX mogą potencjalnie symulować rozruch i przewidywać dynamiczne zachowanie turbin gazowych. Model neuronowy śledzi zmiany parametrów dokładniej niż model konwencjonalny. Wyniki analizy porównawczej dla czterech różnych modeli przedstawiono poniżej, gdzie serie danych M1 i M2 zostały opracowane dla danych treningowych, natomiast M3 i M4 użyto do walidacji możliwości uogólnienia analizy, zgodnie z tabelą 1.2.

Tabela 1.2. Zakres danych dla poszczególnych serii [84].

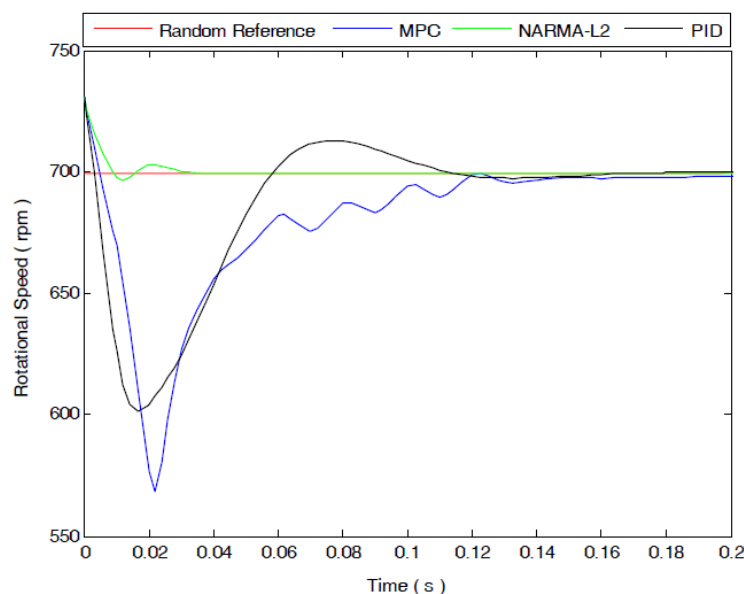
Seria danych	Liczba danych	Zakres danych wejściowych			
		T_{01} [K]	p_{01} [Pa]	$\dot{m}_f$ [kg/s]	$\dot{W}_{load}$ [MW]
M1	1336	<296,48; 301,50>	<99570; 99909>	<3,74; 4,60>	$\approx 0,3$
M2	1165	<297,04; 303,15>	<99570; 99670>	<3,99; 4,50>	<3,30; 18,70>
M3	1506	<290,37; 295,37>	<101940; 102040>	<4,42; 4,80>	<4,42; 4,76>
M4	3296	<299,26; 302,59>	<100950; 101070>	<4,27; 6,90>	<18,20; 23,70>

T_{01} – temperatura na wejściu do sprężarki, p_{01} – ciśnienie na wejściu do sprężarki, $\dot{m}_f$ – masowe natężenie przepływu, $\dot{W}_{load}$ – moc turbiny



Rys. 1.15. Błąd średniokwadratowy modelu Simulink i NARX dla wybranych wyników wszystkich symulacji[84], gdzie M1-M4 – serie danych opisane zgodnie z Tabela 1.2, N – prędkość obrotowa, PRc – spręż sprężarki, T02 – temperatura za sprężarką, T04 – temperatura za turbiną

W trzeciej części pracy [84] porównano sterownik konwencjonalny PID (proporcjonalno-całkująco-różniczkujący) ze sterownikiem opartym na sieciach neuronowych typu MPC (ang. *Model Predictive Control*) w połączeniu z linearyzacją sprzężenia zwrotnego (NARMA-L2). Wykazano, że sterowniki oparte na sieci neuronowej (w tym przypadku NARMA-L2) mogą działać lepiej niż sterowniki konwencjonalne. Czas ustalania, czas narastania i maksymalne przekroczenie dla odpowiedzi NARMA-L2 są mniejsze niż odpowiednie współczynniki dla konwencjonalnego regulatora (rys. 1.16).



Rys. 1.16. Prędkość obrotowa w funkcji czasu. Odpowiedzi trzech różnych kontrolerów silnika odrzutowego[84], gdzie: *Rotational Speed* – prędkość obrotowa [obr/min], *Time* – czas [s]

W podsumowaniu autor wyciąga wniosek, iż pomimo kontrowersyjnych kwestii związanych z SSN mają one duży potencjał i mogą być alternatywą dla tradycyjnych metod modelowania, symulacji i kontroli. Opracowane modele mogą być z powodzeniem używane offline zarówno przy projektowaniu, jak i produkcji oraz online przy eksploatacji, tzn. przy diagnozowaniu stanu technicznego, wykrywaniu usterek, a także przy rozwiązywaniu problemów związanych z turbinami gazowymi. Publikacja [85] zawiera elementy rozprawy [84] i podobnie przedstawia sposoby modelowania, symulowania i sterowania silnikiem turbinowym z wykorzystaniem sieci neuronowych z wykorzystaniem tak zwanego „white-box” i „black-box”. W pracy uwzględniono także dynamikę silnika i opisano sposób doboru SSN do sterowania nim.

Autorzy publikacji [87] proponują zastosowanie sterownika opartego na SSN (DSMNNVB – *Decoupled Sliding-Mode Neural Network Variable-Bound*) do asymptotycznego powrotu do punktu pracy w przypadku wystąpienia niestatecznej pracy sprężarki. Sterownik ten został użyty dla nieznanego, dynamicznego, nieliniowego modelu silnika odrzutowego bez użycia jego modelu matematycznego. Co więcej, spełnia on warunki Lapunowa dla nieliniowych układów dynamicznych.

W pracy [88] przedstawiono problem przewidywania stanu wytwornicy spalin. Zazwyczaj do diagnozowania stosuje się macierze błędów, drzewa decyzyjne lub analizuje się zmiany linii prądu. Takie podejście jest obciążone dużym błędem. W cytowanym artykule zaproponowano użycie sieci neuronowych przygotowanych w zakresie wykrywania, izolowania i oceny błędów w sprężarce jednowirnikowego silnika odrzutowego. Hierarchiczna struktura algorytmu obejmuje szereg zdecentralizowanych sieci przetrenowanych do określonych zadań. Do weryfikacji sieci wykorzystano dane eksperymentalne, które nie były używane w procesie uczenia SSN. W tabeli 1.3 przedstawiono porównanie błędów opisanej metody dla kilku wybranych algorytmów, na przykładzie predykcji stanu podzespołu sprężarki.

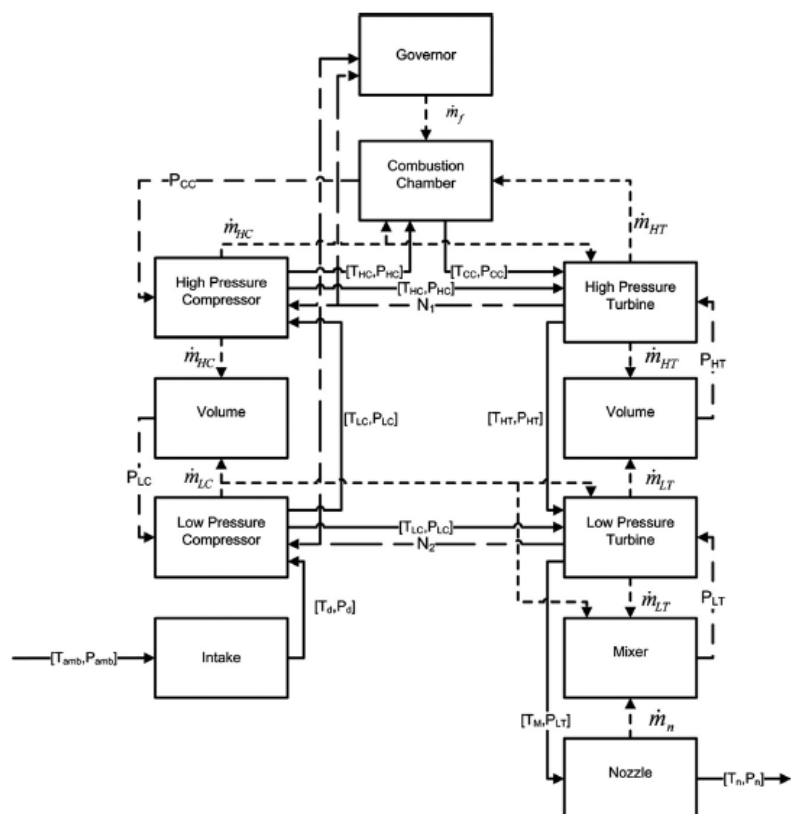
Tabela 1.3. Porównanie wyników diagnozy sprężarki dla różnych algorytmów, gdzie: GPA – *Gas Path Analysis* (analiza linii prądu), ANN – *Artificial Neural Network* (SSN), RMS – błąd średniokwadratowy [88].

Implanted faults		Linear GPA			Non-linear GPA			ANN		
η_{CT}	Γ_{CT}	η_{CT}	Γ_{CT}	RMS	η_{CT}	Γ_{CT}	RMS	η_{CT}	Γ_{CT}	RMS
-0.5	0.5	-0.99	0.38	0.358	-0.51	0.47	0.021	-0.53	0.55	0.038
-0.5	1.0	-1.01	0.86	0.373	-0.51	0.97	0.023	-0.53	1.13	0.098
-0.5	1.5	-1.04	1.30	0.403	-0.51	1.44	0.041	-0.60	1.61	0.109
-0.5	2.0	-1.06	1.75	0.434	-0.49	1.97	0.023	-0.59	1.97	0.065
-1.0	1.0	-2.11	0.69	0.814	-1.01	1.01	0.007	-1.00	1.06	0.040
-1.0	1.5	-2.17	1.13	0.866	-0.99	1.48	0.014	-1.06	1.59	0.075
-1.0	2.0	-2.37	1.55	1.021	-1.00	1.99	0.008	-0.94	1.96	0.053
-1.0	2.5	-2.58	1.95	1.184	-1.00	2.50	0.003	-0.94	2.39	0.085
-1.5	1.5	-3.96	1.01	1.775	-1.51	1.50	0.004	-1.47	1.49	0.020
-1.5	2.5	-4.32	1.73	2.069	-1.67	2.48	0.118	-1.50	2.45	0.033
-1.5	3.5	-4.36	2.46	2.152	-1.52	3.48	0.023	-1.53	3.53	0.028
-1.5	4.5	-4.65	3.16	2.421	-1.49	4.49	0.008	-1.44	4.38	0.097
-2.0	2.0	-5.45	1.17	2.507	-2.01	2.00	0.006	-2.04	1.95	0.042
-2.0	3.0	-5.76	1.89	2.770	-2.02	2.99	0.013	-2.00	3.06	0.041
-2.0	4.0	-6.07	2.58	3.048	-2.01	4.00	0.006	-2.01	4.02	0.018
-2.0	4.5	-6.22	2.91	3.190	-2.01	4.49	0.011	-1.91	4.39	0.096
-2.5	2.5	-6.82	1.34	3.166	NC	NC	–	-2.42	2.39	0.097
-2.5	3.0	-7.01	1.69	3.318	NC	NC	–	-2.43	2.95	0.061
-2.5	3.5	-7.18	2.03	3.469	NC	NC	–	-2.56	3.55	0.059
-2.5	4.5	-7.79	2.69	3.951	NC	NC	–	-2.55	4.56	0.055
-3.0	3.0	-11.53	1.42	6.134	NC	NC	–	-2.95	3.07	0.057
-3.0	3.5	-12.23	1.70	6.648	NC	NC	–	-3.04	3.63	0.095
-3.0	4.0	-11.44	1.98	6.139	NC	NC	–	-3.02	4.16	0.111
-3.0	4.5	-12.37	2.21	6.822	NC	NC	–	-2.99	4.61	0.081
Mean error				2.710	Mean error		0.021	Mean error		0.065

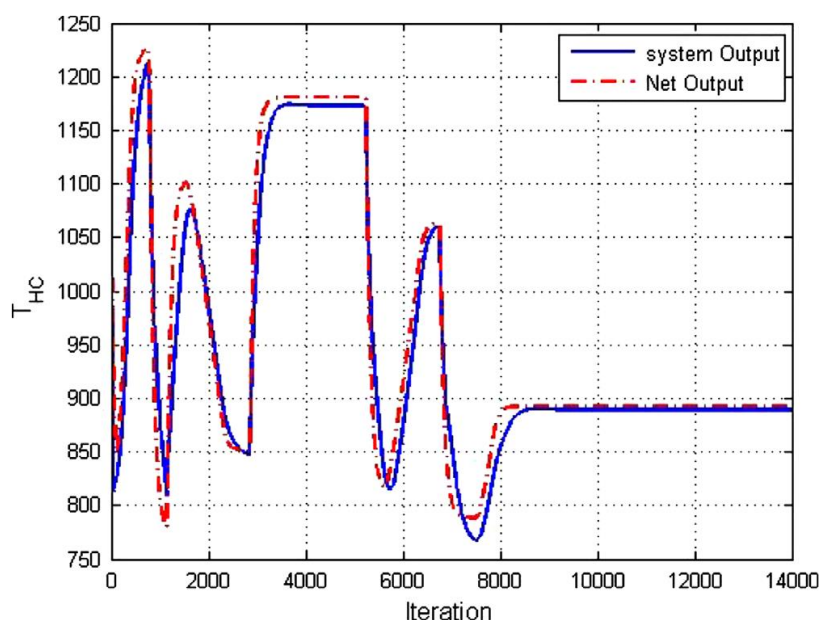
Porównanie wyników zaproponowanego algorytmu neuronowego z innymi metodami pokazuje, że użycie SSN w niektórych przypadkach może zwiększyć dokładność prognoz. Jak stwierdzają autorzy, wyniki te są ilościowe, a nie jakościowe, przez co nie można, bez użycia innych systemów, określić przyczyn uszkodzeń silników odrzutowych.

W czasie eksploatacji, a także przy remontach i naprawach gromadzona jest duża baza danych. Autorzy pracy [89] wykorzystali tę bazę, w połączeniu z SSN, do usprawnienia procesu diagnostyki silników Rolls-Royce RB211-524B4 na latających cysternach od Royal Air Force. W publikacji opisano zastosowaną metodykę z użyciem SSN. Stwierdzono, że w przypadku analizy pojedynczego podzespołu błędy w uczeniu sieci pokrywają się z sygnałem określającym uszkodzenie i wzmacniają je. Pozwala to na łatwiejszą detekcję uszkodzeń. Dzięki analizie par podzespołów (np. HPC-HPT) autorzy zmniejszyli znaczenie błędów uczenia, a wzmocnili sygnał oznaczający usterkę.

W publikacji [44] użyto SSN do detekcji i izolowania przyczyn uszkodzeń, przy wysoko nieliniowej dynamice pracy silnika odrzutowego. Wielowarstwowa, dynamiczna Sztuczna Sieć Neuronowa aproksymuje, na podstawie danych wejście–wyjście, charakterystykę silnika. Analizowany przypadek modelu jest złożony i dynamiczny, co przedstawiono na rysunku 1.17. Z kolei rysunek 1.18 przedstawia wynik testowania sieci neuronowej po zakończeniu procesu uczenia.

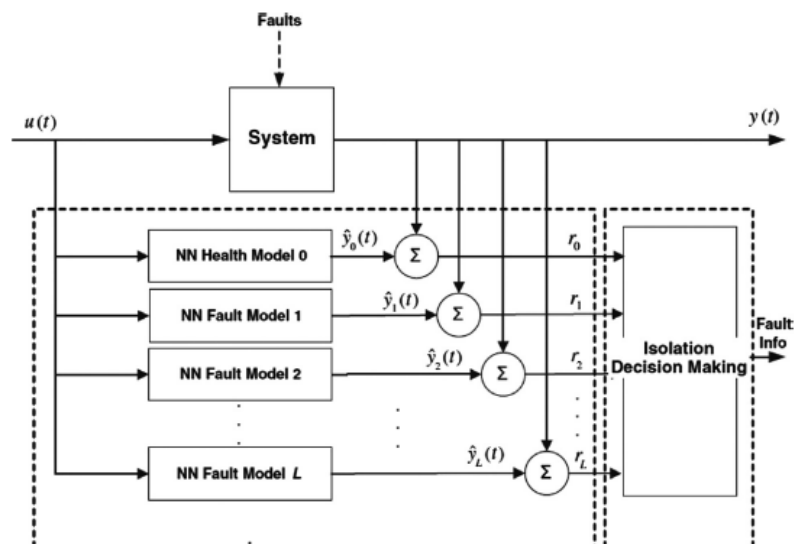


Rys. 1.17. Zależność przepływu informacji między modułami modelu silnika odrzutowego [44]



Rys. 1.18. Wynik testowania przeszkolonej sieci neuronowej dla jednego z silników [44]

Sieć wykorzystuje filtr o nieskończonej odpowiedzi impulsowej, w celu określenia zależności dynamicznej, między wejściem a wyjściem układu, co pozwala wyizolować konkretną usterkę. Algorytm działa także dla dwuwirnikowego silnika odrzutowego. Wykorzystana została analiza różnic pomiędzy idealną a rzeczywistą odpowiedzią układu. Strukturę neuronowego algorytmu decyzyjnego przedstawiono na rysunku 1.19.



Rys. 1.19. Struktura dynamicznej sieci neuronowej do wykrywania usterek sprężarki niskiego ciśnienia dwuwirnikowego silnika odrzutowego [44].

Zaproponowany algorytm dobrze sprawdza się do identyfikacji usterek sprężarki niskiego ciśnienia i tych związanych z prędkością obrotową wirnika. Identyfikacja usterek innych podzespołów (sprężarki i turbiny wysokiego ciśnienia) wymaga dalszego rozwoju algorytmu.

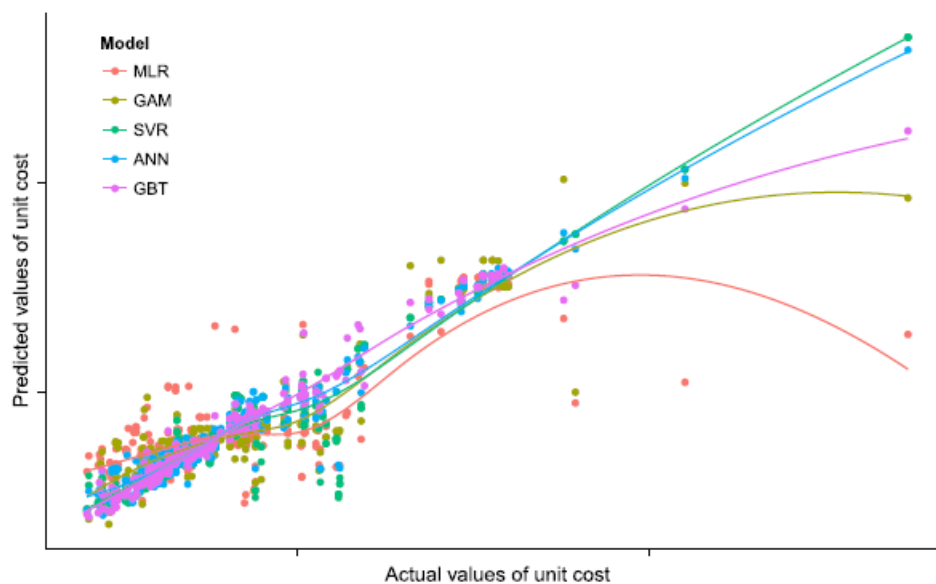
Jeszcze inne podejście do problemu diagnozowania przedstawiono w pracy [45]. Autorzy używają SSN do oceny stanu łopatek turbiny gazowej na podstawie jej zdjęć. W wyniku przeprowadzonych badań stwierdzono przewagę zastosowanej metody nad oceną subiektywną mechanika. Co więcej, zastosowana metoda dzieli łopatki na trzy stany: zdalny, częściowo zdalny i niezdalny (w przeciwieństwie do typowego zdalny–niezdalny). Pozwala to zwiększyć czas pracy łopatek, pod warunkiem dodatkowego nadzoru, zmniejszając całkowite koszty. Dodatkowo metoda ta została także użyta do oceny poprawności procesu technologicznego wygrzewania nowych łopatek.

1.3. INNE ZASTOSOWANIA SSN W LOTNICTWIE

Sztuczne Sieci Neuronowe używane są także do analizy statystycznej przewidywania kosztów i jakości produkcji, czego przykładem jest publikacja [90]. W pracy tej porównano różne algorytmy uczenia maszynowego, takie jak:

- regresja nieliniowa (MLR – ang. *Multiple Linear Regresion*),
- uogólniony model addytywny (GAM – ang. *Generalized Additive Models*),
- Sztuczne Sieci Neuronowe (ANN – ang. *Artificial Neural Network*),
- maszyna wektorów nośnych (SVR – ang. *Support Vector Regression*)
- wariant algorytmu drzewa decyzyjnego (GBT - ang. *Gradient Boosted Trees*).

Na rysunku 1.20 przedstawiono jakość dopasowania danych z użyciem wyżej wymienionych algorytmów dla predykcji kosztu jednostkowego.



Rys. 1.20. Porównanie jakości dopasowania modeli statystycznych z wykorzystaniem danych nieużywanych w czasie przygotowywania algorytmów na przykładzie predykcji kosztu jednostkowego [90].

Sieci neuronowe jako narzędzia do przewidywania trendów, na podstawie rzeczywistych danych statystycznych, wypadają o 40% i 10% lepiej niż kolejno MLR (*Multiple Linear Regression*) i GAMs (*Gradient Boosted Trees*). Jednak sieci mają problem z przewidywaniem zachowania układu dla łopatek sprężarek średniego i wysokiego ciśnienia. Autorzy stwierdzają, że jest to dobry sposób uzupełniający tradycyjne metody przewidywania statystycznego, jednak ze wszystkich porównywanych algorytmów sieci neuronowe wymagają najwięcej mocy obliczeniowej.

Autorzy publikacji [30] przygotowują sieć neuronową w połączeniu z logiką rozmytą. Następnie trenują sieć i wykorzystują ją do diagnozowania i przewidywania zadań związanych z naprawami i konserwacją silników odrzutowych. Przewidują zastosowanie zaproponowanej metody wraz z dodatkowymi czujnikami do teledyktacji w czasie rzeczywistym. Pozwoli to zmniejszyć koszty eksploatacji poprzez wczesne wykrywanie zbliżających się uszkodzeń, zredukowanie przestojów w lotach i lepsze planowanie i zarządzanie aktywami. Łączenie SSN z logiką rozmytą jest badane także w pracy [91]. Podobnie autorzy pracy [56] wykorzystują SSN w celu prognozowania zapotrzebowania na części zamienne do silników lotniczych, co pozwala zmniejszyć błąd prognozy. Dzięki temu możliwe jest także obniżenie kosztów obsługi silników.

Analiza naprężeń za pomocą SSN w łopatkach turbin wiatrowych jest tematem prac [92] i [93]. W pierwszej pracy wykorzystano metodę *Genetic Algorithm-Back Propagation* (GA-BP), zaś w drugiej metodę *Particle Swarm Optimization-Back Propagation* (PSO-BP). W obydwu przypadkach osiągnięto błąd względny około 6% (dla krawędzi natarcia, od strony nadciśnieniowej) w porównaniu z pełnoskalowymi testami statycznymi. Dla krawędzi spływu błąd względny to odpowiednio 6,5% i 18% dla algorytmu PSO-BP i dla GA-BP. W literaturze występują również inne przykłady wykorzystania SSN, np. w obszarze optymalizacji turbin wiatrowych [94, 95].

W pracy [96], podobnie jak w pracach [97, 98], przeprowadzono ocenę niezawodności wirnika turbiny z uwagi na pełzanie, pod kątem luzu wierzchołkowego. Autorzy rozłożyli model wirnika na modele składowe (tarcza, łopatki i korpus), które

przeanalizowali za pomocą MES. Dane te posłużyły do uczenia sieci neuronowej, co pozwoliło osiągnąć dokładność RMSE rzędu $6,32 \cdot 10^{-4}$ dla najgorzej dopasowanego elementu tarczy. W efekcie końcowym osiągnięto ogólną dokładność rozwiązania $\sim 99,93\%$ dla 104 symulacji. Autorzy uważają, że zaprezentowany algorytm przedstawia obiecujące podejście w obszarze dynamicznej analizy niezawodności złożonych konstrukcji.

1.4. PODSUMOWANIE I TEZA PRACY

Biorąc pod uwagę kryterium liczby publikacji, Sztuczne Sieci Neuronowe są zagadnieniem przyszłościowym. Autorzy starają się je implementować wszędzie tam, gdzie złożony charakter zagadnień uniemożliwia bądź znacznie wydłuża analizę zadania postawionego w sposób tradycyjny. W toku analizowania literatury zauważono braki w optymalizacji czasu pracy algorytmów służących do obliczeń zjawisk związanych z wytrzymałością wirników silników turbinowych. Co prawda prace [64, 68-70] poruszają problemy związane z implementacją Sztucznych Sieci Neuronowych do rozwiązywania problemów inżynierskich, jednakże analizy ograniczają się do przepływów albo do zagadnienia kontaktu tarcowego z elementami obciążonymi jednostajnymi siłami. Brakuje natomiast prób wykorzystania SSN do obliczeń wytrzymałościowych elementów maszyn wirnikowych. W praktyce w tego rodzaju analizach stosowane są zwykle algorytmy ograniczające się do iteracyjnego rozwiązywania macierzy sztywności i macierzy sztywności dodatkowej.

Powyższe rozważania i przeanalizowana literatura pozwalają postawić tezę, że **połączenie MES i SSN pozwoli stworzyć efektywny pod względem czasu obliczeń program do analizy wytrzymałościowej elementów zespołów wirnikowych.**

Celem pracy będzie zatem opracowanie algorytmu opartego na SSN do optymalizacji wybranych elementów lotniczego turbinowego silnika odrzutowego z wykorzystaniem schematów uczenia maszynowego, na potrzeby optymalizacji zespołu sprężarkowego.

2. ALGORYTMY SZTUCZNEJ INTELIGENCJI

Sztuczna inteligencja w ujęciu technicznym to wszystkie algorytmy potrafiące korzystać z pewnej formy własnego doświadczenia, w szczególności będące w stanie prognozować stany pośrednie pomiędzy wcześniej napotkanymi. Do algorytmów sztucznej inteligencji zaliczyć można algorytmy uczenia maszynowego naśladujące inteligencję ludzką, a także algorytmy rojowe czy ewolucyjne naśladujące inteligencję zwierzęcą. W dalszej części pracy opisano wykorzystywane przez autora algorytmy zarówno z grupy inteligencji zwierzęcej (algorytm genetyczny z modyfikacjami) i inteligencji ludzkiej (Sztuczne Sieci Neuronowe).

2.1. ALGORYTM GENETYCZNY

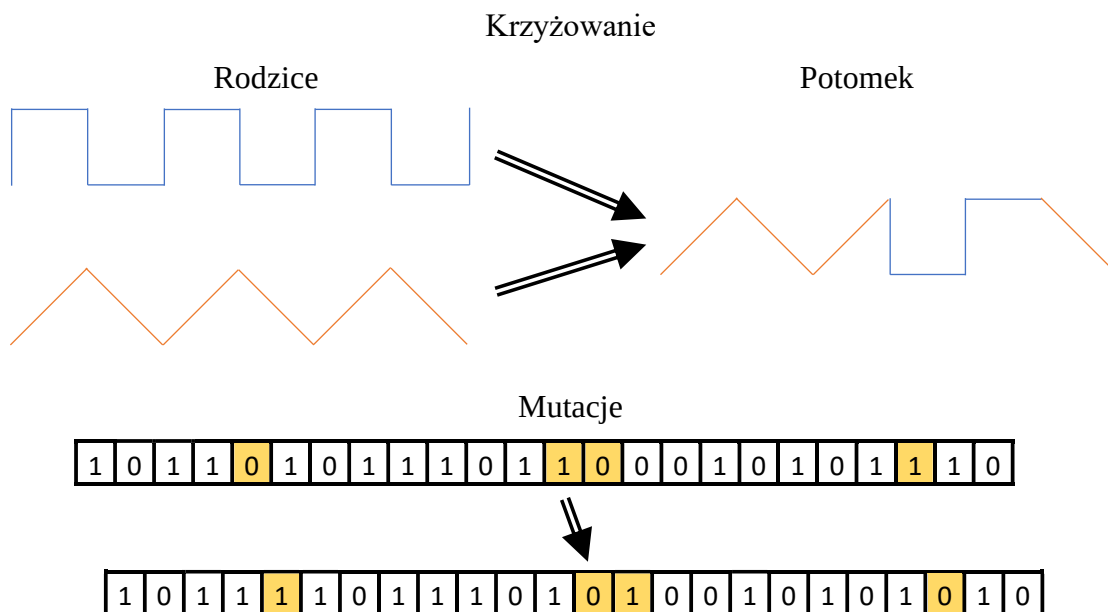
Algorytmy genetyczne, zwane też ewolucyjnymi, to rodzina algorytmów heurystycznych z zasady działania zbliżonych do procesu ewolucji organizmów żywych. Mechanizm działania takiego algorytmu oparty jest na mechanizmach doboru naturalnego, do których należą przede wszystkim krzyżowanie, mutacja oraz dziedziczenie. Algorytm genetyczny został opracowany w 1975 roku przez Johna Hollanda i jego współpracowników. Algorytm ten został później dopracowany i opublikowany w monografii [99].

Algorytmy genetyczne różnią się w założeniach początkowych od większości najczęściej używanych algorytmów. Zmienne decyzyjne zakodowane są w postaci binarnej, a algorytm korzysta bezpośrednio z funkcji celu, nie zaś z ich pochodnych. Poszukiwanie minimum nie rozpoczyna się w jednym punkcie, jak w przypadku np. metod gradientowych, lecz od pewnego zbioru rozwiązań zwanych populacją początkową. Dodatkowo większość procesów wewnątrz programu jest probabilistyczna, a nie deterministyczna jak w przypadku większości najczęściej stosowanych algorytmów do optymalizacji.

Zasada działania algorytmów genetycznych jest stosunkowo prosta. Krzyżowanie, w ogólnym przypadku, polega na wymianie losowej części bitów pomiędzy odpowiednio wyselekcjonowanymi osobnikami (rozwiązaniami). Istnieje na tyle dużo sposobów krzyżowania, że można znaleźć pozycje literaturowe poświęcone tylko tej części algorytmu. Przykładowo w kompendium [100], dla samego krzyżowania, w problemach kodowanych liczbami rzeczywistymi przedstawiono 90 metod.

Przy krzyżowaniu różna też jest ilość potomstwa. Algorytmy $(1+1)$, $(\mu+1)$, $(\mu+\lambda)$, i (μ,λ) [12] opisują, w jaki sposób tworzona jest nowa, tymczasowa populacja, która następnie podlega selekcji i redukcji. Algorytm $(1+1)$ w każdej iteracji z jednego rodzica tworzy jednego potomka. W algorytmie $(\mu+1)$ z μ rodziców tworzony jest również jeden potomek, w $(\mu+\lambda)$ z μ rodziców uzyskuje się λ potomków. Algorytm (μ,λ) działa podobnie jak algorytm $(\mu+\lambda)$ z tą różnicą, że po krzyżowaniu i mutacji wszyscy rodzice z poprzedniego pokolenia giną. Naśladuje on na przykład cykl lęgowy łososi pacyficznych.

Mutacja ogólnie polega na negacji pewnej ilości bitów w wybranych osobnikach. Jednak i ten fragment algorytmu ulegał ewolucji, przykładowo w pozycji [101] przedstawiono 139 wybranych metod mutacji. Ideowy schemat krzyżowania i mutacji przedstawiono na rysunku 2.1.



Rys. 2.1. Krzyżowanie i mutacje.

Wybór osobników do krzyżowania także może być przeprowadzony na wiele sposobów. Najczęściej stosowanymi metodami selekcji są:

- Metoda ruletki [9] – symuluje koło ruletki, podzielone na wczinki, o polu powierzchni proporcjonalnym do wartości funkcji przystosowania każdego z osobników. Algorytm wykorzystuje losowe liczby do wybrania wycinków z prawdopodobieństwem równym ich powierzchni. Wybrane osobniki podlegają krzyżowaniu. Wybór nie musi być losowy. Algorytm może poruszać się wzdłuż obwodu krokami o jednakowej wielkości. Na każdym kroku algorytm przydziela rodzica z sekcji, na którą trafia.
- Selekcja rankingowa [10] – polega na ustawieniu w kolejności, według funkcji przystosowania, wszystkich osobników i dopuszczenie do krzyżowania wybranej liczby najlepszych. Wadą algorytmu jest szybkie ujednolicenie genetyczne populacji.
- selekcja turniejowa [11] – działa podobnie jak w zawodach sportowych rozgrywanych systemem pucharowym. Osobniki podzielone są na grupy, wewnątrz których ma miejsce rywalizacja, a zwycięzcy grup biorą udział w krzyżowaniu. Wybór zwycięzcy w meczu może być realizowany poprzez deterministyczne porównanie funkcji przystosowania lub może być stochastyczny, z prawdopodobieństwem zwycięstwa zależnym od wartości funkcji przystosowania.
- Strategia elitarna – nie jest samodzielną strategią selekcji, a jedynie możliwą modyfikacją. Polega na przeniesieniu najlepszego rozwiązania do kolejnego etapu optymalizacji, zapobiegając utracie optimum, pomimo stochastycznego doboru pozostałych osobników. Stanowi główną różnicę między algorytmami $(\mu+\lambda)$ i (μ,λ) .

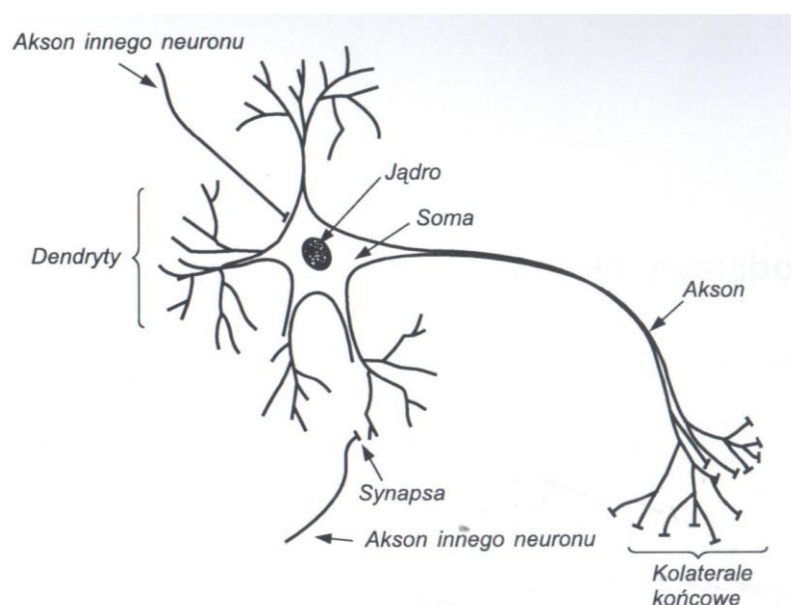
Założenie, że algorytm genetyczny działa na binarnych zmiennych decyzyjnych, nie zawsze musi być spełnione. Genom może być reprezentowany przez liczby zmiennoprzecinkowe, jednak zarówno jego długość, jak i długość alleli musi być znana

i stała od początku pracy algorytmu. Najczęściej w takim przypadku liczba zmiennoprzecinkowa podlega dyskretyzacji.

2.2. SZTUCZNE SIECI NEURONOWE

Sztuczne Sieci Neuronowe są obiektami matematycznymi naśladującymi biologiczne komórki nerwowe (neurony) znajdujące się w mózgu organizmów żywych. Sposób działania komórek nerwowych nie został do końca poznany, dlatego sposób opisu sztucznego neuronu (zwanego perceptronem) jest uproszczony. Pomimo tego faktu może on być użyteczny i wykorzystany z dość dużym powodzeniem.

Komórka nerwowa (rys. 2.2), jak każda żywa komórka, składa się z jądra komórkowego i elementów ciała cytologicznego zwanego somą [20]. Z somy wystają dwie grupy wypustek. Bardzo liczne i cienkie zwane dendrytami i pojedyncza, gruba, rozwidlona na końcu zwana aksonem.



Rys. 2.2. Komórka nerwowa [102].

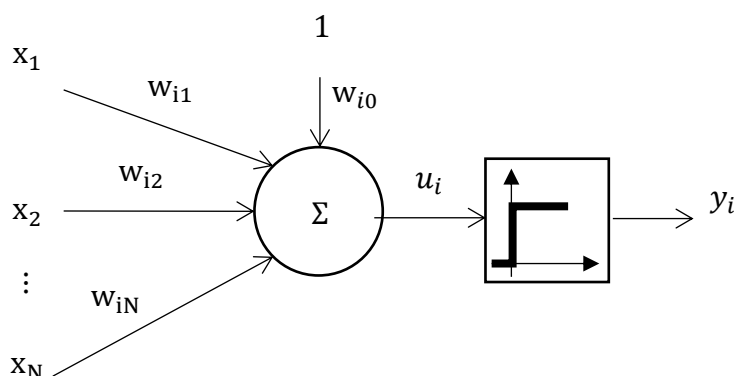
Synapsy są połączeniami pomiędzy kolateralami końcowymi, będącymi końcową częścią aksonu, a dendrytami lub bezpośrednio z somą kolejnych komórek neuronowych.

Jak wcześniej wspomniano, transmisja sygnałów jest bardzo skomplikowanym i nie do końca poznany procesem chemiczno-fizycznym. W dużym uproszczeniu przetwarzanie sygnałów przez neurony opiera się na wydzielaniu neuromediatorów. Neuromediatory są to substancje zmieniające potencjał błony komórkowej. Zmiana ta jest proporcjonalna do ilości wydzielonego neuromediatora i zależy w dużym stopniu od wielkości synaps i możliwości gromadzenia przez nie neuromediatorów. Oznacza to, że każda synapsa może być inna. W modelu matematycznym wejściom do komórki (synapsom) przypisać można odpowiednie współczynniki liczbowe (wagi). Wagi te mogą być dowolnymi liczbami rzeczywistymi, co oznacza, że mogą działać pobudzająco lub tłumić działania poszczególnych komórek.

W przeciwieństwie do techniki cyfrowej sieci neuronowe cechują się odpornością na sygnały zakłócone i uszkodzenia struktury. Dodatkowo działają z dużą szybkością

pomimo powolnego działania pojedynczej komórki. Zostało to osiągnięte przez równoległość przetwarzania informacji.

Sztuczne Sieci Neuronowe znane są badaczom sztucznej inteligencji od ponad 70 lat. W. McCulloch i W. Pitts opracowali pierwszy formalny opis modelu sztucznego neuronu w 1943 roku, zaprezentowany w pracy[103]. Na rysunku 2.3 przedstawiono model komórki nerwowej według McCullocha-Pittsa.



Rys. 2.3. Model komórki nerwowej według McCullocha-Pittsa

Jeżeli całkowity bilans wag w przypadku modelu matematycznego lub zaburzenie równowagi elektrycznej w przypadku komórek żywych nie będzie dostatecznie duże, komórka wraca samorzutnie do stanu początkowego. Mówi się wtedy, że nie został pokonany próg uaktywnienia komórki (opisany funkcją potencjału postsynaptycznego). Jeżeli sygnał wejściowy był dostatecznie duży i próg działania został przekroczony, to ma miejsce lawinowe narastanie sygnału wyjściowego. Komórka w pewnym sensie działa binarnie. Daje maksymalny możliwy impuls (w przypadku dowolnego przekroczenia progu pobudzenia) albo żaden (gdy impuls jest mniejszy od progowego). W Sztucznych Sieciach Neuronowych stwierdzono, że funkcja progu działania wcale nie musi być skokowa. Ogólnie rozumiane najlepsze rezultaty daje sigmoidalna funkcja progowa, co wykazano między innymi w pracy [104].

Model matematyczny neuronu McCullocha-Pittsa [103] opisuje się w za pomocą następującej zależności:

$$y_i = f(u_i) = f\left(\sum_{j=1}^N w_{ij}x_j(t) + w_{i0}\right), \quad (2.1)$$

gdzie u_i – argument funkcji sumacyjnej,
 w_{ij} – wartość wybranego współczynnika wagowego,
 $x_j(t)$ – sygnał wejściowy do neuronu,
 w_{i0} – wartość biasu.

Funkcja $f(u_i)$ jest według McCullocha i Pittsa funkcją skokową Heaviside'a o postaci:

$$f(u_i) = \begin{cases} 1, & \text{dla } u_i > 0 \\ 0, & \text{dla } u_i \leq 0 \end{cases} \quad (2.2)$$

W praktyce częściej stosuje się funkcję sigmoidalną o postaci:

$$f(u_i) = \frac{1}{1 + \exp(-\beta u_i)}, \quad (2.3)$$

gdzie: β – współczynnik stromości funkcji.

W równaniu 2.1 wagi w_{ij} mogą przyjmować dowolne wartości. Przypadek, gdy $w_{ij} > 0$, oznacza połączenie pobudzające, dla $w_{ij} < 0$ połączenie tłumiące, a gdy $w_{ij} = 0$, oznacza brak połączenia synaptycznego. Przyjęcie dyskretnego modelu neuronu uzasadnione jest zjawiskiem refrakcji. Oznacza to, że neuron biologiczny może działać tylko z określoną częstotliwością.

W swojej książce [17] z 1949 roku D. Hebb odkrył, że neuron McCullocha-Pittsa może zapamiętywać informację w swojej strukturze za pomocą wartości wagowych. Dodatkowo zaproponował sposób uczenia sieci za pomocą manipulowania wartościami wag. Swoją myśl uczenia neuronów zaczerpnął z obserwacji przyrody. Zauważył, że waga połączeń między komórkami nerwowymi jest wzmacniana przy ich jednoczesnym uaktywnieniu. Czyli im częściej jest używane dane połączenie, tym staje się mocniejsze. Matematycznie model Hebba przedstawia się w następujący sposób:

$$w_{ij}(k+1) = w_{ij}(k) + \eta y_i(k) y_j(k), \quad (2.4)$$

gdzie: k – w numer cyklu,

η – współczynnik uczenia,

y_i, y_j – wartość wyjściowa z i -tego i j -ego neuronu.

W 1954 roku Minsky [18] zbudował pierwszą sieć neuronową. W 1960 roku powstaje sieć Madaline będąca pierwszym komercyjnym neurokomputerem.

W 1969 roku Minsky i Papert wydają książkę „Perceptrones”. W publikacji tej wykazali oni istotne ograniczenia sieci jednowarstwowej z liniową funkcją aktywacji. Zatrzymało to badania nad sieciami neuronowymi aż do roku 1986. W tym czasie pracowano nad systemami eksperckimi. Jednak kilku pionierów i entuzjastów Sztucznych Sieci Neuronowych kontynuowało swoją pracę, między innymi Bryson, Ho, Werbos, Parker, Rumerumelhart, Linnainmaa, którzy w latach 1970-1976 równolegle opracowali metodę uczenia sieci opartą na wstecznej propagacji błędu. Do najważniejszych wydarzeń zaliczyć można również zaproponowanie przez Hopfielda w 1982 roku pierwszej sieci ze sprzężeniami zwrotnymi. Znaczenie metody wstecznej propagacji błędu przy uczeniu sieci wielowarstwowych podkreślili Rumelhart i Hinton w swoim artykule [25] opublikowanym wraz ze współpracownikami w 1984 roku. Rozpoczęło to na dobre renesans sieci neuronowych zapoczątkowany przez Hopfielda.

Wśród Polaków wymienić należy pioniera Sztucznych Sieci Neuronowych profesora Ryszarda Tadeusiewicza, który między innymi swoją monografią [24] z 1993 roku przybliżył wielu osobom zagadnienia związane ze sztuczną inteligencją i sieciami neuronowymi. Opisał w niej rys historyczny i kierunki rozwoju sieci neuronowych. Opisał także zasady działania i omówił równania wielu typów sieci neuronowych. Następnie przedstawił sposoby ich uczenia i przykłady konkretnych zastosowań. Jego monografia [24] jest swojego rodzaju wykładnią dla wszystkich polskich użytkowników Sztucznych Sieci Neuronowych, o czym może świadczyć liczba cytowań tejże pozycji.

Sztuczne sieci mogą być zorganizowane na wiele różnych sposobów. Przede wszystkim SSN podzielić można na:

- **sieci jednokierunkowe** – gdzie występuje tylko jeden kierunek przepływu informacji. Szczególnym przypadkiem sieci jednokierunkowej jest sieć wielowarstwowa, powstająca z odpowiedniego połączenia ze sobą kolejnych sieci jednowarstwowych.
- **sieci rekurencyjne** – ze sprzężeniem zwrotnym. Przykładami takich sieci mogą być sieć Hopfielda albo sieci uczące się przez współzawodnictwo (Kohonena).

Do każdego zadania należy dobrać odpowiedni model sieci. Przykładowo sieci Hopfielda stosowane są głównie do modelowania pamięci skojarzeniowej, na przykład do rozpoznawania mowy, sieć Kohonena do grupowania przypadków, a nawet do rozpinania siatki 3D wokół skanowanych obiektów. Natomiast sieci jednokierunkowe, będące uniwersalnymi klasyfikatorami, stosowane są między innymi do rozpoznawania wzorców, na przykład do sprawdzania bagażu na lotniskach.

2.3. METODY UCZENIA SZTUCZNYCH SIECI NEURONOWYCH

W zależności od rodzaju i architektury sieci używa się różnych metod ich uczenia. W poniższym podrozdziale przedstawiono kilka najważniejszych metod uczenia Sztucznych Sieci Neuronowych w oparciu o prace [20-25, 73, 79, 81-83].

Metody uczenia SSN można podzielić na dwie grupy: metody uczenia z nauczycielem (pod nadzorem) lub bez nauczyciela (bez nadzoru). Nauczyciel „mówi” sieci neuronowej, że „popelnia błąd” i jak duży on jest, co pozwala sieci dążyć do rozwiązań żądanych. Najciekawsze jest jednak to, że w przypadku, gdy sieć uczy się bez nadzoru, potrafi dokonywać prawidłowej klasyfikacji przypadków na grupy (oczywiście nie nazywając ich w żaden sposób). Może odkryć nowe grupy, o których osoba wykorzystująca sieci mogła nie wiedzieć, że istnieją.

ALGORYTM DELTA [25]

Algorytm DELTA to pierwszy algorytm służący do uczenia jednowarstwowych Sztucznych Sieci Neuronowych. Algorytm ten zakłada, że dla każdego wektora sygnałów wejściowych przypisany jest sygnał wyjściowy. Na podstawie sygnału wyjściowego z sieci możliwe jest obliczenie różnicy między sygnałem rzeczywistym i oczekiwanym. Umożliwia to takie skorygowanie wektora wag sieci, aby zminimalizować różnicę między sygnałem rzeczywistym a oczekiwanym. Wartość wektora wag w kroku $j + 1$ przedstawia zależność (2.5).

$$W^{(j+1)} = W^{(j)} + \eta^{(j)} \delta^{(j)} X^{(j)}, \quad (2.5)$$

gdzie: W – wektor wag,
 η – współczynnik uczenia,
 δ – sygnał błędu,
 X – wektor wejściowy.

Algorytm ten po niewielkich modyfikacjach można zastosować zarówno do sieci liniowej, jak i nieliniowej. Jednak może być używany tylko do sieci jednowarstwowych lub takich, gdzie można zinterpretować w odpowiedni sposób błąd każdej z warstw sieci wielowarstwowych, czego w praktyce raczej się nie stosuje.

WSTECZNA PROPAGACJA BŁĘDU W SIECIACH WIELOWARSTWOWYCH [105]

Podstawową metodą uczenia pod nadzorem wielowarstwowych, jednokierunkowych Sztucznych Sieci Neuronowych jest opracowany przez Linnainma w 1970 roku algorytm [105] oparty o wsteczną propagację błędów, opublikowany później po angielsku w 1976 roku [106].

Algorytm wstecznej propagacji błędów opiera się na założeniu, że mając wyznaczony błąd ($\delta_m^{(j)}$) występujący podczas realizacji j -tego kroku procesu uczenia, w neuronie o numerze m można przesyłać błąd wstecz do wszystkich tych neuronów, których sygnały stanowiły wejścia dla m -tego neuronu (stąd nazwa algorytmu). Wychodząc z tego założenia, do obliczenia wag w $j + 1$ kroku uczenia korzystać można z algorytmu gradientowego lub jego udoskonalonych wersji, szeroko opisywanych w literaturze [20-24].

ALGORYTM GRADIENTOWY [21]

Kolejnym przykładem podstawowego algorytmu uczącego jest algorytm gradientowy, podobny do algorytmu gradientowego stosowanego w optymalizacji funkcji wielu zmiennych. W ostatnich latach podlegał on modyfikacji w celu poprawienia wydajności.

W rozpatrywanym przypadku nie jest znana postać funkcji celu, co pociąga za sobą konieczność poszukiwania kierunku minimum funkcji błędów $Q(W)$. W tym celu okolice punktu początkowego $W + p$ rozwija się w szereg Taylora z pominięciem małych wyższych rzędów. W praktyce wykorzystuje się dwie pierwsze składowe szeregu Taylora, które na podstawie [21] przyjmą postać:

$$Q(W + p) = Q(W)[g(w)]^T \cdot p + \frac{1}{2}p^T H(W) \cdot p + O(h^3). \quad (2.6)$$

Wektor gradientu opisuje się za pomocą równania (2.7), hesjan definiuje się jak w równaniu (2.8), natomiast człon $O(h^3)$ jest błędem obciążenia szeregu, który pomija się jako małą wyższego rzędu.

$$g(W) = \nabla Q = \left[\frac{\partial Q}{\partial W_1}, \dots, \frac{\partial Q}{\partial W_n} \right]^T \quad (2.7)$$

$$H(W) = \begin{bmatrix} \frac{\partial^2 Q}{\partial W_1 \partial W_1} & \dots & \frac{\partial^2 Q}{\partial W_n \partial W_1} \\ \frac{\partial^2 Q}{\partial W_1 \partial W_n} & \dots & \frac{\partial^2 Q}{\partial W_n \partial W_n} \end{bmatrix} \quad (2.8)$$

Punkt $W = W^{(j)}$ (j oznacza j -ty cykl) jest punktem optymalnym funkcji błędu, gdy spełnione są następujące warunki: $G(W^{(j)}) = 0$ i hesjan funkcji $H(W^{(j)})$ jest dodatnio określony. Spełnienie tych warunków gwarantuje znalezienie minimum funkcji błędu, oczywiście z odpowiednią dokładnością. Kierunki poszukiwań minimum w metodzie gradientowej wybiera się według poniższej zależności:

$$p^{(0)} = -\nabla Q(W^{(0)}), \quad (2.9)$$

$$p^{(i+1)} = -\nabla Q(W^{(i+1)}) + \beta^{(i+1)} p^{(i)}, \quad (2.10)$$

gdzie p to kierunek poszukiwań, a β wyraża się równaniem:

$$\beta^{(i+1)} = \frac{\nabla Q(W^{(i+1)})^T \nabla Q(W^{(i+1)})}{\nabla Q(W^{(i)})^T \nabla Q(W^{(i)})}. \quad (2.11)$$

Natomiast wektor wagowy zmieniany jest według poniższej zależności:

$$W^{(j+1)} = W^{(j)} + \eta^{(j)} p^{(j)}. \quad (2.12)$$

By spełnić warunek poprawy wartości funkcji błędu, proces poszukiwań minimum trwa do spełnienia warunku stopu, którym najczęściej jest niewielka zmiana wartości funkcji w dwóch kolejnych krokach.

ALGORYTM NAJWIĘKSZEGO SPADKU [21]

Istnieją także zmodyfikowane metody obliczeń oparte na metodzie gradientowej. Pierwszą modyfikacją jest metoda największego spadku. Polega ona na rozwinięciu funkcji celu w szereg Taylora przy ograniczeniu się tylko do pierwszego członu.

$$Q(W^{(j)} + \pi^{(j)}) = Q(W^{(j)}) [g(W^{(j)})]^T \cdot \pi^{(j)} + O(h^2), \quad (2.13)$$

gdzie: $\pi^{(j)}$ – kierunek poszukiwań.

Aby spełnić warunek poprawy wartości funkcji wraz z kolejnymi iteracjami, wystarczy spełnić warunek:

$$p^{(j)} = -g(W^{(j)}). \quad (2.14)$$

Największą wadą algorytmu największego spadku jest jego mała efektywność.

ALGORYTM RPROP [107]

Jest to opracowany przez Riedmiller Brauna w 1992 roku [107] algorytm uczenia SSN. Jego cechą charakterystyczną to sposób określenia kierunku poszukiwań. Po przeanalizowaniu przez sieć całego zbioru uczącego następuje określenie gradientu, a do dalszej analizy wykorzystuje się jedynie jego znak. Jeżeli znaki kolejnych gradientów różnią się, wielkość kroku jest połowiona. Natomiast jeżeli są zgodne, współczynniki uczenia są obliczane dla każdej wartości wagowej (składowej gradientu).

$$W^{(j+1)} = W^{(j)} - \eta^{(j)} \operatorname{sgn} \left(\frac{\partial E^{(j)^2}}{\partial W^{(j)}} \right), \quad (2.15)$$

gdzie: E^2 – sumaryczny błąd kwadratowy.

Macierz współczynników uczenia $\eta^{(j)}$ wyrażą się wzorem:

$$\eta^{(j)} = \begin{cases} \min\{a\eta^{(j-1)}, \eta_{max}\} & \text{dla } \frac{\partial E^{(j)^2}}{\partial W^{(j)}} \frac{\partial E^{(j-1)^2}}{\partial W^{(j-1)}} > 0 \\ \max\{b\eta^{(j-1)}, \eta_{min}\} & \text{dla } \frac{\partial E^{(j)^2}}{\partial W^{(j)}} \frac{\partial E^{(j-1)^2}}{\partial W^{(j-1)}} < 0 \\ \eta^{(j-1)} & \text{w innym przypadku} \end{cases}, \quad (2.16)$$

gdzie $a = 1,2$; $b = 0,5$; $\eta_{min} = 10^{-6}$; $\eta_{max} = 50$.

METODA MOMENTUM [21]

Metoda „momentum” jest modyfikacją mniej efektywnej metody gradientowej. Opiera się ona na modyfikacji równania (2.12) poprzez dodatkowy człon zawierający momentum. Po przekształceniu równanie z członem momentum przyjmie postać:

$$W^{(j+1)} = W^{(j)} + \Delta W^{(j)}, \quad (2.17)$$

gdzie:

$$\Delta W^{(j)} = \eta^{(j)} \cdot p^{(j)} + \alpha(W^{(j)} - W^{(j-1)}), \quad (2.18)$$

α – współczynnik momentum mieszczący się w przedziale $\langle 0,1 \rangle$.

Składnik drugi jest niezależny od wartości gradientu, a zależy tylko od ostatnich wartości wag. Wartość współczynnika α ma decydujące znaczenie i jego dobór jest funkcją zadania i doświadczenia użytkownika.

METODA GRADIENTÓW SPRZĘŻONYCH [108]

Metoda gradientów sprzężonych [108] jest kolejną modyfikacją metody gradientowej. W metodzie gradientowej kierunek poszukiwania jest zawsze prostopadły do poziomicy funkcji celu przechodzącej przez punkt x (dla przypadku dwuwymiarowego). Jeżeli styczna do poziomicy ma kierunek zbliżony do optymalnego kierunku poszukiwania minimum, obrany przez algorytm gradientowy kierunek będzie jednym z możliwie najgorszych. Algorytm gradientów sprzężonych został opracowany w celu rozwiązania wyżej opisanego problemu określenia kierunku poszukiwania.

W metodzie gradientów sprzężonych rezygnuje się z bezpośredniej informacji o hesjanie. Zamiast tego nowy kierunek poszukiwań określa się tak, aby był ortogonalny i sprzężony ze wszystkimi dotychczas obranymi kierunkami poszukiwań minimum funkcji błędu. Zbiór wektorów p_i jest wzajemnie sprzężony względem macierzy G , gdy spełniona jest zależność:

$$p_i^T G p_j = 0 \text{ dla } i \neq j. \quad (2.19)$$

Wektor spełniający zależność (2.19) ma postać:

$$p_k = -g_k + \sum_{j=0}^{k-1} \beta_{kj} p_j, \quad (2.20)$$

gdzie $g_k = g(W_k)$ i jest aktualną wartością gradientu.

Natomiast sumowanie dotyczy wszystkich poprzednich kierunków. Korzystając z warunku ortogonalności i sprzężenia, powyższe równanie można uprościć do postaci:

$$p_k = -g_k + \beta_{k-1}p_{k-1}. \quad (2.21)$$

Oznacza to, że obecny kierunek poszukiwania minimum funkcji celu zależy od antygradientu funkcji celu w bieżącym punkcie i od poprzedniego kierunku poszukiwań pomnożonego przez współczynnik sprzężenia β . Współczynnik ten określić można na wiele różnych sposobów. Najczęściej wykorzystuje się do tego jedno z dwóch poniższych równań:

$$\beta_{k-1} = \frac{g_k^T(g_k - g_{k-1})}{g_{k-1}^T g_{k-1}}, \quad (2.22)$$

$$\beta_{k-1} = \frac{g_k^T(g_k - g_{k-1})}{-p_{k-1}^T g_{k-1}}. \quad (2.23)$$

Wada tego algorytmu to nawarstwianie się błędów zaokrąglenia. Aby tego uniknąć, po wykonaniu n iteracji (dla n = liczba zmiennych) algorytm powtarza się, określając pierwszy z kierunków zgodny z antygradientem funkcji (tak jak w metodzie największego spadku). Z uwagi na dość małą potrzebną moc obliczeniową i dość szybkie działanie metoda ta jest jedną z najczęściej wykorzystywanych metod optymalizacji, zwłaszcza w przypadku ich dużej złożoności.

METODA GRADIENTÓW SPRĘŻONYCH Z REGULACJĄ [109]

W celu skrócenia czasu obliczeń stosuje się wiele różnych modyfikacji algorytmu gradientów sprzężonych. Jednym z lepszych jest zaproponowany przez Meillera algorytm skalowanego gradientu sprzężonego [109]. Metoda skalowanego gradientu sprzężonego (*Scaled Conjugate Gradient Method* – SCG) polega na połączeniu jednoczesnego wyznaczania sprzężonego kierunku poszukiwań p i optymalnego kroku w tym kierunku. Podobnie jak w metodzie gradientów sprzężonych wychodzi się tutaj z rozwinięcia funkcji celu w kwadratowe równania Taylora z pominięciem małych wyższego rzędu.

$$E(W_k + \eta p_k) \approx E(W_k) + \eta g_k^T p_k + \frac{1}{2} \eta^2 p_k^T G_k p_k, \quad (2.24)$$

gdzie G_k jest przybliżoną wartością hesjanu w k -tej iteracji. Korzystając z minimum funkcji celu, w kierunku p_k otrzymamy zależność:

$$\eta = -\frac{p_k^T g_k}{p_k^T G_k p_k} = \frac{\mu_k}{\delta_k}. \quad (2.25)$$

W przeciwieństwie do określenia $\delta_k = p_k^T G_k p_k$, określenie μ_k jest zadaniem łatwym i szybkim pod względem zużycia mocy obliczeniowej. Dlatego też w praktyce korzysta się z wartości gradientu w punkcie rozwiązania $g(W_k)$ i w punkcie sąsiednim, oddalonym o wartość względną $\sigma_k \in [0; 1]$, co można zapisać w postaci:

$$s_k = G_k p_k \approx \frac{g(W_k + \sigma_k p_k) - g(W_k)}{\sigma_k}. \quad (2.26)$$

Wzór (2.26) jest rozwiązaniem wynikającym z kwadratowego rozwinięcia w równanie Taylora. W praktyce użyteczność tego równania jest niewielka, ponieważ prowadzi ono do minimum funkcji tylko wówczas, gdy hesjan jest dodatnio określony, co w przypadku SSN praktycznie nigdy nie ma miejsca. Aby temu zaradzić, stosuje się regularyzację hesjanu według zależności:

$$s_k = \frac{g(W_k + \sigma_k p_k) - g(W_k)}{\sigma_k} + \lambda_k p_k. \quad (2.27)$$

Parametr regulacyjny λ_k dobiera się tak, aby spełnić zależność $\delta_k = p_k^T s_k > 0$. Jeżeli po wprowadzeniu poprawki δ_k jest nadal ujemna, należy dalej zwiększać wartość współczynnika λ_k . Jeżeli założy się, że nowa wartość λ_{rk} spełnia warunek dodatniości hesjanu, a odpowiadającą jej wartość s_k zastąpimy s_{rk} , otrzymamy:

$$s_{rk} = s_k + (\lambda_{rk} - \lambda_k) p_k \quad (2.28)$$

Dla nowej wartości λ_{rk} spełniona jest nierówność:

$$\delta_{rk} = p_k^T G_k p_k + p_k^T (\lambda_{rk} - \lambda_k) p_k = \delta_k + (\lambda_{rk} - \lambda_k) |p_k|^2 > 0. \quad (2.29)$$

Z tej zależności wyznaczyć można λ_{rk} :

$$\lambda_{rk} > \lambda_k - \frac{\delta_k}{|p_k|^2}. \quad (2.30)$$

Autor algorytmu [109] w tym miejscu wstępnie zaproponował wartość opisaną równaniem (2.31)

$$\lambda_{rk} = 2 \left(\lambda_k - \frac{\delta_k}{|p_k|^2} \right). \quad (2.31)$$

Z tego wynika, że ostateczny wzór na optymalny krok uczenia η spełniający warunek dodatniości hesjanu wyraża się równaniem:

$$\eta = \frac{\mu_k}{\delta_k} = \frac{\mu_k}{-p_k^T s_k + \lambda_k |p_k|^2}. \quad (2.32)$$

Z powyższej zależności wynika odwrotna proporcjonalność kroku uczenia η i współczynnika λ_k . Aby uzyskać maksymalizację kroku uczenia i tym samym skrócić czas działania algorytmu, wprowadza się współczynnik Δ_k definiowany w następujący sposób:

$$\Delta_k = \frac{E(W_k) - E(W_k + \eta p_k)}{E(W_k) - E_a(\eta p_k)}, \quad (2.33)$$

gdzie: $E_a(\eta p_k)$ – to aproksymowana wartość funkcji celu. Uwzględniając wcześniejsze rozważania zapisać można:

$$\Delta_k = \frac{E(W_k) - E(W_k + \eta p_k)}{\frac{\mu_{rk}^2}{2\delta_{rk}}}. \quad (2.34)$$

Powyższe równanie określa, jak bardzo zniekształcona została funkcja celu na kierunku p_k . Meiller w pracy [109] zaproponował:

$$\left\{ \begin{array}{l} \text{dla } \Delta_k \geq \frac{3}{4}, \lambda_k \leftarrow \frac{1}{4} \lambda_k; \\ \text{dla } \Delta_k < \frac{1}{4}, \lambda_k \leftarrow \lambda_k + \frac{\delta_k(1 - \Delta_k)}{|p_k|^2}; \\ \text{w przeciwnym wypadku } \lambda_k \text{ pozostaje bez zmian} \end{array} \right. \quad (2.35)$$

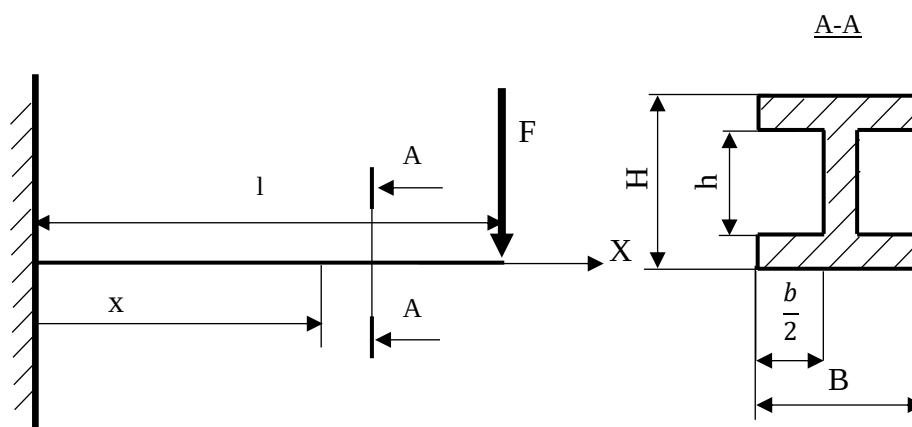
W pracy do uczenia Sztucznych Sieci Neuronowych wykorzystano metodę gradientów sprzężonych, w celu rozwiązania problemu tarczy prostej, a także tarczy profilowanej (rozdział 2.4).

2.4. BADANIA TESTOWE SSN

W podrozdziale przedstawiono algorytmy zawierające Sztuczne Sieci Neuronowe do obliczeń w obszarze inżynierii mechanicznej. Podjęto próby udowodnienia, że zastosowanie tego typu algorytmów jest możliwe oraz że zaproponowane rozwiązania są poprawne fizycznie. Wszystkie obliczenia wykonano w programie MATLAB *Neural Network Tools*. Jest to narzędzie zawierające funkcje, wewnątrz których uwikłane są równania opisujące zarówno działanie jak i algorytmy uczenia Sztucznych Sieci Neuronowych.

BELKA WYSIĘGNIKOWA

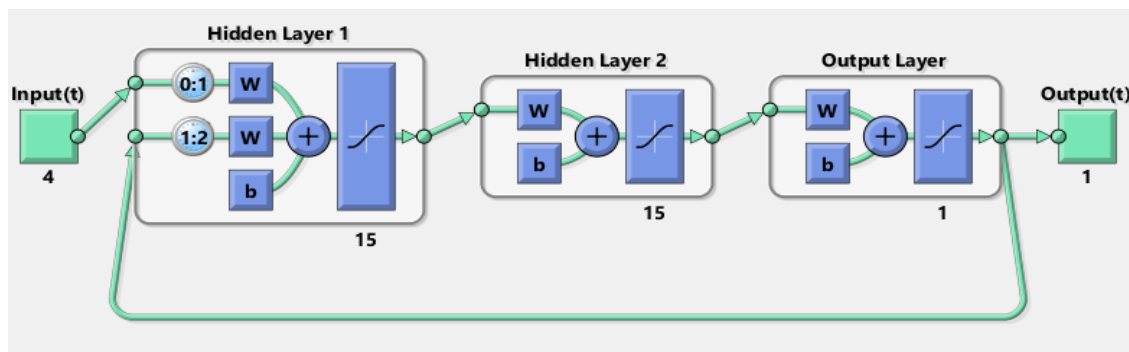
Pierwszym zaproponowanym algorytmem jest sieć neuronowa zrealizowana w podprogramie programu MATLAB o nazwie Neural Network Tools. Do testowania SSN wybrano belkę wysięgnikową przedstawioną na rysunku 2.4, jako przypadek możliwy do walidacji analitycznej.



Rys. 2.4. Dwuteowa belka wysięgnikowa obciążona siłą skupioną, gdzie: F – wektor siły, l – całkowita długość belki, x – współrzędne przekroju obliczeniowego wzdłuż osi belki, H – wysokość dźwigara, h – wysokość ścianki dźwigara, B – szerokość pasa dźwigara, b – szerokość pasa bez ścianki, X – oś odciętych

Kolejnym krokiem było opracowanie Sztucznej Sieci Neuronowej z wykorzystaniem programu MATLAB z pakietem Neural Network Toolbox. Narzędzie pozwala tworzyć prawie dowolną sieć neuronową i jej połączenia, a także uczyć ją za pomocą zbioru

algorytmów uczących. Zastosowano sieć o architekturze przedstawionej na rys. 2.5. Jest to sieć neuronowa typu NARX (ang. *Nonlinear Autoregressive with External (Exogenous) Input*) z czterema wejściami, dwiema warstwami ukrytymi po 15 neuronów każda i jednej warstwie wyjściowej z jednym neuronem. Jako funkcję aktywacyjną wybrano funkcję sigmoidalną dla wszystkich neuronów. Zastosowano także biasy dla wszystkich neuronów.

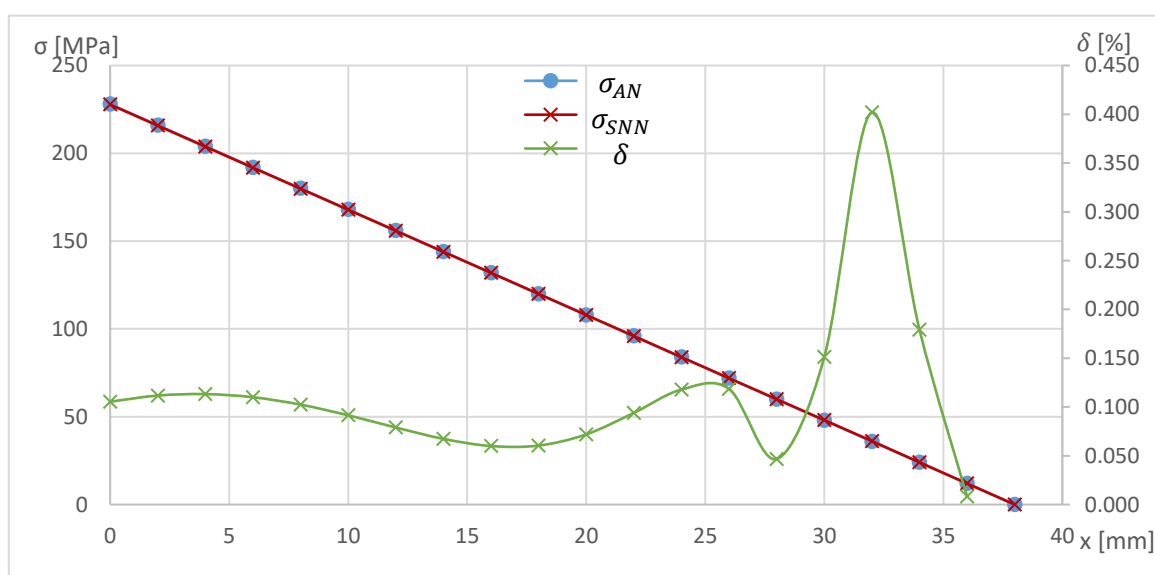


Rys. 2.5. Schemat Sztucznej Sieci Neuronowej [110]

Cztery parametry wejściowe to: długość belki l , bieżąca współrzędna przekroju x i siła obciążająca belkę F oraz wskaźnik wytrzymałości na zginanie W_y , który był obliczony analitycznie. Wylosowano 500 przykładowych kombinacji tych parametrów. Długość belki l losowano z przedziału $(0, 100)$ [mm], siłę F z przedziału $(0, 100)$ [N] i współrzędna przekroju x z przedziału $(0, l)$. Były to dane wejściowe do uczenia sieci.

Korzystając z zależności analitycznych, obliczono naprężenia dla każdego z wylosowanych przykładów. W ten sposób otrzymano dane wyjściowe potrzebne do uczenia sieci. Po zakończeniu procesu uczenia metodą GDM (ang. *Gradient Descent with Momentum weight and bias learning function*) wybrano losowo jeden nowy przypadek belki o długości $l = 38$ mm obciążonej siłą $F = 12$ N, której wskaźnik wytrzymałości na zginanie był równy $W_y = 2$ mm³.

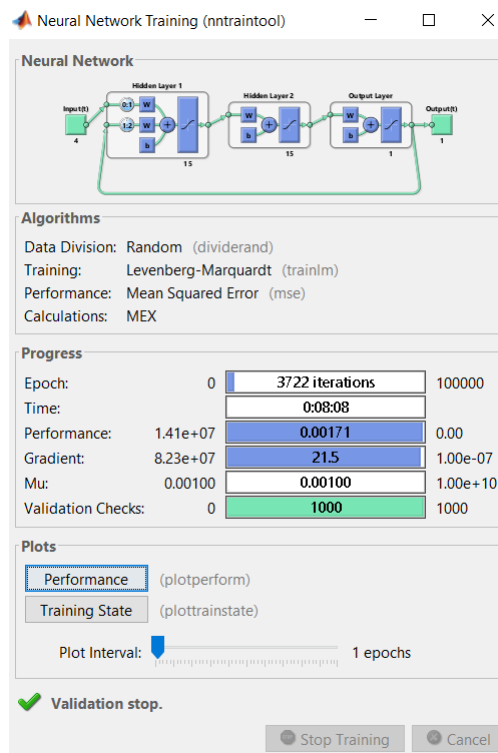
Wykonano obliczenia naprężeń dwoma metodami: analitycznie i za pomocą nauczonej Sztucznej Sieci Neuronowej. Wyniki zestawiono na rys. 2.6.



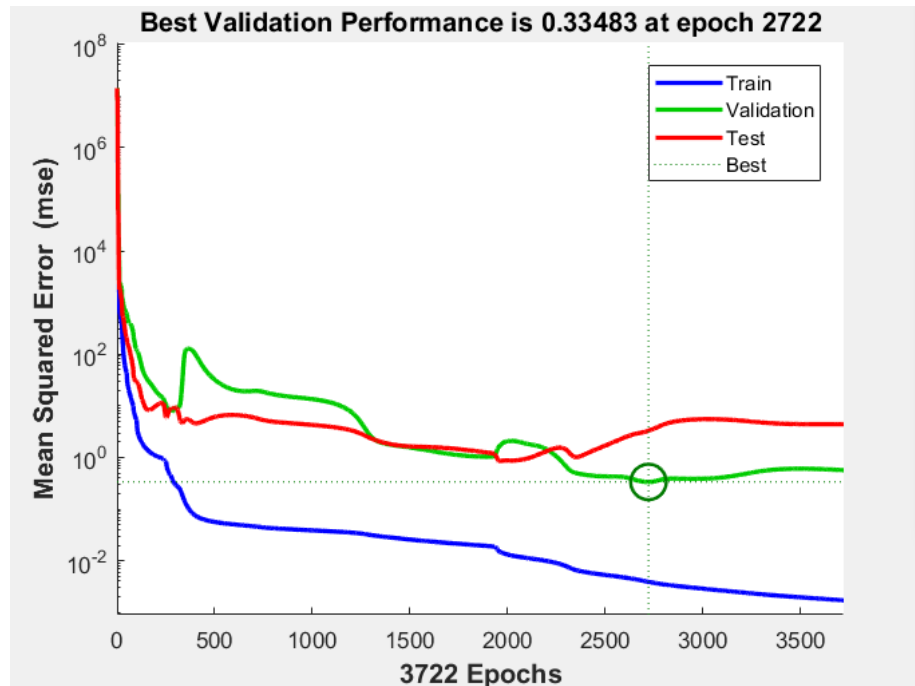
Rys. 2.6. Naprężenia obliczone za pomocą Sztucznej Sieci Neuronowej (σ_{SNN}) i analitycznie (σ_{ANAL}). Na osi pomocniczej zaznaczono zakres możliwych wartości błędu względnego (δ)

Jak można zauważyć na rys. 2.6, błąd względny obliczeń nie przekracza 0,15% z wyłączeniem swobodnego końca belki. Naprężenia na swobodnym końcu belki są niewielkie (około 36 MPa), a sieć neuronowa popełnia błąd względny (około 0,4%), przy błędzie względnym nie większy niż 0,15 MPa. Jest to bardzo dobry wynik. Sam przebieg uczenia odbył się według ustawień przedstawionych na Rys. 2.7 i 2.8, gdzie:

- „Data Division – Random” określa losową kolejność wprowadzania danych do SSN;
- „Training – Levenberg-Marquardt” określa algorytm uczenia sieci neuronowej (LM);
- „Performance – Mean Squared Error” funkcją celu uczenia maszynowego była minimalizacja błędu średniokwadratowego;



Rys. 2.7. Przebieg procesu uczenia Sztucznej Sieci Neuronowej



Rys. 2.8. Błąd średniokwadratowy uczenia Sztucznej Sieci Neuronowej

Podczas procesu uczenia wyznaczony błąd średni (rys. 2.8), definiowany równaniem (2.36), w funkcji epok uczących dla danych uczących (linia niebieska), testowych (linia czerwona) i walidacji (linia zielona). Zielonym markerem zaznaczono minimum funkcji błędu MSE, które jest opisane równaniami:

$$MSE(\hat{\theta}) = D^2(\hat{\theta}) + (b(\hat{\theta}))^2, \quad (2.36)$$

gdzie: D^2 – wariancja estymatora $\hat{\theta}$,

$b(\hat{\theta})$ – obciążenie estymatora definiowane równaniem (2.37).

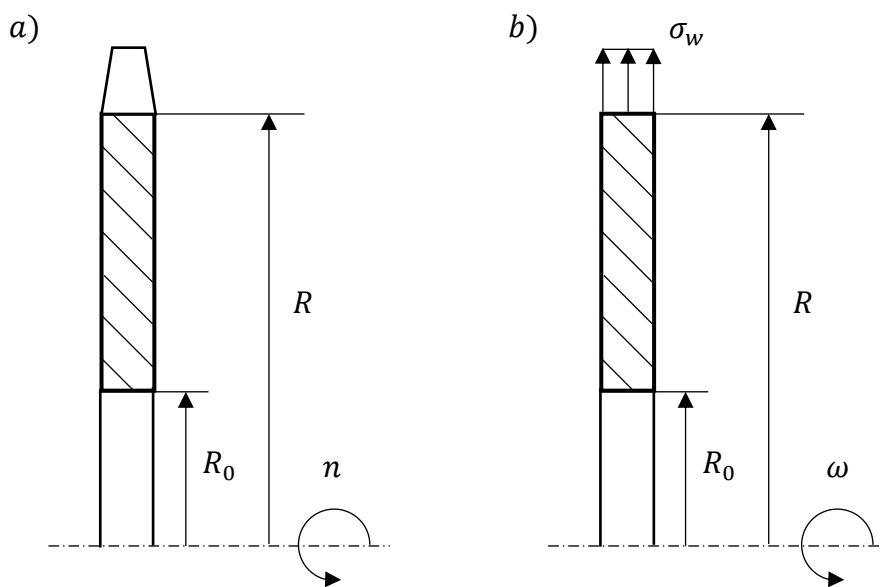
$$b(\hat{\theta}) = E[(\hat{\theta})] - \theta \quad (2.37)$$

Z rysunku 2.8 wynika, że SSN zakończyła proces uczenia z zadowalającą dokładnością po 2722 epokach. Pracowała przez kolejne 1000 epok w celu sprawdzenia, czy nie wpadła w lokalne, nie dość dokładne minimum funkcji błędu, tak jak dla okolic 350 i 1900 epoki uczenia, gdzie nastąpił gwałtowny wzrost błędu. Jest to typowe zjawisko dla procesu uczenia się Sztucznych Sieci Neuronowych i dobrze świadczy o tym procesie. Sieć na początku „nie zrozumiała” zjawiska i nie do końca poprawnie się go nauczyła, a następnie „zmieniła tok rozumowania”, co pozwoliło osiągnąć zadowalające minimum. Sieć w wyniku uczenia osiągnęła wartości dostatecznie dobre ($MSE < 1$) wcześniej niż przy 2722 epoce. Zaprogramowanie procesu uczenia nie było nastawione na jego szybkość, miało jedynie za zadanie sprawdzić możliwość zastosowania SSN w problemach wybranego rodzaju, z odpowiednią dokładnością.

W wyniku przeprowadzonej analizy stwierdzono, że Sztuczne Sieci Neuronowe spełniły postawione zadanie w postaci wyznaczenia naprężeń.

TARCZA SPRĘŻARKI OSIOWEJ

Zastosowano sieci neuronowe do wyznaczenia rozkładu naprężeń w tarczy sprężarki osiowej. Sprawdzono możliwości SSN i jej architekturę na przykładzie tarczy prostej (rys 2.9). Wykorzystany przykład zapewnia szybkość i prostotę generowania danych potrzebnych do uczenia sieci. W obliczeniach wykorzystano własny program napisany w języku MATLAB do wyznaczania naprężeń za pomocą metody elementów skończonych (Załącznik 1 - Tarcza_1D.m). Schemat blokowy całego algorytmu przedstawia rysunek 2.15.



Rys. 2.9. Tarcza prosta; a) przypadek rzeczywisty, b) przypadek uproszczony do obliczeń, gdzie: n – prędkość obrotowa, ω – prędkość kątowna, R_0 – promień otworu centralnego, R – promień zewnętrzny tarczy, σ_w – ciągnienie wieńcowe

Rozpatrywano ważką tarczę prostą z otworem, obciążoną ciągnieniem pochodzącym od masy końcówki tarczy (masy łopatek i części zamkowej tarczy Rys. 2.9 b). Naprężenia promieniowe i obwodowe pochodzące od sił masowych dla tarczy z otworem wyrażają się równaniami [111]:

$$\sigma_r = \frac{3 + \nu}{8} \rho \omega^2 R^2 \left[1 + \left(\frac{R_0}{R} \right)^2 - \left(\frac{R_0}{r} \right)^2 - \left(\frac{r}{R} \right)^2 \right], \quad (2.38)$$

$$\sigma_t = \frac{3 + \nu}{8} \rho \omega^2 R^2 \left[1 + \left(\frac{R_0}{R} \right)^2 + \left(\frac{R_0}{r} \right)^2 - \frac{1 + 3\nu}{3 + \nu} \left(\frac{r}{R} \right)^2 \right], \quad (2.39)$$

gdzie: σ_r – naprężenia promieniowe,

ν – liczba Poissona,

ρ – gęstość materiału,

ω – prędkość kątowna tarczy,

R – promień zewnętrzny tarczy,

R_0 – promień otworu,

r – promień bieżący,

σ_t – naprężenia obwodowe.

Naprężenia promieniowe i obwodowe pochodzące od obciążeń wieńcowych wyrażają się równaniami [111]:

$$\sigma_r = \frac{\sigma_w}{1 - \left(\frac{R_0}{R}\right)^2} \left[1 - \left(\frac{R_0}{r}\right)^2 \right], \quad (2.40)$$

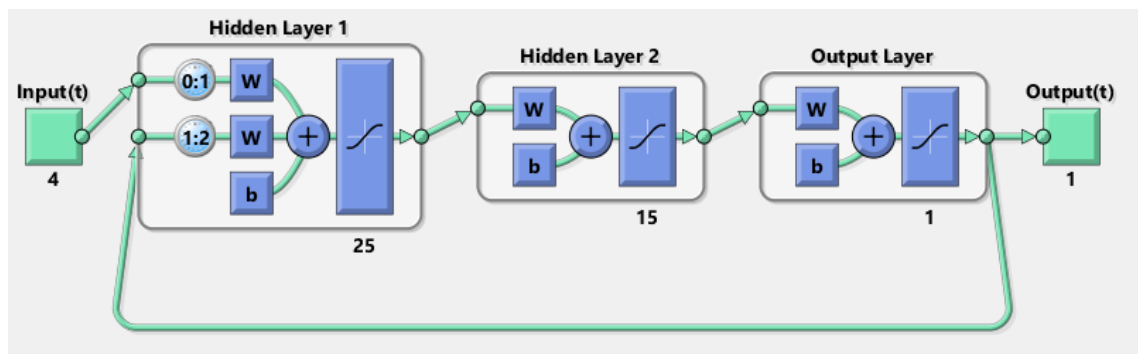
$$\sigma_r = \frac{\sigma_w}{1 - \left(\frac{R_0}{R}\right)^2} \left[1 + \left(\frac{R_0}{r}\right)^2 \right], \quad (2.41)$$

gdzie: σ_w – naprężenia odpowiadające ciągnięciu wieńcowemu.

W przypadku gdy tarcza jest jednocześnie ważka i obciążona ciągnięciem, wypadkowa naprężeń promieniowych i obwodowych może być zastąpiona naprężeniami zredukowanymi:

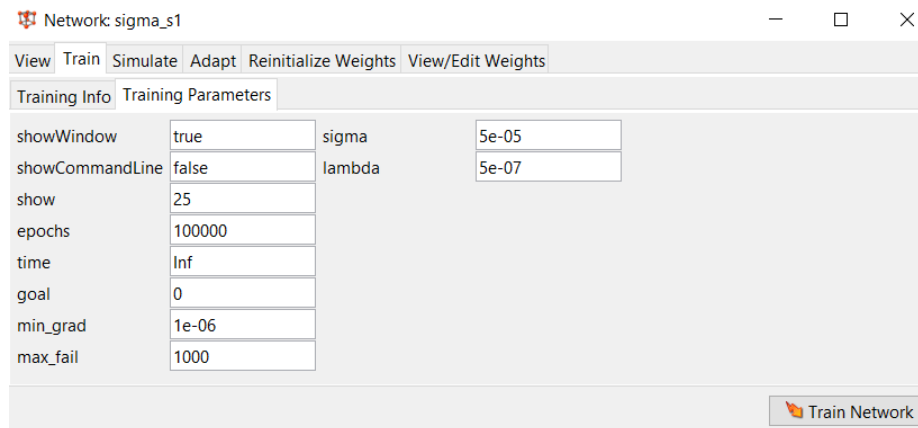
$$\sigma_{red} = \sqrt{\sigma_s^2 + \sigma_t^2 - \sigma_s \sigma_t}. \quad (2.42)$$

Do analizy tego układu konstrukcyjnego zespołu sprężarki osiowej, wykorzystano cztery wielowarstwowe SSN ze wsteczną propagacją błędu, o budowie jak na rys. 2.10. Sieci były uczone metodą „*trainscg*” (*Scaled Conjugate Gradient backpropagation* - metoda gradientów sprzężonych z regulacją, i ze wsteczną propagacją błędu). Sieć uczono do momentu aż osiągnięto 1000 iteracji uczenia bez poprawy wartości wagowych. Każda z sieci liczyła inną składową naprężeń. Cztery zastosowane do nich parametry wejściowe to r (bieżący promień obliczeniowy, R , R_0 i ω , w przypadku obliczeń tarczy ważkiej z zakresów kolejno $\langle 0,1 \rangle$, $\langle 0, \frac{R}{2} \rangle$ i $\langle 0,10^5 \rangle$, natomiast $r = \{R_0, \frac{1}{4}(R - R_0), \frac{1}{2}(R - R_0), \frac{3}{4}(R - R_0), R\}$ lub σ_w (w przypadku obliczeń tarczy obciążonej ciągnięciem z zakresu $\langle 0,10^{10} \rangle$). Ponieważ rozpatrywana tarcza jest prosta, nie ma potrzeby rozpatrywać jej grubości, pod warunkiem obliczenia naprężeń odpowiadających ciągnięciu wieńcowemu σ_w . Dodatkowo za stałe przyjęto dane materiałowe odpowiadające stopowi tytanu Ti-6Al-4V ($\nu = 0,4031$, $\rho = 4,43 \frac{g}{m^3}$) często wykorzystywanemu w zespołach sprężarkowych lotniczych silników turbinowych.



Rys. 2.10. Schemat Sztucznej Sieci Neuronowej do obliczeń tarczy o stałej grubości

Sieci poddano krótkiemu, równoległemu uczeniu, którego parametry przedstawiono na rys. 2.11.



Rys. 2.11. Ustawienia procesu uczenia sieci [110]

Sieć neuronową uczono, wykorzystując 1200 przypadków wyznaczonych analitycznie. Tabela 2.1 zawiera zakresy, z jakich losowane były potrzebne dane wejściowe do sieci. Do jej testowania wylosowano dwa nowe, nieznane sieci przypadki, które przedstawiono w tabeli 2.2. W trakcie uczenia część z 1200 przypadków jest również nadal używana do testowania sieci zgodnie z zastosowanym algorytmem uczenia. Błąd średniokwadratowy walidacji tego procesu uczenia w każdym z przypadków mieścił się pomiędzy 10^5 a 10^6 . Dokładność ta jest dostatecznie dobra na potrzeby obliczeń. W rozpatrywanym przypadku błąd wynosił 0,015%. Maksymalne wartości błędu nie przekraczają 6% nawet dla niewielkich naprężeń.

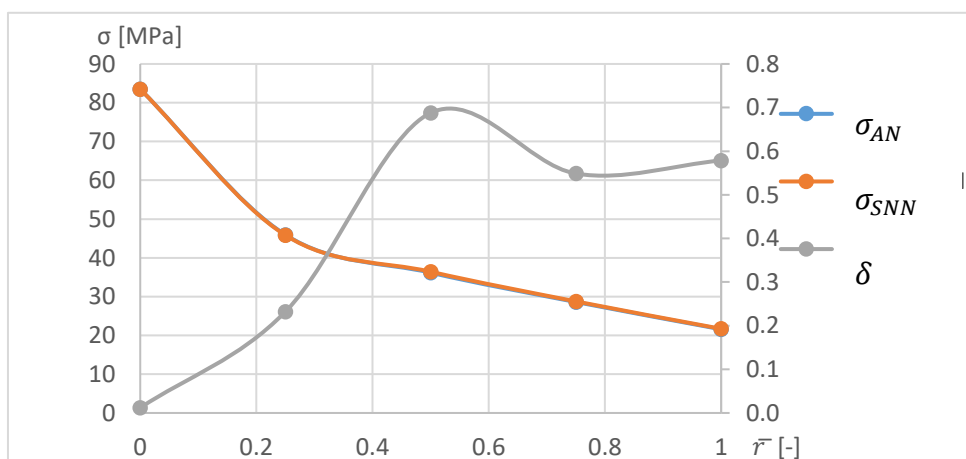
Tabela 2.1. Zakresy danych wejściowych do uczenia SSN

Parametr	Oznaczenie [jednostka]	Dopuszczalny przedział
Prędkość kątowna	ω [rad/s]	$(0; 10^4)$
Ciągnienie	σ_w [kPa]	$(0; 10^4)$
Promień wewnętrzny tarczy	R_0 [m]	$(0; 1)$
Promień zewnętrzny tarczy	R [m]	$(R_0; 1)$
Bieżący promień obliczeniowy	r [m]	$\frac{R - R_0}{4}k + R_0$ dla $k = \{0,1,2,3,4\}$

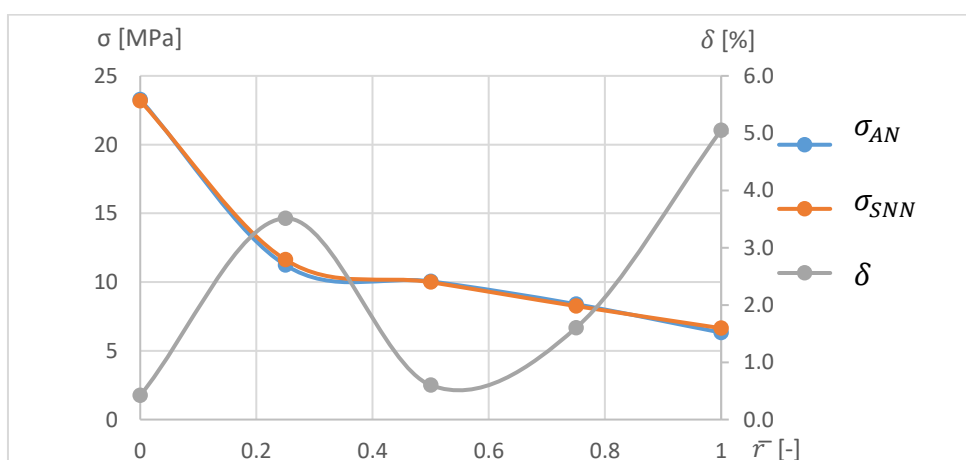
Wyniki naprężeń i błędy nauczonej sieci neuronowej przedstawiono na rysunkach 2.12 i 2.13, dla obydwu przypadków zestawionych w tabeli 2.2. Dane zostały dobrane losowo.

Tabela 2.2. Dane użyte do sprawdzenia sieci.

Nr przykładu		1	2
Prędkość obrotowa	ω [rad/s]	6313	7708
Napężenia wieńcowe	σ_w [kPa]	7171	4607
Promień bieżący	$r(1)$ [m]	0,1688	0,0200
	$r(2)$ [m]	0,2942	0,0775
	$r(3)$ [m]	0,4195	0,1350
	$r(4)$ [m]	0,5440	0,1925
	$r(5)$ [m]	0,6701	0,2500



Rys. 2.12. Napężenia (σ) i błąd względny (δ) popełniony przez sieć na promieniach bieżących (r) – dla przykładu 1 (tabela 2.2).



Rys. 2.13. Napężenia (σ) i błąd względny (δ) popełniony przez sieć promieni bieżących (r [-]) – dla przykładu 2 (tabela 2.2).

Otrzymane wyniki są poprawne pod względem teoretycznym. W przypadkach rzeczywistych mogą wystąpić większe wartości naprężeń, jednak rozwiązywany problem miał na celu wykazać poprawność pracy SSN. Dla wszystkich przedstawionych przykładów błąd popełniony przez sieć był nie większy niż 5%. Jest to wynik zadowalający. Można osiągnąć lepszą dokładność poprzez zmianę struktury sieci i ponowne, dłuższe uczenie. Celem zadania było udowodnienie, że możliwe jest nauczenie sieci rozwiązywać problem tego typu, więc nie było wymagane nauczenie sieci dokładnej predykcji rozkładu naprężeń.

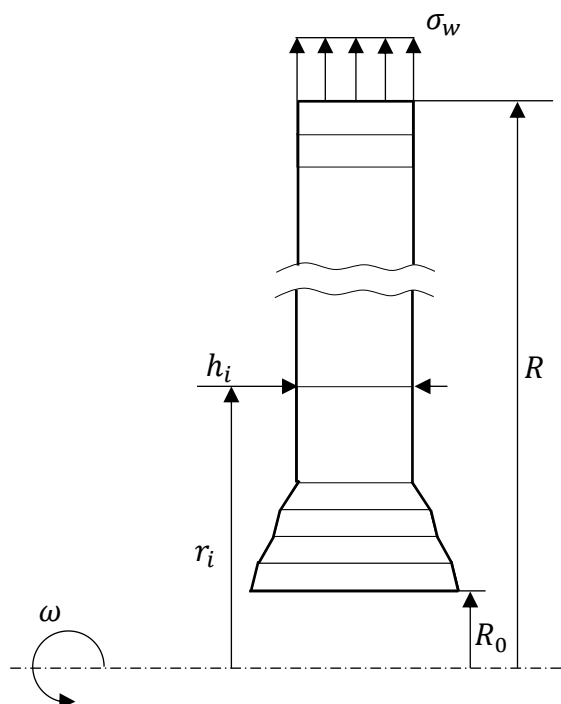
W pracy podjęto także próby przystosowania sieci do wyznaczania rozkładu naprężeń w całej tarczy jednocześnie, niestety okazało się, że otrzymane wyniki są gorsze jakościowo (błąd średniokwadratowy wynosił 10^{16} Pa), a czas potrzebny na uczenie sieci znacznie się wydłużył.

Porównując wyniki dla dwóch podejść (holistycznego i fragmentarycznego) do obliczeń tarczy prostej, wywnioskować można, że sieć działa lepiej w przypadku rozłożenia problemu na kilka zadań powtarzalnych. Oznacza to, że w dalszej pracy warto rozważyć iteracyjne wyznaczanie elementów macierzy sztywności w algorytmie MES za pomocą SSN. Jednak dla porównania wyników i potwierdzenia wniosków można zastosować podejście holistyczne i wyznaczyć całą macierz sztywności w jednej iteracji.

TARCZA PROFILOWANA

Kolejnym etapem pracy badawczej była budowa sieci neuronowej rozwiązującej problem wyznaczenia rozkładu naprężeń w profilowanej, symetrycznej, ważkiej tarczy sprężarki osiowej, obciążonej ciągnięciem.

Faza wstępna obejmowała opisanie i usystematyzowanie problemu oraz tworzenie bazy danych przypadków niezbędnych do uczenia sieci. Założono, że w każdym przypadku tarcza składać się będzie z 18 elementów stożkowych o liniowo zmiennej grubości. Rozpatrywana sieć ma obliczać naprężenia w tarczach wykonanych tylko z jednego rodzaju materiału. Wylosowano 2200 przypadków zgodnych z ograniczeniami ujętymi w tabeli 2.3. W efekcie utworzono program, który dla podanych 24 parametrów wejściowych oblicza naprężenia dla zadanych 18 elementów przedstawionych na rysunku 2.14.



Rys. 2.14. Model tarczy profilowanej z przedstawionym zagęszczeniem siatki obliczeniowej, gdzie: ω – prędkość kątowa, R_0 – Promień otworu centralnego, R – promień zewnętrzny tarczy, σ_w – ciągnięcie wieńcowe, r_i – promieniowa współrzędna i-tego węzła obliczeniowego, h_i - grubość i-tego elementu.

Tabela 2.3. Zakres danych wejściowych do programu MES liczącego naprężenia

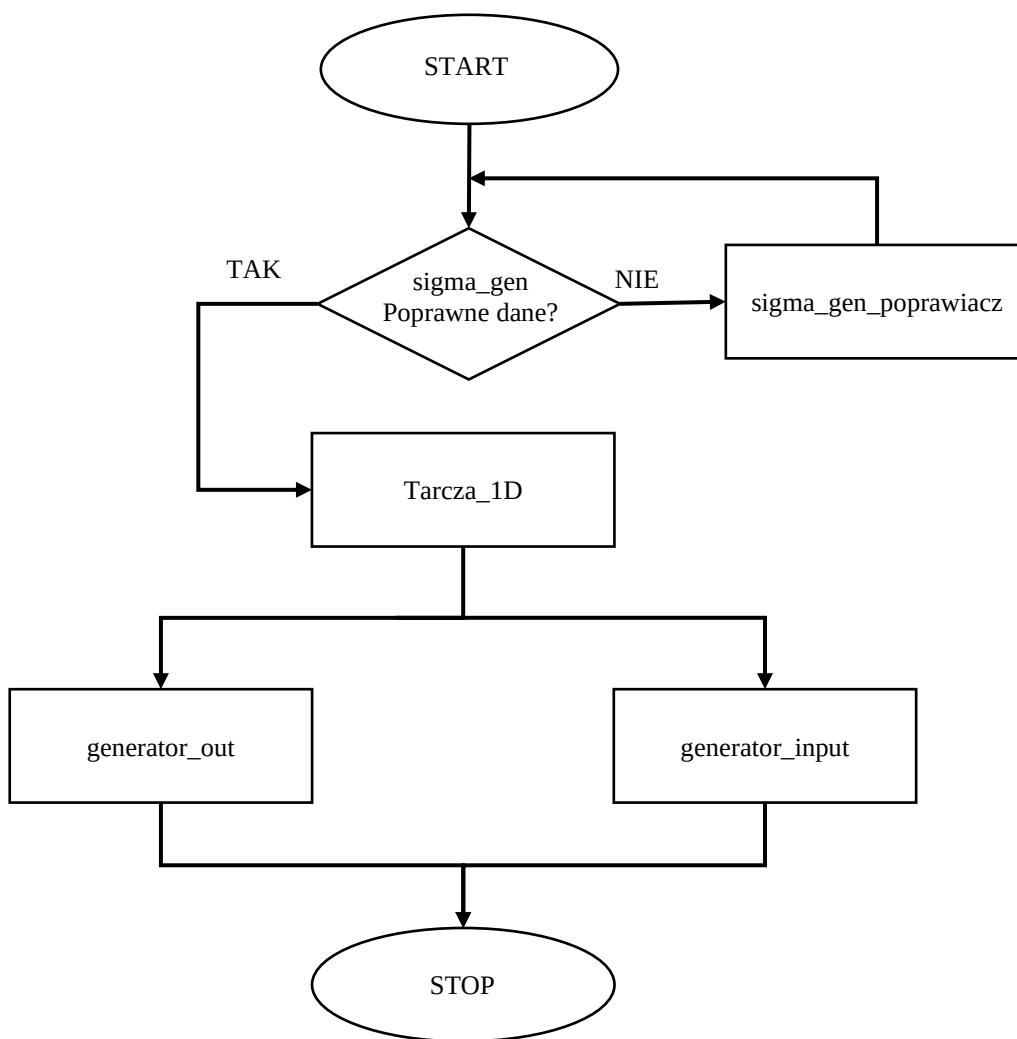
Parametr	Oznaczenie	Przedział	
Promień wewnętrzny tarczy	$r_{min} [m]$	0,01	0,5
Promień zewnętrzny tarczy	$r_{max} [m]$	r_{min}	1
Prędkość kątowa	$\omega [rad/s]$	50	2000
Masa pojedynczej łopatki wraz z jej częścią zamkową	$m_{li} [kg]$	0,01	1
Liczba łopatek	$z_{li} [-]$	20	100
Promień położenia środka masy łopatek	$r_{li} [m]$	r_{max}	1,5
Grubość tarczy dla r_1	$h_1 [m]$	0,005	0,250
Grubość tarczy dla r_{18}	$h_{18} [m]$	0,005	0,233

Dane dobrano w sposób losowy tak, aby były zgodne z przedziałami, a tarcze nie zwiększały swojej grubości wraz z promieniem. Szansa na zmniejszenie grubości tarczy wraz ze wzrostem promienia była tym mniejsza, im mniejsza była grubość. Założono, że wszystkie elementy tarczy będą miały znormalizowany wymiar promieniowy, zgodnie z tabelą 2.4. Sumując łączną długość elementów, otrzymamy długość całej tarczy równą $r_{max} - r_{min}$. W celu realizacji zadań napisano w języku MATLAB programy służące do rozwiązywania problemu metodą elementów skończonych. Rysunek 2.15 przedstawia schemat blokowy stosowanego algorytmu. W celu realizacji zadań napisano w języku MATLAB programy służące do rozwiązywania problemu metodą elementów skończonych.

Tabela 2.4. Znormalizowane długości elementów

Nr elementu	Zależność na obliczenie długości
1-4	$\frac{r_{max} - r_{min}}{40}$
5-16	$\frac{r_{max} - r_{min}}{10}$
17-18	$\frac{r_{max} - r_{min}}{20}$

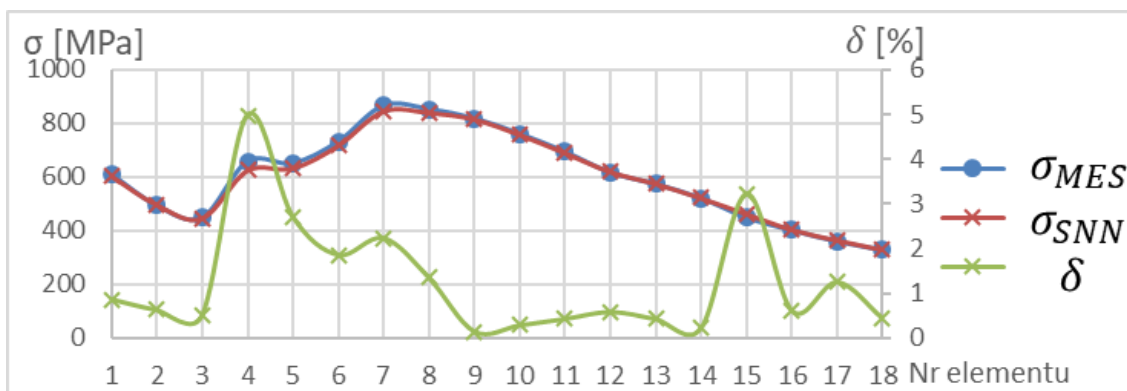
Korzystając z przedstawionych założeń, wylosowano 3200 przypadków tarcz profilowanych. Za pomocą napisanego programu opartego na MES obliczono rozkład naprężeń dla 2200 przypadków. Pozostałe tworzyły rezerwę dla przypadków niepoprawnych z punktu widzenia konstrukcji silników lotniczych lub zasad zawartych w MES. W sytuacji niemożliwości wykonania obliczeń, ze względu na zbyt duże skokowe zmiany grubości powodujące błędy numeryczne, takie przypadki zastępowano przypadkami rezerwowymi i operację powtarzano. Przypadki, dla których występował błąd obliczeń, były niezgodne ze sztuką inżynierską. Program o nazwie „Tarcza_1D” (Załącznik 1 - Tarcza_1D.m 1) liczący naprężenia z wykorzystaniem MES działa w oparciu o algorytm opisany w pracy [112]. Jest to program stanowiący jądro całego algorytmu. Drugim, nadrzędnym opracowanym programem jest „sigma_gen” (Załącznik 2). Ma on za zadanie zmieniać nazwy poszczególnych parametrów tak, aby były możliwie małymi tablicami, na potrzeby działania programu „Tarcza_1D”. Jego drugim zadaniem jest kontrolowanie, czy wszystkie obliczone naprężenia są poprawne pod względem numerycznym, czyli czy nie ma ujemnych objętości, zbyt dużych skokowych zmian grubości czy innych błędów numerycznych. Jeżeli tak, to jest wywoływany program „sigma_gen_poprawiacz” (Załącznik 3), którego zadaniem jest usunięcie n niepoprawnych wyników obliczeń i zastąpienie ich wynikami poprawnymi. Wyniki te generuje się poprzez ponowne, n -krotne wywoływanie programu „Tarcza_1D”. Dodatkowo występują jeszcze dwa pomocnicze programy: „generator_input” (Załącznik 4) i „generator_out” (Załącznik 5), których rolą jest wczytywanie i zapisywanie danych z i do arkusza kalkulacyjnego oraz tworzenie tablic przygotowanych jako dane wejściowe („generator_input”) i wyjściowe („generator_iout”) do uczenia SSN. Na rys. 2.15 przedstawiono schemat blokowy zastosowanego algorytmu.



Rys. 2.15. Schemat blokowy zastosowanego algorytmu

Końcowy etap obejmował budowę SSN. Podobnie jak w testowym przypadku użyto sieci typu NARX o architekturze odpowiadającej schematowi na rysunku 2.10. Wystąpiła różnica w ilości neuronów wejściowych i w warstwie ukrytej, wykorzystano układ 24-60-40-1 zamiast 4-25-15-1. Sieć uczono, wykorzystując algorytm SCG (metoda gradientu sprzężonego). Użyto osiemnastu sieci. Każda z nich miała ten sam zestaw danych wejściowych, ale liczyła naprężenia dla innego elementu.

W wyniku przeprowadzonej analizy uzyskano program zdolny doliczyć naprężenia dla dowolnej tarczy z przedziału zawartego w tabeli 2.3. Dokładność działania algorytmu sprawdzono za pomocą dziesięciu nowo wylosowanych przypadków spełniających podobne warunki jak dane uczące sieć. Porównywano naprężenia w tarczy profilowanej obliczone za pomocą algorytmu opartego o MES, służącego wcześniej do generacji danych z naprężeniami obliczonymi przez SSN. Po wstępnym uczeniu sieci uzyskano maksymalny błąd względny mniejszy od 5% przy średnim błędzie 1,3%. Wyniki dla jednego z przebadanych przypadków przedstawiono na rysunku 2.16.



Rys. 2.16. Rozkład naprężeń w tarczy profilowanej wyznaczony za pomocą MES (σ_{MES}) SSN (σ_{SSN}) wraz z błędem względnym (δ)

Przeprowadzona analiza pozwala wysunąć wniosek, że dalsze uczenie sieci, dla których błąd był większy (element 4, 5, 7 i 15), pozwoliłoby zbliżyć się całemu algorytmowi do dokładności 2%. Dalsze uczenie prowadziłoby do uzyskania jeszcze większej dokładności.

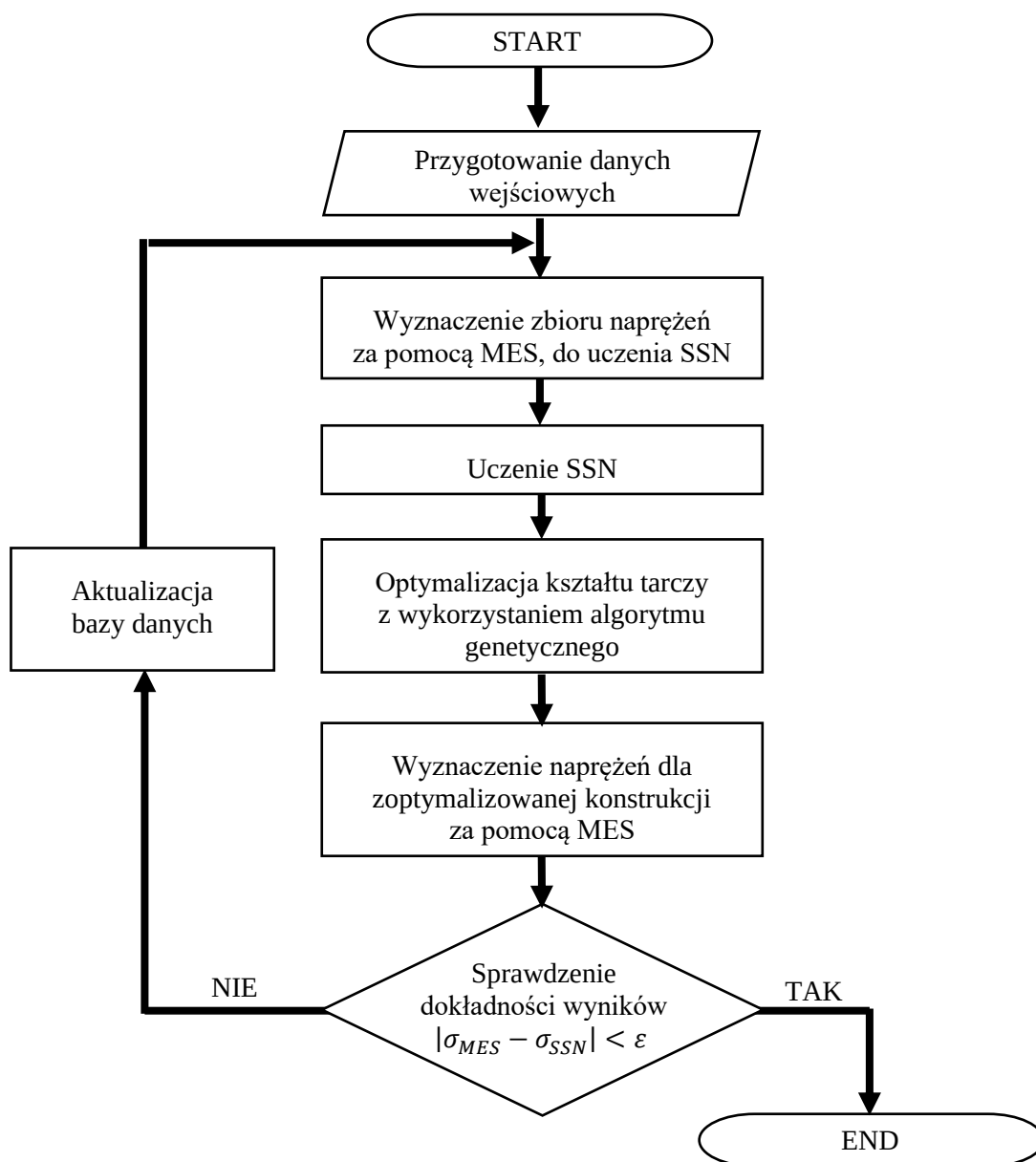
Dodatkowo wykonano analizę porównawczą czasu pracy algorytmu generującego dane uczące i samej SSN. Obliczone naprężenia posłużyły jako dane wejściowe do uczenia sieci. Czas obliczenia 2200 przypadków za pomocą MES wynosił, według wbudowanych w program MATLAB komend tic i toc, 854,9 sekundy, natomiast te same obliczenia wykonane za pomocą SSN zajęły 0,45 sekundy.

3. METAMODEL OPARTY NA SSN DO OPTYMALIZACJI TARCZY SPRĘŻARKI OSIOWEJ

W rozdziale przedstawiono wyniki badań numerycznych nad możliwością zastosowania Sztucznych Sieci Neuronowych (SSN) do obliczeń wytrzymałościowych elementów maszyn wirnikowych. Opisano strukturę i zasadę działania algorytmu, przechodząc od ogólnej architektury do dokładnego opisu kolejnych modułów. Algorytm wykorzystano do analizy tarczy sprężarki i określono dalsze obszary badań.

Opracowany algorytm ma za zadanie przeprowadzić optymalizację kształtu tarczy sprężarki osiowej silnika odrzutowego przy uwzględnieniu kryterium czasu. Funkcją celu algorytmu jest minimalizacja masy tarczy przy spełnieniu warunku wytrzymałościowego. Jednym z istotnych problemów w rozwiązywanym tego typu przypadku jest czasochłonność obliczeń. Przy wykorzystaniu komercyjnego oprogramowania opartego na metodzie elementów skończonych, przy optymalizacji, program albo automatycznie usuwa nadmiarowe elementy, albo robi to konstruktor. Następnie wygładza się powierzchnię zoptymalizowanego obiektu i powtarza proces wyznaczania naprężeń i usuwania zbędnych elementów. Cykl obliczeń powtarzany jest w pętli do osiągnięcia zadowalającego rezultatu. Należy zauważyć, że w tym przypadku algorytm nie może dodać materiału, aby odciążyć inne części elementu. Proces zarówno wyznaczania naprężeń, jak i usuwania nadmiarowych elementów jest stosunkowo długi i wydłuża się wraz z ilością iteracji i zwiększaniem dokładności. W przedstawionym rozwiązaniu zastosowano optymalizację z wykorzystaniem metody Monte Carlo. Na rysunku 3.1 zaprezentowano ogólną strukturę opracowanego algorytmu, gdzie zastąpiono znaczną część obliczeń wykonywanych przez MES algorytmem opartym na SSN. Podobny algorytm wykorzystano w publikacji [71], przy innym modelu aproksymacyjnym i algorytmie genetycznym. Model aproksymacyjny zastosowany w pracy [71] to radialne sieci neuronowe; w niniejszej pracy zastosowano SSN typu Feed Forward. Dodatkowo w opracowaniu [71] baza danych zwiększała się z każdą iteracją, natomiast w zaproponowanym algorytmie wielkość bazy danych jest ograniczona do maksymalnie 50 przypadków uczących.

Struktura algorytmu przedstawionego na rys. 3.1 obejmuje wprowadzenie danych wejściowych do programu, po czym następuje wyznaczenie rozkładu naprężeń niezbędnych do uczenia SSN. Następnie podprogram, z wykorzystaniem algorytmu genetycznego, optymalizuje masę tarczy z uwzględnieniem warunku wytrzymałościowego. W kolejnym etapie, z użyciem MES, sprawdzana jest dokładność warunku wytrzymałościowego względem SSN. Jeżeli dokładność ta jest niezadowalająca lub wyniki różnią się od wyników optymalnych z poprzedniej iteracji, następuje powrót do uczenia SSN i ponowna optymalizacja. W celu walidacji kryterium czasu mierzy się czas generowania danych niezbędnych do uczenia SSN oraz czas działania algorytmu genetycznego.



Rys. 3.1. Schemat blokowy algorytmu do optymalizacji tarczy sprężarki

3.1. GŁÓWNE ELEMENTY ALGORYTMU

Ogólną strukturę algorytmu zaprezentowano na rys. 3.1. Główne elementy algorytmu, które zostały przedstawione, to:

- rodzaj danych wejściowych;
- wykorzystywany algorytm MES;
- struktura wykorzystanych SSN;
- funkcja celu i kary;
- ograniczenia algorytmu genetycznego.

Dane wejściowe to: właściwości materiałowe (moduł Younga, liczba Poissona, gęstość materiału tarczy), promień otworu, zewnętrzny promień tarczy, prędkość kątowna tarczy, masa łopatki wraz z jej zamkiem i fragmentem tarczy niezbędnym do zadziałania zasady de Saint-Venanta, liczba łopatek, promień położenia środka masy łopatki wraz

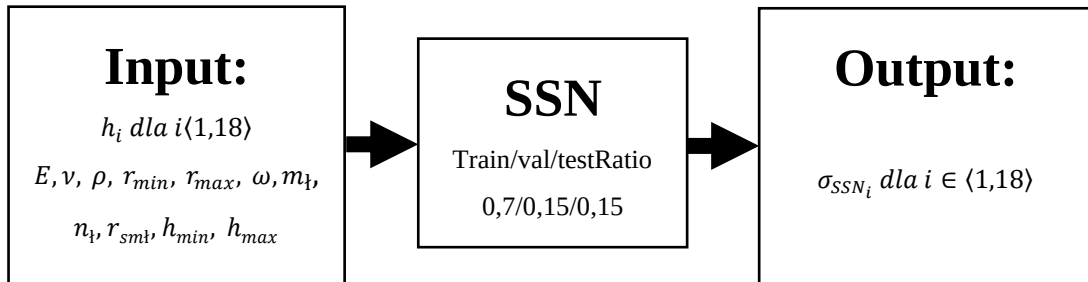
z częścią zamkową, a także maksymalna i minimalna dopuszczalna grubość tarczy. Do algorytmu dodatkowo wprowadzono 18 grubości tarczy na znormalizowanych promieniach, zgodnie z równaniem (3.1), grubość u podstawy tarczy została narzucona odgórnie jako maksymalna dopuszczalna wartość. Początkowo tarczę podzielono na 10 równych elementów skończonych. Następnie pierwsze dwa elementy podzielono na cztery, a ostatnie dwa na dwa, co w efekcie pozwoliło otrzymać tarczę podzieloną na 18 elementów skończonych z zagęszczeniem siatki w okolicy krawędzi, zgodnie z regułą (3.1). W przypadku tarczy o dowolnych rozmiarach położenie kolejnych węzłów siatki zawsze odpowiada takiemu samemu promieniowi względnemu, co umożliwia parametryzację różnych tarcz celem uczenia Sztucznych Sieci Neuronowych.

$$r_i = \begin{cases} r_1 = r_{min} \\ \frac{r_{max} - r_{min}}{40} + r_{i-1} & \text{dla } i = \langle 2,9 \rangle \\ \frac{r_{max} - r_{min}}{10} + r_{i-1} & \text{dla } i = \langle 10,16 \rangle \\ \frac{r_{max} - r_{min}}{20} + r_{i-1} & \text{dla } i = \langle 17,19 \rangle \end{cases} \quad (3.1)$$

gdzie: r_{min} – promień otworu tarczy,

r_{max} – promień zewnętrzny części optymalizowanej tarczy.

Wszystkie dane wprowadzone do algorytmu (w tym materiałowe) były w postaci macierzowej. Pozwala to uwzględnić zmianę właściwości materiałowych np. w przypadku uwzględnienia gradientu temperatury.



Rys. 3.2. Parametry wejściowo-wyjściowe SSN ze współczynnikami szkolenia, testowania i walidacji,

gdzie: h_i – grubość tarczy dla kolejnego, znormalizowanego promienia,
 E – moduł Younga,
 ν – liczba Poissona,
 ρ – gęstość materiału,
 r_{min} – promień otworu,
 r_{max} – promień zewnętrzny tarczy,
 ω – prędkość kątowna,
 m_l – masa łopatki,
 n_l – liczba łopatek,
 r_{sm} – promień środka masy łopatki,
 h_{min} – minimalna dopuszczalna grubość tarczy,
 h_{max} – maksymalna dopuszczalna grubość tarczy,
 σ_{SSN_i} – naprężenia zredukowane, obliczone przez SSN dla każdego znormalizowanego promienia

Walidację wyników uzyskanych z algorytmu opartego na MES przeprowadzono dwutorowo. Porównano je z obliczeniami analitycznymi dla tarczy prostej i z wynikami programu komercyjnego (ANSYS) dla tarczy profilowanej. Różnica naprężeń algorytmu

MES (uproszczonego poprzez pominięcie macierzy sztywności dodatkowej) dla tarczy po optymalizacji nie przekracza 0,5%.

W celu przygotowania danych do uczenia SSN wylosowano zbiór tarcz z przedziału $\pm 5\%$ grubości tarczy względem pierwszego lub ostatniego optymalnego rozwiązania. Jeżeli na kolejnym (licząc od osi obrotu) promieniu grubość tarczy była większa niż na poprzednim, to zastępowano ją wartością z poprzedniego promienia. Pozwoliło to uzyskać rozkład grubości zgodny z obiektami rzeczywistymi (rys 2.14). Sprawdzano także, czy nie została przekroczona maksymalna i minimalna dopuszczalna grubość tarczy.

W kolejnym etapie wyznaczono macierz przypadków uczących. Posłużyła ona do generowania właściwych danych wejściowych i wyjściowych. Algorytm, dla 10 przypadków z wcześniej przygotowanej macierzy, oblicza rozkład naprężeń za pomocą metody elementów skończonych. Zastosowana metoda została szerzej opisana w pracy [112] i opiera się na aproksymacji tarczy elementami powłoki stożkowej o liniowo zmiennej grubości. W przypadku problemów numerycznych związanych ze zbyt dużym skokiem grubości tarczy przypadek jest nadpisywany kolejnym. W każdej pętli działania algorytmu nowe 10 przypadków jest dopisywane do tych przygotowanych już we wcześniejszych iteracjach, a 10 najstarszych usuwane. Program został skonfigurowany tak, aby jednocześnie baza danych liczyła nie więcej niż 50 przypadków uczących. Pozwala to na szybkie i w małym obszarze, dostatecznie dokładne uczenie SSN oraz generowanie danych do bazy. Na koniec program przesyła dane do kolejnego podprogramu.

W celu określenia struktury Sztucznej Sieci Neuronowej i doboru algorytmu uczącego przeprowadzono badania numeryczne, w wyniku których wybrano sieć typu Feed Forward. Rozważano także sieć NARX, jednak okazało się, że jej czas uczenia był dłuższy przy mniejszej dokładności. Następnie przeprowadzono analizę porównawczą następujących 9 algorytmów uczących:

- *Scaled conjugate gradient backpropagation (SCG);*
- *Levenberg-Marquardt backpropagation (L-M);*
- *Gradient descent with momentum and adaptive learning rate backpropagation (GDA);*
- *BFGS quasi-Newton backpropagation (BFGS);*
- *Gradient descent with momentum backpropagation (GDM);*
- *One-step secant backpropagation (OSS);*
- *Conjugate gradient backpropagation with Powell-Beale restarts (CGB);*
- *Conjugate gradient backpropagation with Fletcher-Reeves updates (CGF);*
- *Conjugate gradient backpropagation with Polak-Ribière updates (CGP).*

Metodę uczenia SSN sprawdzano w oparciu o kryterium grubości tarczy na dwa sposoby. Pierwszy dotyczył przypadków $(100 \pm 2)\%$ grubości tarczy, drugi $(100 \pm 5)\%$. W pierwszym przypadku algorytmy SCG, L-M, BFGS uczyły się znacznie dłużej (średnio 24, 79, 126 sekund), pozostałe natomiast poniżej 1 s, co więcej wszystkie z porównywalną dokładnością. Natomiast w przypadku danych $\pm 5\%$, tylko algorytm SCG był w stanie nauczyć sieć o wybranej architekturze. Sprawdzono także, że dwukrotne uczenie sieci wybraną metodą znacząco zmniejsza prawdopodobieństwo występowania poważnych błędów uczenia, które przy pojedynczym uczeniu

występowały mniej więcej 1 na 5 przypadków, natomiast przy dwukrotnym poważnych błędów nie stwierdzono.

Ilość warstw w sieci została dobrana na podstawie efektywności uczenia sieci η opartej na iloczynie zredukowanego błędu względnego i zredukowanego czasu (3.2):

$$\eta = \frac{\delta_1 t_1}{\delta_i t_i}, \quad (3.2)$$

gdzie: η – efektywność uczenia sieci,
 δ_1 – błąd względny pierwszej sieci,
 δ_i – błąd względny i-tej sieci,
 t_1 – czas uczenia pierwszej sieci,
 t_i – czas uczenia i-tej sieci.

Badano sieci od dwóch do pięciu warstw ukrytych po 50 neuronów w każdej. Wyniki analizy przedstawiono w tabeli 3.1.

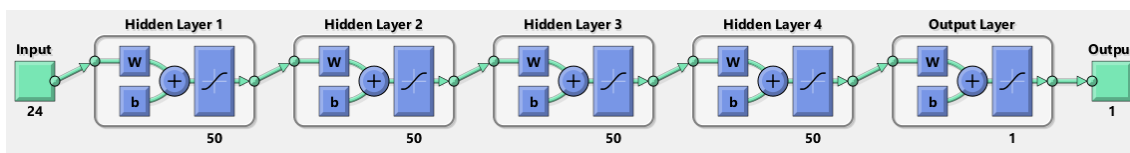
Tabela 3.1. Efektywność sieci w zależności od ilości warstw ukrytych

Warstwy ukryte	δ [%]	t [s]	$\bar{\delta}$ [–]	$\bar{t}$ [–]	η [–]
2	10,37	0,84	1	1	1
3	13,23	1,18	0,783	0,7119	0,558
4	7,24	0,97	1,432	0,8660	1,240
5	6,01	1,22	1,726	0,6885	1,188

Wybrano sieć neuronową o czterech warstwach ukrytych, ponieważ miała największą efektywność. Początkowy błąd uczenia był duży. Został on zredukowany do około 1,5% przez cykliczne działanie całego algorytmu (sieć była uczona jeszcze kilka razy), przy jednoczesnej modyfikacji bazy danych przypadków uczących.

Ilość neuronów przyjęto, sprawdzając wartości co 20 w warstwach (począwszy od 10 neuronów). Dla wartości 30 neuronów i mniejszej sieć miała tendencję do uczenia się na pamięć, zwłaszcza dla przypadków, gdy grubość tarczy zawierała się w przedziale $\pm 2\%$. Dla sieci o większej ilości neuronów czas uczenia znacznie wzrastał przy niedużym wzroście dokładności.

Kolejny podprogram uczył SSN (rys. 3.3). Zastosowano tu Sztuczne Sieci Neuronowe typu Feed Forward, zaimplementowane w programie MATLAB z użyciem Neural Network Toolbox. Sieć zbudowano z czterech warstw ukrytych, po 50 neuronów w każdej, i jednej warstwy wyjściowej. Każdy pojedynczy neuron posiadał biasy i sigmoidalną funkcję aktywacyjną.



Rys. 3.3. Schemat SSN do obliczania naprężeń, gdzie: Input 24 – dane wejściowe; Hidden Layer 1, 2, 3, 4 – warstwy ukryte, po 50 neuronów każda; Output Layer – pojedynczy neuron warstwy wyjściowej z sieci; Output 1 – parametr wyjściowy z sieci

Dane wejściowe do sieci to 24 wspomniane wcześniej parametry. Jedyne parametry wyjściowe to naprężenia. Zastosowano 18 Sztucznych Sieci Neuronowych typu Feed Forward, z których każda liczyła naprężenia w jednym z elementów zwymiarowanych wcześniej. Do nauki sieci wykorzystano algorytm skalowanych gradientów sprzężonych (SCG – *Scaled Conjugate Gradient method*) [109], który podobnie jak w metodzie gradientów sprzężonych wywodzi się z rozwinięcia funkcji celu w kwadratowe równanie Taylora. Różnica polega na połączeniu jednoczesnego wyznaczania sprzężonego kierunku poszukiwań i optymalnego kroku w tym kierunku. Proces uczenia trwał aż do osiągnięcia braku poprawy w 50 kolejnych iteracjach i był powtarzany dwukrotnie dla każdej sieci. Czas uczenia wszystkich SSN w czasie jednej iteracji działania całego zaproponowanego algorytmu to około 40 sekund.

Podprogram optymalizujący kształt tarczy służył do optymalizacji grubości w węzłach (węzły zewnętrzne każdego z elementów skończonych), do czego zastosowano algorytm genetyczny. Grubość tarczy, dla węzła odpowiadającego wewnętrznej krawędzi tarczy, została zadana. Górne i dolne ograniczenia przyjęto jak dla grubości tarcz przygotowywanych do uczenia i stanowiły one $\pm 5\%$ grubości tarczy optymalnej z poprzedniej iteracji. Funkcja celu w algorytmie to minimalizacja masy tarczy z uwzględnieniem warunku wytrzymałościowego. Warunek wytrzymałościowy uwzględniono w postaci funkcji kary.

W trakcie działania algorytm genetyczny generuje nowe przypadki tarcz i oblicza ich masy, sprawdzając warunek wytrzymałościowy dla całej tarczy (z wykorzystaniem nauczonych SSN). W przypadku przekroczenia naprężeń dopuszczalnych, do masy wynikającej z geometrii, dodawana była kara, wyliczona jako kwadrat tysięczności różnicy pomiędzy naprężeniami obliczonymi w każdym elemencie a naprężeniami dopuszczalnymi. Pozwoliło to przejść z funkcji celu z ograniczeniami na funkcję celu bez ograniczeń, ale z karą. Wartość kary została dobrana eksperymentalnie tak, aby był zapewniony gwałtowny wzrost wartości funkcji poza dopuszczalnymi granicami obszaru, przy spełnieniu warunku ciągłości pochodnej funkcji celu. Po każdym pokoleniu następowała walidacja dokładności algorytmu z wykorzystaniem MES. Jeżeli działanie algorytmu SSN było dokładne i rozwiązanie optymalne nie zmieniło się znacząco względem ostatniego, następowało zatrzymanie algorytmu. W przeciwnym przypadku powracano do generowania danych niezbędnych do uczenia SSN w nowo ograniczonym obszarze rozwiązań dopuszczalnych.

W czasie działania podprogramu optymalizującego czas pracy algorytmu genetycznego został ograniczony do jednej generacji, pomimo iż dłuższy czas pracy zbliża rozwiązanie do minimum globalnego. Ograniczenie to zastosowano, ponieważ kara naliczana jest na podstawie algorytmu SSN, który ma skończoną dokładność. Im bliżej granic przestrzeni rozwiązań przewidzianych przy treningu SSN znajduje się obliczany przypadek, tym mniejsza jest dokładność obliczeń. Po każdej iteracji algorytmu genetycznego usuwano wszystkie rozwiązania za wyjątkiem najlepszego (strategia

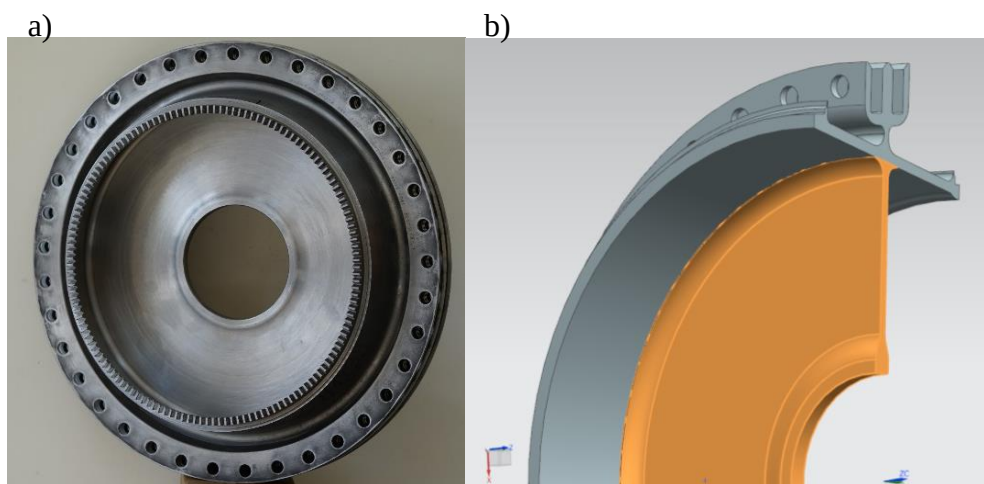
elitarna), następnie w jego okolicy tworzą nową przestrzeń poszukiwania rozwiązania optymalnego. Zastosowaną modyfikację algorytmu genetycznego można porównać do biologicznych wielkich wymierań. Osobniki, które przetrwały, stają się źródłem całych nowych rodzajów czy rodzin.

Przerwanie poszukiwań minimum i douczenie SSN nowych przypadków, z okolicy ostatniego optymalnego rozwiązania, znacząco zmniejsza czas i ilość iteracji algorytmu genetycznego niezbędnych do osiągnięcia minimum z założoną dokładnością.

Sprawdzenie algorytmu oparte jest na warunku wytrzymałościowym, sprawdzanym za pomocą MES. W przypadku niespełnienia warunku pętla jest powtarzana.

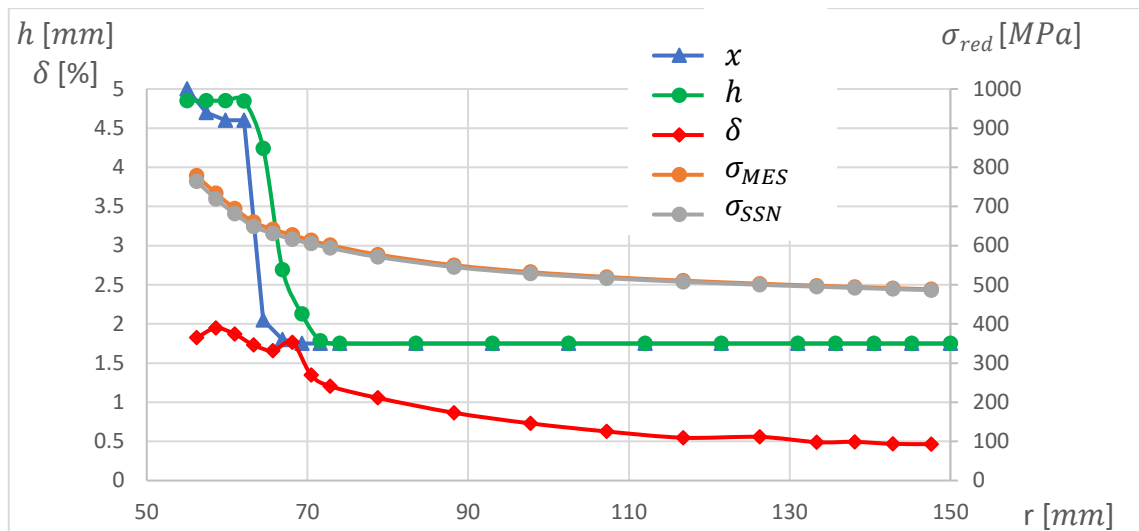
3.2. OPTYMALIZACJA TARCZY SPRĘŻARKI

Z wykorzystaniem wcześniej przedstawionego algorytmu przeprowadzono proces optymalizacji (część tarczy zaznaczona kolorem pomarańczowym na rysunku 3.4). Optymalizacja takiego układu konstrukcyjnego umożliwiła analizę zasadności oraz poprawności stosowania SSN w połączeniu ze zmodyfikowanym algorytmem genetycznym. Wyciągnięte wnioski z obliczeń wykorzystano do dopracowania całego algorytmu.



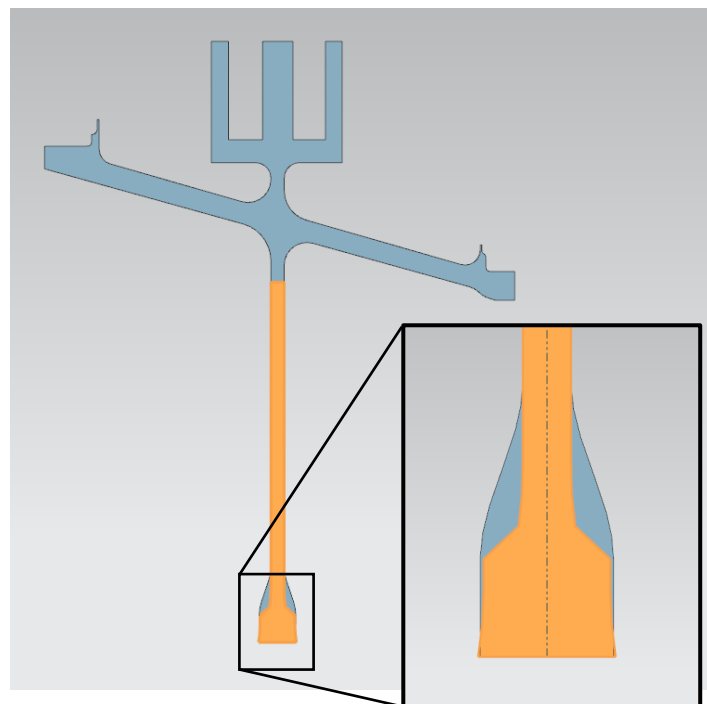
Rys. 3.4. Tarcza stopnia sprężarki osiowej turbinowego silnika odrzutowego: obiekt rzeczywisty (a) oraz model wzorcowy tarczy z zaznaczonym obszarem optymalizowanym (b)

W procesie optymalizacji przeprowadzono pięć iteracji. Jedna iteracja trwała około 98 s, w tym przez około 40 s algorytm przygotowywał dane wejściowe do uczenia i zajmował się treningiem Sztucznych Sieci Neuronowych. Wyniki procesu optymalizacji kształtu i rozkładu naprężeń przedstawiono na rysunku 3.5.



Rys. 3.5. Kształt zoptymalizowanej tarczy i odpowiadający jej rozkład naprężeń zredukowanych, gdzie x – profil tarczy po optymalizacji; h – profil tarczy przed optymalizacją; δ – błąd względny naprężeń; σ_{MES} – naprężenia wyznaczone za pomocą MES; σ_{SSN} – naprężenia wyznaczone za pomocą SSN

Naprężenia σ_{MES} przedstawione na rysunku 3.5 zostały wyznaczone jako ostatni element pracy algorytmu. Uzyskana masa zoptymalizowanej tarczy (rys. 3.6) to około 7,6 kg. W wyniku procesu optymalizacji udało się zmniejszyć masę tarczy o 0,5 kg w stosunku do tarczy wzorcowej.



Rys. 3.6. Zestawienie profilu tarczy wyjściowej (kolor szary) i zoptymalizowanej (kolor pomarańczowy). Powiększono obszar największych zmian

Ponieważ kształt uzyskany w wyniku optymalizacji z technologicznego punktu widzenia takie podcięcie skutkujące karbem, w dalszej części pracy zrezygnowano z parametryzacji tarczy elementami trapezowymi.

Sprawdzono także dopasowanie sieci neuronowych do bazy przypadków uczących. Wartości RMSE (pierwiastek błędu średniokwadratowego), MAPE (średni bezwzględny

błąd procentowy) i R^2 (współczynnik determinacji) zostały obliczone dla ostatniej iteracji algorytmu, co przedstawia tabela 3.2.

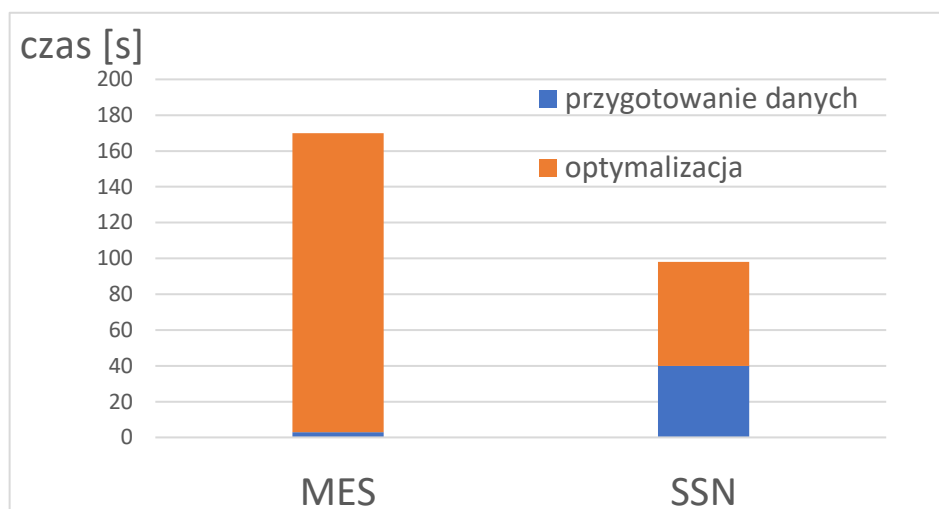
Tabela 3.2 RMSE, MAPE i R^2 dla ostatniej iteracji algorytmu

Numer SSN	<i>RMSE</i>	<i>MAPE</i>	R^2
1	1,3041	0,1242	0,99915
2	1,2487	0,1099	0,99890
3	1,1778	0,1213	0,99904
4	1,0996	0,1312	0,99924
5	1,0478	0,1256	0,99916
6	1,0340	0,1414	0,99910
7	0,5993	0,0811	0,99914
8	0,5274	0,0782	0,99923
9	0,4842	0,0669	0,99915
10	0,3630	0,0525	0,99908
11	0,3010	0,0440	0,99897
12	0,2593	0,0400	0,99890
13	0,2439	0,0397	0,99876
14	0,2446	0,0348	0,99850
15	0,2109	0,0324	0,99881
16	0,2169	0,0325	0,99859
17	0,2030	0,0325	0,99869
18	0,2046	0,0323	0,99870
Wartość średnia	0,5983	0,0733	0,99895

Wartość średnia wymienionych parametrów to kolejno $RMSE = 0,5983$, $MAPA = 0,0733$ i $R^2 = 0,99895$.

W celu porównania czasu pracy algorytmu zmodyfikowano wykorzystany algorytm poprzez usunięcie z niego sieci neuronowych. Zmodyfikowany algorytm dalej będzie nazywany algorytmem wzorcowym. Naprężenia były wyznaczane bezpośrednio za pomocą metody elementów skończonych. Czas działania całego algorytmu wzorcowego to około 170 s. Czas przygotowania danych wejściowych do algorytmu w tym przypadku to około 3 s, a pozostały czas to praca genetycznego algorytmu optymalizującego. Na

rysunku 3.7 przedstawiono porównanie czasu pracy zaproponowanego algorytmu używającego SSN i algorytmu wzorcowego MES.



Rys. 3.7. Porównanie czasu pracy pojedynczej iteracji algorytmu wzorcowego, opartego wyłącznie na MES i algorytmu wykorzystującego SSN

Zaproponowany algorytm łączący SSN z modułem genetycznym działa stosunkowo szybko. Czas działania algorytmu to około 7-8 minut, od momentu rozpoczęcia obliczeń, co w przypadku analizowanego problemu objęło 5 iteracji. Czas trwania pojedynczej iteracji całego algorytmu wzorcowego to ~ 170 s, przy ~ 98 s jednej iteracji zaproponowanego algorytmu opartego na SSN. Różnica czasu w przedstawionym przypadku to ~ 70 s, co stanowi w przybliżeniu 40% czasu pracy algorytmu wzorcowego, przy dokładności względem MES na poziomie $(97 \div 98)\%$. Czas potrzebny do wprowadzenia danych wejściowych i zamodelowania zoptymalizowanej tarczy to kroki, które należy wykonać zarówno dla algorytmu SSN, jak i wzorcowego. Z tego powodu ich czasy zostały pominięte jako niemające wpływu na bezwzględną różnicę czasu działania algorytmów.

Zaproponowany algorytm łączący moduł genetyczny i Sztuczne Sieci Neuronowe (SSN) pozwala znacznie skrócić czas optymalizacji. Na przedstawionym przykładzie minimalizacji masy tarczy lotniczego silnika odrzutowego, z uwzględnieniem warunku wytrzymałościowego, skrócono czas pracy algorytmu o 40% względem algorytmu wzorcowego. Sam wzorcowy algorytm genetyczny został zmodyfikowany względem klasycznego podejścia literaturowego poprzez zmianę w każdej iteracji obszar rozwiązań dopuszczalnych. Nowy obszar określano jako $\pm 5\%$ wartości optymalnej z poprzedniej iteracji. Zastosowanie takiej modyfikacji algorytmu genetycznego jest dopuszczalne, gdy zna się ogólny charakter funkcji celu lub spodziewaną wartość rozwiązania optymalnego.

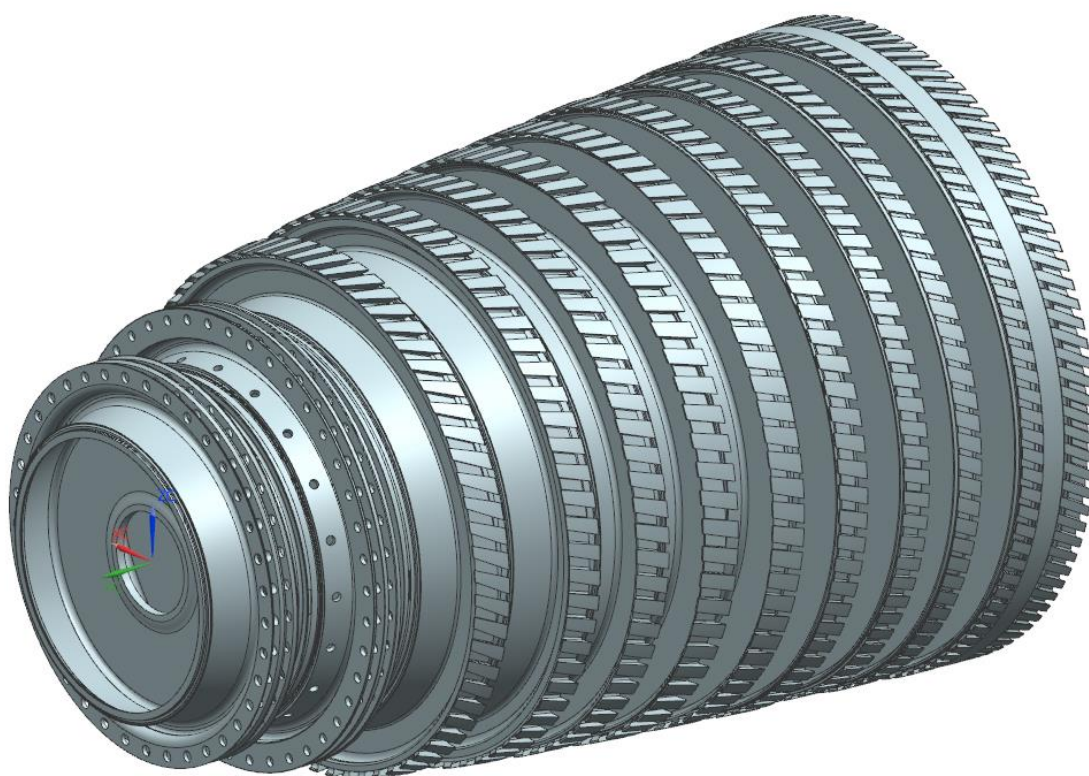
Przedstawiony algorytm może być z powodzeniem zastosowany wszędzie tam, gdzie wymagana jest optymalizacja, w której obliczenie wartości funkcji celu dla dowolnego punktu jest czasochłonne. Część z tych obliczeń można z powodzeniem zastąpić Sztucznymi Sieciami Neuronowymi, nawet w przypadku, gdy równania różniczkowe są nieliniowe (np. do wyznaczenia rozkładu naprężeń w tarczy wirującej). Celem dalszej pracy będzie potwierdzenie, że zaproponowany algorytm można zastosować także w przypadku bardziej skomplikowanego modelu tarczy silnika turbinowego (konstrukcja tarczowo-bębnowa).

4. FUNKCJA CELU I KARY DO OPTIMALIZACJI KONSTRUKCJI WIRNIKA TARCZOWO-BĘBNOWEGO

W celu przeprowadzenia optymalizacji konstrukcji wirnika tarczowo-bębnowego sprężarki lotniczego silnika turbinowego przygotowano odpowiedni model. Za przykład posłużyła czternastostopniowa sprężarka silnika Lyulka AL-21F3 (rys. 4.1). Obiektem badań w obszarze optymalizacji konstrukcji tarczowo-bębnowej była sprężarka osiowa lotniczego silnika turbinowego AL-21F3. Wykonano model tego zespołu oraz zaplanowano i przeprowadzono optymalizację równoległą stopni o konstrukcji bębnowo-tarczowej (od 3 do 14 na rysunku 4.2). W tym celu wykorzystano algorytm genetyczny poszukujący minimum funkcji celu. Stopnie o zastosowaniu specjalnym (1, 8 i 14 stopień) nie podlegały optymalizacji, tak jak i drugi stopień o konstrukcji bębnowej.



Rys. 4.1. Wirnik silnika AL-21F3



Rys. 4.2. Model optymalizowanych stopni sprężarki silnika AŁ-21F3

Sztuczne Sieci Neuronowe zostały wykorzystane w celu zmniejszenia czasu określania funkcji kary. Naprężenia niezbędne do uczenia SSN określono za pomocą metody elementów skończonych z wykorzystaniem języka APDL. Ogólny schemat algorytmu jest podobny do algorytmu przedstawionego na rysunku 3.1.

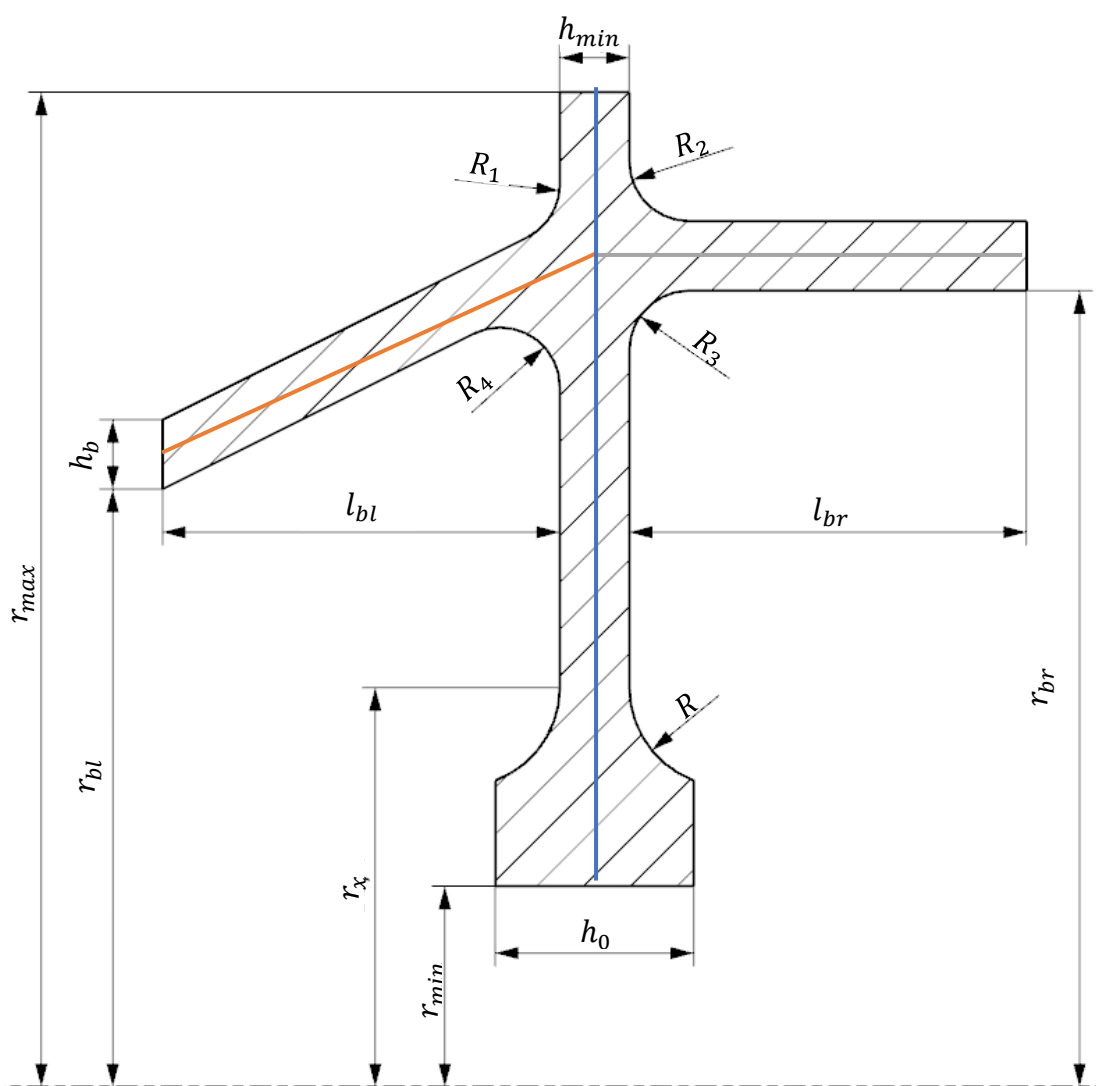
Pierwszym etapem optymalizacji było przygotowanie modelu obiektu rzeczywistego w celu określenia wymiarów gabarytowych i geometrycznych warunków brzegowych. W tabeli 4.1 zestawiono dane wejściowe do programu optymalizującego dla wybranych czternastu stopni sprężarki.

Tabela 4.1. Wymiary gabarytowe i geometryczne warunków brzegowych dla optymalizowanych stopni optymalizowanego wirnika, gdzie E – moduł Younga, ρ – gęstość, ν – liczba Poissona, m_l – masa łopatki, r_{sm} – promień środka masy części zamkowej, r_{sml} – promień środka masy łopatki, V – objętość części zamkowej. Pozostałe oznaczenia zgodne z rys. 3.4

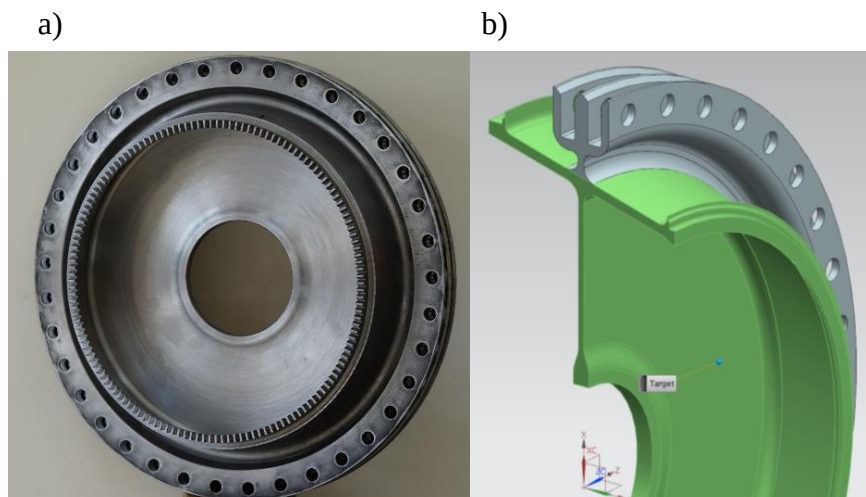
Nr stopnia	3	4	5	6	7	8	9	10	11	12	13	Jednostka
Parametr												
r_{max}	181,6	213	266,8	283,3	290,1	299,4	306,8	314,408	315,3533	323,1	328	mm
r_{min}	55	55	55	55	55	48	150	150	73	73	90	mm
h_0	10											mm
h_{min}	3,5											mm
l_{bl}	50,6	53,3	57,0	67,6	41,8	43,0	43,0	37,0	37,5	33,7	29,0	mm
r_{bl}	151,5	181,6	210,5	237,6	247,4	247,4	247,4	239,8	224,6	211,6	196,3	mm
l_{br}	56,5	52,8	42,3	41,8	42,3	39,0	41,0	40,9	36,9	38,5	21,4	mm
r_{br}	181,6	210,5	237,6	247,4	247,4	247,4	239,8	224,6	211,6	196,3	186,4	mm
r_b	165,7	195,5	226,1	247,4	247,4	247,4	247,4	232,6	216,4	203,8	188,8	mm
E	113,6											GPa
ρ	4430											kg/m ³
ν	0,3420	0,3420	0,3420	0,3420	0,3420	0,3420	0,3420	0,3420	0,3420	0,3420	0,3420	–
m_l	150,7	121,0	72,1	46,8	43,4	40,8	28,8	21,2	20,0	13,4	12,3	g
$ilosc_lopatek$	37	44	67	66	68	64	74	73	80	88	97	–
r_{sm}	195,3	226,6	270,5	283,8	296,3	304,2	312,1	318,4	320,0	327,0	331,9	mm
r_{sml}	261,0	278,6	318,0	323,0	325,0	329,0	330,0	338,0	341,0	344,0	349,0	mm
V	17616	20707	9846	12341	7224	8391	7395	9745	9241	8174	3486	mm ³
n	8310											obr/min

4.1. ZASTOSOWANIE JĘZYKA APDL DO WYZNACZANIA NAPRĘŻENIA W KONSTRUKCJI TARCZOWO-BĘBNOWEJ

Największą zaletą języka APDL (*Ansys Parametric Design Language*) jest przystosowanie do modelowania parametrycznego, co pozwala uzyskać zbiór rozwiązań. Za pomocą języka APDL można także przeprowadzić dyskretyzację modelu, zadać parametryczne warunki brzegowo-początkowe i przeprowadzić analizę numeryczną. Możliwe jest także wygenerowanie ścieżek, wzdłuż których – niezależnie od wariantu modelu – odczytuje się wybrane wyniki analizy numerycznej. Z uwagi na te zalety zdecydowano się na wykorzystanie języka APDL. Celem pracy było stworzenie parametrycznego modelu konstrukcji tarczowo-bębnowej sprężarki lotniczego silnika turbinowego, którego pojedynczy stopień przedstawiono na rysunku 4.4. Model parametryczny stopnia w widoku przekrojowym przedstawiono na rysunku 4.3.

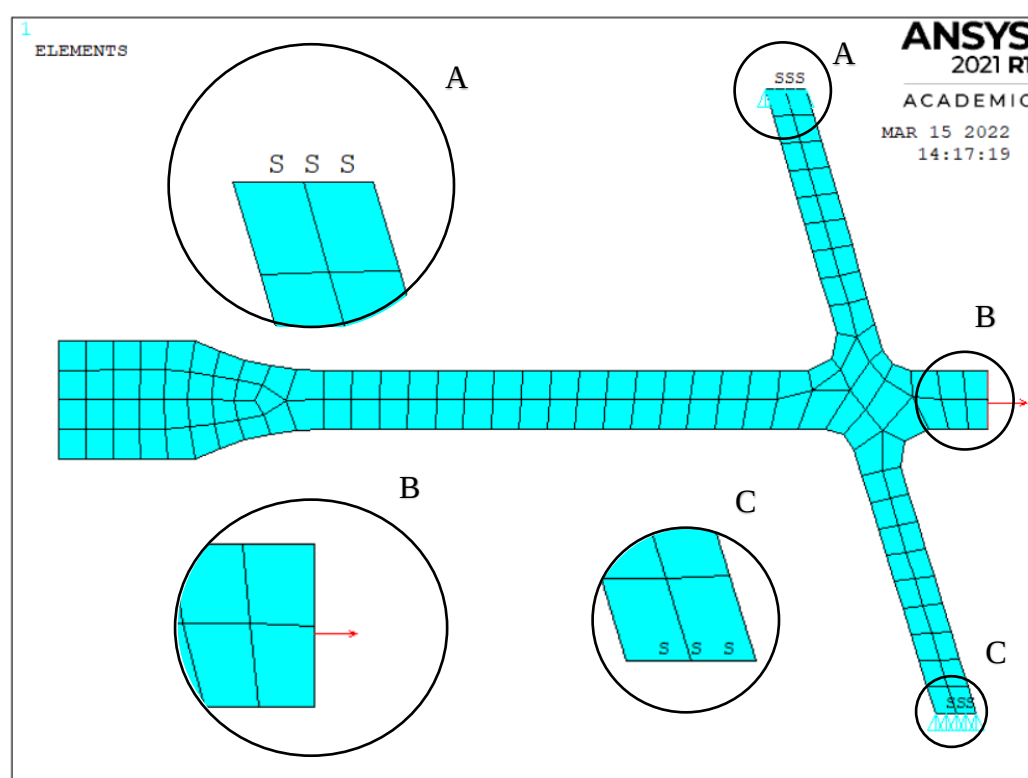


Rys. 4.3. Model parametryczny tarczy z oznaczeniem głównych wymiarów. Kolorami zaznaczono ścieżki, wzdłuż których wyznaczano naprężenia do optymalizacji



Rys. 4.4. Optymalizowana tarcza sprężarki: a) obiekt rzeczywisty, b) model CAD

Model przedstawiony na rysunku 4.3 został skonstruowany tak, aby zapewnić styczność krzywych (klasa połączenia G1) w miejscach połączenia elementów układu, z wyłączeniem połączenia piasta-łuk R , w którym zastosowano połączenie klasy G0, czyli brak ciągłości pochodnych. Użyty model jest osiowosymetryczny względem osi OY (rys 4.5), ponieważ wymaga tego struktura języka APDL.



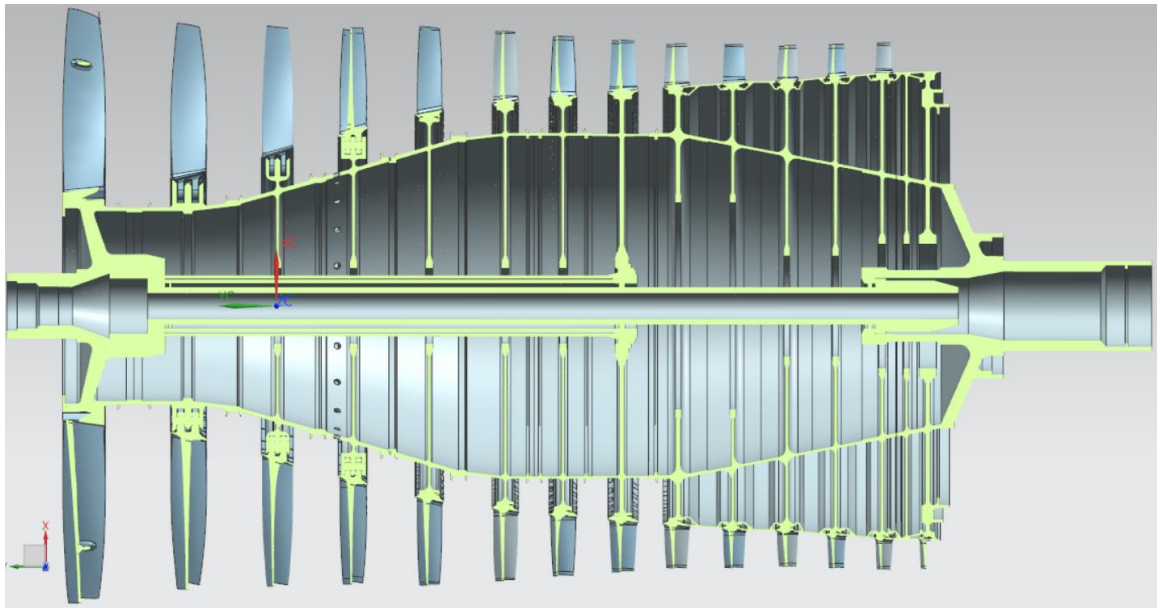
Rys. 4.5. Zdyskretyzowany model tarczy z zaznaczonymi warunkami brzegowymi, gdzie A i C – strefa warunku symetrii, B – strefa ciągnięcia wieńcowego

Dyskretyzację układu przeprowadzono za pomocą funkcji „AMESH” z zadaniem rozmiarem siatki na poziomie 0,005. Koniec tarczy (B) obciążony jest ciągnięciem zgodnie z równaniem (4.1).

$$\sigma_w = \frac{(F_t + F_z) n_l}{2 \pi r_{max} h_{min}}, \quad (4.1)$$

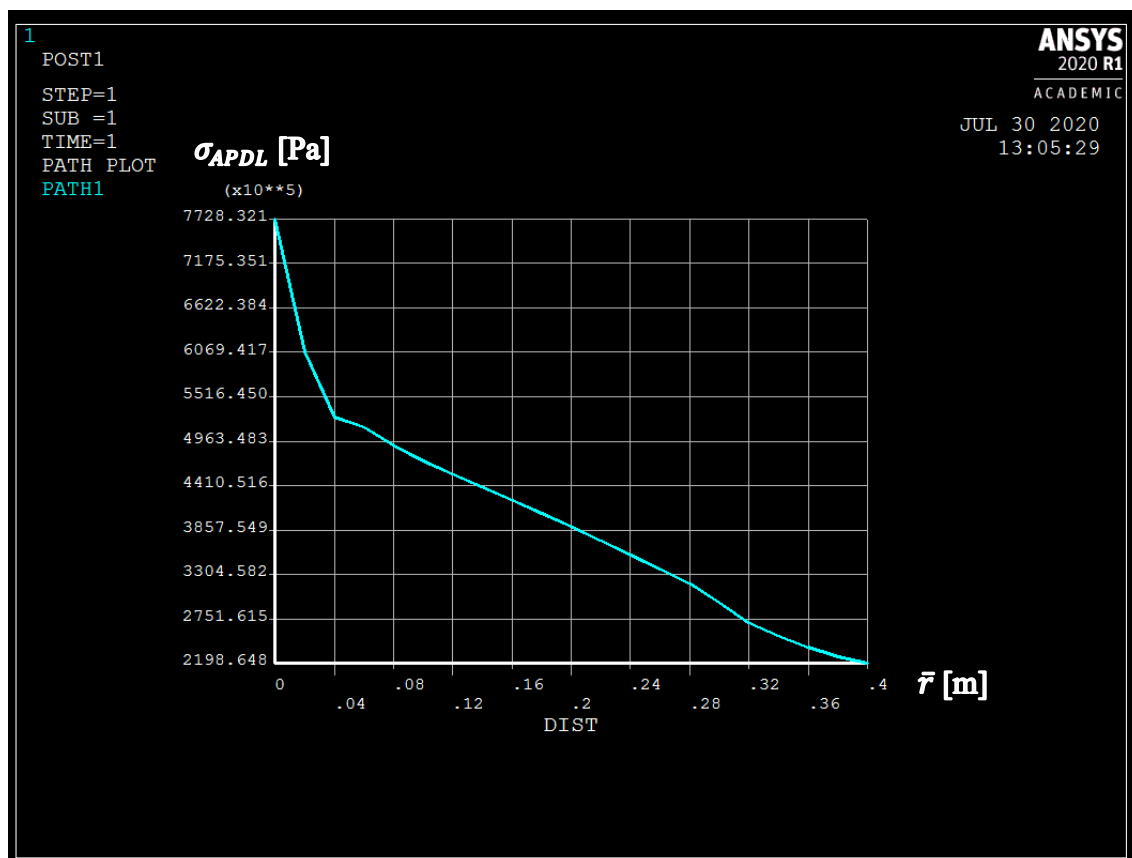
gdzie: σ_w – ciągnienie wieńcowe,
 F_t – siła odśrodkowa od masy części zamkowej tarczy,
 F_z – siła odśrodkowa od masy łopatk,
 n_l – liczba łopatek,
 r_{max} – promień zewnętrzny tarczy,
 h_{min} – grubość tarczy.

Ciągnienie zostało zrealizowane za pomocą funkcji SFL (Specifies Surface Loads). Prędkość obrotową układu tarczy zadano za pomocą funkcji OMEGA. Warunek symetrii przemieszczeń zadany jest w miejscu łączenia bębna z bębniem kolejnego stopnia (na rys. 4.5 strefa A i B). Takie podejście pozwala rozłożyć analizę całego wirnika sprężarki (rys. 4.6) na poszczególne stopnie. Założono, że połączenie stopni jest dostatecznie sztywne i warunek symetrii w tym obszarze nie wprowadza znaczących błędów.



Rys. 4.6. Wirnik do optymalizacji zespołu sprężarkowego

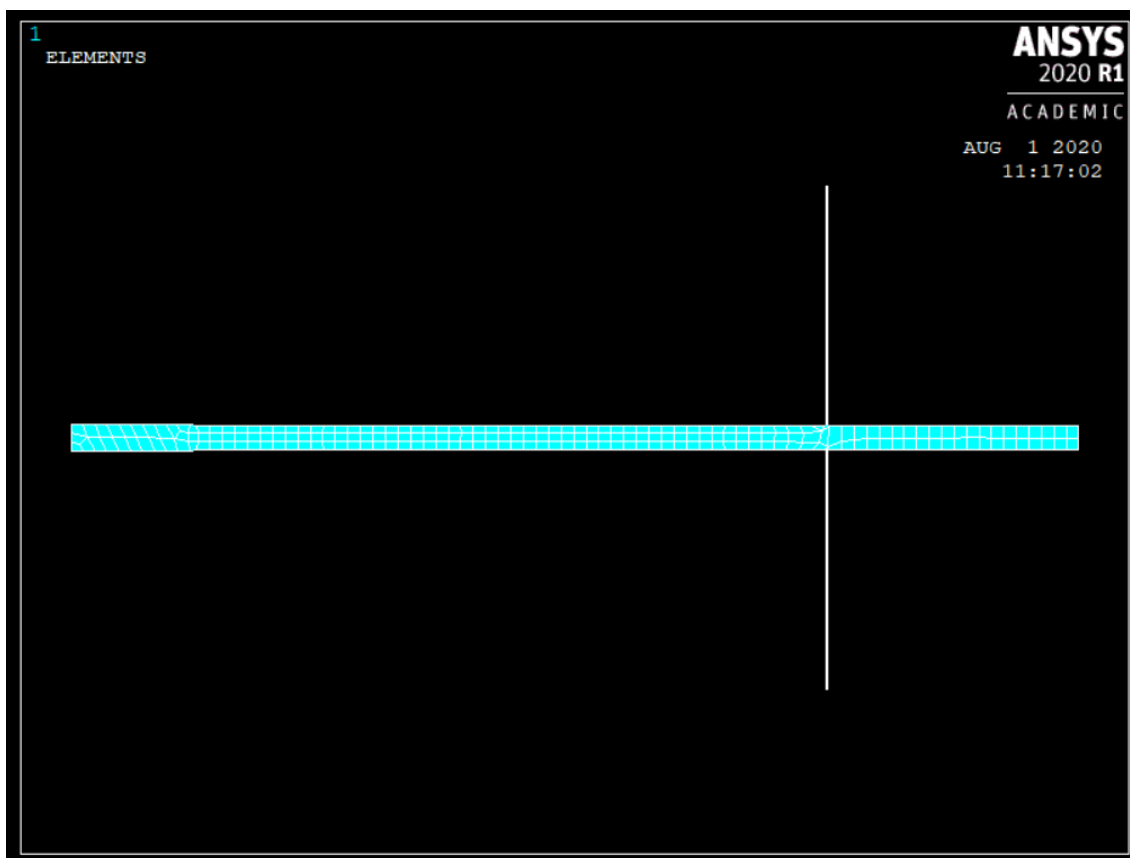
Rozwiązując problem numeryczny, w postprocesorze wygenerowano trzy ścieżki, wzdłuż których odczytano dane niezbędne do obliczeń. Każda ze ścieżek jest określona przez dwa punkty. Pierwsza ścieżka opiera się o środek podstawy piasty i środek końcówki tarczy (rys. 4.3, niebieska linia), druga biegnie przez środek lewej (rys. 4.3, czerwona linia), a trzecia przez środek prawej części bębnowej (rys. 4.3, zielona linia). Pozwoliło to uzyskać rozkład naprężeń obliczony zgodnie z hipotezą największej energii odkształcenia postaciowego (teoria Hubera–Misesa–Hencky’ego) w przekrojach niezbędnych do profilowania konstrukcji. Przykładowy przebieg naprężeń zredukowanych wzdłuż pierwszej ścieżki przedstawiono na rysunku 4.7.



Rys. 4.7. Rozkład naprężeń wzdłuż pierwszej ścieżki

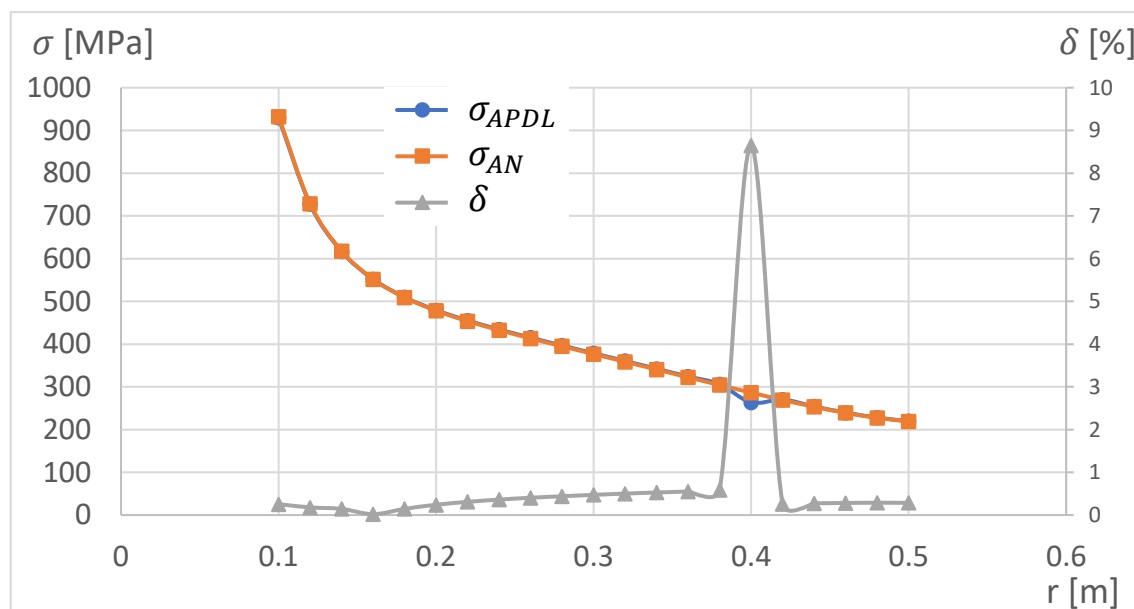
Ścieżka części tarczowej została podzielona na dwadzieścia węzłów, a ścieżki części bębnowej na pięć.

Utworzony model numeryczny poddano walidacji za pomocą stochastycznie wybranych parametrów geometrycznych modelu i obciążeń. Wyniki analizy numerycznej porównano z obliczeniami analitycznymi, zbliżając geometrię modelu do tarczy prostej. Zbieżność tę zapewniono poprzez zmniejszenie grubości piasty do 5,1 mm przy grubości tarczy 5 mm, zmniejszeniu grubości bębna do 0,5 mm i promieni zaokrągleń R_i do 5 mm. Można zauważyć, że uzyskany model odpowiada układowi tarczy prostej (jak na rys. 4.8).



Rys. 4.8. Model tarczy o stałej grubości przygotowany do walidacji programu

Na rysunek 4.9 przedstawiono błąd względny (δ) i rozkład naprężeń dla obliczeń analitycznych (σ_{AN}) i obliczeń wykonanych przez APDL (σ_{APDL}).



Rys. 4.9. Walidacja kodu napisanego w języku APDL

Dla promienia $r = 0,4$ m wystąpił największy błąd, z powodu występowania w tym obszarze dodatkowego elementu konstrukcyjnego (połączenie tarcza-bęben). W pozostałej części błąd nie przekracza 0,6%.

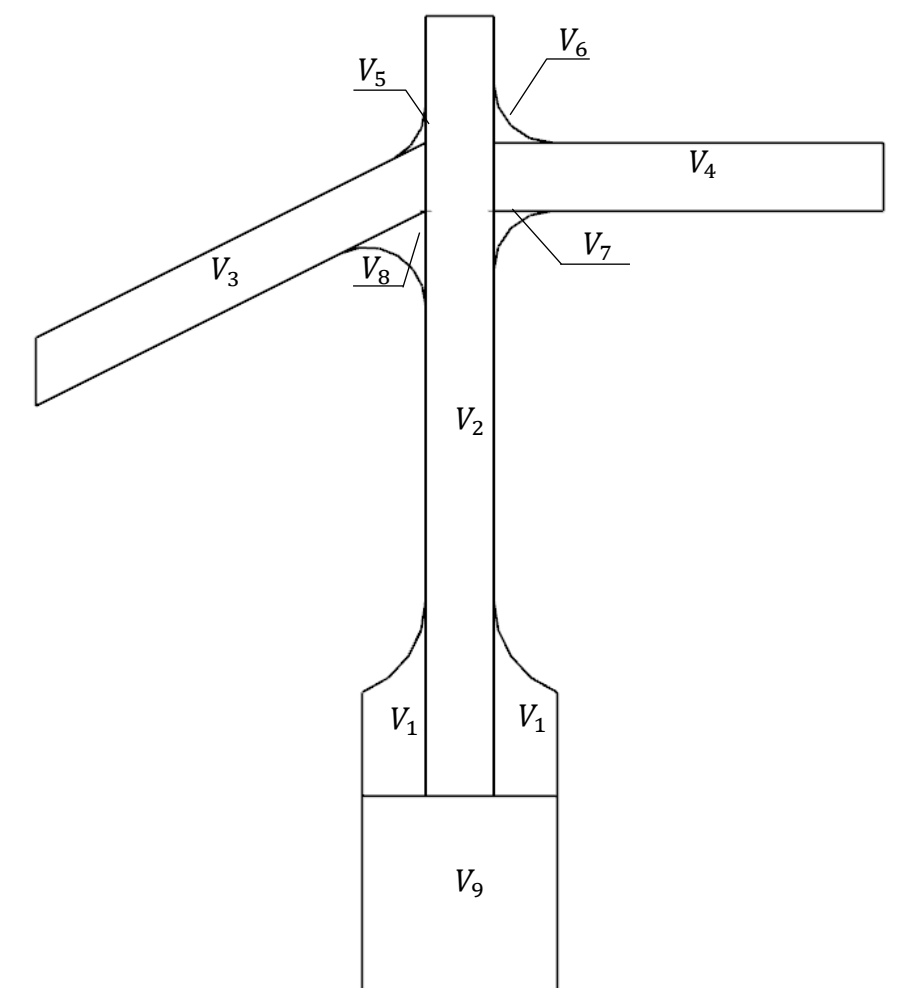
4.2. FUNKCJA CELU I FUNKCJA KARY

Funkcja celu jest to przyjęte kryterium, podlegające optymalizacji w oparciu o zmienne decyzyjne. Funkcją celu może być między innymi koszt eksploatacji, cena detalu, masa itp. Z kolei funkcja kary pozwala zastosować metody poszukiwania ekstremum funkcji celu bez ograniczeń obszaru rozwiązań, jednakże realnie istniejące ograniczenia powinny być uwzględnione w wyrażeniu definiującym nakładaną karę.

Celem optymalizacji realizowanej w oparciu o przedstawiony algorytm jest minimalizacja masy tarczy przy jednoczesnym braku przekroczenia naprężeń dopuszczalnych. Za spełnienie warunku wytrzymałościowego odpowiedzialna jest funkcja kary.

FUNKCJA CELU

W celu obliczenia masy tarczę podzielono zgodnie z Rys. 4.10 na dziewięć części.



Rys. 4.10. Model tarczy z oznaczonymi kolejnymi obszarami

Objętość części 1-8 policzono w oparciu o metody przedstawione poniżej i na rys. 4.11 (na przykładzie lewego górnego zaokrąglenia połączenia tarcza-bęben (V_5)):

$$V_5 = V_{51} - V_{52}, \quad (4.2)$$

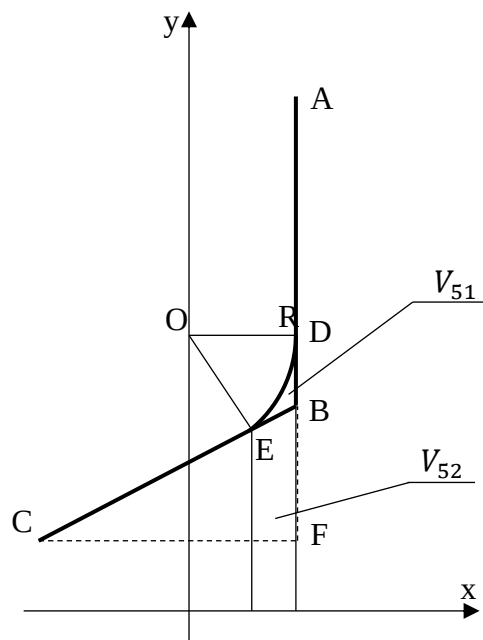
$$V_{51} = \pi \int_a^b y^2 dx = \pi \int_{x_F}^R \left(-\sqrt{R^2 - x^2} + y_D \right)^2 dx, \quad (4.3)$$

$$V_{52} = \frac{\pi}{3} (R + x_E) (rb^2 + rb y_E + y_E^2). \quad (4.4)$$

Całka V_{51} została rozwiązana numerycznie i oznacza ona objętość bryły ograniczonej łukiem ER i osią obrotu, a V_{52} to objętość stożka ściętego, ograniczona zewnętrzną powierzchnią lewej części bębnowej na odcinku EB (rys. 4,11). W celu określenia granic całkowania i wymiarów stożka (określeniu współrzędnych punktów E i D) rozwiązano układ równań:

$$\begin{cases} x_E^2 + (y_E - y_D)^2 = R^2 \\ y_E = t x_E + b \\ y_E = -\frac{1}{t} x_E + y_D \end{cases}. \quad (4.5)$$

Pozostałe oznaczenia są zgodne z modelem (rys. 4.3).



Rys. 4.11. Oznaczenia punktów niezbędnych do wyznaczenia objętości bryły powstałej przez obrót wokół osi OX

Współrzędne punktów A,B,C:

$$\begin{aligned} & A(R, r_max), \\ & B(R, rb + hb), \\ & C(lbl, rbl + hb), \\ & tg\alpha = t = \frac{rb - rbl}{lbl}. \end{aligned} \tag{4.6}$$

Z punktu B i prostej CB:

$$\begin{aligned}rb + hb &= tR + b, \\ b &= rb + hb - tR.\end{aligned}\tag{4.7}$$

Rozwiązanie układu (4.5) daje następujące rozwiązania:

$$\left\{ \begin{array}{l} x_E = -\frac{R t}{\sqrt{t^2 + 1}} \\ y_E = \frac{b \sqrt{t^2 + 1} - R t^2}{\sqrt{t^2 + 1}} \\ y_D = \frac{b \sqrt{t^2 + 1} - R t^2 - R}{\sqrt{t^2 + 1}} \end{array} \right. \cup \left\{ \begin{array}{l} x_E = \frac{R t}{\sqrt{t^2 + 1}} \\ y_E = \frac{b \sqrt{t^2 + 1} + R t^2}{\sqrt{t^2 + 1}} \\ y_D = \frac{b \sqrt{t^2 + 1} + R t^2 + R}{\sqrt{t^2 + 1}} \end{array} \right. . \quad (4.8)$$

Pierwsze rozwiązanie odpowiada przypadkowi, gdy łuk znajduje się poniżej punktu B (prawdziwe dla V_7 i V_8), drugie, gdy łuk jest powyżej (dla V_1 , V_5 i V_6). Podstawiając (4.8) do równań (4.2)-(4.4), otrzymano objętość odpowiednich obszarów. Objętości od V_5 do V_8 zostały policzone według powyższych zależności. V_1 policzono analogicznie, jednak z pominięciem odjęcia części walcowej. Objętości V_2 i V_9 policzono ze wzorów na objętość walca. Objętość V_3 i V_4 obliczono zgodnie z równaniami (4.9)-(4.11) na objętość ściętego stożka (na przykładzie V_3):

$$V_{31} = \frac{\pi}{3} l b l [(r b l + h b)^2 + (r b l + h b)(r b + h b) + (r b + h b)^2], \quad (4.9)$$

$$V_{32} = \frac{\pi}{3} l b l (r b l^2 + r b l r b + r b^2), \quad (4.10)$$

$$V_3 = V_{31} - V_{32}. \quad (4.11)$$

Całkowitą objętość tarczy obliczono zgodnie z równaniem:

$$V = \sum_{i=1}^8 V_i - V_9. \quad (4.12)$$

Masę tarczy obliczono, znając gęstość materiału. W rozpatrywanym przypadku jest to stop tytanu o gęstości $\rho = 4430 \frac{\text{kg}}{\text{m}^3}$.

Określona funkcja celu pozwala sprowadzić wymiary tarczy (15 parametrów) do jednej wartości liczbowej (masa/objętość). Dzięki temu możliwa jest optymalizacja konstrukcji bez wykorzystania bardziej czasochłonnych algorytmów do optymalizacji wielokryterialnej. Tak zdefiniowana funkcja celu nie pozwala uwzględnić spełnienia warunku wytrzymałościowego. W tym celu zastosowano funkcję kary.

FUNKCJA KARY

Funkcję kary stosuje się w celu usunięcia ograniczeń przestrzeni rozwiązań funkcji celu. Dodatkowo funkcja kary pozwala uwzględnić dodatkowe warunki niezawarte w funkcji celu, bez jej nadmiernej komplikacji, a także bez ograniczania przestrzeni rozwiązań dopuszczalnych. Pożądane jest, aby przestrzeń zmiennych decyzyjnych była nieskończona, ponieważ ułatwia to numeryczne badanie funkcji. W analizowanym przypadku, gdy naprężenia obliczone za pomocą SSN przekroczyły naprężenia dopuszczalne w dowolnym punkcie dowolnej ścieżki, do wyznaczonej masy dodano karę obliczaną zgodnie z zależnością (4.13).

$$kara = k * (\sigma_{red} - \sigma_{dop}), \quad (4.13)$$

gdzie: σ_{red} – naprężenia wyznaczone za pomocą SSN,
 σ_{dop} – naprężenia dopuszczalne,
 k – stała wzmacniająca karę równa 1000.

Zależność tę wyznaczono empirycznie. Dla mniejszych wartości współczynnika wzmocnienia kary (k) funkcja miała tendencję do szukania minimum dla konstrukcji, w której przekroczono naprężenia dopuszczalne. Natomiast w przypadku zwiększenia współczynnika k funkcja miała tendencję do wskazywania minimum znacznie oddalonego od rozwiązania podejrzanego za optymalne, czyli gdy naprężenia zredukowane są równe naprężeniom dopuszczalnym.

Wyniki optymalizacji z wykorzystaniem przedstawionych algorytmów po implikacji algorytmu neuronowego (rozdział 5) przedstawiono w rozdziale 6.

5. DOBÓR STRUKTURY I METODY UCZENIA SIECI NEURONOWEJ

W rozdziale 3 wykazano, że SSN są narzędziem zdolnym do zastąpienia lub wspomagania obliczeń numerycznych. Analiza numeryczna dostarczyła przesłanek, że zastosowana struktura sieci prawdopodobnie nie została dobrana optymalnie. W poniższym rozdziale przeprowadzono badania numeryczne w celu doboru optymalnej metody uczenia SSN i jej architektury. Obiekt poddany optymalizacji to część bębnowo-tarczowa sprężarki lotniczego silnika turbinowego. Zmienne decyzyjne, funkcja celu i kary zostały opisane w rozdziale 4.

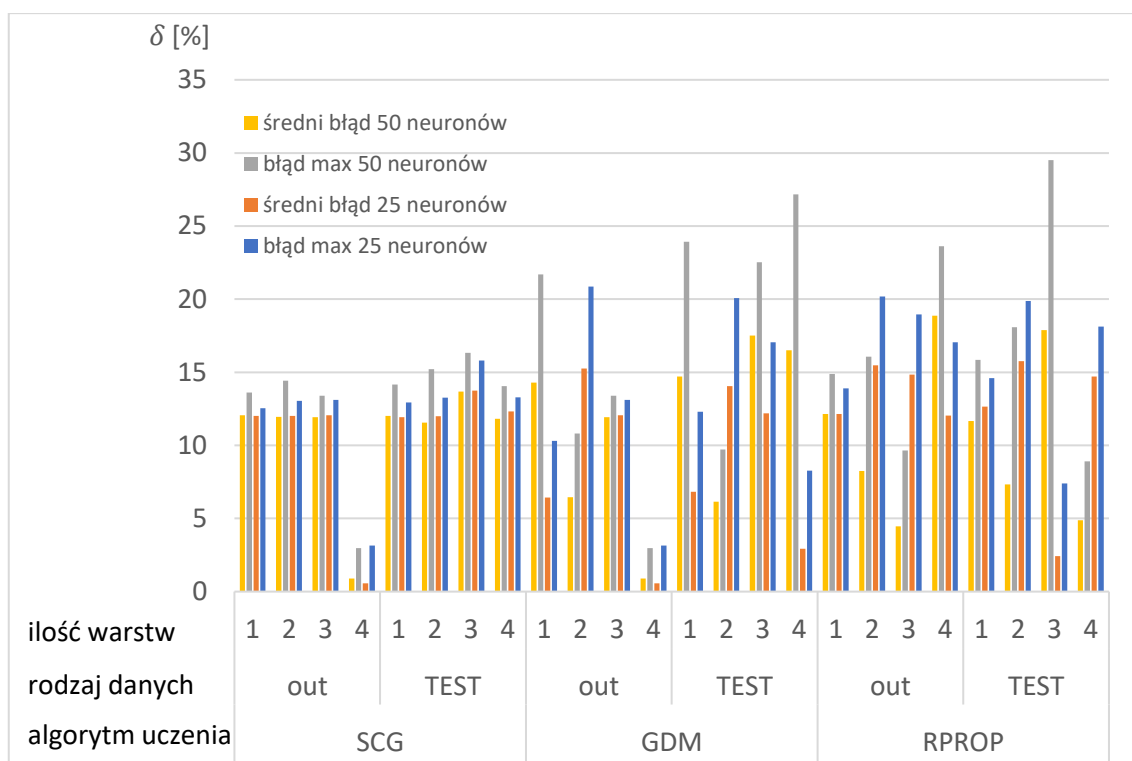
5.1. DOBÓR METODY UCZENIA

Dobór metody uczenia oparty został na przeglądzie literatury. Dostarczył on przesłanek, że SSN są narzędziem zdolnym do wspomagania lub zastąpienia klasycznych obliczeń numerycznych. Autor, w wyniku wstępnej analizy numerycznej przedstawionej w pracy [109], uzyskał wiedzę z zakresu problemów doboru optymalnej sieci. Zbudowana sieć nie w pełni spełnia oczekiwania autora w zadaniu optymalizacji. Dlatego w kolejnym etapie pracy przeprowadzono badania numeryczne w celu doboru optymalnej struktury i metody uczenia SSN. Wybrane metody poddano analizie porównawczej. Porównane zostały:

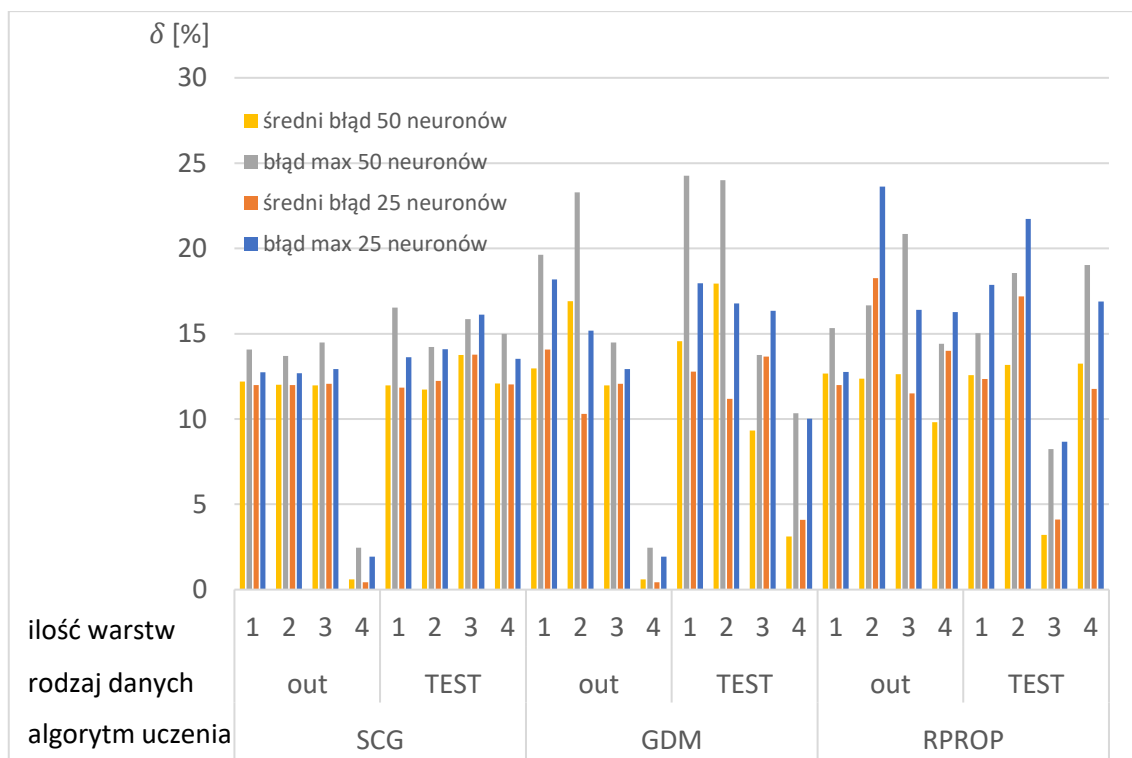
- Metoda gradientów sprzężonych z regulacją (SCG) [109];
- Metoda momentum (GDM) [113];
- Algorytm RPROP [114].

Sieć uczono od jednego do czterech razy. Danymi wykorzystanymi do uczenia SSN i porównywania wyników były naprężenia, które zostały wygenerowane w oparciu o algorytm w języku APDL. Sieć poddawano treningowi do osiągnięcia pięćdziesiątej iteracji algorytmu uczącego bez poprawy wartości wagowych.

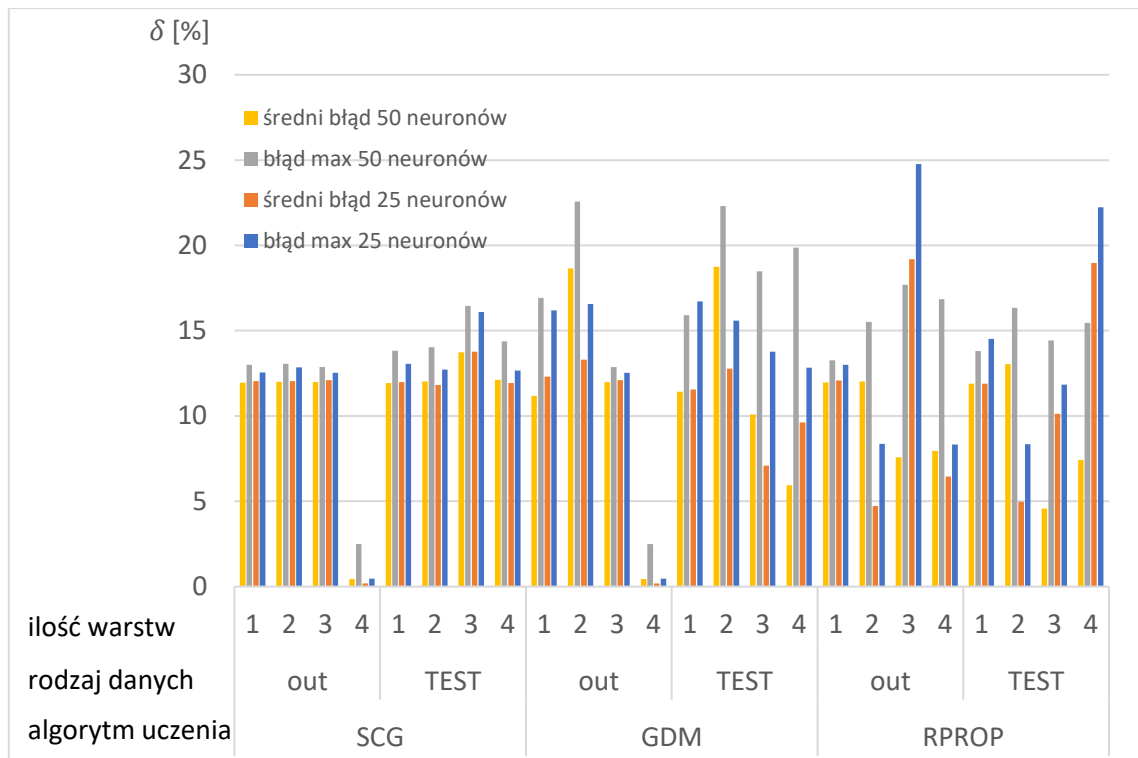
Przeprowadzono obliczenia sieci, wykorzystując od jednej do czterech warstw ukrytych i 25 lub 50 neuronów w każdej warstwie. Z każdej sieci uzyskano 150 wyników, dla których wyznaczono wartość bezwzględną średniego i największego błędu względnego pomiędzy naprężeniami wyznaczonymi przez SSN a algorytmem APDL. Wynik analizy przedstawiono na rysunkach 5.1-5.3, gdzie dane „out” to baza danych uczących SSN, baza „TEST” to nowo wylosowane dane, nie wykorzystywane do procesu uczenia sieci.



Rys. 5.1. Maksymalny i średni błąd względny dla wybranych struktur sieci neuronowych i algorytmów uczących po pierwszym cyklu uczenia



Rys. 5.2. Maksymalny i średni błąd względny dla wybranych struktur sieci neuronowych i algorytmów uczących po drugim cyklu uczenia.



Rys. 5.3. Maksymalny i średni błąd względny dla wybranych struktur sieci neuronowych i algorytmów uczących po trzecim cyklu uczenia

Na powyższych wykresach można zauważyć, iż w analizowanym przypadku:

- niemal wszystkie sieci osiągają większe błędy dla sieci o 50 neuronach w warstwie;
- wielokrotne uczenie sieci ma największy wpływ na poprawę jakości wyników dla algorytmu RPROP;
- algorytm RPROP uzyskuje względnie duże rozbieżności w dokładności obliczeń w zależności od liczby warstw ukrytych, szczególnie w zależności od ich parzystości;
- błędy dla pierwszej iteracji procesu uczenia dla sieci o jednej warstwie ukrytej dla algorytmów SCG i RPROP są zbliżone;
- algorytmy SCG i GDM cechują się gwałtownym spadkiem błędu dla sieci czterowarstwowych;
- algorytm GDM cechuje się małą wrażliwością na ilość warstw ukrytych (z wyjątkiem sieci czterowarstwowej) i powtarzanie procesu uczenia.

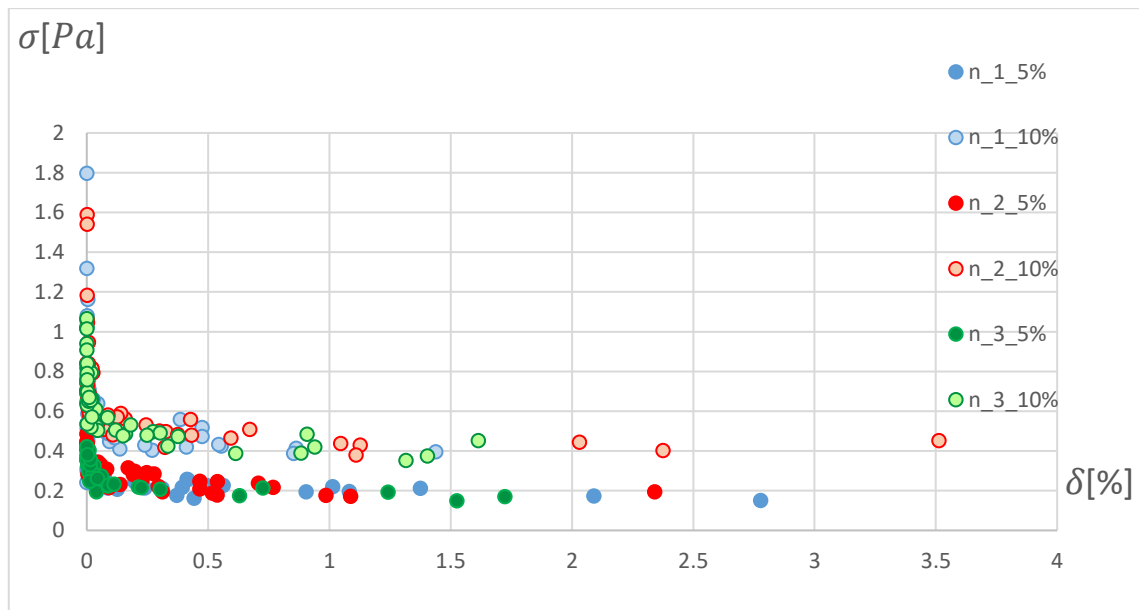
Ponieważ żaden algorytm nie wyróżnia się pod względem jakościowym, to w dalszych badaniach nie wykluczono żadnego z nich. Z tego powodu podjęto próbę zminimalizowania błędu poprzez proces optymalizacji struktury SSN. W wyniku przeprowadzonej analizy stwierdzono brak potrzeby powtarzania procesu uczenia sieci.

5.2. DOBÓR STRUKTURY SSN

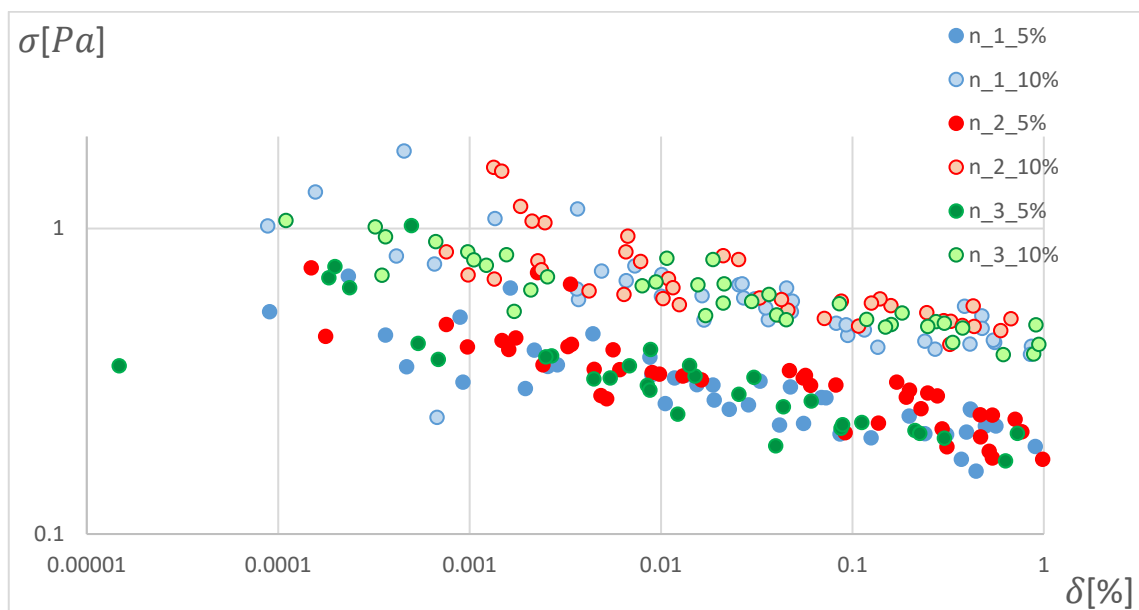
W celu określenia optymalnej struktury SSN przeprowadzono za pomocą algorytmu genetycznego optymalizację dwukryterialną. Badano wpływ ilości warstw ukrytych i neuronów w warstwie na wartość bezwzględną średniego błędu względnego i odchylenia standardowego. Optymalizację przeprowadzono dla dwóch algorytmów uczących (RPROP, SCG i GDM) dobranych na podstawie badań numerycznych, opisanych w poprzednim rozdziale. Algorytm genetyczny wykonywał obliczenia dla 20 pokoleń, optymalizując równolegle 50 sieci neuronowych o zadanej ilości warstw ukrytych. Badano sieci od jednej do trzech warstw ukrytych i od jednego do 50 neuronów w każdej warstwie. Porównywano także dwa przypadki rozmiaru danych uczących dla wartości: $\pm 5\%$ i $\pm 10\%$ względem zmiennych decyzyjnych odpowiadających tarczy wzorcowej. Zmienne te zostały szerzej opisane w punkcie 4.1. Uzyskane wyniki procesu optymalizacji przedstawiono poniżej. Są to naprężenia obliczone w siódmym węźle pierwszej ścieżki, liczone za pomocą SSN i algorytmu APDL. Dla pozostałych węzłów zauważono zbliżoną korelację do wyników uzyskanych dla węzła siódmego.

Z uwagi na stochastyczny proces optymalizacji (algorytm genetyczny) i dualną funkcję celu (jednoczesna minimalizacja odchylenia standardowego (σ) i błędu względnego (δ)), otrzymane wyniki w postaci frontów Pareto różnią się od typowych. Zamiast pojedynczej linii łączącej punkty optymalne w sensie Pareto mamy do czynienia z obszarem rozwiązań potencjalnie optymalnych. Wyniki, w postaci punktów rozwiązań potencjalnie równoważnych, przedstawiono w następujący sposób:

- dla algorytmu GDM na rys. 5.4 i 5.5;
- dla algorytmu SCG na rys. 5.6-5.9;
- dla algorytmu RPROP na rys. 5.10-5.13.

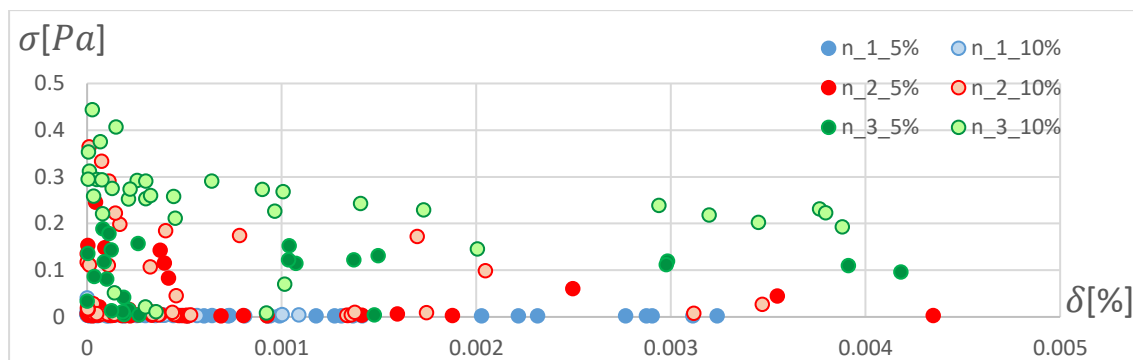


Rys. 5.4. Fronty Pareto dla wszystkich sieci uczonych za pomocą algorytmu GDM

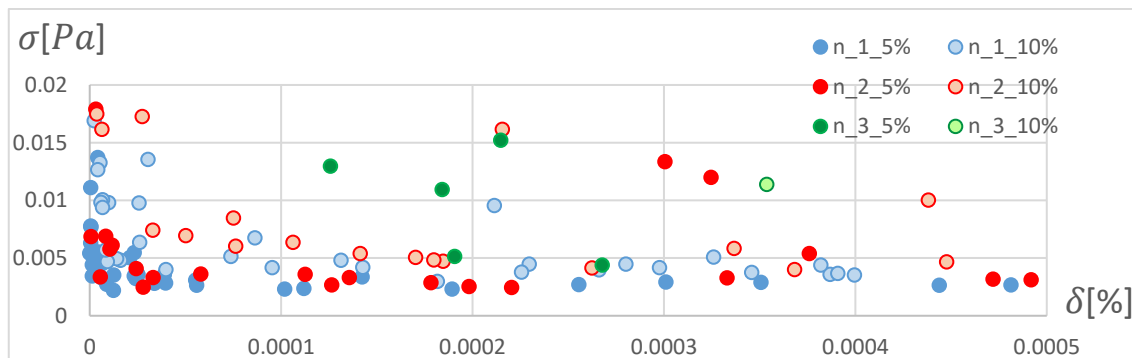


Rys. 5.5. Fronty Pareto w skali logarytmicznej dla wszystkich sieci uczonych za pomocą algorytmu GDM. Zbliżenie na największą dokładność

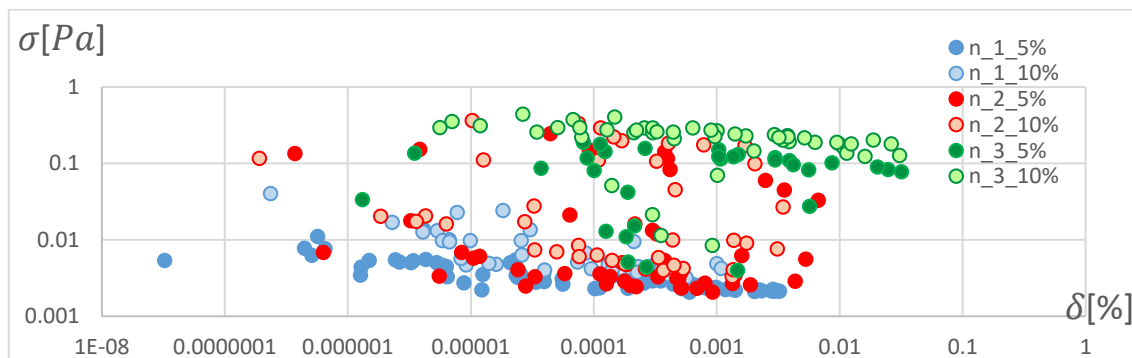
Na podstawie rozkładu błęd wyznaczenia naprężeń przez SSN i APDL i odchylenia standardowego stwierdzono, że algorytm GDM nie jest w stanie zrealizować zadania niezależnie od struktury sieci. Zaobserwowano wrażliwość algorytmu tylko na rozpiętość bazy danych. Na rysunku 5.5 wyraźnie widać dwa pasma odpowiadające rozpiętości bazy danych 10% i 5%. Przy optymalizacji zauważono brak wrażliwości na dokładność uczenia w zależności od ilości warstw ukrytych i ilości neuronów w warstwie.



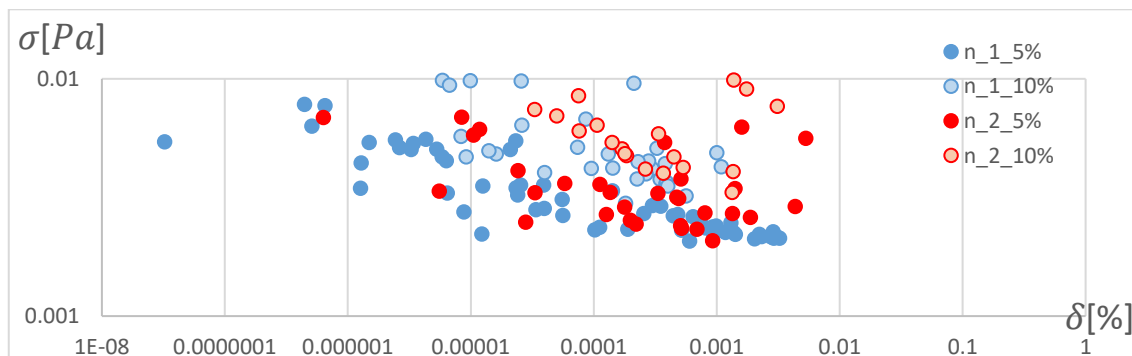
Rys. 5.6. Fronty Pareto dla wszystkich sieci uczonych za pomocą algorytmu SCG



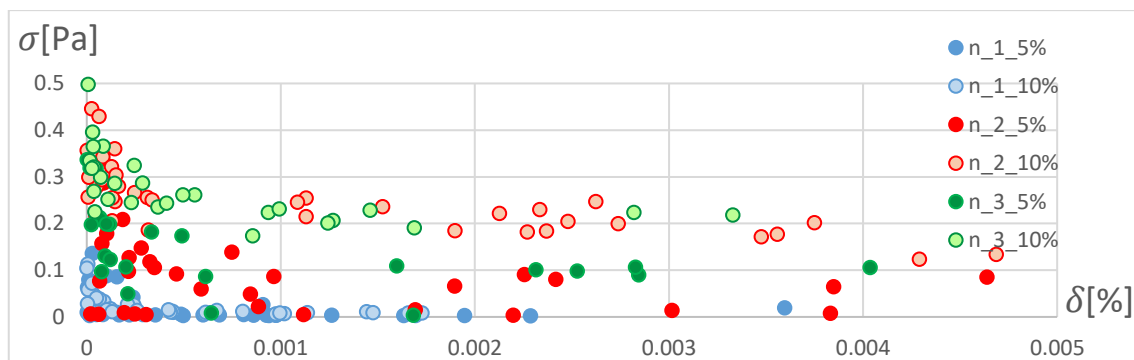
Rys. 5.7. Fronty Pareto dla wszystkich sieci uczonych za pomocą algorytmu SCG. Zbliżenie na największą dokładność



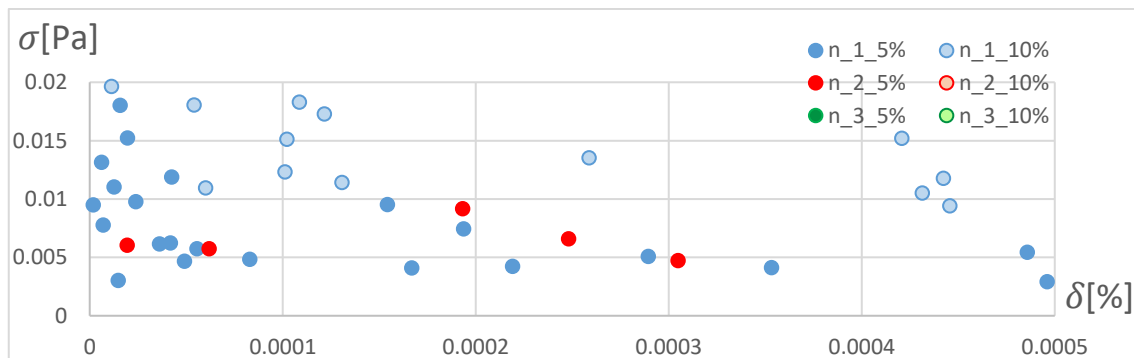
Rys. 5.8. Fronty Pareto w skali logarymicznej dla wszystkich sieci uczonych za pomocą algorytmu SCG



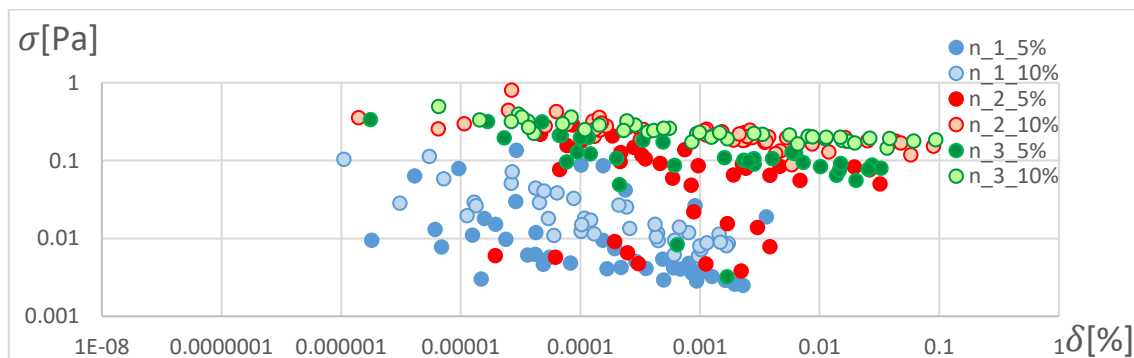
Rys. 5.9. Fronty Pareto w skali logarymicznej dla dwóch najlepiej dopasowanych sieci uczonych za pomocą algorytmu SCG



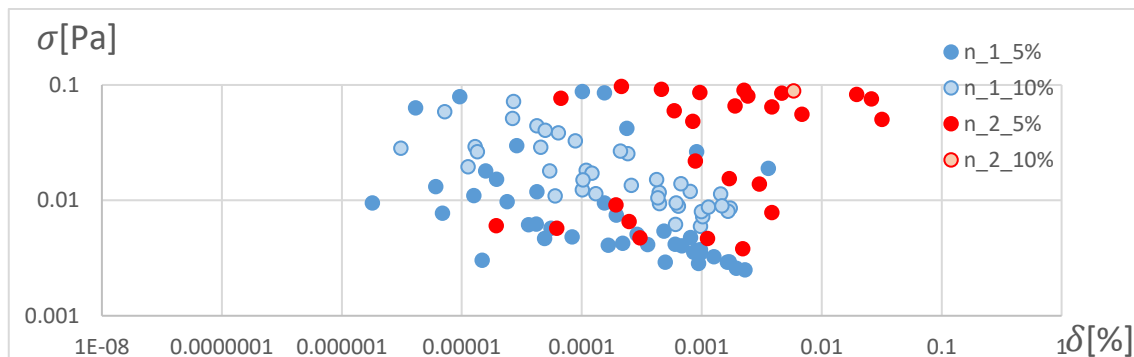
Rys. 5.10. Front Pareto dla wszystkich badanych sieci uczonych za pomocą algorytmu RPROP



Rys. 5.11. Front Pareto dla wszystkich badanych sieci uczonych za pomocą algorytmu RPROP. Zbliżenie na największą dokładność



Rys. 5.12. Front Pareto w skali logarytmicznej dla wszystkich badanych sieci uczonych algorytmem RPROP



Rys. 5.13. Front Pareto w skali logarytmicznej dla dwóch najlepiej dopasowanych sieci uczonych za pomocą algorytmu RPROP

Na rys. 5.6-5.13 przedstawiono fronty Pareto dla wybranych sieci neuronowych dla dwóch zmiennych (wartości bezwzględnej średniego błędu względnego δ i jego odchylenia standardowego σ). W wyniku przeprowadzonych badań zauważono, że wzrost ilości warstw ukrytych zmniejsza wpływ rozpiętości bazy danych uczących na dokładność wyników. Jednak dokładność ta jest tym mniejsza, im więcej sieć posiada warstw. Dla obydwu metod uczących (SCG i RPROP) punkty frontu Pareto dla sieci neuronowej o jednej warstwie ukrytej przy rozpiętości danych $\pm 5\%$ mają najmniejsze wartości błędu względnego. W przypadku algorytmu RPROP dla sieci o jednej i dwóch warstwach ukrytych (dla danych $\pm 5\%$) fronty Pareta układają się równolegle do siebie, tworząc pasma o zbliżonej szerokości, przy mniejszym błędzie średnim sieci o pojedynczej warstwie ukrytej. Dla algorytmu SCG pasmo frontu dla sieci jednowarstwowej wygląda podobnie jak dla sieci uczonej algorytmem RPROP, jednak pasmo dla sieci dwuwarstwowej jest dużo szersze (Rys. 5.6 i 5.10). Jego środek znajduje się bliżej pasma sieci jednowarstwowej, a jego wartości minimalne odpowiadają jej wartościom. Przyczyny zmiany szerokości i położenia frontu można szukać w różnej ilości neuronów w warstwie, co przedstawiono w tabelach 5.1 i 5.2. Ilości neuronów w danej sieci są wartościami średnimi, zaokrąglonymi do liczb całkowitych, uzyskanymi w procesie optymalizacji.

Tabela 5.1. Porównanie optymalnej ilości neuronów w warstwie ukrytej dla wybranych algorytmów uczących i różnych rozpiętości bazy danych

Ilość warstw ukrytych		1		2				3					
Rozpiętość bazy danych		5%	10%	5%		10%		5%			10%		
Średnia ilość neuronów w warstwie	SCG	2	3	5	8	4	7	23	27	8	21	19	5
	RPTOP	3	3	12	6	16	6	17	22	9	33	23	6

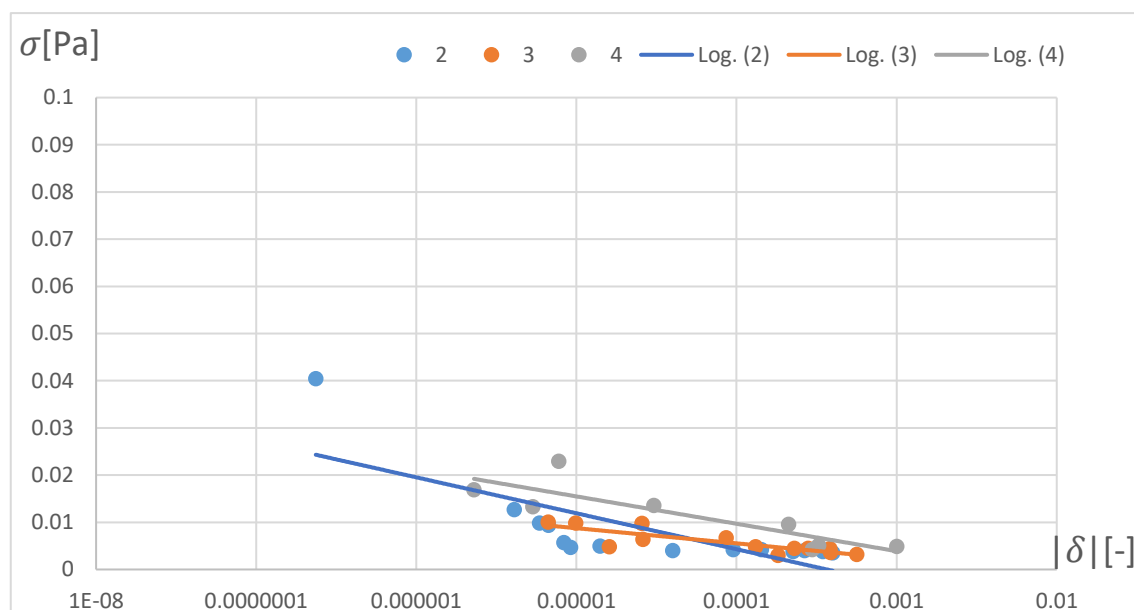
Poniżej, w tabeli 5.2 zestawiono uśrednione wartości bezwzględne błędu względnego i jego odchylenia standardowego dla wybranych algorytmów uczących oraz ilości neuronów w warstwie. Wyniki uzyskano dla sieci o jednej warstwie ukrytej, ponieważ sieć ta cechuje się mniejszym błędem od sieci o większej liczbie warstw.

Tabela 5.2. Wartość bezwzględna średniego błędu względnego $|\delta|$ i jego odchylenie standardowe σ dla wybranych algorytmów uczących i ilości neuronów w jednej warstwie ukrytej

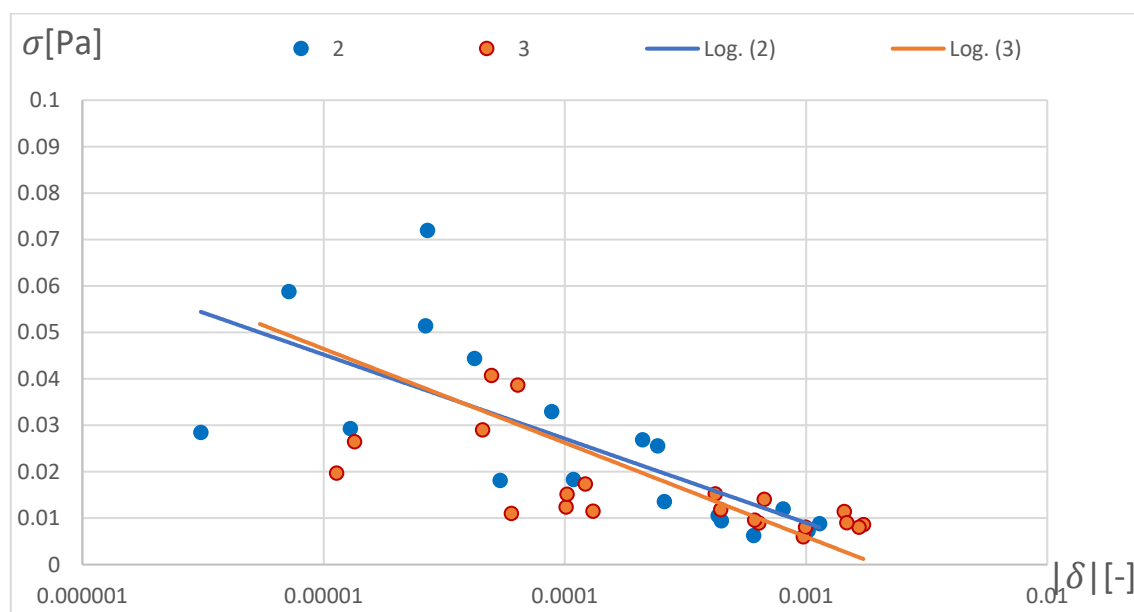
Ilość neuronów	SCG		RPROP	
	$ \delta [-]$	σ [Pa]	$ \delta [-]$	σ [Pa]
2	1,30E-04	1,30E-04	3,07E-04	2,53E-02
3	1,79E-04	1,79E-04	5,34E-04	5,34E-04
4	3,30E-04	3,30E-04	-	-

Algorytm RPROP dla jednej warstwy ukrytej, z wyłączeniem sieci o dwóch neuronach, gorzej dopasowuje się do bazy danych uczących niż algorytm SCG. Najmniejszym błędem względnym i najmniejszym odchyleniem standardowym charakteryzuje się sieć uczona algorytmem SCG o 2 neuronach w warstwie ukrytej. Z połączenia $|\delta|$ i σ na podstawie linii trendu określić można jakościowo średnie

dopasowanie SSN do wybranej bazy danych, co przedstawiono na Rys. 5.14 i 5.15. Wartość bezwzględna średniego błędu względnego $|\delta|$ przedstawiono na skali logarytmicznej.



Rys. 5.14. Front Pareto w zależności od ilości neuronów w warstwie ukrytej z zaznaczonymi liniami trendu. Sieć uczona metodą SCG

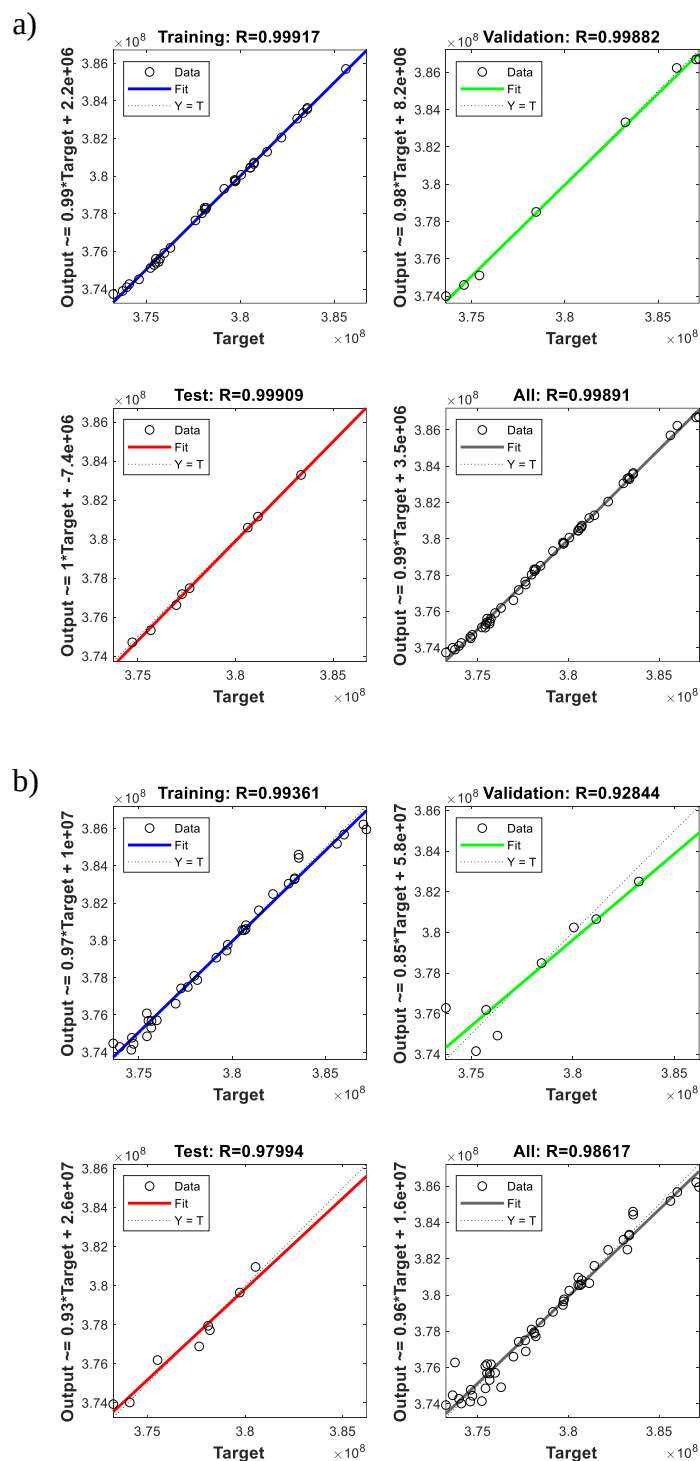


Rys. 5.15. Front Pareto w zależności od ilości neuronów w warstwie ukrytej z zaznaczonymi liniami trendu. Sieć uczona metodą RPROP

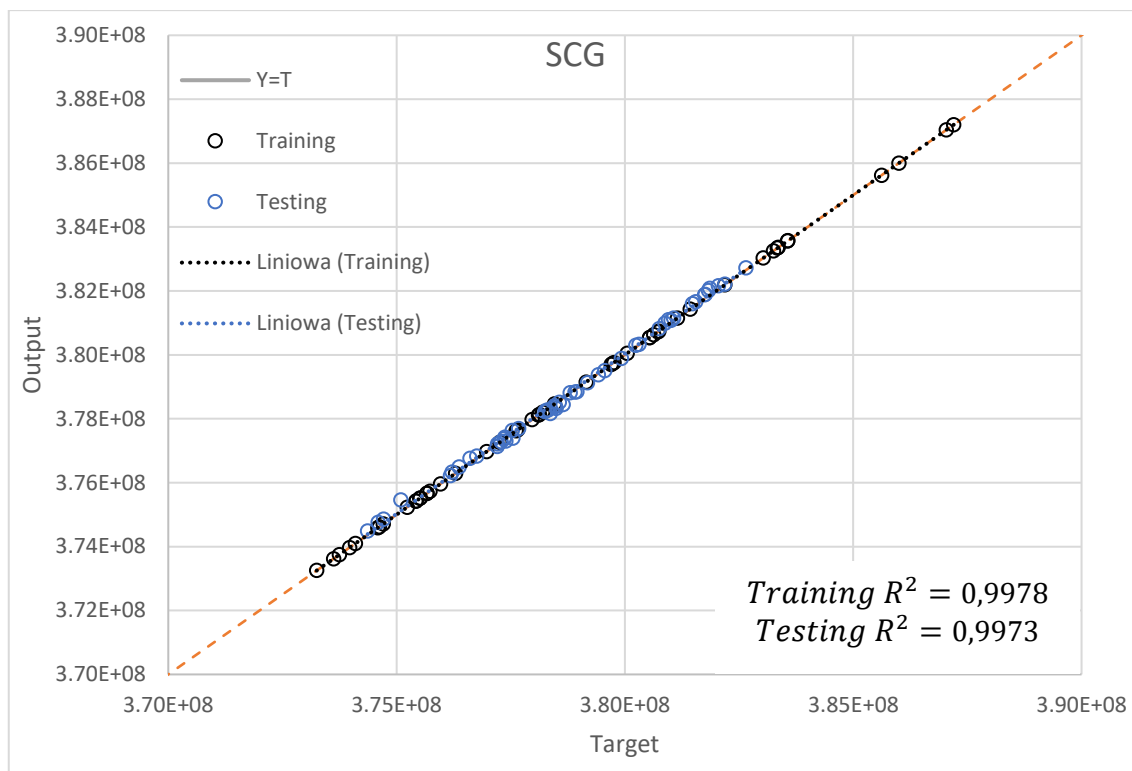
Porównano także współczynnik determinacji (R^2) dla algorytmu SCG i RPROP o jednej warstwie ukrytej i od 2 do 4 neuronów dla bazy danych $\pm 5\%$. Na

Na rys. 5.16-5.18 przedstawiono dokładność uczenia wybranych algorytmów dla najlepiej dopasowanych sieci neuronowych. Wynosi ona odpowiednio $R^2 = 0.9978$ i $R^2 = 0.9725$ dla algorytmów SCG (NN 20-2-1) i RPROP (NN 20-3-1).

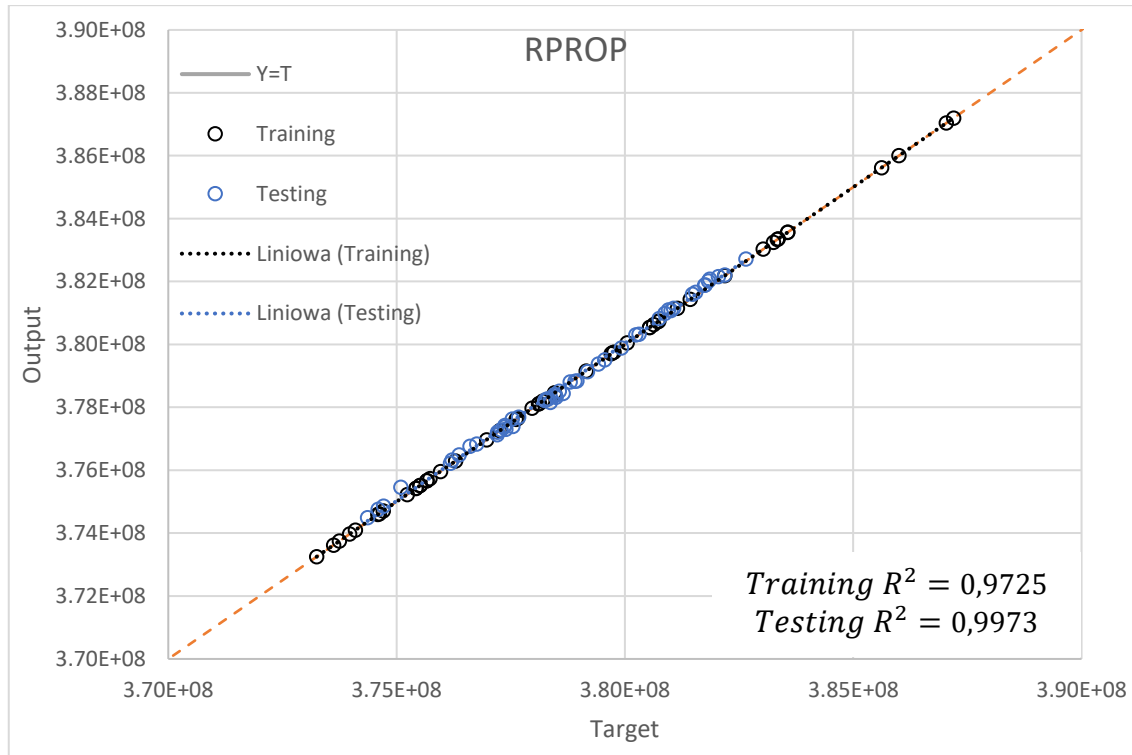
Dokładność obliczono na podstawie nowo wylosowanego zbioru danych, niewykorzystywanego w procesie uczenia sieci.



Rys. 5.16. Dopasowanie danych uczących dla algorytmu a) SCG (NN 20-2-1) i b) RPROP (NN 20-3-1), gdzie dane *input* to naprężenia wyliczone przez APDL, a *output* to dane przewidziane przez SSN.



Rys. 5.17. Porównanie dopasowania danych wykorzystanych do uczenia i sprawdzających dla algorytmu SCG (NN 20-2-1), gdzie dane *input* to naprężenia wyliczone przez APDL, a *output* to dane przewidziane przez SSN.



Rys. 5.18. Porównanie dopasowania danych wykorzystanych do uczenia i sprawdzających dla algorytmu RPROP (NN 20-3-1), gdzie dane *input* to naprężenia wyliczone przez APDL, a *output* to dane przewidziane przez SSN.

Na podstawie powyższej analizy, zgodnie z podejściem Pareto, wybrano sieć neuronową o jednej warstwie ukrytej z dwoma neuronami, uczoną za pomocą algorytmu SCG z wykorzystaniem bazy danych rozpiętości $\pm 5\%$, jako optymalną do danego zadania. Odrzucono sieć o trzech i czterech neuronach, ponieważ mają one kilka punktów znacznie odbiegających od średniej, co zwiększa odchylenie standardowe.

Przeprowadzona analiza literatury, obliczenia i analiza wyników wykazały, że możliwe jest wykorzystanie Sztucznych Sieci Neuronowych do rozwiązywania problemów inżynierskich i naukowych. Przeprowadzono optymalizację metody uczenia i struktury Sztucznej Sieci Neuronowej w zastosowaniu do badań elementów wirujących maszyn wirnikowych na przykładzie turbinowego silnika lotniczego. Wykonano badania trzech algorytmów uczenia SSN (GDM, SCG i RPROP). W wyniku przeprowadzonej analizy wykresów okazało się, że GDM cechował się najgorszym dopasowaniem zarówno do danych uczących, jak i testowych. Na wyniki jakościowe nie miało istotnego wpływu powtarzanie procesu uczenia sieci. Obliczenia zespołu sprężarkowego wykazały, że algorytmy SCG i RPROP cechują się porównywalną dokładnością.

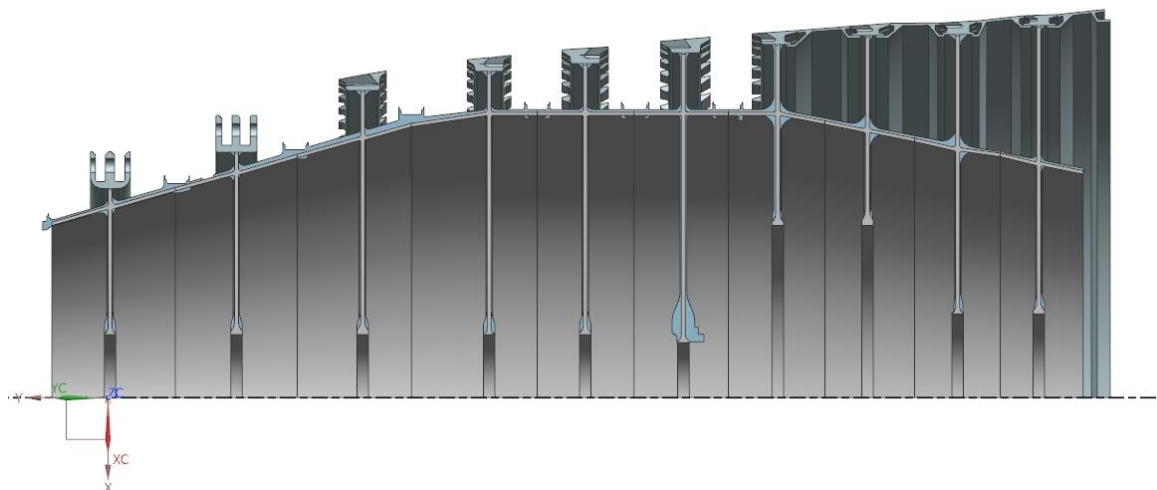
Przedstawiono koncepcję wykorzystania połączenia języków APDL i MATLAB w procesie optymalizacji zespołu sprężarkowego. Optymalizacja struktur sieci za pomocą algorytmu genetycznego dała możliwość wyznaczenia frontów Pareta dla optymalnych struktur sieci. Fronty te pogrupowano w zależności od metody uczenia i ilości neuronów w warstwie. Pozwoliło to, zgodnie z podejściem Pareto, wyselekcjonować sieć o optymalnej strukturze i metodzie uczenia. Przeprowadzona analiza umożliwiła wytypowanie optymalnej struktury sieci neuronowej według metody Pareto. Sieć optymalna jest zbudowana z jednej warstwy ukrytej i dwóch neuronów oraz uczona za pomocą metody gradientów sprzężonych (SCG).

Zastosowane w pracy połączenie optymalizacji wielokryterialnej w ujęciu Pareto do predykcji naprężeń celem minimalizacji masy konstrukcji podzespołu silnika lotniczego można uznać za celowe. Optymalizacja wielokryterialna w ujęciu Pareto jest dobrym narzędziem pozwalającym określić optymalną strukturę Sztucznej Sieci Neuronowej (NN 20-2-1, SCG), minimalizując jednocześnie zarówno jej średni błąd, jak i odchylenie standardowe.

6. WYNIKI OPTIMALIZACJI

Korzystając z opisanego wcześniej (w rozdziałach 3-5) algorytmu, przeprowadzono optymalizację wirnika sprężarki silnika AŁ21-F3. Wyniki optymalizacji przedstawiono na rys. 6.1 i w tabeli 6.1.

Kolorem szarym zaznaczono przekrój struktury wirnika po optymalizacji, natomiast kolorem stalowym przekrój wirnika wzorcowego. Wykorzystano programy z załączników 2-11, gdzie program z załącznika 8 – `main_program.m` jest programem nadrzędnym, wywołującym wszystkie pozostałe (z wyjątkiem importu danych). Przykładowe okno wyników programu przedstawiono w Załącznik 17 – Przykładowa odpowiedź programu po optymalizacji. Geometrię utworzono, korzystając z części wcześniej opracowanej geometrii i programu z załącznika 16, tłumaczącego wyniki optymalizacji na geometrię w języku GRIP. Program `resetAPDL` (załącznik 10) służy do sterowania kursorem celem restartu programu APDL.



Rys. 6.1. Model wirnika sprężarki silnika AŁ21-F3

Optymalizowano stopnie od trzeciego do trzynastego z wyłączeniem ósmego. Nieoptymalizowane stopnie mają konstrukcję inną niż bębnowo-tarczowa (2 stopień) lub zastosowanie specjalne (1, 8 i 14 stopień). Stopnie 1 i 14 mają inną konstrukcję, ponieważ w ich części wchodzi łożyska, natomiast stopień 8 służy jako podpora dla tylnej części dzielonego fragmentu wału. Optymalizowano część tarczowo-bębnową z wyłączeniem wzmocnienia wielowypustów i uszczelnień labiryntowych. Część zamkowa także nie podlegała optymalizacji. Celem optymalizacji była minimalizacja masy przy spełnieniu warunku wytrzymałościowego. Do wyznaczenia naprężeń użyto programu ANSYS wywoływanego poprzez instrukcje parametryczne zapisane w języku APDL. W optymalizacji pominięto analizę drgań, analizę zmęczenia i wpływ potencjalnych uszkodzeń. Są to zagadnienia albo zbyt losowe (uszkodzenia), albo potrzebujące tylko kilkukrotnej analizy, co poddaje w wątpliwość zasadność stosowania Sztucznych Sieci Neuronowych w tych przypadkach. W wyniku optymalizacji porównywano objętości stopni przed optymalizacją i po niej, przy założeniu, że wirnik wykonano ze stopu tytanu Ti-6Al-4V.

Tabela 6.1. Wartości parametrów wymiarowych tarcz, wyliczane w trzech cyklach optymalizacji, dla określonych stopni sprężarki. Pogrubiono najlepsze rozwiązania.

nr_{st}	3			4			5			6			7		
nr_{it}	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
R	0,0277	0,0194	0,0130	0,0073	0,0079	0,0125	0,0176	0,0142	0,0080	0,0056	0,0101	0,0101	0,0062	0,0077	0,0250
R_1	0,0028	0,0020	0,0020	0,0026	0,0020	0,0026	0,0020	0,0026	0,0015	0,0018	0,0020	0,0020	0,0024	0,0013	0,0023
R_2	0,0019	0,0016	0,0017	0,0022	0,0018	0,0013	0,0023	0,0010	0,0020	0,0029	0,0011	0,0037	0,0033	0,0023	0,0020
R_3	0,0010	0,0012	0,0013	0,0016	0,0006	0,0011	0,0016	0,0008	0,0010	0,0015	0,0012	0,0011	0,0010	0,0014	0,0011
R_4	0,0016	0,0015	0,0012	0,0020	0,0010	0,0014	0,0017	0,0010	0,0011	0,0010	0,0008	0,0017	0,0013	0,0007	0,0010
r_x	0,0830	0,0746	0,0681	0,0624	0,0630	0,0676	0,0728	0,0694	0,0631	0,0606	0,0652	0,0652	0,0612	0,0628	0,0803
h_b	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030
nr_{st}	9			10			11			12			13		
nr_{it}	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
R	0,0116	0,0149	0,0207	0,0264	0,0114	0,0103	0,0133	0,0060	0,0069	0,0120	0,0119	0,0124	0,0079	0,0104	0,0278
R_1	0,0032	0,0032	0,0030	0,0023	0,0023	0,0026	0,0021	0,0015	0,0017	0,0025	0,0013	0,0026	0,0013	0,0016	0,0020
R_2	0,0037	0,0034	0,0024	0,0017	0,0029	0,0036	0,0018	0,0022	0,0024	0,0022	0,0015	0,0024	0,0013	0,0022	0,0037
R_3	0,0014	0,0019	0,0017	0,0020	0,0023	0,0014	0,0014	0,0013	0,0010	0,0013	0,0014	0,0011	0,0009	0,0015	0,0015
R_4	0,0017	0,0010	0,0017	0,0010	0,0015	0,0011	0,0010	0,0010	0,0013	0,0011	0,0010	0,0017	0,0013	0,0011	0,0009
r_x	0,1618	0,1650	0,1709	0,1767	0,1616	0,1604	0,0864	0,0790	0,0799	0,0852	0,0850	0,0855	0,0980	0,1005	0,1180
h_b	0,0030	0,0030	0,0030	0,0030	0,0031	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0030	0,0031	0,0031	0,0030

Właściwości materiałowe stopu Ti-6Al-4V to:

moduł Younga $E = 113,6$ GPa;

liczba Poissona $\nu = 0,342$;

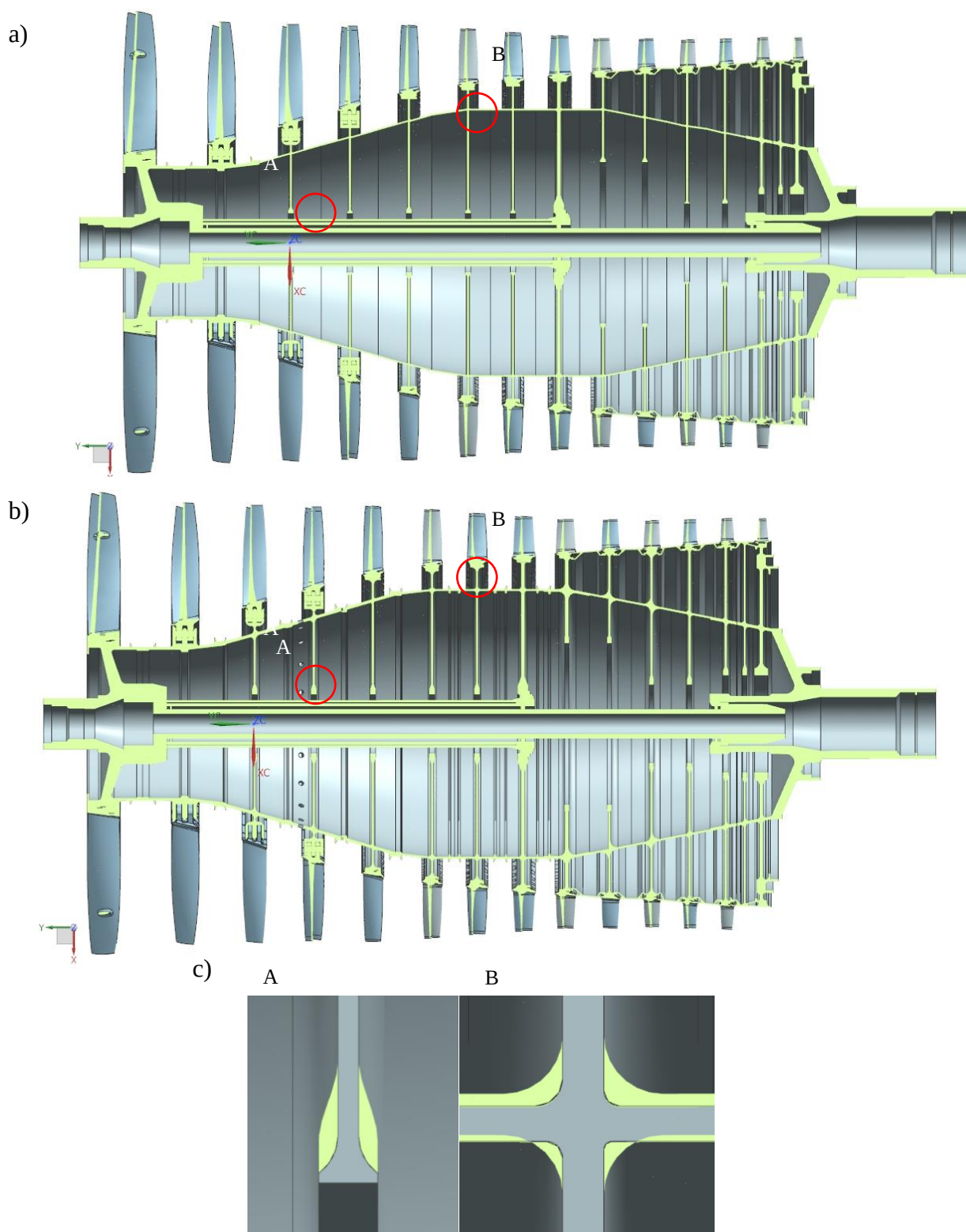
gęstość $\rho = 4430 \frac{kg}{m^3}$;

granica plastyczności $R_e = 1080$ MPa;

wytrzymałość na rozciąganie $R_m = 1170$ MPa;

naprężenia dopuszczalne przy uwzględnieniu współczynnika bezpieczeństwa $\sigma_{dop} = 780$ MPa.

Dla wybranego stopu tytanu maksymalne naprężenia dopuszczalne w warunkach quasi-statycznych to około 1080 MPa w temperaturze pokojowej, przy szybkości odkształceń $1 s^{-1}$ (źródło [115]). Dla temperatury około 900 K, przy szybkości odkształceń $2 s^{-1}$, naprężenia dopuszczalne to około 650 MPa [116]. Współczynnik bezpieczeństwa dla temperatury pokojowej to około 1,5. Maksymalna temperatura w sprężarce to około 730 K. Optymalizację przeprowadzono dla stałej temperatury (20°C) i naprężeń dopuszczalnych 780 MPa. W optymalizacji pominięto wpływ temperatury na moduł Younga, liczbę Poissona i pozostałe parametry materiałowe. Wszystkie dane materiałowe w kodzie programu wprowadzono w postaci macierzowej, co umożliwia, w razie potrzeby, uzależnienie zmiany właściwości materiałowych od rozkładu temperatury. W celu ewentualnego uwzględnienia naprężeń termicznych pomijalnych przy występującym w sprężarce gradiencie temperatur należałoby zmodyfikować algorytm APDL. Na rys. 6.2 porównano geometrię wirnika przed optymalizacją (a) i po niej (b). Położenie połączeń tarcza-bęben nie podlegało modyfikacji.



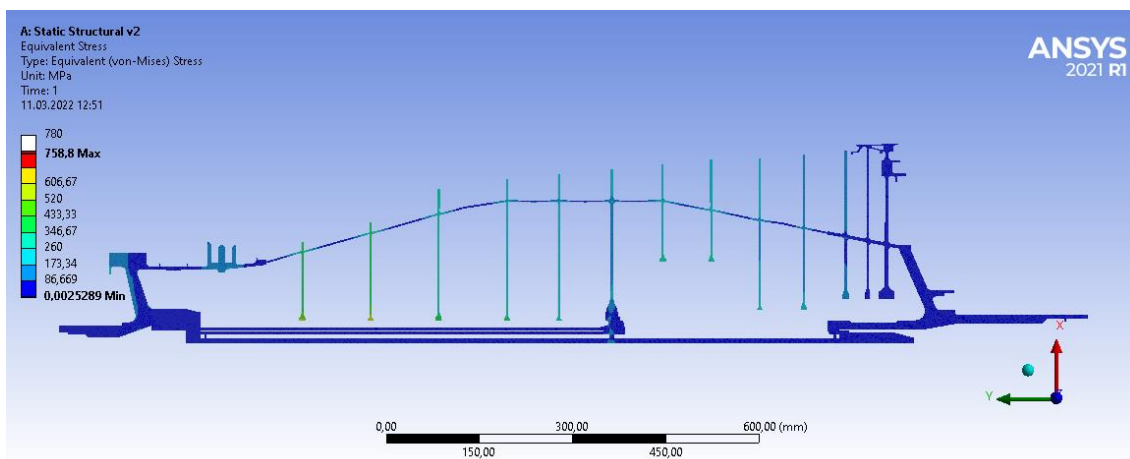
Rys. 6.2. Porównanie wirnika wzorcowego (a) z wirnikiem po optymalizacji (b); (c) obszary największych zmian geometrii przed optymalizacją i po niej

W tabeli 6.2 porównano objętości stopni przed optymalizacją i po niej. Różnica masy wynosi około 41 kg, co stanowi około 28% względem masy początkowej. W przypadku porównywania tylko elementów podlegających optymalizacji zysk na masie stanowi 56% masy początkowej optymalizowanych części.

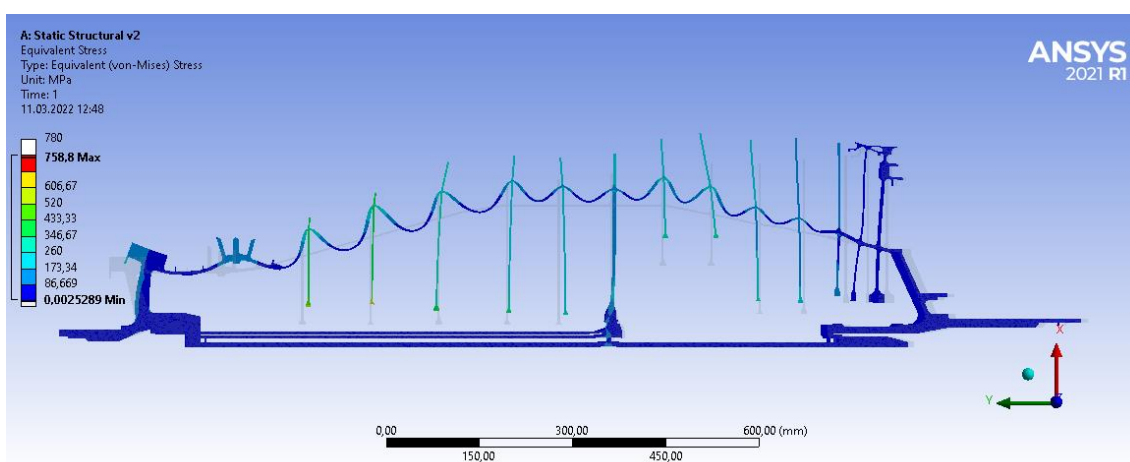
Tabela 6.2. Zestawienie objętości optymalizowanych elementów konstrukcyjnych

Nr stopnia	Objętość wzorcowa [mm ³]	Objętość po optymalizacji [mm ³]	δ [%]
1	4163140	-	-
2	1634650	-	-
3	1780565	697642	61
4	2240954	854804	62
5	2199726	1178117	46
6	2098528	1351238	36
7	2077445	1278567	38
8	2736546	-	-
9	2641188	1208300	54
10	1857114	1206079	35
11	2134128	1335237	37
12	1971784	1374736	30
13	1991020	1282944	36
14	3701408	-	-
Σ	33228196	24003408	28

Tak duża różnica w masie pomiędzy rzeczywistą konstrukcją i konstrukcją po optymalizacji powinna wskazywać na potencjalne błędy w procesie optymalizacji. Przeprowadzono analizę MES wirnika celem ostatecznej walidacji procesu optymalizacji. Rysunki 6.3 i 6.4 przedstawiają naprężenia (Hubera-von Misesa) dla kolejno rzeczywistych i stukrotnie przeskalowanych przemieszczeń. Naprężenia dopuszczalne (780 MPa) nie zostały przekroczone w żadnym obszarze.

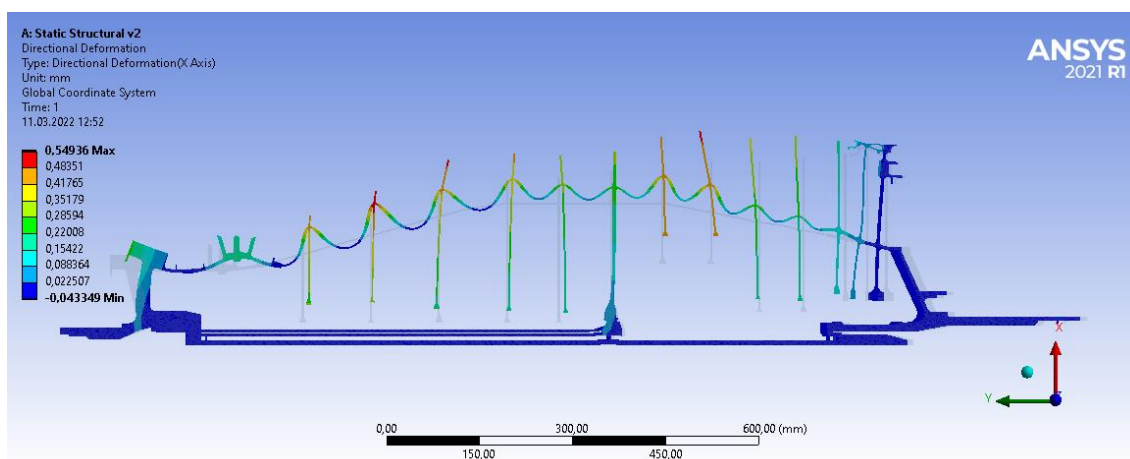


Rys. 6.3. Naprężenia Hubera-von Misesa wirnika po optymalizacji



Rys. 6.4. Naprężenia Hubera-von Misesa wirnika po optymalizacji. Odształcenia przeskalowane 100-krotnie celem ich uwydatnienia

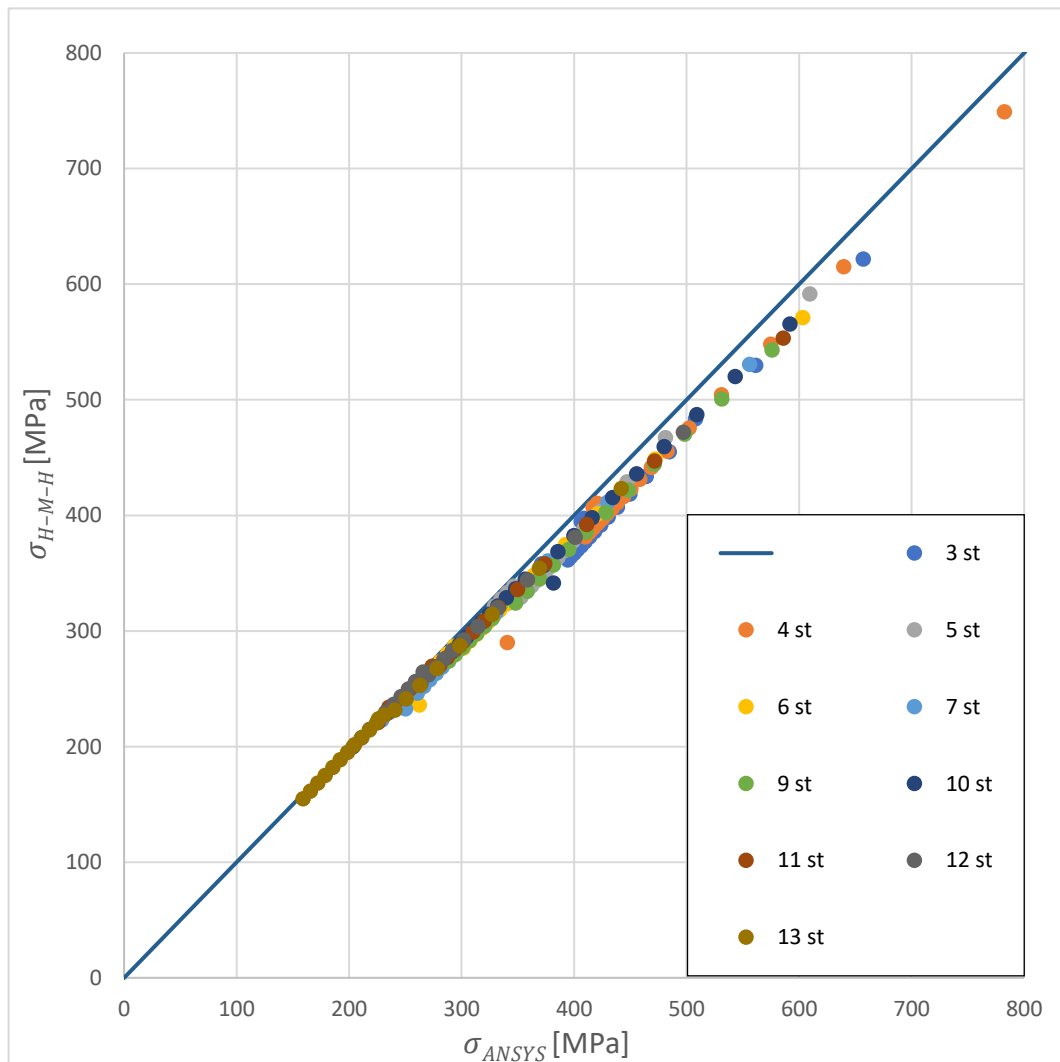
Na rysunku 6.5 przedstawiono składową promieniową odkształceń wirnika. Odształcenia przeskalowano 100-krotnie celem ich uwydatnienia.



Rys. 6.5. Składowa promieniowa przemieszczeń wirnika. Odształcenia przeskalowano 100-krotnie

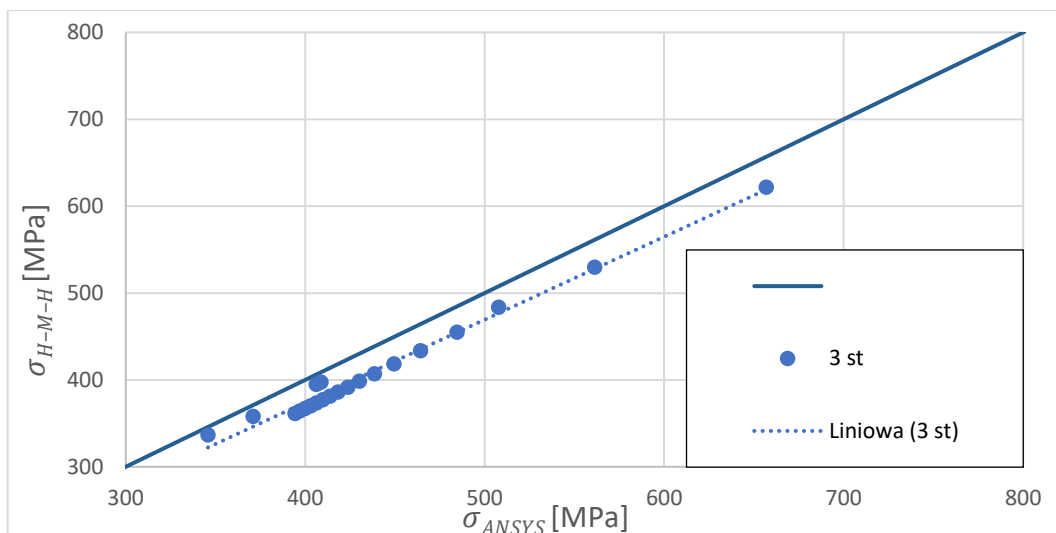
Na rysunku 6.6 porównano rozkład naprężeń redukowanych (Hubera-von Misesa) wyznaczonych przez stosowany algorytm optymalizujący z rozkładem naprężeń wyznaczonych w programie ANSYS workbench. Naprężenia sprawdzające wyznaczono dla całego stopnia z uwzględnieniem łopatek, lecz z pominięciem zamków. Łopatki

zamodelowano uwzględniając ich sztywne mocowanie do tarczy. Zastosowanie takiego połączenia tarcza-łopatka nie ma wpływu na rozkład naprężeń w tarczy w przypadku obliczeń statycznych. W przypadku analizy drgań takie uproszczenie jest oczywiście niedopuszczalne. Rozkłady naprężeń wyznaczone przez MES i SSN są zbieżne.

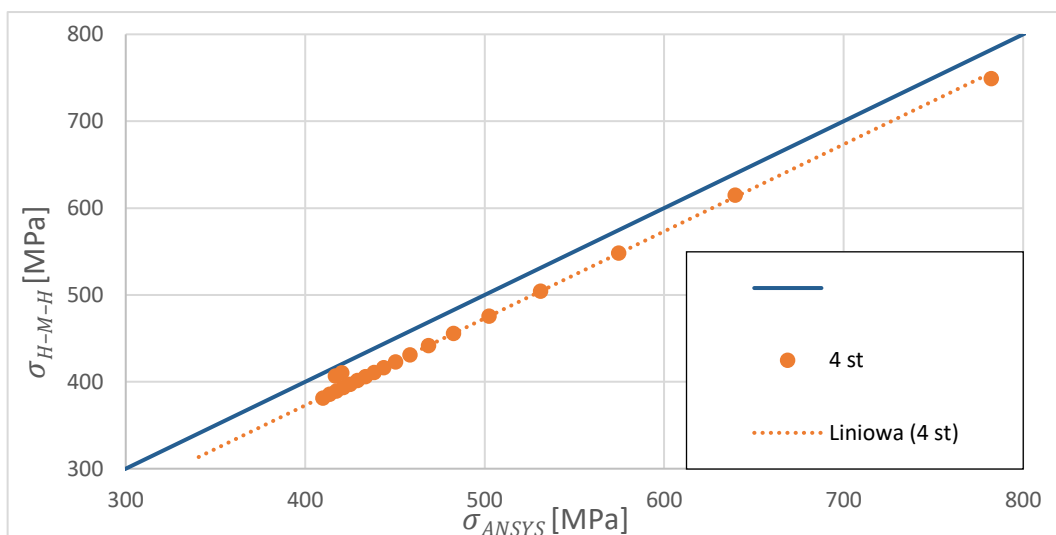


Rys. 6.6. Dopasowanie naprężeń zredukowanych Hubera-von Misesa-Huckiego (σ_{H-M-H}) wyznaczone przez wykorzystywany algorytm i wyznaczone w programie ANSYS workbench (σ_{ANSYS}), dla wszystkich optymalizowanych stopni

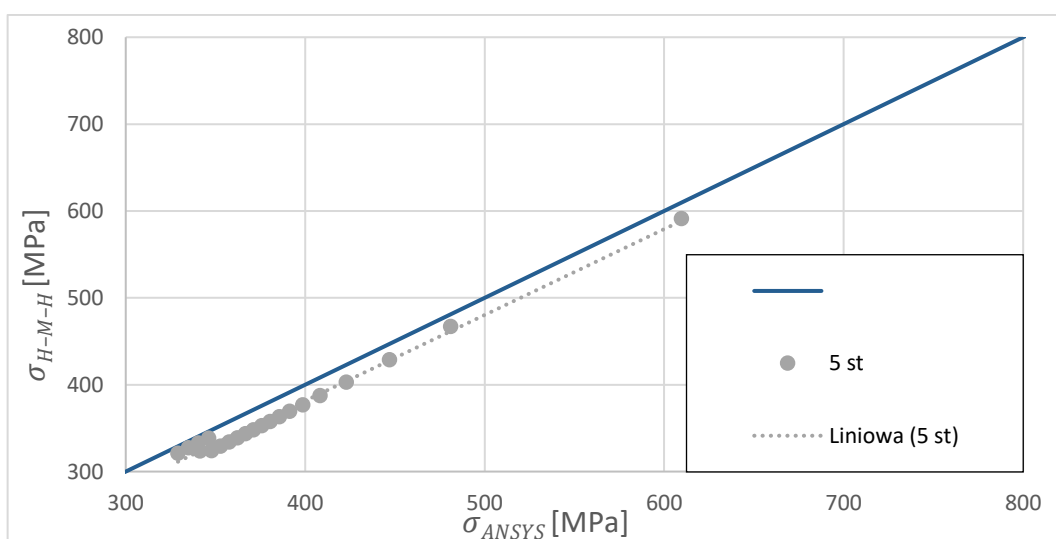
Na rysunku 6.6 przedstawiono dopasowanie naprężeń zredukowanych Hubera-von Misesa-Huckiego (σ_{H-M-H}) wyznaczone przez wykorzystywany algorytm i wyznaczone w programie ANSYS workbench (σ_{ANSYS}), dla wszystkich optymalizowanych stopni. Poniżej przedstawiono dopasowanie wyników uzyskanych dla poszczególnych stopni.



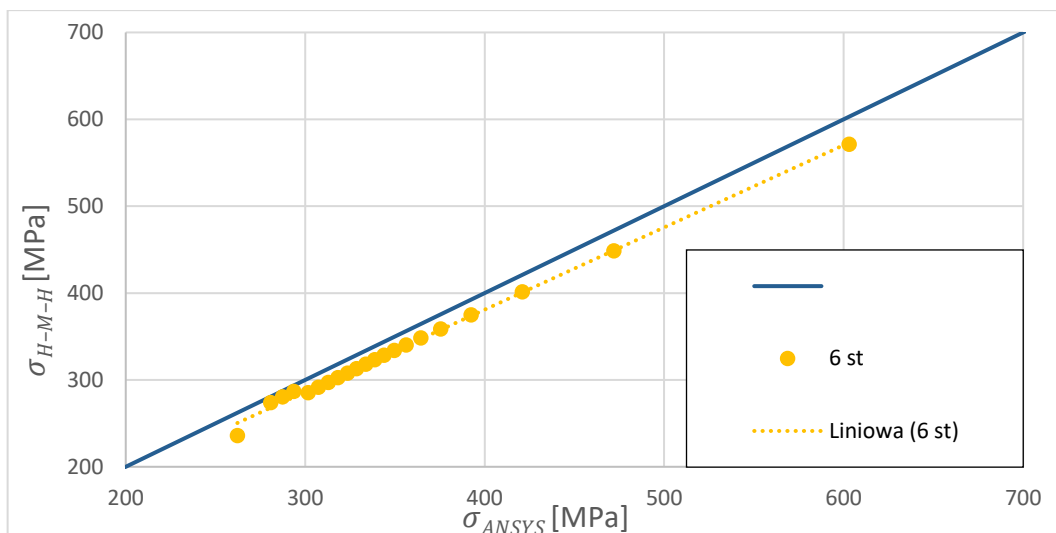
Rys. 6.7. Dopasowanie naprężeń zredukowanych dla 3 stopnia



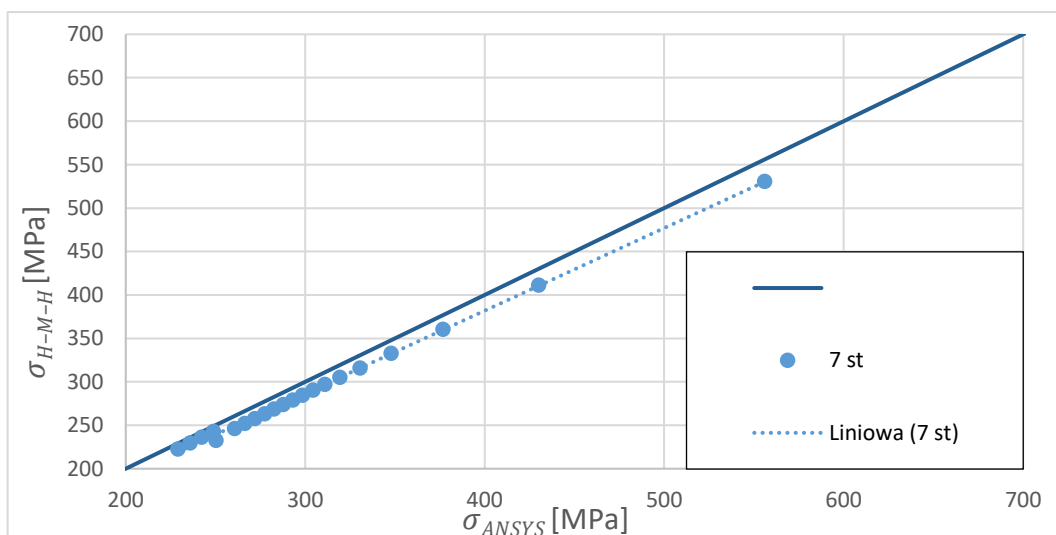
Rys. 6.8. Dopasowanie naprężeń zredukowanych dla 4 stopnia



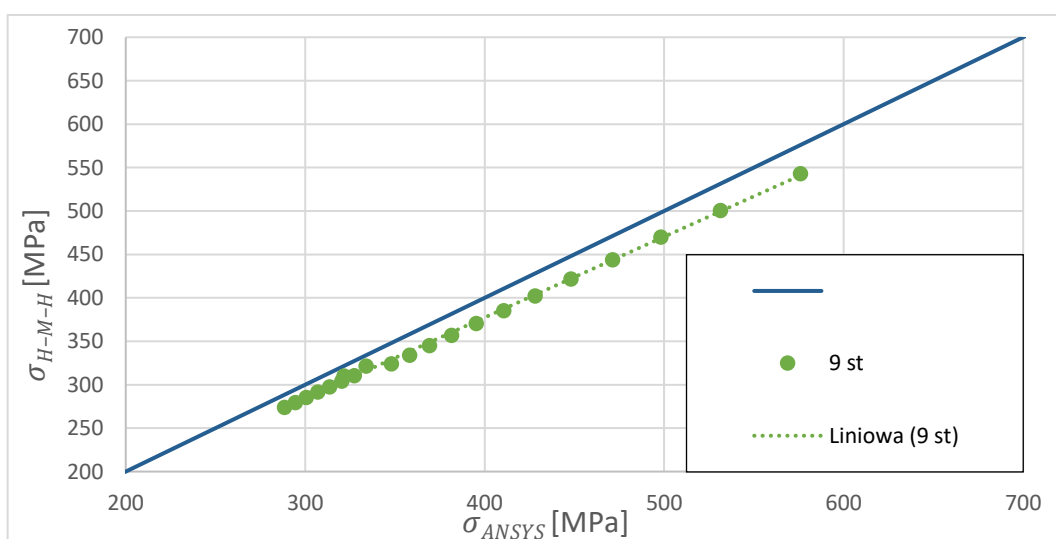
Rys. 6.9. Dopasowanie naprężeń zredukowanych dla 5 stopnia



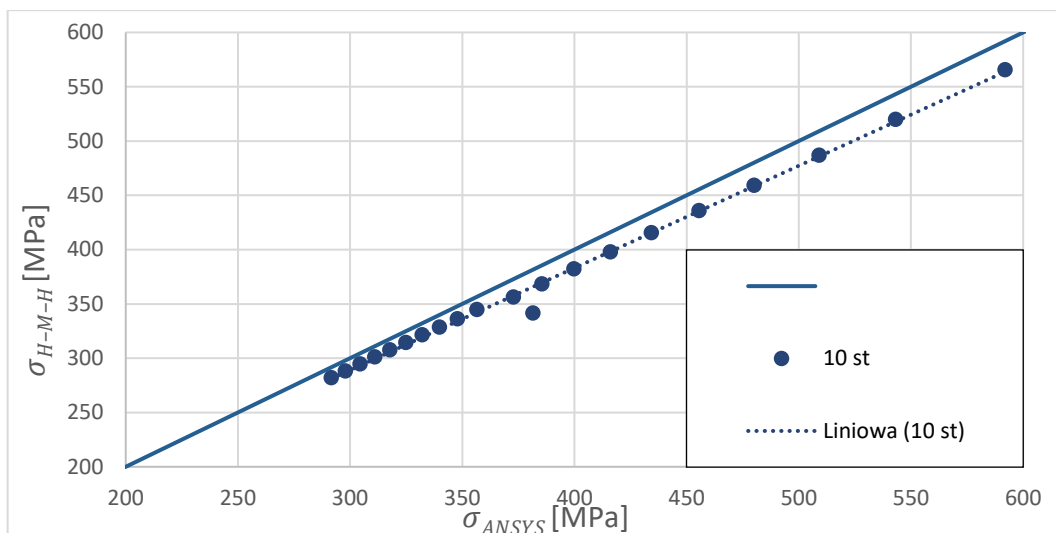
Rys. 6.10. Dopasowanie naprężeń zredukowanych dla 6 stopnia



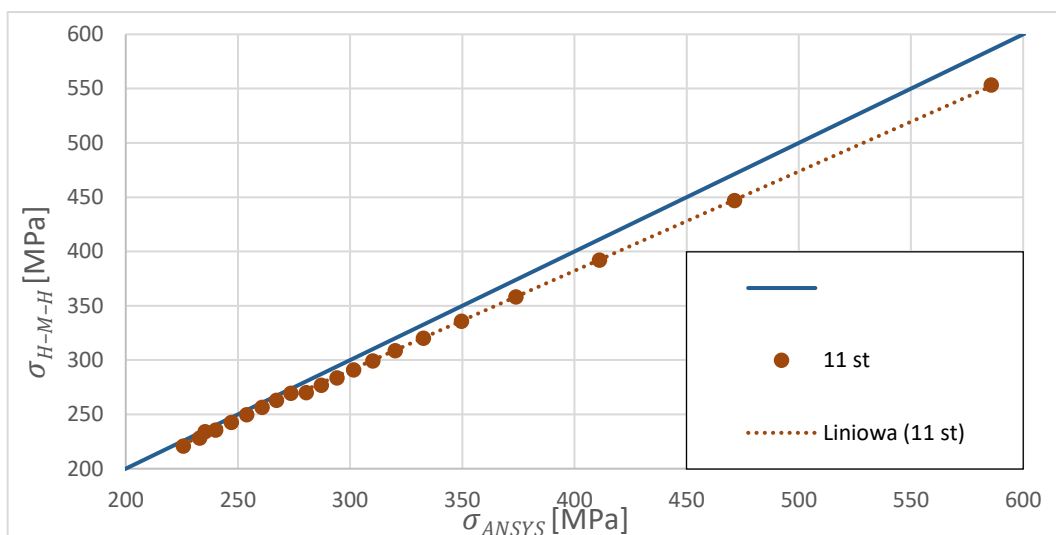
Rys. 6.11. Dopasowanie naprężeń zredukowanych dla 7 stopnia



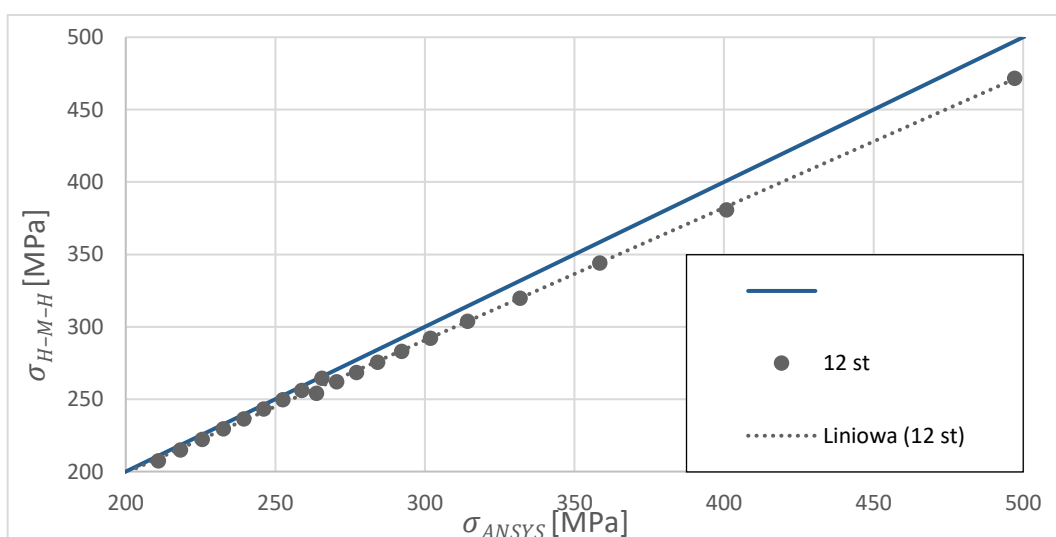
Rys. 6.12. Dopasowanie naprężeń zredukowanych dla 9 stopnia



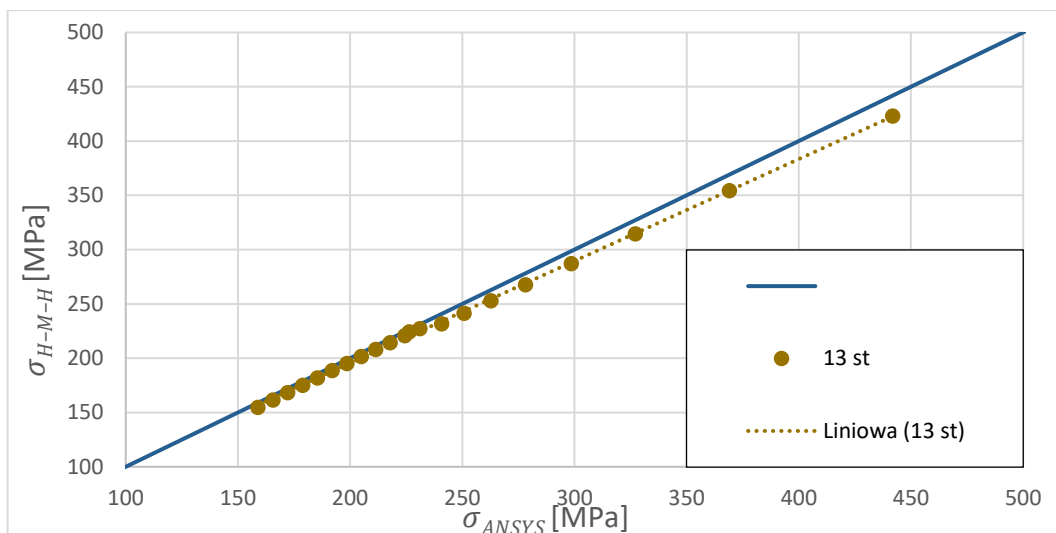
Rys. 6.13. Dopasowanie naprężeń zredukowanych dla 10 stopnia



Rys. 6.14. Dopasowanie naprężeń zredukowanych dla 11 stopnia



Rys. 6.15. Dopasowanie naprężeń zredukowanych dla 12 stopnia



Rys. 6.16. Dopasowanie naprężeń zredukowanych dla 13 stopnia

Na rysunkach 6.7-6.16 przedstawiono dopasowanie wyników naprężeń dla wszystkich stopni. Dla wszystkich stopni aproksymacja liniowa dopasowania danych wyznaczonych dla izolowanego stopnia (za pomocą SSN) i całego wirnika (za pomocą programu ANSYS) jest niemal równoległa do linii idealnego dopasowania. Wysłano wniosek, że wszystkie naprężenia wyznaczone przez SSN są niedoszacowane. Wynika to z uproszczeń wprowadzonych w celu uczenia SSN, a naprężenia w izolowanym stopniu są mniejsze średnio o 4,5% w stosunku do wyników uzyskanych dla konstrukcji całego wirnika. W tabeli 6.3 przedstawiono błąd względny naprężeń dla analizowanych węzłów w tarczach wszystkich stopni oraz współczynnik determinacji R^2 .

Zauważyć można tendencję wzrostową dokładności dopasowania danych wraz z odległością od stopni o konstrukcji specjalnej (nieoptymalizowanych: stopień 1, 2, 8 i 14).

Tabela 6.3. Wartość średnia $\bar{\delta}$ [%] i odchylenie standardowe σ [%] błędu względnego dopasowania oraz współczynnik determinacji R^2 naprężeń wyznaczonych przez algorytm SSN i program ANSYS δ [%]

Nr stopnia	3	4	5	6	7	9	10	11	12	13
δ [%]	5,3	4,2	3,0	5,3	4,6	5,7	4,4	5,5	5,1	4,2
	5,6	3,8	2,9	4,9	4,3	5,8	4,3	5,1	5,0	4,0
	4,8	4,6	4,0	4,6	4,3	5,6	4,3	4,6	4,0	3,9
	6,1	5,0	4,6	4,5	4,3	5,7	4,3	4,2	3,6	3,8
	6,6	5,3	5,1	4,5	4,4	5,9	4,3	3,9	3,3	3,8
	6,9	5,6	5,4	4,5	4,4	6,0	4,3	3,7	3,2	3,8
	7,2	5,8	5,6	4,5	4,5	6,1	4,3	3,6	3,1	3,8
	7,4	5,9	5,8	4,5	4,5	6,3	4,4	3,5	3,1	3,8
	7,6	6,1	5,9	4,6	4,6	6,4	4,4	3,5	3,1	1,0
	7,7	6,2	6,0	4,6	4,7	6,5	4,3	3,5	3,1	1,7
	7,8	6,3	6,1	4,7	4,8	6,7	10,4	3,5	3,6	1,7
	7,9	6,4	6,3	4,8	4,9	6,8	3,3	3,7	0,4	1,6
	8,0	6,5	6,4	4,9	5,0	3,7	3,3	0,5	1,1	1,6
	8,1	6,5	6,5	5,0	5,2	3,6	3,2	1,5	1,1	1,7
	8,2	6,6	6,6	5,1	5,3	5,1	3,2	1,6	1,1	1,7
	8,3	6,7	6,7	5,2	5,5	5,0	3,2	1,6	1,2	1,8
	8,3	6,8	5,1	5,4	7,0	5,0	3,2	1,7	1,3	1,9
	3,5	7,0	2,1	10,0	2,3	5,0	3,2	1,8	1,4	2,1
	2,6	14,9	2,1	2,3	2,4	5,0	3,2	1,9	1,5	2,2
	2,8	2,4	2,2	2,4	2,6	5,0	3,2	2,0	1,7	2,4
	2,8	2,4	2,3	2,5	2,7	5,0	3,2	2,1	1,8	2,6
$\bar{\delta}$ [%]	6,36	5,95	4,79	4,70	4,40	5,52	4,09	3,00	2,51	2,63
σ [%]	1,98	2,45	1,67	1,51	1,12	0,88	1,56	1,36	1,35	1,07
R^2	0,9859	0,9936	0,9912	0,9966	0,9985	0,9987	0,9951	0,9996	0,9990	0,9993

Innym powodem znacznej zmiany kształtu konstrukcji w wyniku optymalizacji może być przypuszczalne zastosowanie w oryginalnej konstrukcji większego współczynnika bezpieczeństwa, co odpowiadałoby znaczną zmianą w obszarach A na rys. 6.2.

Autorowi nie udało się dotrzeć do zaleceń konstrukcyjnych stosowanych w ZSRR w latach sześćdziesiątych minionego stulecia. Różnice w obszarach B mogą mieć na celu wydłużenie żywotności zmęczeniowej wirnika lub zapobiegnięcie niepożądanym rezonansom. Zmiana techniki i technologii projektowania na przestrzeni ostatnich 60 lat nie powinna mieć aż takiego wpływu na zmianę masy konstrukcji, natomiast technologia materiałowa już tak. Może to tłumaczyć wysokie współczynniki bezpieczeństwa, ponieważ, zgodnie z wiedzą autora, w ZSRR w czasie projektowania silnika AŁ21-F3 były problemy z powtarzalnym wytwarzaniem stopów wysokiej jakości. Możliwe także, że autor dotarł do mylących danych na temat wykorzystywanego w silniku materiału i jego specyfikacje znacząco różnią się od zastosowanego w procesie optymalizacji stopu Ti-6Al-4V.

Znaczna zmiana kształtu tarcz może także wynikać z analizy drgań lub z analizy zmęczeniowej. Aspekty drgań, aerosprężystości łopatek i zmęczenia elementów wirnikowych to problemy wybiegające poza temat rozprawy, z powodu małej

powtarzalności obliczeń, mogących posłużyć do uczenia SSN. Temat tej pracy nie przewidywał konieczności obliczania prędkości krytycznych rezonansów, wyznaczania diagramów Campbella, czy wyznaczania krytycznych prędkości flatteru łopatek na wieńcach wirnikowych. Niemniej, gdyby elementy po optymalizacji wytrzymałościowej miały być wykorzystane w rzeczywistej konstrukcji, należałoby przeprowadzić takie obliczenia.

WNIOSKI KOŃCOWE

Celem rozprawy doktorskiej było opracowanie algorytmu opartego na SSN do optymalizacji wybranych elementów lotniczego turbinowego silnika odrzutowego z wykorzystaniem algorytmów uczenia maszynowego, aby zoptymalizować zespół sprężarkowy. Przed przystąpieniem do budowy algorytmu i badań numerycznych przeprowadzono analizę literatury, w wyniku której nie stwierdzono stosowalności tego typu podejścia do rozwiązywania wyznaczonego celu. Natknięto się na prace związane z wykorzystaniem Sztucznych Sieci Neuronowych, ale nie wykorzystywano ich w połączeniu z Metodą Elementów Skończonych do obliczeń wytrzymałościowych maszyn wirnikowych.

Aby zrealizować zaplanowany cel rozprawy, najpierw udowodniono, że możliwe jest wykorzystanie Sztucznych Sieci Neuronowych do rozwiązywania problemów inżynierskich i naukowych, co zostało przedstawione w rozdziałach 2 i 3. Wykorzystano SSN do obliczeń naprężeń w belce wysięgnikowej obciążonej siłą skupioną, z dość dużą dokładnością. W tym przypadku błąd względny nie przekraczał 0,5% (dla sieci typu NARX). Kolejnym analizowanym zagadnieniem były obliczenia rozkładu naprężeń w ważkiej tarczy prostej, obciążonej ciągnięciem. Ten problem został rozwiązany z dobrą dokładnością, wynoszącą 5% dla sieci typu Feed Forward. Analizę przeprowadzono dla wielu mniejszych sieci neuronowych, z których każda wyznaczała naprężenia w pojedynczym węźle siatki obliczeniowej, w zależności od działających obciążeń i danych geometrycznych całego badanego elementu. Zbiór uczący liczył 50 spośród 2200 przypadków uczących i był zmienny w czasie uczenia. Czas wyznaczania naprężeń przez sieci wyniósł 0,45 sekundy przy 854,5 sekundach potrzebnych na rozwiązanie tego samego problemu przez MES. Przeprowadzone badania uczenia sieci wyznaczania naprężeń jednocześnie w całej tarczy dostarczyły wiedzy, że sieć uczy się lepiej złożonych zależności dla poszczególnych elementów, ale rozłożonych na mniejsze powtarzalne problemy niż całego zagadnienia jednocześnie.

W rozdziale trzecim tarczę profilowaną poddano optymalizacji za pomocą opracowanej modyfikacji algorytmu genetycznego. Algorytm genetyczny w każdej iteracji liczył 50 osobników, które odpowiadały 50 przypadkom uczącym sieci, dla których naprężenia były wyznaczane za pomocą MES. Następnie algorytm poszukiwał minimum, które było punktem wyjściowym (środkiem chwilowej przestrzeni rozwiązań dopuszczalnych) kolejnej iteracji. Samo zastosowanie zmiennej chwilowej przestrzeni rozwiązań stanowi udoskonalenie algorytmu genetycznego, które skraca czas każdej iteracji, zwiększając ich liczbę. Takie podejście, pomimo większej liczby iteracji, zwiększa prawdopodobieństwo osiągnięcia minimum globalnego i skraca czas działania algorytmu. Dodatkowo nie wymaga zbyt dokładnego uczenia sieci w każdej iteracji, ponieważ dokładność SSN zwiększa się z każdą iteracją. Czas pracy algorytmu realizującego proces obliczeniowy obejmujący 5 iteracji wynosił około 7-8 minut. Czas pracy pojedynczej iteracji algorytmu bez SSN to około 170 s. Czas pracy jednej iteracji zaproponowanego algorytmu opartego na SSN to około 98 s, gdzie 40 s to czas uczenia sieci, a pozostałe 58 s to czas pracy algorytmu genetycznego (dalej zwanego wzorcowym). Różnica czasu w przedstawionym przypadku to około 70 s, co stanowi około 40% czasu pracy algorytmu wzorcowego, przy dokładności względem MES na poziomie $(97 \div 98)\%$.

Kolejnym krokiem do osiągnięcia założonego celu było opracowanie programu, w oparciu o język APDL, do wyznaczania rozkładu naprężeń wzdłuż wybranych ścieżek

dla stopnia sprężarki o konstrukcji tarczowo-bębnowej. Program ten został opisany w rozdziale 4.

W dalszej części pracy (rozdział 5) opisano proces optymalizacji struktury i metody uczenia Sztucznych Sieci Neuronowych pod kątem minimalizacji błędu średniokwadratowego i odchylenia standardowego naprężeń w konstrukcji tarczowo-bębnowej. Dane do uczenia sieci generowano losowo z ograniczonego, zmiennego w czasie zbioru rozwiązań dopuszczalnych. Ponownie były to obciążenia i dane geometryczne tarczy. Na ich podstawie, za pomocą algorytmu zakodowanego w języku APDL (w oparciu o MES), obliczono rozkłady naprężeń wzdłuż wybranych ścieżek. Odczytano z nich naprężenia w wybranych, znormalizowanych względem długości ścieżki węzłach, które stanowiły dane wyjściowe, niezbędne do uczenia sieci pod nadzorem. Optymalizowano zarówno algorytm uczenia, jak i strukturę sieci dotyczącą ilości warstw i neuronów w warstwie. Porównano trzy algorytmy uczenia o 25 lub 50 neuronach w każdej z od 1 do 4 warstw ukrytych, a także wpływ powtarzalności procesu uczenia na dokładność predykcji. Sieci uczono: metodą gradientów sprzężonych z regulacją (SCG), metodą momentum (GDM) i algorytmem sprężystej propagacji wstecznej (RPROP). Wszystkie trzy algorytmy wskazały, że sieci o 50 neuronach uczą się z mniejszą dokładnością. Kolejnym etapem była optymalizacja struktury sieci z rozróżnieniem wykorzystanego algorytmu uczącego. Badano sieci zawierające od 1 do 3 warstw ukrytych i od 1 do 50 neuronów w każdej warstwie. Określono także wpływ rozpiętości bazy danych uczących w zakresie, $\pm 5\%$ i $\pm 10\%$. Pozwoliło to otrzymać zbiór rozwiązań optymalnych w ujęciu Pareto. Na jego podstawie odrzucono algorytm GDM, a także sieci wielowarstwowe. Kolejnym krokiem była walidacja wyników poprzez ponowną optymalizację sieci w ujęciu Pareto. Za optymalną uznano sieć typu Feed Forward (20-2-1), uczoną metodą gradientów sprzężonych (SCG) i rozpiętości bazy danych uczących $\pm 5\%$. Sieć ta charakteryzowała się najmniejszą regresją dla danych testowych $R^2 = 0,9973$.

W końcowym rozdziale przedstawiono optymalizację elementów czternastostopniowej sprężarki silnika AŁ 21F-3 z użyciem opracowanej modyfikacji algorytmu genetycznego i SSN wspomagających obliczenia twarde (MES rozdział 6). Podejście to pozwoliło zmniejszyć objętość optymalizowanych elementów od 30% do 62%. Zastosowany algorytm miękki (SSN) oparty na Sztucznych Sieciach Neuronowych cechuje się dużą dokładnością. Jego najmniejszy współczynnik determinacji R^2 dla rozkładu naprężeń w pojedynczym, izolowanym stopniu wynosi 0,9859. Jest on najmniejszy dla stopni sąsiadujących ze stopniami o konstrukcji specjalnej i wzrasta on dla stopni sąsiadujących ze stopniami o konstrukcji bębnowo-tarczowej.

Cel pracy, jakim było opracowanie algorytmu opartego na SSN do optymalizacji wybranych elementów lotniczego turbinowego silnika odrzutowego z wykorzystaniem algorytmów uczenia maszynowego, na rzecz optymalizacji zespołu sprężarkowego, został osiągnięty.

ELEMENTY NOWOŚCI

W rozprawie przedstawiono autorską modyfikację algorytmu do optymalizacji potencjalnie każdego problemu naukowego, której celem było skrócenie czasu pracy, poprzez połączenia Algorytmu Genetycznego i Sztucznych Sieci Neuronowych. Modyfikacja algorytmu genetycznego polegała na wprowadzeniu zmiennej w każdej

iteracji (co kilka pokoleń), chwilowej przestrzeni rozwiązań dopuszczalnych. Zwiększa to ilość iteracji, ale znacząco zmniejsza czas pracy algorytmu, odsuwając także problem przedwczesnego ujednoludnienia genomu dla większej liczby iteracji. Pozwala to przeprowadzić optymalizację dla mniejszych populacji. Zastąpienie części obliczeń twardych przez Sztuczne Sieci Neuronowe znacząco zmniejsza czas obliczeń, ponieważ kolejne pokolenia rozwiązań są obliczane przez szybkie Sztuczne Sieci Neuronowe, a nie za pomocą MES.

Połączenie zmodyfikowanego Algorytmu Genetycznego i Sztucznych Sieci Neuronowych ma wpływ na istotność pewnych wad sieci, do których należy długi czas uczenia i „uczenie na pamięć”. Należy podkreślić, że te negatywne cechy są neutralizowane przez zmienną w każdej iteracji przestrzeń rozwiązań dopuszczalnych. Zmniejsza to rozpiętość danych uczących, a w konsekwencji skraca czas uczenia. Co więcej, fakt, że rozwiązanie optymalne nie zostanie wskazane w pierwszej iteracji, nie wymaga dużej dokładności w predykcji sieci, co także ogranicza czas niezbędny na trening sieci. Iteracyjna wędrówka przestrzeni rozwiązań połączona z kontynuacją uczenia sieci pozwala na finalne osiągnięcie dużej dokładności predykcji.

KIERUNKI DALSZYCH PRAC

Kierunek dalszych prac będzie obejmował analizę drgań i analizę zmęczeniową całego wirnika. W przypadku drgań rezonansowych lub szybkiego zużycia zmęczeniowego badania zostaną przeprowadzone z użyciem zaproponowanego algorytmu. Proces optymalizacji będzie prowadzony przy zmianie ograniczeń w przestrzeni rozwiązań dopuszczalnych dla zbioru zmiennych parametrów geometrycznych wykorzystanych w pracy. W przypadku błędów w obliczeniach lub zbyt wąskich technologicznie ograniczeń uwzględnianych parametrów należy zmienić je na inne.

Planowane prace będą obejmowały walidację postawionej we wnioskach tezy, że zaproponowany algorytm umożliwia optymalizację także innych problemów inżynierskich oraz naukowych, wymagających obliczeń MES lub MOS.

Dalsze prace obejmą także modyfikacje algorytmu genetycznego. Zostanie to zrealizowane poprzez implementację jednocześnie zmiennej chwilowej przestrzeni rozwiązań dopuszczalnych i zdarzeń wzorowanych na wielkich wymieraniach. Pozwoli to zmniejszyć rozmiar populacji, ilość iteracji i zwiększyć prawdopodobieństwo osiągnięcia lepszego minimum funkcji celu.

ZALĄCZNIKI

ZALĄCZNIK 1 - TARCZA_1D.M

```
%function
[sigma_red]=tarcza_1D(rmin,rmax,omega,mli,zli,rli);

% % Flag
% % 0 - bicgstab converged to the desired tolerance
tol within maxit iterations.
% % 1 - bicgstab iterated maxit times but did not
converge.
% % 2 - Preconditioner M was ill-conditioned.
% % 3 - bicgstab stagnated. (Two consecutive iterates
were the same.)
% % 4 - One of the scalar quantities calculated during
bicgstab became too small or too large to continue
computing.
obliczenia_wstepne=1;

m_zz=0;
global rr %x
global rho

s_=0.5;%in(20);
Fxi=0;%in(19)
Fri=0;%in(18)

Msi=0;%in(14)
tol=10^-8;
maxit=1000;
format short

lele=18;

fi=pi/2; %[rad]
%
wsp_zag1=4; %wsp>=1 4
wsp_zag2=2; %wsp>=1 2
obszar_zag1=2; %2
obszar_zag2=2;

LLL=(rmax-rmin)/10;
```

```

LL=(ones(1,lele))*LLL;

for ele=1:wsp_zag1*obszar_zag1
    LL(ele)=LLL/4;
end
for ele=lele-wsp_zag2*obszar_zag2+1:lele
    LL(ele)=LLL/(wsp_zag2);
end

for ele=1:lele
    if ele==1
        rr(1,ele)=rmin;

    else
        rr(1,ele)=rr(2,ele-1);

    end
    rr(2,ele)=rr(1,ele)+sin(fi)*LL(ele);
end

hh=ones(2,18).*x;
%hh(1,1)=0.01;%OK
hh(2,1)=hh(1,1);

for k=2:18
    hh(2,k)=x(k);
    hh(1,k)=hh(2,k-1);
end

%% okreslenie potrzebnych pustej macierzy K
K_=zeros(lele*3+3);
Fomega_=zeros(lele*3+3,1);
u=zeros(6,lele);
sigma_t=zeros(1,lele);
sigma_s=zeros(1,lele);
sigma_red=zeros(1,lele);

%% globalizacja ukladu wsp.
if fi==pi/2
    lambdai=[0,1,0;
-1,0,0;
0,0,1];
else
    lambdai=[cos(fi),sin(fi),0;
-sin(fi),cos(fi),0;
0,0,1];
end
lambdaj=lambdai;

```

```

lambda=[lambdai,zeros(3);
        zeros(3),lambdaj];

%% petla po elementach
for ele=1:lele
    % disp('-----ele-----')
    % okreslenie potrzebnych pustej macierzy T
    T=zeros(6,lele*3+3);

    %% przypisanie wymiarow z odpowiednich macierzy do
    skalarów w celu uproszczenia zapisu
    L=LL(ele);
    hi=hh(1,ele);
    hj=hh(2,ele);
    ri=rr(1,ele);
    rj=rr(2,ele);

    C=[1,0,0,0,0,0;
        -1/L,0,0,1/L,0,0;
        0,1,0,0,0,0;
        0,0,1,0,0,0;
        0,-3/L.^2,-2/L,0,3/L.^2,-1/L;
        0,2/L.^3,1/L.^2,0,-2/L.^3,1/L.^2];

    %% tworzenie macierzy BT*D*B*r i jej calkowanie
    % pat2:
    pat1l=@(s_)(-(ri-s_.*(ri-rj)).*((sin(fi)./(ri-
s_.*(ri-rj))-(s_.*sin(fi))./(ri-s_.*(ri-
rj))).*(E.*(hi-s_.*(hi-hj)).*(sin(fi)./(ri-s_.*(ri-
rj))-(s_.*sin(fi))./(ri-s_.*(ri-rj)))))./(ni.^2-1)-
(E.*ni.*(hi-s_.*(hi-hj))./(L.*(ni.^2-1)))+(E.*(hi-
s_.*(hi-hj))./(L.*(ni.^2-1))-(E.*ni.*(hi-s_.*(hi-
hj)).*(sin(fi)./(ri-s_.*(ri-rj))-(s_.*sin(fi))./(ri-
s_.*(ri-rj)))))./(ni.^2-1))./L));
    pat2l=@(s_)(-(ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)).*(sin(fi)./(ri-s_.*(ri-rj))-(s_.*sin(fi))./(ri-
s_.*(ri-rj))).*(cos(fi)./(ri-s_.*(ri-rj)))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))))./(ni.^2-
1)-(E.*ni.*(hi-s_.*(hi-hj)).*(cos(fi)./(ri-s_.*(ri-
rj))-(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj)))))./(L.*(ni.^2-1)));
    pat3l=@(s_)(-(ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)).*(sin(fi)./(ri-s_.*(ri-rj))-(s_.*sin(fi))./(ri-
s_.*(ri-rj))).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-

```

```

(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj))+L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(ni.^2-
1)-(E.*ni.*(hi-s_.*(hi-hj)).*((L.*s_.^2.*cos(fi))./(ri-
s_.*(ri-rj)))+(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj))+L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))))./(L.*(ni.^2-1))));
pat41=@(s_)(-(ri-s_.*(ri-rj)).*((E.*ni.*(hi-s_.*(hi-
hj)))./(L.*(ni.^2-1)))+(E.*s_.*sin(fi).*(hi-s_.*(hi-
hj)))./((ni.^2-1).*(ri-s_.*(ri-rj)))).*(sin(fi)./(ri-
s_.*(ri-rj)))-(s_.*sin(fi))./(ri-s_.*(ri-rj)))-
((E.*(hi-s_.*(hi-hj)))./(L.*(ni.^2-
1)))+(E.*ni.*s_.*sin(fi).*(hi-s_.*(hi-hj)))./((ni.^2-
1).*(ri-s_.*(ri-rj))))./(L));
pat51=@(s_)(-((E.*(hi-s_.*(hi-hj)).*(sin(fi)./(ri-
s_.*(ri-rj)))-(s_.*sin(fi))./(ri-s_.*(ri-
rj)))).*((3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj)))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(ni.^2-1)-
(E.*ni.*(hi-s_.*(hi-hj)).*((3.*s_.^2.*cos(fi))./(ri-
s_.*(ri-rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))))./(L.*(ni.^2-1)).*(ri-s_.*(ri-rj)));
pat61=@(s_)((E.*(hi-s_.*(hi-
hj)).*((L.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj)))-
(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))).*(sin(fi)./(ri-
s_.*(ri-rj)))-(s_.*sin(fi))./(ri-s_.*(ri-
rj))))./(ni.^2-1)-(E.*ni.*(hi-s_.*(hi-
hj)).*((L.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj)))-
(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(L.*(ni.^2-
1)).*(ri-s_.*(ri-rj)));

pat12=@(s_)(-(ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)).*(sin(fi)./(ri-s_.*(ri-rj)))-(s_.*sin(fi))./(ri-
s_.*(ri-rj))))./(ni.^2-1)-(E.*ni.*(hi-s_.*(hi-
hj)))./(L.*(ni.^2-1)).*(cos(fi)./(ri-s_.*(ri-rj)))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))));
pat22=@(s_)(-(ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj))))./(L.^2.*(ri-s_.*(ri-rj))))./(12.*(ni.^2-
1))).*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))).+(12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj))))./(L.^2.*(ri-s_.*(ri-rj)))).*(E.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj))))./(L.^2.*(ri-s_.*(ri-rj))))./(12.*(ni.^2-

```

```

1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*(hi-s_.*(hi-
hj)).*(cos(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).^2)./(ni.^2-1));
pat32=@(s_)(-(ri-s_.*(ri-rj)).*(((12.*s_)/L.^2-
(3.*(2.*ri-2.*s_.*(ri-rj)))./(L.^2.*(ri-s_.*(ri-
rj))))).*(E.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-
(2.*(2.*ri-2.*s_.*(ri-rj)))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*(sin(fi))./(ri-s_.*(ri-rj))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-
1)))+(6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-rj)))).*(E.*(hi-
s_.*(hi-hj)).^3.*(sin(fi))./(ri-s_.*(ri-rj))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_)/L-(2.*(2.*ri-2.*s_.*(ri-
rj)))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1)))+(E.*(hi-s_.*(hi-hj)).*(cos(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj)))).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(ni.^2-
1));
pat42=@(s_)(-(E.*ni.*(hi-s_.*(hi-hj)))./(L.*(ni.^2-
1)))+(E.*s_.*sin(fi).*(hi-s_.*(hi-hj)))./(ni.^2-
1).*(ri-s_.*(ri-rj))).*(ri-s_.*(ri-
rj)).*(cos(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))));
pat52=@(s_)((ri-s_.*(ri-rj)).*(((E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj)))./(L.^2.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))).*(6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))+(12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-

```



```

rj)))/(L.^2.*(ri-s_.*(ri-rj)))).*(E.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)./L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj)))/(L.^2.*(ri-s_.*(ri-rj))))./(12.*(ni.^2-
1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj))))./(12.*(ni.^2-1)))-(E.*(hi-s_.*(hi-
hj)).*(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))).*(cos(fi)./(ri-
s_.*(ri-rj))-(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(ni.^2-
1)));

```

```

pat62=@(s_)((ri-s_.*(ri-
rj)).*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-rj)))).*(E.*(hi-
s_.*(hi-hj)).^3.*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj))))./(12.*(ni.^2-
1)))-(E.*ni.*(hi-s_.*(hi-hj)).^3.*((6.*s_)./L-(2.*ri-
2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-rj))))./(12.*(ni.^2-
1)))-((12.*s_)./L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj)))/(L.^2.*(ri-s_.*(ri-rj)))).*(E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_)./L-(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-
s_.*(ri-rj))))./(12.*(ni.^2-1)))-(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj))))./(12.*(ni.^2-
1)))+(E.*(hi-s_.*(hi-hj)).*(L.*s_.^2.*cos(fi))./(ri-
s_.*(ri-rj))-(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).*(cos(fi)./(ri-s_.*(ri-rj))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(ni.^2-
1)));

```

```

pat13=@(s_)(-(ri-s_.*(ri-rj)).*(E.*(hi-s_.*(hi-
hj)).*(sin(fi)./(ri-s_.*(ri-rj))-(s_.*sin(fi))./(ri-
s_.*(ri-rj))))./(ni.^2-1)-(E.*ni.*(hi-s_.*(hi-
hj)))/(L.*(ni.^2-1))).*(L.*s_.*cos(fi))./(ri-
s_.*(ri-rj))-(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)));

```

```

pat23=@(s_)(-(ri-s_.*(ri-rj)).*((6.*s_)./L-
(2.*(2.*ri-2.*s_.*(ri-rj)))/(L.*(ri-s_.*(ri-
rj)))).*(E.*(hi-s_.*(hi-hj)).^3.*((12.*s_)./L.^2-
(3.*(2.*ri-2.*s_.*(ri-rj)))/(L.^2.*(ri-s_.*(ri-
rj))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj))))./(12.*(ni.^2-1)))+(E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-

```

```

(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj))./(L.^2.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))).*(sin(fi)./(ri-s_.*(ri-rj)))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj)))+(E.*(hi-
s_.*(hi-hj)).*(cos(fi)./(ri-s_.*(ri-rj)))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))))./(ni.^2-
1));
pat33=@(s_)(-(ri-s_.*(ri-rj)).*(((6.*s_)/L-
(2.*(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-
rj))))).*(E.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-
(2.*(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*sin(fi)./(ri-s_.*(ri-rj)))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*(hi-s_.*(hi-
hj)).^3.*sin(fi)./(ri-s_.*(ri-rj)))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_)/L-(2.*(2.*ri-2.*s_.*(ri-
rj))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))).*(sin(fi)./(ri-s_.*(ri-rj)))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj)))+(E.*(hi-
s_.*(hi-hj)).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))))).^2)./(ni.^2-1));
pat43=@(s_)(-(E.*ni.*(hi-s_.*(hi-hj)))./(L.*(ni.^2-
1))+(E.*s_.*sin(fi).*(hi-s_.*(hi-hj)))./(ni.^2-
1).*(ri-s_.*(ri-rj))).*(ri-s_.*(ri-
rj)).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))));
pat53=@(s_)((ri-s_.*(ri-rj)).*(((6.*s_)/L-
(2.*(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-
rj))))).*(E.*(hi-s_.*(hi-hj)).^3.*((12.*s_)/L.^2-
(3.*(2.*ri-2.*s_.*(ri-rj))./(L.^2.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-

```

```

hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj)))./(L.^2.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))).*(sin(fi)./(ri-s_.*(ri-rj))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj))-(E.*(hi-
s_.*(hi-hj)).*((3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(ni.^2-
1)));
pat63=@(s_)((ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)).^3.*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj))))./(12.*(ni.^2-
1))-(E.*ni.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-(2.*ri-
2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))).*(sin(fi)./(ri-s_.*(ri-rj))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj)))-
((6.*s_)/L-(2.*(2.*ri-2.*s_.*(ri-rj)))./(L.*(ri-
s_.*(ri-rj)))).*(E.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-
(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))-(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj))))./(12.*(ni.^2-
1)))+(E.*(hi-s_.*(hi-hj)).*(L.*s_.^2.*cos(fi))./(ri-
s_.*(ri-rj))-(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(ni.^2-
1)));
pat14=@(s_)((ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)))./(L.*(ni.^2-1))-(E.*ni.*(hi-s_.*(hi-
hj)).*(sin(fi)./(ri-s_.*(ri-rj))-(s_.*sin(fi))./(ri-
s_.*(ri-rj))))./(ni.^2-1))/L-(s_.*sin(fi)).*(E.*(hi-
s_.*(hi-hj)).*(sin(fi)./(ri-s_.*(ri-rj))-
(s_.*sin(fi))./(ri-s_.*(ri-rj))))./(ni.^2-1)-
(E.*ni.*(hi-s_.*(hi-hj)))./(L.*(ni.^2-1))))./(ri-
s_.*(ri-rj))));

```

```

pat24=@(s_)(-((E.*ni.*(hi-s_.*(hi-
hj)).*(cos(fi)./(ri-s_.*(ri-rj))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))))/(L.*(ni.^2-1)+(E.*s_.*sin(fi).*(hi-s_.*(hi-
hj)).*(cos(fi)./(ri-s_.*(ri-rj))-
(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))))/((ni.^2-
1).*(ri-s_.*(ri-rj)))).*(ri-s_.*(ri-rj));
pat34=@(s_)(-(ri-s_.*(ri-rj)).*((E.*ni.*(hi-s_.*(hi-
hj)).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))))/(L.*(ni.^2-1)+(E.*s_.*sin(fi).*(hi-s_.*(hi-
hj)).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))))/((ni.^2-
1).*(ri-s_.*(ri-rj))));
pat44=@(s_)(-(((E.*(hi-s_.*(hi-hj)))./(L.*(ni.^2-
1)))+(E.*ni.*s_.*sin(fi).*(hi-s_.*(hi-hj)))./((ni.^2-
1).*(ri-s_.*(ri-rj))))./L+(s_.*sin(fi).*((E.*ni.*(hi-
s_.*(hi-hj)))./(L.*(ni.^2-1)))+(E.*s_.*sin(fi).*(hi-
s_.*(hi-hj)))./((ni.^2-1).*(ri-s_.*(ri-rj))))./(ri-
s_.*(ri-rj))).*(ri-s_.*(ri-rj)));
pat54=@(s_)(-(ri-s_.*(ri-rj)).*((E.*ni.*(hi-s_.*(hi-
hj)).*(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(L.*(ni.^2-
1)))+(E.*s_.*sin(fi).*(hi-s_.*(hi-
hj)).*(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./((ni.^2-
1).*(ri-s_.*(ri-rj))));
pat64=@(s_)((ri-s_.*(ri-rj)).*((E.*ni.*(hi-s_.*(hi-
hj)).*(L.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./(L.*(ni.^2-
1)))+(E.*s_.*sin(fi).*(hi-s_.*(hi-
hj)).*(L.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))./((ni.^2-
1).*(ri-s_.*(ri-rj))));

pat15=@(s_)(-(ri-s_.*(ri-
rj)).*(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))).*(E.*(hi-
s_.*(hi-hj)).*(sin(fi)./(ri-s_.*(ri-rj))-
(s_.*sin(fi))./(ri-s_.*(ri-rj))))./(ni.^2-1)-
(E.*ni.*(hi-s_.*(hi-hj)))./(L.*(ni.^2-1))));
pat25=@(s_)((ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-

```

```

(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj))./(L.^2.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))).*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj))))+(12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj))./(L.^2.*(ri-s_.*(ri-rj))))).*((E.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj))./(L.^2.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))-(E.*(hi-s_.*(hi-
hj)).*((3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))*cos(fi)./(ri-
s_.*(ri-rj))-(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))))./(ni.^2-
1)));
pat35=@(s_)((ri-s_.*(ri-rj)).*((12.*s_)/L.^2-
(3.*(2.*ri-2.*s_.*(ri-rj))./(L.^2.*(ri-s_.*(ri-
rj))))).*((E.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-
(2.*(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*sin(fi)./(ri-s_.*(ri-rj)))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-
1)))+(6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-rj))))).*((E.*(hi-
s_.*(hi-hj)).^3.*sin(fi)./(ri-s_.*(ri-rj)))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_)/L-(2.*(2.*ri-2.*s_.*(ri-
rj))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-1)))-
(E.*(hi-s_.*(hi-hj)).*((3.*s_.^2.*cos(fi))./(ri-
s_.*(ri-rj))-(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj)))*cos(fi))./(ri-s_.*(ri-rj)))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))))./(ni.^2-
1)));
pat45=@(s_)(-((E.*ni.*(hi-s_.*(hi-hj)))/L.*(ni.^2-
1))+(E.*s_.*sin(fi).*(hi-s_.*(hi-hj)))/((ni.^2-
1).*(ri-s_.*(ri-rj))))).*(ri-s_.*(ri-
rj)).*((3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))));

```

```

pat55=@(s_)((- (ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj)))./(L.^2.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))).*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj))))+(12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj)))./(L.^2.*(ri-s_.*(ri-rj))))*(E.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj)))./(L.^2.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*(hi-s_.*(hi-
hj)).*(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))).^2)./(ni.^2-
1)));
pat65=@(s_)((ri-s_.*(ri-rj)).*((12.*s_)/L.^2-
(3.*(2.*ri-2.*s_.*(ri-rj)))./(L.^2.*(ri-s_.*(ri-
rj))))*(E.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-(2.*ri-
2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))-(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))-(6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-rj)))).*(E.*(hi-
s_.*(hi-hj)).^3.*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))-(E.*ni.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-(2.*ri-
2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))+(E.*(hi-s_.*(hi-hj)).*(L.*s_.^2.*cos(fi))./(ri-
s_.*(ri-rj))-L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).*(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))))./(ni.^2-1)));
pat16=@(s_)((L.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))).*(ri-s_.*(ri-
rj)).*(E.*(hi-s_.*(hi-hj)).*(sin(fi))./(ri-s_.*(ri-
rj))-(s_.*sin(fi))./(ri-s_.*(ri-rj)))))./(ni.^2-1)-
(E.*ni.*(hi-s_.*(hi-hj)))./(L.*(ni.^2-1)));
pat26=@(s_)((ri-s_.*(ri-rj)).*((E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-

```

```

rj)))/(L.^2.*(ri-s_.*(ri-rj)))))/(12.*(ni.^2-
1)).*(2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj)))-((6.*s_)/L-
(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-
rj))))).*(E.*(hi-s_.*(hi-hj)).^3.*((12.*s_)/L.^2-
(3.*(2.*ri-2.*s_.*(ri-rj))./(L.^2.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*(hi-s_.*(hi-
hj)).*(L.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))).*(cos(fi))./(ri-
s_.*(ri-rj))-(3.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))/(ni.^2-
1));
pat36=@(s_)((ri-s_.*(ri-
rj)).*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj))).*(E.*(hi-
s_.*(hi-hj)).^3.*(sin(fi))./(ri-s_.*(ri-rj))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_)/L-(2.*(2.*ri-2.*s_.*(ri-
rj))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-1))-
((6.*s_)/L-(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-
rj))))).*(E.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-
(2.*(2.*ri-2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*(sin(fi))./(ri-s_.*(ri-rj))-
(4.*s_.*sin(fi))./(ri-s_.*(ri-
rj)))+(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+(E.*(hi-s_.*(hi-
hj)).*(L.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).*(L.*s_.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*L.*s_.^2.*cos(fi))./(ri-s_.*(ri-
rj)))+(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj))))/(ni.^2-
1));
pat46=@(s_)((E.*ni.*(hi-s_.*(hi-hj))./(L.*(ni.^2-
1)))+(E.*s_.*sin(fi).*(hi-s_.*(hi-hj))./((ni.^2-
1).*(ri-s_.*(ri-rj))))).*(L.*s_.^2.*cos(fi))./(ri-
s_.*(ri-rj))-(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).*(ri-s_.*(ri-rj)));
pat56=@(s_)((ri-s_.*(ri-rj)).*((6.*s_)/L-(2.*ri-
2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-rj))))).*(E.*(hi-
s_.*(hi-hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-
2.*s_.*(ri-rj))./(L.^2.*(ri-s_.*(ri-

```

```

rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))-((E.*(hi-s_.*(hi-
hj)).^3.*((6.*s_.^2.*sin(fi))./(L.*(ri-s_.*(ri-rj)))-
(6.*s_.*sin(fi))./(L.*(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1))+(E.*ni.*(hi-s_.*(hi-
hj)).^3.*((12.*s_)/L.^2-(3.*(2.*ri-2.*s_.*(ri-
rj))./(L.^2.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))).*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj)))+(E.*(hi-
s_.*(hi-hj)).*((L.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).*((3.*s_.^2.*cos(fi))./(ri-s_.*(ri-rj))-
(2.*s_.^3.*cos(fi))./(ri-s_.*(ri-rj)))))./(ni.^2-1));
pat66=@(s_)(-(ri-s_.*(ri-rj)).*((6.*s_)/L-(2.*ri-
2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-rj))))).*((E.*(hi-
s_.*(hi-hj)).^3.*((6.*s_)/L-(2.*ri-2.*s_.*(ri-
rj))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-1))-
(E.*ni.*(hi-s_.*(hi-hj)).^3.*((2.*s_.*sin(fi))./(ri-
s_.*(ri-rj))-(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj)))))./(12.*(ni.^2-1)))+((2.*s_.*sin(fi))./(ri-
s_.*(ri-rj))-(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-
rj))).*((E.*(hi-s_.*(hi-
hj)).^3.*((2.*s_.*sin(fi))./(ri-s_.*(ri-rj))-
(3.*s_.^2.*sin(fi))./(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1))-(E.*ni.*(hi-s_.*(hi-hj)).^3.*((6.*s_)/L-(2.*ri-
2.*s_.*(ri-rj))./(L.*(ri-s_.*(ri-rj)))))./(12.*(ni.^2-
1)))+(E.*(hi-s_.*(hi-hj)).*((L.*s_.^2.*cos(fi))./(ri-
s_.*(ri-rj))-(L.*s_.^3.*cos(fi))./(ri-s_.*(ri-
rj))).^2)./(ni.^2-1));

```

```

Ke(1,1)=integral(pat11,0,1);
Ke(2,1)=integral(pat21,0,1);
Ke(3,1)=integral(pat31,0,1);
Ke(4,1)=integral(pat41,0,1);
Ke(5,1)=integral(pat51,0,1);
Ke(6,1)=integral(pat61,0,1);

```

```

Ke(1,2)=integral(pat12,0,1);
Ke(2,2)=integral(pat22,0,1);
Ke(3,2)=integral(pat32,0,1);
Ke(4,2)=integral(pat42,0,1);
Ke(5,2)=integral(pat52,0,1);
Ke(6,2)=integral(pat62,0,1);

```

```

Ke(1,3)=integral(pat13,0,1);

```



```

Ke(2,3)=integral(pat23,0,1);
Ke(3,3)=integral(pat33,0,1);
Ke(4,3)=integral(pat43,0,1);
Ke(5,3)=integral(pat53,0,1);
Ke(6,3)=integral(pat63,0,1);

Ke(1,4)=integral(pat14,0,1);
Ke(2,4)=integral(pat24,0,1);
Ke(3,4)=integral(pat34,0,1);
Ke(4,4)=integral(pat44,0,1);
Ke(5,4)=integral(pat54,0,1);
Ke(6,4)=integral(pat64,0,1);

Ke(1,5)=integral(pat15,0,1);
Ke(2,5)=integral(pat25,0,1);
Ke(3,5)=integral(pat35,0,1);
Ke(4,5)=integral(pat45,0,1);
Ke(5,5)=integral(pat55,0,1);
Ke(6,5)=integral(pat65,0,1);

Ke(1,6)=integral(pat16,0,1);
Ke(2,6)=integral(pat26,0,1);
Ke(3,6)=integral(pat36,0,1);
Ke(4,6)=integral(pat46,0,1);
Ke(5,6)=integral(pat56,0,1);
Ke(6,6)=integral(pat66,0,1);

%Kee(:, :, ele) = ((2*pi*L) .* Ke);
Kee_(:, :) = lambda' * ((2*pi*L) .* Ke) * lambda;
clear Ke

%% okreslenie polozenia Ke w K za pomoca wektora T

T(1, 3*ele-2)=1;
T(2, 3*ele-1)=1;
T(3, 3*ele)=1;
T(4, 3*ele+1)=1;
T(5, 3*ele+2)=1;
T(6, 3*ele+3)=1;

KK_ = T' * Kee_ * T;

K_ = K_ + KK_;
clear KK

%% obciarzenia od masy tarczy

```

```

    patt1=@(s_) ( sin(fi).* (hi-s_.*(hi-hj)).*(ri-s_.*(ri-
rj)).^2 );
    patt2=@(s_) ( L.*s_.*sin(fi).* (hi-s_.*(hi-hj)).*(ri-
s_.*(ri-rj)).^2 );
    patt3=@(s_) ( cos(fi).* (hi-s_.*(hi-hj)).*(ri-s_.*(ri-
rj)).^2 );
    patt4=@(s_) ( L.*s_.*cos(fi).* (hi-s_.*(hi-hj)).*(ri-
s_.*(ri-rj)).^2 );
    patt5=@(s_) ( L.^2.*s_.^2.*cos(fi).* (hi-s_.*(hi-
hj)).*(ri-s_.*(ri-rj)).^2 );
    patt6=@(s_) ( L.^3.*s_.^3.*cos(fi).* (hi-s_.*(hi-
hj)).*(ri-s_.*(ri-rj)).^2 );

    FFomega(1)=integral(patt1,0,1);
    FFomega(2)=integral(patt2,0,1);
    FFomega(3)=integral(patt3,0,1);
    FFomega(4)=integral(patt4,0,1);
    FFomega(5)=integral(patt5,0,1);
    FFomega(6)=integral(patt6,0,1);
    FFFomega=FFomega';

    Fomega=(-2*pi*L*rho*omega.^2).*C'*FFFomega;

    Fomegaa_=lambda'*Fomega;
    clear Fomega
    if fi==pi/2
        Fomegaa_(1)=0;
        Fomegaa_(3)=0;
        Fomegaa_(4)=0;
        Fomegaa_(6)=0;
    end
    Fomega_=Fomega_+T'*Fomegaa_;

end
%% warunki brzegowe
%
R0(3*lele+1)=(2*pi*rmax).*Fxi;
R0(3*lele+2)=(2*pi*rmax).*Fri+((mli*zli)+m_zz)*rli*ome
ga.^2);
R0(3*lele+3)=(2*pi*rmax).*Msi;
R0_=R0';

%% suma obciazen

Fomega_;
R_=(-Fomega_)+R0_./(2*pi()*rmax);

%% wynik obliczen

```

```

format long
[u_,fll]=bicgstab(K_,R_,tol,maxit);

for ele=1:lele
    L=LL(ele);
    hi=hh(1,ele);
    hj=hh(2,ele);
    ri=rr(1,ele);
    rj=rr(2,ele);
    uuu_(1)=u_(3*(ele-1)+1);
    uuu_(2)=u_(3*(ele-1)+2);
    uuu_(3)=u_(3*(ele-1)+3);
    uuu_(4)=u_(3*(ele-1)+4);
    uuu_(5)=u_(3*(ele-1)+5);
    uuu_(6)=u_(3*(ele-1)+6);

    u(:,ele)=lambda*uuu_';

    ui1=u(1,ele);
    ui2=u(2,ele);
    ui3=u(3,ele);
    uj1=u(4,ele);
    uj2=u(5,ele);
    uj3=u(6,ele);

    h=hi+(hj-hi)*s_;

    N=[ui1*((E*(hi-s_*(hi-hj)))/(L*(ni^2-1))-(E*ni*(hi-
s_*(hi-hj))*(sin(fi)/(ri-s_*(ri-rj))-(s_*sin(fi))/(ri-
s_*(ri-rj))))/(ni^2-1))-uj1*((E*(hi-s_*(hi-
hj)))/(L*(ni^2-1))+(E*ni*(hi-s_*(hi-
hj))*((s_*sin(fi))/(ri-s_*(ri-rj)))+(L*s_*cos(fi))/(ri-
s_*(ri-rj))))/(ni^2-1))+(E*ni*ui3*(hi-s_*(hi-
hj))*((2*L*s_^2*cos(fi))/(ri-s_*(ri-rj))-
(L*s_^3*cos(fi))/(ri-s_*(ri-rj))))/(ni^2-
1)+(E*ni*uj3*(hi-s_*(hi-hj))*((L*s_^2*cos(fi))/(ri-
s_*(ri-rj))-(L*s_^3*cos(fi))/(ri-s_*(ri-rj))))/(ni^2-
1)-(E*ni*ui2*(hi-s_*(hi-hj))*(cos(fi)/(ri-s_*(ri-rj))-
(3*s_^2*cos(fi))/(ri-s_*(ri-rj))+(2*s_^3*cos(fi))/(ri-
s_*(ri-rj))))/(ni^2-1)-(E*ni*uj2*(hi-s_*(hi-
hj))*((3*s_^2*cos(fi))/(ri-s_*(ri-rj))-
(2*s_^3*cos(fi))/(ri-s_*(ri-rj))))/(ni^2-1);
    (E*ui3*(hi-s_*(hi-hj))*((2*L*s_^2*cos(fi))/(ri-
s_*(ri-rj))-(L*s_^3*cos(fi))/(ri-s_*(ri-rj))))/(ni^2-
1)-ui1*((E*(hi-s_*(hi-hj))*(sin(fi)/(ri-s_*(ri-rj))-
(s_*sin(fi))/(ri-s_*(ri-rj))))/(ni^2-1)-(E*ni*(hi-
s_*(hi-hj)))/(L*(ni^2-1)))-uj1*((E*(hi-s_*(hi-

```

$$\begin{aligned}
& \text{hj})) * ((s_ * \sin(fi)) / (ri - s_ * (ri - rj)) + (L * s_ * \cos(fi)) / (ri - \\
& s_ * (ri - rj))) / (ni^2 - 1) + (E * ni * (hi - s_ * (hi - \\
& hj))) / (L * (ni^2 - 1)) + (E * uj3 * (hi - s_ * (hi - \\
& hj)) * ((L * s_^2 * \cos(fi)) / (ri - s_ * (ri - rj)) - \\
& (L * s_^3 * \cos(fi)) / (ri - s_ * (ri - rj)))) / (ni^2 - 1) - \\
& (E * ui2 * (hi - s_ * (hi - hj)) * (\cos(fi) / (ri - s_ * (ri - rj)) - \\
& (3 * s_^2 * \cos(fi)) / (ri - s_ * (ri - rj)) + (2 * s_^3 * \cos(fi)) / (ri - \\
& s_ * (ri - rj)))) / (ni^2 - 1) - (E * uj2 * (hi - s_ * (hi - \\
& hj)) * ((3 * s_^2 * \cos(fi)) / (ri - s_ * (ri - rj)) - \\
& (2 * s_^3 * \cos(fi)) / (ri - s_ * (ri - rj)))) / (ni^2 - 1); \\
& ui2 * ((E * (hi - s_ * (hi - hj))^3 * ((12 * s_ / L^2 - (6 * ri - \\
& 6 * s_ * (ri - rj)) / (L^2 * (ri - s_ * (ri - rj)))) / (12 * (ni^2 - \\
& 1)) + (E * ni * (hi - s_ * (hi - hj))^3 * ((6 * s_^2 * \sin(fi)) / (L * (ri - \\
& s_ * (ri - rj))) - (6 * s_ * \sin(fi)) / (L * (ri - s_ * (ri - \\
& rj)))) / (12 * (ni^2 - 1)) - uj2 * ((E * (hi - s_ * (hi - \\
& hj))^3 * ((12 * s_ / L^2 - (6 * ri - 6 * s_ * (ri - rj)) / (L^2 * (ri - \\
& s_ * (ri - rj)))) / (12 * (ni^2 - 1)) + (E * ni * (hi - s_ * (hi - \\
& hj))^3 * ((6 * s_^2 * \sin(fi)) / (L * (ri - s_ * (ri - rj))) - \\
& (6 * s_ * \sin(fi)) / (L * (ri - s_ * (ri - rj)))) / (12 * (ni^2 - \\
& 1)) + ui3 * ((E * (hi - s_ * (hi - hj))^3 * ((6 * s_ / L - (4 * ri - \\
& 4 * s_ * (ri - rj)) / (L * (ri - s_ * (ri - rj)))) / (12 * (ni^2 - 1)) - \\
& (E * ni * (hi - s_ * (hi - hj))^3 * ((4 * s_ * \sin(fi)) / (ri - s_ * (ri - \\
& rj)) - (3 * s_^2 * \sin(fi)) / (ri - s_ * (ri - rj)))) / (12 * (ni^2 - \\
& 1)) + uj3 * ((E * (hi - s_ * (hi - hj))^3 * ((6 * s_ / L - (2 * ri - \\
& 2 * s_ * (ri - rj)) / (L * (ri - s_ * (ri - rj)))) / (12 * (ni^2 - 1)) - \\
& (E * ni * (hi - s_ * (hi - hj))^3 * ((2 * s_ * \sin(fi)) / (ri - s_ * (ri - \\
& rj)) - (3 * s_^2 * \sin(fi)) / (ri - s_ * (ri - rj)))) / (12 * (ni^2 - \\
& 1)) + (E * ni * uj1 * \sin(fi) * (hi - s_ * (hi - hj))^3 / (12 * (ni^2 - \\
& 1) * (ri - s_ * (ri - rj))))); \\
& ui2 * ((E * (hi - s_ * (hi - hj))^3 * ((6 * s_^2 * \sin(fi)) / (L * (ri - \\
& s_ * (ri - rj))) - (6 * s_ * \sin(fi)) / (L * (ri - s_ * (ri - \\
& rj)))) / (12 * (ni^2 - 1)) + (E * ni * (hi - s_ * (hi - \\
& hj))^3 * ((12 * s_ / L^2 - (6 * ri - 6 * s_ * (ri - rj)) / (L^2 * (ri - \\
& s_ * (ri - rj)))) / (12 * (ni^2 - 1)) - uj2 * ((E * (hi - s_ * (hi - \\
& hj))^3 * ((6 * s_^2 * \sin(fi)) / (L * (ri - s_ * (ri - rj))) - \\
& (6 * s_ * \sin(fi)) / (L * (ri - s_ * (ri - rj)))) / (12 * (ni^2 - \\
& 1)) + (E * ni * (hi - s_ * (hi - hj))^3 * ((12 * s_ / L^2 - (6 * ri - \\
& 6 * s_ * (ri - rj)) / (L^2 * (ri - s_ * (ri - rj)))) / (12 * (ni^2 - 1)) - \\
& ui3 * ((E * (hi - s_ * (hi - hj))^3 * ((4 * s_ * \sin(fi)) / (ri - s_ * (ri - \\
& rj)) - (3 * s_^2 * \sin(fi)) / (ri - s_ * (ri - rj)))) / (12 * (ni^2 - 1)) - \\
& (E * ni * (hi - s_ * (hi - hj))^3 * ((6 * s_ / L - (4 * ri - 4 * s_ * (ri - \\
& rj)) / (L * (ri - s_ * (ri - rj)))) / (12 * (ni^2 - 1)) - uj3 * ((E * (hi - \\
& s_ * (hi - hj))^3 * ((2 * s_ * \sin(fi)) / (ri - s_ * (ri - rj)) - \\
& (3 * s_^2 * \sin(fi)) / (ri - s_ * (ri - rj)))) / (12 * (ni^2 - 1)) - \\
& (E * ni * (hi - s_ * (hi - hj))^3 * ((6 * s_ / L - (2 * ri - 2 * s_ * (ri - \\
& rj)) / (L * (ri - s_ * (ri - rj)))) / (12 * (ni^2 -
\end{aligned}$$

```

1))) + (E*uj1*sin(fi)*(hi-s_*(hi-hj))^3)/(12*(ni^2-
1)*(ri-s_*(ri-rj))))];

sigma_s(ele)=(N(1)/h+sqrt((6*N(3)/(h.^2)).^2));
sigma_t(ele)=(N(2)/h+sqrt((6*N(4)/(h.^2)).^2));
sigma_red(ele)=sqrt(sigma_s(ele).^2+sigma_t(ele).^2-
sigma_s(ele)*sigma_t(ele));
end

format short
ele=1:lele;

if fl1~=0
    j=j+1;
    blad(j)=i;
    sigma_red(:)=0;
end

```

ZALĄCZNIK 2 - SIGMA_GEN.M

```
clear blad parametr

m=1; %pierwszy przypadek
n=10; %ostatni przypadek
j=0;
i=1;
blad=0;
sigma_wyn=zeros(18,n);
parametr=0;
=inp
for i=m:n

    rmin=(input(1,i));
    rmax=(input(2,i));
    omega=(input(3,i));
    mli=input(4,i);
    zli=input(5,i);
    rli=(input(6,i));

    hh=ones(2,18);
    hh(1,1)=input(7,i);
    hh(2,1)=hh(1,1);

    for k=2:18
        hh(1,k)=input(k+6,i);
        hh(2,k)=hh(1,k-1);
    end

    tarcza_1D;

    sigma_wyn(:,i)=sigma_red';
end

while blad~=0

    parametr=[parametr,blad];
    sigma_gen_poprawiacz;

end
parametr(1)=[];
```

```

clear wyniki
wynik(:,m:i)=[input(:,m:i);sigma_wyn];

ii=1;
parametr2=size(parametr);
while ii<2+i-m-parametr2(1,2)
    if wynik(25,ii)==0
        wynik(:,ii)=[];
    else
        ii=ii+1;
    end
end
% disp('wynik')
% disp(wynik)

inpSSN=wynik(1:24,:);
out=wynik(25:end,:);

disp(' ----- Policzyló się ----- ')
disp('zakonczylo na ')
disp(i)

```

ZAŁĄCZNIK 3 - SIGMA_GEN_POPRAWIACZ.M

sigma_gen_poprawiacz.m

```
aa=blad;  
jj=size(aa);  
j=jj(2);  
  
for l=1:j  
  
    i=i+1;  
  
    rmin=input(1,i);  
    rmax=input(2,i);  
    omega=input(3,i);  
    mli=input(4,i);  
    zli=input(5,i);  
    rli=input(6,i);  
  
    hh=ones(2,18);  
    hh(1,1)=input(7,i);  
    hh(2,1)=hh(1,1);  
  
    for k=2:18  
        hh(1,k)=input(k+6,i);  
        hh(2,k)=hh(1,k-1);  
    end  
  
    tarcza_1D;  
    %a=blad(l);  
    %sigma_wyn(:,a)=sigma_red';  
    sigma_wyn(:,i)=sigma_red';  
end  
  
bbb=blad;  
bb=blad;  
cc=size(aa);  
bb(1:cc(2))=[];  
blad=bb;
```


ZALĄCZNIK 4 - GENERATOR_INPUT.M

```
tic
clc
clear all

input=zeros(10000,24);

%inputt = xlsread('input.xlsx',2,'G3:G4')%DSH26');
input= (xlsread('input.xlsx',10,'C10:Z8013'))';
disp(' ----- Wczytano dane ----- ')

toc
```

ZALĄCZNIK 5 - GENERATOR_OUT.M

```
out1=out(1,:)*10^-6;  
out2=out(2,:)*10^-6;  
out3=out(3,:)*10^-6;  
out4=out(4,:)*10^-6;  
out5=out(5,:)*10^-6;  
out6=out(6,:)*10^-6;  
out7=out(7,:)*10^-6;  
out8=out(8,:)*10^-6;  
out9=out(9,:)*10^-6;  
out10=out(10,:)*10^-6;  
out11=out(11,:)*10^-6;  
out12=out(12,:)*10^-6;  
out13=out(13,:)*10^-6;  
out14=out(14,:)*10^-6;  
out15=out(15,:)*10^-6;  
out16=out(16,:)*10^-6;  
out17=out(17,:)*10^-6;  
out18=out(18,:)*10^-6;  
disp(' OK ')
```

ZALĄCZNIK 6 – FUNKCJA_MASA.M

```
function masa=funkcja_masa(xx,~)

global r_max r_min rbl rbr lbl lbr rb h0 hmin rho hb
sigma_dop inp_y liczba_sieci

%xx=[R,R1,R2,R3,R4,rx,hb]
R=xx(1);
R1=xx(2);
R2=xx(3);
R3=xx(4);
R4=xx(5);
rx=xx(6);
hb=xx(7);

P=zeros(9,1);

fun= @(x) (-sqrt(R.^2-x.^2)+rx).^2;
P(1)=pi*integral(fun,-R,-R+(h0-hmin)/2);
P(2)=pi*hmin*r_max^2;
P(9)=pi*h0*r_min^2;
P(3)=pi/3*lbl*((rbl+hb)^2+(rbl+hb)*(rb+hb)+(rb+hb)^2-
-(rbl^2+rbl*rb+rb^2));
P(4)=pi/3*lbr*((rbr+hb)^2+(rbr+hb)*(rb+hb)+(rb+hb)^2-
-(rbr^2+rbr*rb+rb^2));

%% sprawdzenie wzorów
% xE=x yE=y yD=z
% xE^2+(yE-yD)^2==R1^2
% yE==t*xE+B
% yE==-1/t*xE+yD
% https://www.wolframalpha.com/input/?i=x%5E2%2B%28y-z%29%5E2%3DR%5E2%3B+y%3Dt\*x%2BB%3B+y%3D-1%2Ft\*x%2Bz
%% zaokraglenie lewe górne (R1)
t=(rb-rbl)/lbl;
B=rb+hb-t*R1;
if t~=0
    % xE=-((R1)*t)/sqrt(t^2 + 1);
    % yE=(B*sqrt(t^2+1)-(R1)*t^2)/sqrt(t^2+1);
    % yD=(B*sqrt(t^2+1)-(R1)*t^2-(R1))/sqrt(t^2+1);
    xE=((R1)*t)/sqrt(t^2 + 1);
    yE=(B*sqrt(t^2+1)+(R1)*t^2)/sqrt(t^2+1);
    yD=(B*sqrt(t^2+1)+(R1)*t^2+(R1))/sqrt(t^2+1);
    p(1)=pi/3.*(R1-xE).*(rb.^2+rb.*yE+yE.^2);
else
    xE=0;
```

```

    %yD=rb-R1;
    yD=rb+R1;
    p(1)=pi*rb^2*R1;
end
fun= @(x) (-sqrt(R1.^2-x.^2)+yD(1)).^2;
p(2)=pi*integral(fun,xE,R1);
P(5)=p(2)-p(1);
%% zaokraglenie prawe górne (R2)
t=(rbr-rb)/lbr;
B=rb+hb+R2*t;
if t~=0
    % xE=-((R2)*t)/sqrt(t^2 + 1)
    % yE=(B*sqrt(t^2+1)-(R2)*t^2)/sqrt(t^2+1)
    % yD=(B*sqrt(t^2+1)-(R2)*t^2-(R2))/sqrt(t^2+1)
    xE=((R2)*t)/sqrt(t^2 + 1);
    yE=(B*sqrt(t^2+1)+(R2)*t^2)/sqrt(t^2+1);
    yD=(B*sqrt(t^2+1)+(R2)*t^2+(R2))/sqrt(t^2+1);
    p(1)=pi/3.*(R2+xE).*(rb.^2+rb.*yE+yE.^2);
else
    xE=0;
    %yD=rb-R2;
    yD=rb+R2;
    p(1)=pi*rb^2*R2;
end
fun= @(x) (-sqrt(R2.^2-x.^2)+yD(1)).^2;
p(2)=pi*integral(fun,-R2,xE);
P(6)=p(2)-p(1);
%% zaokraglenie prawe dolne (R3)
t=(rb-rbr)/lbr;
B=rb+t*R3;
if t~=0
    xE=-((R3)*t)/sqrt(t^2 + 1);
    yE=(B*sqrt(t^2+1)-(R3)*t^2)/sqrt(t^2+1);
    yD=(B*sqrt(t^2+1)-(R3)*t^2-(R3))/sqrt(t^2+1);
    % xE=((R3)*t)/sqrt(t^2 + 1)
    % yE=(B*sqrt(t^2+1)+(R3)*t^2)/sqrt(t^2+1)
    % yD=(B*sqrt(t^2+1)+(R3)*t^2+(R3))/sqrt(t^2+1)
    p(1)=pi/3.*(R3+xE).*(rb.^2+rb.*yE+yE.^2);
else
    xE=0;
    yD=rb-R3;
    %yD=rb+R3;
    p(1)=pi*rb^2*R3;
end
fun= @(x) (-sqrt(R3.^2-x.^2)+yD(1)).^2;
p(2)=pi*integral(fun,-R3,xE);
P(7)=p(1)-p(2);

```

```

%% zaokrąglanie lewe dolne (R4)
t=(rb-rbl)/lbl;
B=rb-t*R4;
if t~=0
    xE=-((R4)*t)/sqrt(t^2 + 1);
    yE=(B*sqrt(t^2+1)-(R4)*t^2)/sqrt(t^2+1);
    yD=(B*sqrt(t^2+1)-(R4)*t^2-(R4))/sqrt(t^2+1);
    % xE=((R4)*t)/sqrt(t^2 + 1)
    % yE=(B*sqrt(t^2+1)+(R4)*t^2)/sqrt(t^2+1)
    % yD=(B*sqrt(t^2+1)+(R4)*t^2+(R4))/sqrt(t^2+1)
    p(1)=pi/3.*(R4+xE).*(rb.^2+rb.*yE+yE.^2);
else
    xE=0;
    yD=rb-R4;
    %yD=rb+R4;
    p(1)=pi*rb^2*R4;
end
fun= @(x) (-sqrt(R4.^2-x.^2)+yD).^2;
p(2)=pi*integral(fun,-xE,R4);
P(8)=p(1)-p(2);

%funkcja_masa(inp_x(:,1))

%P(6)

V=sum(P(1:8))+P(2)-P(9);

clear masa;

masa=V*rho;

y=NN([xx';inp_y(:,1)]);

for ii=1:liczba_sieci-1
    if y(ii)>sigma_dop(ii)
        masa=masa+1000*(y(ii)-sigma_dop(ii));
    else
        end
end

end

```

ZALĄCZNIK 7 – SIGMY.M

```
function[sigma_tarcza,sigma_bebna]=sigmy(x,y)

global iteracja
global orb omega
%% uruchomienie APDL
orb=initialize_orb();
%load_ansys_aas;

iCoMapdlUnit=actmapdlserver(orb,'C:\WorkingFolder\aaS_
MapdlId.txt');
char(iCoMapdlUnit.executeCommandToString('/CLEAR,
START '));
char(iCoMapdlUnit.executeCommandToString('/units,SI'))
;
%% wprowadzanie parametrów

%execwbcommand('systems=GetAllSystems()')
char(iCoMapdlUnit.executeCommandToString(['omega='
num2str(omega)]));
char(iCoMapdlUnit.executeCommandToString(['r_max='
num2str(y(1))]]));
char(iCoMapdlUnit.executeCommandToString(['r_min='
num2str(y(2))]]));
char(iCoMapdlUnit.executeCommandToString(['h0='
num2str(y(3))]]));
char(iCoMapdlUnit.executeCommandToString(['hmin='
num2str(y(4))]]));
char(iCoMapdlUnit.executeCommandToString(['lbl='
num2str(y(5))]]));
char(iCoMapdlUnit.executeCommandToString(['rbl='
num2str(y(6))]]));
char(iCoMapdlUnit.executeCommandToString(['rbr='
num2str(y(7))]]));
char(iCoMapdlUnit.executeCommandToString(['lbr='
num2str(y(8))]]));
char(iCoMapdlUnit.executeCommandToString(['rb='
num2str(y(9))]]));

char(iCoMapdlUnit.executeCommandToString(['R='
num2str(x(1))]]));
char(iCoMapdlUnit.executeCommandToString(['R1='
num2str(x(2))]]));
char(iCoMapdlUnit.executeCommandToString(['R2='
num2str(x(3))]]));
```

```

char(iCoMapdlUnit.executeCommandToString(['R3='
num2str(x(4))]));
char(iCoMapdlUnit.executeCommandToString(['R4='
num2str(x(5))]));
char(iCoMapdlUnit.executeCommandToString(['rx='
num2str(x(6))]));
char(iCoMapdlUnit.executeCommandToString(['hb='
num2str(x(7))]));

char(iCoMapdlUnit.executeCommandToString(['sigma_w='
num2str(y(10))]));

char(iCoMapdlUnit.executeCommandToString('/prep7'));
%% własności elementu
char(iCoMapdlUnit.executeCommandToString('ET,1,PLANE18
3,,,1'));
%% dane materialowe (OK)
char(iCoMapdlUnit.executeCommandToString('MPTEMP,1,0
'));
char(iCoMapdlUnit.executeCommandToString(['MPDATA,EX,1
,, ' num2str(y(11))]));
char(iCoMapdlUnit.executeCommandToString(['MPDATA,DENS
,1,, ' num2str(y(12))]));
char(iCoMapdlUnit.executeCommandToString(['MPDATA,PRXY
,1,, ' num2str(y(13))]));

%% !! punkty do tarczy

char(iCoMapdlUnit.executeCommandToString('K,1,r_min,lb
r+h0/2,0, '));
char(iCoMapdlUnit.executeCommandToString('K,2,r_min,lb
r-h0/2,0, '));
char(iCoMapdlUnit.executeCommandToString('K,3,rx,lbr+h
min/2,0'));
char(iCoMapdlUnit.executeCommandToString('K,4,rx,lbr-
hmin/2,0'));
char(iCoMapdlUnit.executeCommandToString('K,5,r_max,lb
r+hmin/2,0'));
char(iCoMapdlUnit.executeCommandToString('K,6,r_max,lb
r-hmin/2,0'));

%%!! punkty srodkowe bebna
char(iCoMapdlUnit.executeCommandToString('K,7,rb+hb/2,
lbr+hmin/2,0'));

```

```

char(iCoMapdlUnit.executeCommandToString('K,8,rb-
hb/2,lbr+hmin/2,0'));

char(iCoMapdlUnit.executeCommandToString('K,19,rb+hb/2
,lbr-hmin/2,0'));
char(iCoMapdlUnit.executeCommandToString('K,20,rb-
hb/2,lbr-hmin/2,0'));

%% punkty do bebna prawego

char(iCoMapdlUnit.executeCommandToString('K,9,rbr+hb/2
,0,0'));
char(iCoMapdlUnit.executeCommandToString('K,10,rbr-
hb/2,0,0'));

%% punkty do bebna lewego

char(iCoMapdlUnit.executeCommandToString('K,11,rbl+hb/
2,lbl+lbr,0'));
char(iCoMapdlUnit.executeCommandToString('K,12,rbl-
hb/2,lbl+lbr,0'));

char(iCoMapdlUnit.executeCommandToString('K,17,rx,lbr+
h0/2,0,');
char(iCoMapdlUnit.executeCommandToString('K,18,rx,lbr-
h0/2,0,');

%% definiowanie prostych

char(iCoMapdlUnit.executeCommandToString('LSTR, 1, 2
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 2, 18
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 1, 17
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 3, 8
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 7, 5
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 4, 20
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 19, 6
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 6, 5
'));

```



```

char(iCoMapdlUnit.executeCommandToString('LSTR, 10, 9
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 12, 11
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 12, 8
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 20, 10
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 11, 7
'));
char(iCoMapdlUnit.executeCommandToString('LSTR, 19, 9
'));

% zaokraglenia

char(iCoMapdlUnit.executeCommandToString('K,13,rx,lbr+
hmin/2+R,0'));
char(iCoMapdlUnit.executeCommandToString('K,14,rx-
R,lbr+hmin/2+R,0'));

char(iCoMapdlUnit.executeCommandToString('K,15,rx,lbr-
hmin/2-R,0'));
char(iCoMapdlUnit.executeCommandToString('K,16,rx-
R,lbr-hmin/2-R,0'));

char(iCoMapdlUnit.executeCommandToString('LARC,3,14,13
,R, '));
char(iCoMapdlUnit.executeCommandToString('
LARC,16,4,15,R, '));

char(iCoMapdlUnit.executeCommandToString('LSBL,3,15,,K
EEP,KEEP'));
char(iCoMapdlUnit.executeCommandToString('LSBL,15,18'
));
char(iCoMapdlUnit.executeCommandToString('LSBL,2,16,
,KEEP,KEEP'));
char(iCoMapdlUnit.executeCommandToString('LSBL,16,18'
));
char(iCoMapdlUnit.executeCommandToString('
LDELE,20,22,2'));
char(iCoMapdlUnit.executeCommandToString('
LDELE,2,3'));

%% zaokraglenia tarcza-beben

char(iCoMapdlUnit.executeCommandToString('LFILLT,4,11,
R1, , '));

```

```

char(iCoMapdlUnit.executeCommandToString('LFILLT,13,5,
R2, , '));
char(iCoMapdlUnit.executeCommandToString('LFILLT,14,7,
R3, , '));
char(iCoMapdlUnit.executeCommandToString('LFILLT,6,12,
R4, , '));

%% powierzchnia

char(iCoMapdlUnit.executeCommandToString('AL,all'));

%% siatka
char(iCoMapdlUnit.executeCommandToString('AESIZE,1,0.0
02, '));
char(iCoMapdlUnit.executeCommandToString('AMESH,1'));

%% warunki brzegowe i obciazenia (z palca, sprawdzic
wartosci)
% symetria
char(iCoMapdlUnit.executeCommandToString('DL, 10,
,SYMM'));
char(iCoMapdlUnit.executeCommandToString('DL, 9,
,SYMM'));

%predkosc ratowa
char(iCoMapdlUnit.executeCommandToString('OMEGA,0,omeg
a,0, '));

% sila od lopatek
char(iCoMapdlUnit.executeCommandToString('SFL,8,pres,-
sigma_w, '));

%% rozwiazywanie
%char(iCoMapdlUnit.executeCommandToString('FINISH'));
char(iCoMapdlUnit.executeCommandToString('/SOL'));

char(iCoMapdlUnit.executeCommandToString('/STATUS,SOLU
'));
char(iCoMapdlUnit.executeCommandToString('SOLVE'));

char(iCoMapdlUnit.executeCommandToString('finish'));
%% sciezka wzdluz tarczy
char(iCoMapdlUnit.executeCommandToString('/POST1 '));
char(iCoMapdlUnit.executeCommandToString('PATH,PATH1,2
,30,20, '));
char(iCoMapdlUnit.executeCommandToString('PPATH,1,0,r_
min,lbr,0,0, '));

```


ZAŁĄCZNIK 8 – MAIN_PROGRAM.M

```
%% program główny
%clc
%clear all

global r_max r_min rbl rbr lbl lbr rb h0 hmin rho
sigma_dop inp_y
global orb sigma_w omega

global liczba_sieci
global iteracja
%https://www.google.com/search?ei=HFYMX535G9D1qwhAoomIAw&q=apdl+sercere+%2847%29%3A+write+to+readonly+file%2C+unit+70+&oq=apdl+sercere+%2847%29%3A+write+to+readonly+file%2C+unit+70+&gs\_lcp=CgZwc3ktYWIQDFDTJ1jdMWDTUGgBcAB4AYABbYgBkgKSAQMyLjGYAQCGAQGqAQdnd3Mtd2l6&sclient=psy-ab&ved=0ahUKEwid987nn8rqAhXQ-ioKHUBRAjEQ4dUDCAw

liczba_sieci=21+12; %21
nn=50; %wielkosc bazy uczacej
nr_st=12;
bemben_min=0.003;
iteracja =1;

sigma_dop=ones(liczba_sieci,1)*270000000;

if petla==0
    masa=99;
    %% zmienne decyzyjne (x)
    R=0.01; %promień zaokrąglenia stopki
    R1=0.005; %promienie połączenia tarcza-bęben
    R2=0.005;
    R3=0.005;
    R4=0.005;
    rx=0.08; % położenie środka okręgu R
    hb=0.0035; %grubosc bebna 0.0035

    %% stale geometria itp (y)
    r_max=stopnie(2,nr_st);
    r_min=stopnie(3,nr_st);
    h0=stopnie(4,nr_st);
    hmin=stopnie(5,nr_st);
```

```

lbl= stopnie(6,nr_st); % położenie końca bębna lewego
rbl=stopnie(7,nr_st);
lbr=stopnie(8,nr_st); % położenie końca bębna prawego
rbr=stopnie(9,nr_st);
rb=stopnie(10,nr_st); %położenie połączenia tarcza-
bęben

E=stopnie(11,nr_st); %moduł Younga
rho=stopnie(12,nr_st) ; %gęstość
eta=stopnie(13,nr_st) ; %liczba Poissona

ml=stopnie(15,nr_st); % masa łopatki
ilosc_lopatek=stopnie(16,nr_st);
r_sm=stopnie(17,nr_st); %położenie środka masy

%położenie środka masy łopatki, pierwsza część to
środek masy łopatki
%względem sworznia, druga to promień położenia
sworznia
r_sml=stopnie(18,nr_st);
Vol=stopnie(19,nr_st); % objętość cz. zamkowej
n=stopnie(20,nr_st); % źródło
: http://www.k204.ru/books/vrd/wiki2/PDF/Lyulka.pdf
strona 18
omega=n*pi/30; %zmieniać ręcznie w "sigmy"
Ft=Vol*rho*r_sm*omega^2; %siła od części zamkowej
Fz=ml*r_sml*omega^2; %siła od łopatek
sigma_w=(Ft+Fz)*ilosc_lopatek/(2*pi*r_max*hmin); %z
niebieskiej serii s 342 (5.34)

end

resetAPDL
while petla<15
    petla=petla+1

    orb=initialize_orb();
    %StartAPDL

    disp('Start Loading AnsysAAS toolbox...');
    matlab_version=version();
    toolbox_version='1.1.9';
    jarPath=fileparts(which('initialize_orb'));

    jarPathAndFileJava17=sprintf('%s/%s',jarPath,'ANSYS_AA
S.jar');

```

```

jarPathAndFileJava18=sprintf('%s/%s',jarPath,'ANSYS_AA
S_jdk_1.8.0_121.jar');
javaaddpath(jarPathAndFileJava18);
fprintf('AnsysAAS toolbox %s loaded in Matlab
%s.\n',toolbox_version,matlab_version)

% load_ansys_aas;

pause(12)
%% zmienne i stałe do funkcji
% test programu "sigmy"
if rx-R>r_min
else
rx=R*1.01+r_min;
end

if x==0
xx=[R,R1,R2,R3,R4,rx,hb]';
else
xx=x';
end

yy=[r_max,r_min,h0,hmin,lbl,rbl,lbr,rbr,rb,sigma_w,E,r
ho,eta]';

%sigmy(xx,yy)
%[sigma_tarcza,sigma_bebna]=sigmy(xx,yy)
% generowanie bazy do uczenia SSN
% losowanie wartosci
inp_x=(rand(7,nn)-0.5)*0.2.*xx+xx; %macierz zmiennych
inp_y=ones(13,nn).*yy; %macierz stalych
inp=[inp_x;inp_y]; %macierz stalych i zmiennych do
uczenie SSN
%inp_TEST=[inp_x;inp_y];

% disp(masa)
% if masa>100
% return
% end
%% generowanie danych uczacych
disp('----- generowanie danych uczacych -----
')
tic
out=zeros(liczba_sieci,nn);
for i=1:nn
if inp_x(6,i)<inp_x(1,i)

```

```

else
inp_x(6,i)=inp_x(1,i)*1.01+r_min;
end
[yyy,zzz]=sigmy(inp_x(:,i),inp_y(:,i));
out(:,i)=[yyy;zzz];

end
toc

%% trenowanie sieci
disp('----- trenowanie sieci -----
')
tic
Trener_Sieci(inp,out);
toc
disp('----- test sieci -----
')
tic
y=NN(inp(:,1));
toc
%% optymalizacja

funkcja_masa(inp_x(:,1)')

A = [];
b = [];
Aeq = [];
beq = [];

%xx=[R,R1,R2,R3,R4,rx,hb]
zakres=20;
lb=xx-zakres/100*xx;
ub=xx+zakres/100*xx;
if lb(6,:)-lb(1,:)<r_min
else
lb(6,:)=lb(1,:)*1.01+r_min;
end
if lb(7,:)<bemben_min
lb(7,:)=bemben_min;
end
disp(' ----- optymalizacja ----- ')
tic
% https://www.mathworks.com/help/gads/ga.html
options =
optimoptions('ga','MaxGenerations',2,'PopulationSize',
50,...
'InitialPopulationMatrix',xx');

```



```

    %'InitialPopulationRange'
    [x,masa] =
ga(@funkcja_masa,7,A,b,Aeq,beq,lb,ub,[],[],options);

    disp('x')
    disp (x')
    if x(6)<x(1)
    else
    x(6)=x(1)*1.01+r_min;
    disp('wyniki optymalizacji poza zakresem
dopuszczalnym')
    masa=funkcja_masa(x);
    end
    % disp('sigma APDL i SSN')
    % [yyyy,zzzz]=sigmy(x,yy);
    % pp1=[yyyy;zzzz];
    % pp2=NN([x';inp_y(:,1)]);
    % disp([pp1,pp2])
    %
    % disp('V')
    % format long
    % disp (masa/rho)
    % format short
    %
    % disp('R,R1,R2,R3,R4,rx,hb')
    % disp([x;xx'])
    disp ('masa')
    disp (masa)
    % if masa>100
    % return
    % end
    disp ('petla')
    disp (petla)
    clear orb
    save last.mat
end
clear orb
save last.mat

z=yy;
z(10:13)=[];
xxx=x';
fprintf('r_max= %8.3f \nr_min= %8.3f \nh0= %8.3f
\nhmin= %8.3f \nlbl= %8.3f \nrbl= %8.3f \nlbr= %8.3f
\nrbr= %8.3f \nr= %8.3f \n',z)

```

```

fprintf('R=%8.4f\nR1=%8.4f\nR2=%8.4f\nR3=%8.4f\nR4=%8.
4f\nrx=%8.4f\nhb=%8.4f\n',x')
disp ('masa')
disp (masa)

%do gripa
fprintf('r_max= %8.3f \nr_min= %8.3f \nh0= %8.3f
\nh_min= %8.3f \nlbl= %8.3f \nrbl= %8.3f \nlbr= %8.3f
\nrbr= %8.3f \nr= %8.3f \n',z*1000)

fprintf('R=%8.4f\nR1=%8.4f\nR2=%8.4f\nR3=%8.4f\nR4=%8.
4f\nrx=%8.4f\nhb=%8.4f\n',x'*1000)

```

ZALĄCZNIK 9 – NN.M

```
function y=NN(x)
%NN(inp(:,1))
global liczba_sieci
global k

y=zeros(liczba_sieci,1);

for i=1:liczba_sieci
y(i)=sim(k{i},x);
end
%y(1)=sim(k{1},inp(:,1))
end
```

ZALĄCZNIK 10 – RESETAPDL

```
function resetAPDL

global iteracja
%% usuwanie plików z pamięci
system('taskkill /F /IM ANSYS.EXE'); %zamknij -AAS
delete('C:\WorkingFolder\*'); %usuń pliki
z WorkingFolder
%% uruchom -AAS za pomocą myszki
import java.awt.Robot; %uruchom -AAS za pomocą myszki
import java.awt.event.*;
mouse = Robot;
mouse.mouseMove(0,0);
mouse.mouseMove(650,800);
mouse.mousePress(InputEvent.BUTTON1_MASK);
mouse.mouseRelease(InputEvent.BUTTON1_MASK);
pause(20)
%%

iteracja=1;

end
```

ZALĄCZNIK 11 – TRENER_SIECI

```
function Trener_Sieci(iinp, oout)

global liczba_sieci
global k
global net
net = feedforwardnet([2]);
net.trainFcn = 'trainscg';
%net = train(net, inp, out(1,:));
%view(net)

sigma_=net;

sigma_.trainParam.showWindow=0;
sigma_.trainParam.max_fail = 50;

j=3;

k=cell(1,liczba_sieci);
% sigma_test=sigma_
% train(sigma_test,inp,oout(1,:));

for kk=1:liczba_sieci
    for i=1:j
        k{kk}=sigma_;
        k{kk}=train(k{kk},iinp,oout(kk,:));
    end
end

%y(1)=sim(k{1},inp(:,1));

end
```

ZALĄCZNIK 12 – PORÓWNANIE_SIECI.M

```
%porównanie sieci
clc
clear all
%load 'inp_out+-10%'
load 'inp_out+-5%'
n=2; % ilość uczenia sieci
nn=50; %ilość neuronów w stopniu
k=3; %nr testowanego wezla
out2=out(k,:);
%out_TEST=out_TEST(k,:);
%
% network_3_50 = feedforwardnet([nn]);
% network_4_50 = feedforwardnet([nn,nn]);
network_3_50 = feedforwardnet([nn,nn,nn]);
network_4_50 = feedforwardnet([nn,nn,nn,nn]);
%% SCG

network_3_50_1_out=network_3_50;
network_4_50_1_out=network_4_50;
for i=1:n
    disp('scg')
    %siec 3 warstwy uktyte
    network_3_50_1_out.trainFcn = 'trainscg';
    network_3_50_1_out.trainParam.showWindow=0;
    network_3_50_1_out.trainParam.max_fail = 50;
    disp('czas uczenia 3 war')
    tic

network_3_50_1_out=train(network_3_50_1_out,inp,out2);
toc
    out_TEST_3_50_1_out=sim(network_3_50_1_out,inp_TEST);
    out_3_50_1_out=sim(network_3_50_1_out,inp);

    % 4 warstwy ukrte
    network_4_50_1_out.trainFcn = 'trainscg';
    network_4_50_1_out.trainParam.showWindow=0;
    network_4_50_1_out.trainParam.max_fail = 50;
    disp('czas uczenia 4 war')
    tic

network_4_50_1_out=train(network_4_50_1_out,inp,out2);
toc
    out_4_50_1_out=sim(network_4_50_1_out,inp);
    out_TEST_4_50_1_out=sim(network_4_50_1_out,inp_TEST);
```

```

% wyniki zbiorcze
scg_=[out(2,:) ',out_3_50_1_out',out_4_50_1_out'];

scg_TEST=[out_TEST(2,:) ',out_TEST_3_50_1_out',out_TEST_4_50_1_out'];

%disp('scg')
%disp(' out 3 war 4 war TEST 3 war 4 war ');
scg=[scg_,scg_TEST];
%disp (scg);
end
clear out_4_50_1_out out_TEST_4_50_1_out
%% gdm
disp('gdm')
network_3_50_1_out=network_3_50;
network_4_50_1_out=network_4_50;
for i=1:n
    % siec 3 warstwy uktyte
    network_3_50_1_out.trainFcn = 'traingdm';
    network_3_50_1_out.trainParam.showWindow=0;
    network_3_50_1_out.trainParam.max_fail = 50;
    disp('czas uczenia 3 war')
    tic

network_3_50_1_out=train(network_3_50_1_out,inp,out2);
    toc
    out_TEST_3_50_1_out=sim(network_3_50_1_out,inp_TEST);
    out_3_50_1_out=sim(network_3_50_1_out,inp);

    % 4 warstwy ukrte
    network_4_50_1_out.trainFcn = 'traingdm';
    network_4_50_1_out.trainParam.showWindow=0;
    network_4_50_1_out.trainParam.max_fail = 50;
    disp('czas uczenia 4 war')
    tic

network_4_50_1_out=train(network_4_50_1_out,inp,out2);
    toc
    out_4_50_1_out=sim(network_4_50_1_out,inp);
    out_TEST_4_50_1_out=sim(network_4_50_1_out,inp_TEST);

% wyniki zbiorcze
gdm_=[out(2,:) ',out_3_50_1_out',out_4_50_1_out'];

gdm_TEST=[out_TEST(2,:) ',out_TEST_3_50_1_out',out_TEST_4_50_1_out'];

```

```

% disp('scg')
% disp(' out 3 war 4 war TEST 3 war 4 war ');
gdm=[gdm_,gdm_TEST];
% disp (gdm);
end
clear out_4_50_1_out out_TEST_4_50_1_out
%% RP
disp('rp')
network_3_50_1_out=network_3_50;
network_4_50_1_out=network_4_50;
for i=1:n
    % siec 3 warstwy uktyte
    network_3_50_1_out.trainFcn = 'trainrp';
    network_3_50_1_out.trainParam.showWindow=0;
    network_3_50_1_out.trainParam.max_fail = 50;
    disp('czas uczenia 3 war')
    tic

network_3_50_1_out=train(network_3_50_1_out,inp,out2);
toc
out_TEST_3_50_1_out=sim(network_3_50_1_out,inp_TEST);
out_3_50_1_out=sim(network_3_50_1_out,inp);

    % 4 warstwy ukrte
    network_4_50_1_out.trainFcn = 'traingdm';
    network_4_50_1_out.trainParam.showWindow=0;
    network_4_50_1_out.trainParam.max_fail = 50;
    disp('czas uczenia 4 war')
    tic

network_4_50_1_out=train(network_4_50_1_out,inp,out2);
toc
out_4_50_1_out=sim(network_4_50_1_out,inp);
out_TEST_4_50_1_out=sim(network_4_50_1_out,inp_TEST);

    % wyniki zbiorcze
    rp_=[out(2,:) ',out_3_50_1_out',out_4_50_1_out'];

rp_TEST=[out_TEST(2,:) ',out_TEST_3_50_1_out',out_TEST_
4_50_1_out'];

    % disp('scg')
    disp(' out 3 war 4 war TEST 3 war 4 war ');
    rp=[rp_,rp_TEST];
    % disp (rp);
end
clear out_4_50_1_out out_TEST_4_50_1_out

```



```
porownanie=[scg,gdm,rp]  
disp('END')
```

ZALĄCZNIK 13 – FUNKCJA.M

```
function blad=funkcja(x)
%tic

global inp inp_TEST out out2 out_TEST nn

x=ceil(x);
%% SCG
network_3_50 = feedforwardnet(x);
network_3_50_1_out=network_3_50;

%disp('scg')
%siec 3 warstwy uklyte
network_3_50_1_out.trainFcn = 'trainscg';
network_3_50_1_out.trainParam.showWindow=0;
network_3_50_1_out.trainParam.max_fail = 50;
%disp('czas uczenia 3 war')

for i=1:nn

network_3_50_1_out=train(network_3_50_1_out,inp,out2);
end
out_TEST_3_50_1_out=sim(network_3_50_1_out,inp_TEST);
%out_3_50_1_out=sim(network_3_50_1_out,inp);

% wyniki zbiorcze
%scg_=[out(2,:)','out_3_50_1_out'];
scg_TEST=[out_TEST(2,:)','out_TEST_3_50_1_out'];

%disp('scg')
%disp(' out 3 war 4 war TEST 3 war 4 war ');
scg=scg_TEST;
%disp (scg);

%czas=toc;
bladd=((scg(:,1)-scg(:,2))./scg(:,1))*100;
%max(bladd);
blad=std(bladd);
%blad_czas=blad*czas;

end
```

ZALĄCZNIK 14 – FUNKCJA2.M

```
function y=funkcja2(x)%porównanie sieci
tic

global inp inp_TEST out out_TEST
nn=1; %ile razy uczymy siec
out7=out(7,:);
x=ceil(x);

%% SCG
network_3_50 = feedforwardnet(x);
network_3_50_1_out=network_3_50;
network_3_50_1_out.trainFcn = 'trainscg';
%disp('scg')
%siec 3 warstwy uktyte

network_3_50_1_out.trainParam.showWindow=0;
network_3_50_1_out.trainParam.max_fail = 50;
%disp('czas uczenia 3 war')

for i=1:nn

network_3_50_1_out=train(network_3_50_1_out,inp,out7);
end
out_TEST_3_50_1_out=sim(network_3_50_1_out,inp_TEST);
%out_3_50_1_out=sim(network_3_50_1_out,inp);

% wyniki zbiorcze
%scg=[out(2,:)','out_3_50_1_out'];
scg_TEST=[out_TEST(7,:)','out_TEST_3_50_1_out'];

%disp('scg')
%disp(' out 3 war 4 war TEST 3 war 4 war ');
scg=scg_TEST;
%disp (scg);

%czas=toc;
bladd=( (scg(:,1)-scg(:,2))./scg(:,1))*100;
%blad=max(abs(bladd));
%blad=std(bladd);
y(1)=abs(mean(bladd));
y(2)=std(bladd);
%blad_czas=blad*czas;

end
```

ZALĄCZNIK 15 - FUNKCJA3.M

```
function y=funkcja3(x)%porównanie sieci
tic

global inp inp_TEST out out_TEST
nn=1; %ile razy uczymy siec
out7=out(7,:);
x=ceil(x);

%% SCG
network_3_50 = feedforwardnet(x);
network_3_50_1_out=network_3_50;
network_3_50_1_out.trainFcn = 'traingdm';

%disp('scg')
%siec 3 warstwy uktyte

network_3_50_1_out.trainParam.showWindow=0;
network_3_50_1_out.trainParam.max_fail = 50;
%disp('czas uczenia 3 war')

for i=1:nn

network_3_50_1_out=train(network_3_50_1_out,inp,out7);
end
out_TEST_3_50_1_out=sim(network_3_50_1_out,inp_TEST);
%out_3_50_1_out=sim(network_3_50_1_out,inp);

% wyniki zbiorcze
%scg=[out(2,:)','out_3_50_1_out'];
scg_TEST=[out_TEST(7,:)','out_TEST_3_50_1_out'];

%disp('scg')
%disp(' out 3 war 4 war TEST 3 war 4 war ');
scg=scg_TEST;
%disp (scg);

%czas=toc;
bladd=( (scg(:,1)-scg(:,2))./scg(:,1))*100;
%blad=max(abs(bladd));
%blad=std(bladd);
y(1)=abs(mean(bladd));
y(2)=std(bladd);
%blad_czas=blad*czas;

end
```

ZALĄCZNIK 16 – GENERATOR_KODU_GRIP.M

```
clc
%clear all

%load('3_st_v1_po_15','x')

%% wczytywanie wyników
if exist('stopnie')
else
    load('stopnie')
end

wyn_opt =
xlsread('C:\WAT\doktorat_aktualne\wyniki_optymalizacji',2,'F2:A10');

%% czyszczenie wyników nieoptymalnych
i=30;
while i>1
    wyn_opt(:,i)=[];
    wyn_opt(:,i-1)=[];
    i=i-3;
end

%% generowanie kodu GRIP
przesow=0;

E=stopnie(11,1); %moduł Younga
rho=stopnie(12,1) ; %gęstość
eta=stopnie(13,1) ; %liczba Poissona
disp('ENTITY/pkt(600),ln(600),cr(600),solid(20)')

przesow=1.74;
ll_ln=20; %liczba linii
ll_p=40; %liczba punktów
ll_cr=10;
j=-1;
for i=3:12
    nr_st=i;
    j=j+1;
```

```

l_ln=ll_ln*j; %liczba linni
l_p=ll_p*j; %liczba punktow
l_cr=ll_cr*j;

r_max=stopnie(2,nr_st)*1000;
r_min=stopnie(3,nr_st)*1000;
h0=stopnie(4,nr_st)*1000;
hmin=stopnie(5,nr_st)*1000;
lbl= stopnie(6,nr_st)*1000; % położenie końca bębna
lewego
rbl=stopnie(7,nr_st)*1000;
lbr=stopnie(8,nr_st)*1000; % położenie końca bębna
prawego
rbr=stopnie(9,nr_st)*1000;
rb=stopnie(10,nr_st)*1000; %położenie połączenia
tarcza-bęben

ml=stopnie(15,nr_st); % masa łopatk
ilosc_lopatek=stopnie(16,nr_st);
r_sm=stopnie(17,nr_st); %położenie środka masy

%położenie środka masy łopatk, pierwsza część to
środek masy łopatk
%względem sworznia, druga to promień położenia
sworznia
r_sml=stopnie(18,nr_st);
Vol=stopnie(19,nr_st); % objętość cz. zamkowej
n=stopnie(20,nr_st); % źródło
: http://www.k204.ru/books/vrd/wiki2/PDF/Lyulka.pdf
strona 18
omega=n*pi/30; %zmieniać ręcznie w "sigmy"
Ft=Vol*rho*r_sm*omega^2; %siła od części zamkowej
Fz=ml*r_sml*omega^2; %siła od lopatek
sigma_w=(Ft+Fz)*ilosc_lopatek/(2*pi*r_max*hmin); %z
niebieskiej serii s 342 (5.34)

y=[r_max,r_min,h0,hmin,lbl,rbl,rbr,lbr,rb,sigma_w,E,rh
o,eta]';
disp(['omega=' num2str(omega)])

```

```

disp(['r_max=' num2str(y(1))])
disp(['r_min=' num2str(y(2))])
disp(['h0=' num2str(y(3))])
disp(['hmin=' num2str(y(4))])
disp(['lbl=' num2str(y(5))])
disp(['rbl=' num2str(y(6))])
disp(['rbr=' num2str(y(7))])
disp(['lbr=' num2str(y(8))])
disp(['rb=' num2str(y(9))])

x=wyn_opt(3:9,nr_st-2);
disp(['R=' num2str(x(1)*1000)])
disp(['R1=' num2str(x(2)*1000)])
disp(['R2=' num2str(x(3)*1000)])
disp(['R3=' num2str(x(4)*1000)])
disp(['R4=' num2str(x(5)*1000)])
disp(['rx=' num2str(x(6)*1000)])
disp(['hb=' num2str(x(7)*1000)])

if j==0

else

przesow=przesow+lbl;
end
disp(['przesow=' num2str(przesow)])
disp('')

%% !! punkty do tarczy

fprintf('pkt(1+%d)=POINT/r_min,lbl+h0/2-
przesow,0\n',l_p);

fprintf('pkt(1+%d)=POINT/r_min,przesow,0\n',l_p);
fprintf('pkt(2+%d)=POINT/r_min,h0/2-
przesow,0\n',l_p);
fprintf('pkt(3+%d)=POINT/r_min,-h0/2-
przesow,0\n',l_p);

fprintf('ln(1+%d)=LINE/pkt(2+%d),pkt(3+%d)\n',l_ln,l_p,
l_p);

fprintf('pkt(4+%d)=POINT/r_max,hmin/2-
przesow,0\n',l_p);
fprintf('pkt(5+%d)=POINT/r_max,-hmin/2-
przesow,0\n',l_p);

```

```

fprintf('ln(2+%d)=LINE/pkt(4+%d),pkt(5+%d)\n',l_ln,l_p
,l_p);

fprintf('pkt(6+%d)=POINT/rb+hb/2,+hmin/2-
przesow,0\n',l_p);

fprintf('ln(3+%d)=LINE/pkt(6+%d),pkt(4+%d)\n',l_ln,l_p
,l_p);
fprintf('pkt(7+%d)=POINT/rb+hb/2,-hmin/2-
przesow,0\n',l_p);

fprintf('ln(4+%d)=LINE/pkt(7+%d),pkt(5+%d)\n',l_ln,l_p
,l_p);

fprintf('pkt(8+%d)=POINT/rbl+hb/2,+lbl-
przesow,0\n',l_p);

fprintf('ln(5+%d)=LINE/pkt(6+%d),pkt(8+%d)\n',l_ln,l_p
,l_p);
fprintf('pkt(9+%d)=POINT/rbr+hb/2,-lbr-
przesow,0\n',l_p);

fprintf('ln(6+%d)=LINE/pkt(7+%d),pkt(9+%d)\n',l_ln,l_p
,l_p);

fprintf('pkt(10+%d)=POINT/rb-hb/2,hmin/2-
przesow,0\n',l_p);
fprintf('pkt(11+%d)=POINT/rbl-hb/2,lbl-
przesow,0\n',l_p);

fprintf('ln(7+%d)=LINE/pkt(10+%d),pkt(11+%d)\n',l_ln,l
_p,l_p);

fprintf('ln(8+%d)=LINE/pkt(11+%d),pkt(8+%d)\n',l_ln,l_
p,l_p);

fprintf('pkt(12+%d)=POINT/rb-hb/2,-hmin/2-
przesow,0\n',l_p);
fprintf('pkt(13+%d)=POINT/rbr-hb/2,-lbr-
przesow,0\n',l_p);

fprintf('ln(9+%d)=LINE/pkt(12+%d),pkt(13+%d)\n',l_ln,l
_p,l_p);

fprintf('ln(10+%d)=LINE/pkt(13+%d),pkt(9+%d)\n',l_ln,l
_p,l_p);

```



```

fprintf('pkt (14+%d)=POINT/rx,h0/2-przesow,0\n',l_p);

fprintf('ln (11+%d)=LINE/pkt (14+%d),pkt (2+%d)\n',l_ln,l_p,l_p);
fprintf('pkt (15+%d)=POINT/rx,-h0/2-przesow,0\n',l_p);

fprintf('ln (12+%d)=LINE/pkt (15+%d),pkt (3+%d)\n',l_ln,l_p,l_p);

fprintf('pkt (16+%d)=POINT/rx,hmin/2-przesow,0\n',l_p);

fprintf('ln (13+%d)=LINE/pkt (16+%d),pkt (10+%d)\n',l_ln,l_p,l_p);

fprintf('pkt (17+%d)=POINT/rx,-hmin/2-przesow,0\n',l_p);

fprintf('ln (14+%d)=LINE/pkt (17+%d),pkt (12+%d)\n',l_ln,l_p,l_p);

fprintf('pkt (18+%d)=POINT/rx,hmin/2+R-przesow,0\n',l_p);

fprintf('cr (1+%d)=CIRCLE/CENTER,pkt (18+%d),RADIUS,R,START,180,END,270\n',l_cr,l_p);

fprintf('pkt (19+%d)=POINT/rx,-hmin/2-R-przesow,0\n',l_p);

fprintf('cr (2+%d)=CIRCLE/CENTER,pkt (19+%d),RADIUS,R,START,90,END,180\n',l_cr,l_p);

fprintf('pkt (20+%d) =
POINT/INTOF,ln (11+%d),cr (1+%d)\n',l_p,l_ln,l_cr);
fprintf('pkt (21+%d) =
POINT/INTOF,ln (12+%d),cr (2+%d)\n',l_p,l_ln,l_cr);

fprintf('pkt (22+%d)=POINT/cr (1+%d),ATANGL,269\n',l_p,l_cr);

fprintf('pkt (23+%d)=POINT/cr (2+%d),ATANGL,91\n',l_p,l_cr);

```

```

fprintf('DELETE/cr(1+%d),cr(2+%d),ln(11+%d),ln(12+%d)\n',l_cr,l_cr,l_ln,l_ln);
%

fprintf('ln(11+%d)=LINE/pkt(20+%d),pkt(2+%d)\n',l_ln,l_p,l_p);

fprintf('ln(12+%d)=LINE/pkt(21+%d),pkt(3+%d)\n',l_ln,l_p,l_p);
%

fprintf('cr(1+%d)=CIRCLE/pkt(20+%d),pkt(22+%d),pkt(16+%d)\n',l_cr,l_p,l_p,l_p);

fprintf('cr(2+%d)=CIRCLE/pkt(17+%d),pkt(23+%d),pkt(21+%d)\n',l_cr,l_p,l_p,l_p);
%

fprintf('cr(3+%d)=FILLET/XLARGE,ln(6+%d),YSMALL,ln(4+%d),RADIUS,R1\n',l_cr,l_ln,l_ln);

fprintf('cr(4+%d)=FILLET/XLARGE,ln(5+%d),YLARGE,ln(3+%d),RADIUS,R2\n',l_cr,l_ln,l_ln);

fprintf('cr(5+%d)=FILLET/XSMALL,ln(7+%d),YLARGE,ln(13+%d),RADIUS,R3\n',l_cr,l_ln,l_ln);

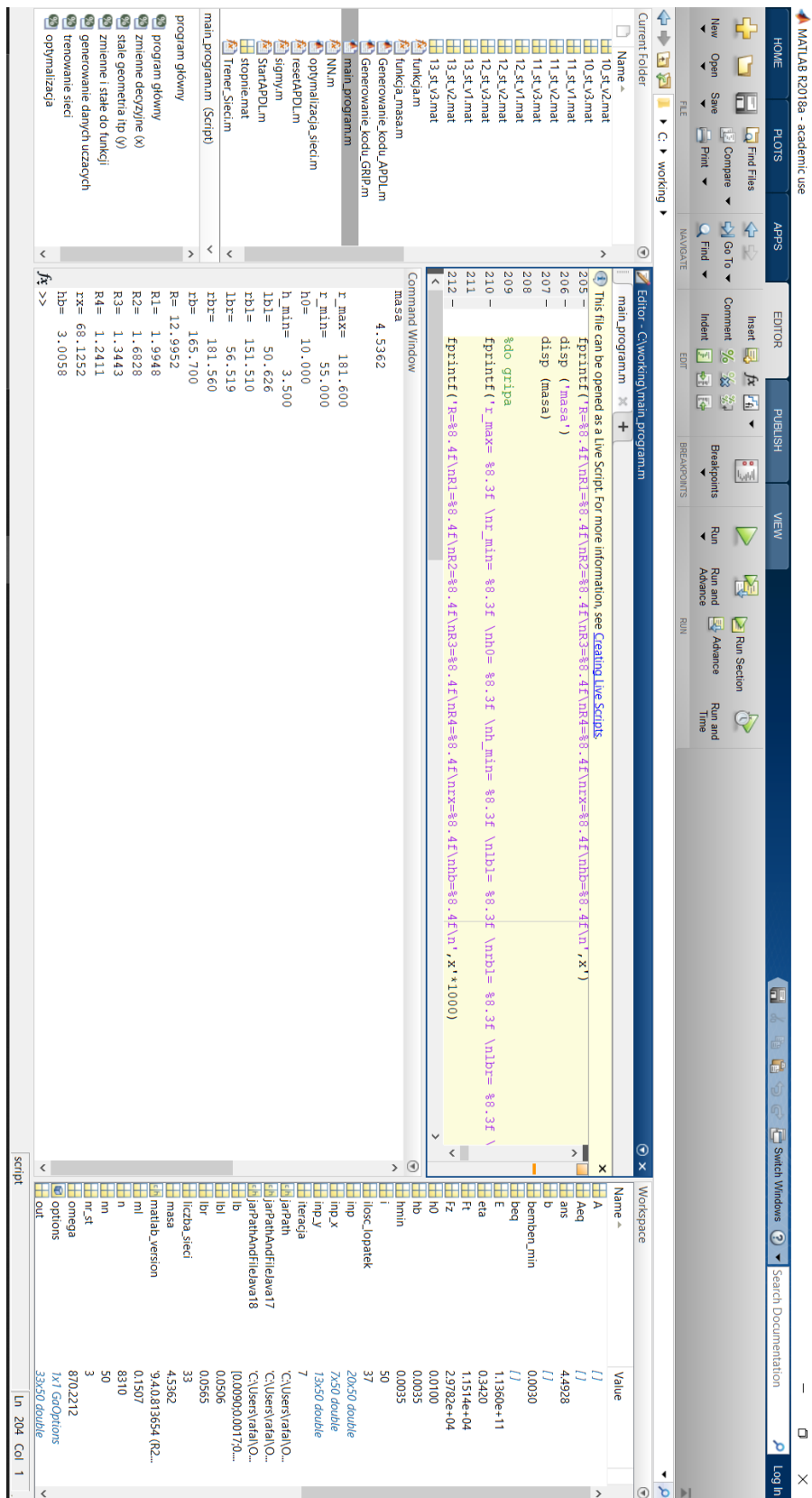
fprintf('cr(6+%d)=FILLET/XSMALL,ln(9+%d),YSMALL,ln(14+%d),RADIUS,R4\n',l_cr,l_ln,l_ln);

fprintf('solid(1+%d) = SOLREV/ln(1+%d..14+%d),cr(1+%d..6+%d),$ \n',j,l_ln,l_ln,l_cr,l_cr);
disp('ORIGIN,0,0,0,ATANGL,360,AXIS,0,1,0')

przesow=przesow+lbr;
end
disp('HALT')

```

ZAŁĄCZNIK 17 – PRZYKŁADOWA ODPOWIEDŹ PROGRAMU PO OPTYMALIZACJI



BIBLIOGRAFIA

1. Holland J. H. Genetic algorithms. *Scientific American*. 267.1: 66-73., 1992.
2. LEE Carmen Kar Hang. A review of applications of genetic algorithms in operations management. . *Engineering Applications of Artificial Intelligence*. 76: 1-12., 2018.
3. Katoch S., Chauhan S. S. i Kumar V. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*. 80.5: 8091-8126., 2021.
4. Deb K. i Agrawal R. B. Simulated binary crossover for continuous search space. . *Complex systems*, . 9(2), 115-148., 1995.
5. Eshelman L. J., Caruana R. A. i Schaffer J. D. Biases in the crossover landscape. *Proceedings of the third international conference on Genetic algorithms* . pp. 10-19, 1989.
6. Michalewicz Z. *Genetic Algorithms+ Data Structure= Evolution Programs* Springer. . New York : brak nazwiska, 1992.
7. Ono I. Real-coded genetic algorithm for function optimization using unimodal normal distribution crossover. *Proceedings of 7th ICGA*. pp. 246-253, 1997.
8. Deep K. i Thakur M. A new crossover operator for real coded genetic algorithms. *Applied mathematics and computation*. 188(1), 895-911, 2007.
9. Blicke T. i Thiele L. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*. 4.4: 361-394., 1996.
10. Baker J. E. Adaptive selection methods for genetic algorithms. *Proceedings of an International Conference on Genetic Algorithms and their applications*. 1985.
11. Goldberg D. E., Korb B. i Drb K. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*. 3.5: 493-530, 1989.
12. Dianati M., Song I. i Treiber M. An introduction to genetic algorithms and evolution strategies. Technical report, University of Waterloo, Ontario, N2L 3G1, Canada. 2002.
13. Sivaram M. i inni. Exploiting the local optima in genetic algorithm using Tabu search. *Indian Journal of Science and Technology*. 12(1), 1-13., 2019.
14. Hamamoto A. H. i inni. Network anomaly detection system using genetic algorithm and fuzzy logic. . *Expert Systems with Applications*. 92, 390-402., 2018.
15. Reynolds J. i inni. A zone-level, building energy optimisation combining an artificial neural network, a genetic algorithm, and model predictive control. *Energy*. 151, 729-739., 2018.
16. Kozakiewicz A., Kieszek R. i Rogólski R. Optimization of a Jet Engine Compressor Disc with Application of Artificial Neural Networks for Calculations Related to Time and Mass Criteria. . *Advances in Science and Technology Research Journal*. 15(2), 2021.
17. Hebb D. O. The organization of behavior: a neuropsychological theory. brak miejsca : J. Wiley; Chapman & Hall,, 1949.
18. Minsky M. L. Theory of neural-analog reinforcement systems and its application to the brain model problem. brak miejsca : Princeton University, 1954.

19. Ambroch K. <https://smp.uph.edu.pl/msn/32/ambroch.pdf>. [Online] [Zacytowano: 12 04 2022.]
20. Osowski S. Sieci neuronowe do przetwarzania informacji. Warszawa : Oficyna Wydawnicza Politechniki Warszawskiej, 2006. ISBN 83-7207-615-4.
21. Witryna sieci Web Katedry Inżynierii Komputerowej Uniwersytetu Rzeszowskiego. [Online] 30 08 2017. www.neurosoft.edu.pl/media/pdf/tkwater/sztuczna_inteligencja/2_alg_ucz_ssn.pdf.
22. Wantoch-Rekowski R. Sieci neuronowe w zadaniach. Perceptron wielowarstwowy. Warszawa : BEL Studio Sp. z o. o., 2003. ISBN 83-88442-67-8.
23. Osowski S. Sieci neuronowe w ujęciu algorytmicznym. Warszawa : Wydawnictwo Naukowo-Techniczne, 1996. ISBN 83-204-1976-X.
24. Tadeusiewicz R. Sieci neuronowe. Warszawa : Akademicka Oficyna Wydawnicza, 1993. ISBN 83-85769-03-X.
25. Rumelhart D. E., Hinton G. E. i Williams R. J. Learning representations by back-propagating errors. *Nature*. 1986, 323, strony 533-536.
26. Apoorv MAHESHWARI, Navindran DAVENDRALINGAM i A DELAURENTIS Daniel. A comparative study of machine learning techniques for aviation applications. In: 2018 Aviation Technology, Integration, and Operations Conference. 2018. p. 3980.
27. Kozakiewicz A. i Kieszek R. Zastosowanie sztucznych sieci neuronowych do obliczeń wytrzymałościowych maszyn wirnikowych. *Przegląd Mechaniczny*, LXXVII. 2018, strony 28-30.
28. R. Kieszek, A. Kozakiewicz i R. Rogólski. Optimization of a Jet Engine Compressor Disc with Application of Artificial Neural Networks for Calculations Related to Time and Mass Criteria. *Advances in Science and Technology Research Journal*. 2021.
29. Kumar A. i inni. Performance Based Anomaly Detection Analysis of a Gas Turbine Engine by Artificial Neural Network Approach. *Annual Conference of the Prognostics and Health Management Society* 2012. 2012, Tom 3, 107.
30. DePold H. R. i Gass F. D. The application of expert systems and neural networks to gas turbine prognostics and diagnostics. *THE AMERICAN SOCIETY OF MECHANICAL ENGINEERS*. 1998.
31. NASCIMENTO Renato Giorgiani i VIANA Felipe AC. Cumulative Damage Modeling with Recurrent Neural Networks. *AIAA Journal*. 2020, 58.12: 5459-5471.
32. NYULÁSZI Ladislav, et al. Fault detection and isolation of an aircraft turbojet engine using a multi-sensor network and multiple model approach. *Acta Polytechnica Hungarica*. 2018, 15.2: 189-209.
33. DE GIORGI Maria Grazia, FICARELLA Antonio i DE CARLO Laura. Jet engine degradation prognostic using artificial neural networks. *Aircraft Engineering and Aerospace Technology*. 2019.
34. CHEN Ho-Sheng, LAN Tian-Syung i LAI Yu-Ming. Prediction Model of Working Hours of Cooling Turbine of Jet Engine with Back-propagation Neural Network. *Sensors and Materials*. 2021, 33.2: 843-858.

35. MORINAGA Makoto i al. et. Identification of jet aircraft model based on frequency characteristics of noise by convolutional neural network. . *Acoustical Science and Technology*. 2019, 40.6: 391-398.
36. KUMARIN A., KUZNETSOV A. i MAKARYANTS G. Hardware-in-the-loop neuro-based simulation for testing gas turbine engine control system. . In: 2018 Global Fluid Power Society PhD Symposium (GFPS). IEEE. 2018. p. 1-5.
37. Grazia DE GIORGI Maria i QUARTA Marco. Hybrid multigene genetic programming-artificial neural networks approach for dynamic performance prediction of an aeroengine. *Aerospace Science and Technology*,. 2020, 103: 105902.
38. Yu Y. i inni. Neural-network based analysis and prediction of a compressor's characteristic performance map. *Applied energy*, 84(1), 48-55. 84, 2007, Tomy 48-55.
39. K. Ghorbanian M. Gholamrezaei. An artificial neural network approach to compressor performance prediction. *Applied Energy*. 86, 2009, strony 1210–1221.
40. Cortés O., Urquiza G. i Hernández J. A. . Optimization of operating conditions for compressor performance by means of neural network inverse. *Applied Energy*. 86, 2009, 2487–2493.
41. Roberts R i Eastbourn S. Modeling techniques for a computational efficient dynamic turbofan engine model. *International Journal of Aerospace Engineering*. 2014.
42. Pogorelov G. I. i inni. Application of neural network technology and high-performance computing for identification and real-time hardware-in-the-loop simulation of gas turbine engines. *Procedia Engineering*. 2017, 176, strony 402–408.
43. Bazazzadeh M., Badihi H. i Shahriari A. Gas Turbine Engine Control Design Using Fuzzy Logic and Neural Networks. *International Journal of Aerospace Engineering*. 2011, Tom 2011.
44. S. Sina Tayarani-Bathaie Z. N. Sadough Vanini, K. Khorasani. Dynamic neuralnetwork-basedfaultdiagnosisofgasturbineengines. *Neurocomputing*. 2014, 125, strony 153–165.
45. M. Bogdan J. Błachnio. Metoda wspomagania komputerowego w badaniach. *Biuletyn WAT*. Vol. LX, Nr1, 2011.
46. FRĄCKOWIAK A. CIAŁKOWSKI M., WRÓBLEWSKA A. Application of iterative algorithms for gas-turbine blades cooling optimization. . *International Journal of Thermal Sciences*, . 2017, 118: 198-206.
47. ORKISZ M. STAWARZ S. Modeling of turbine engine axial-flow compressor and turbine characteristics. *Journal of Propulsion and Power*. 2000, 16.2: 336-339.
48. FRĄCKOWIAK A. WOLFERSDORF J. v, CIAŁKOWSKI, M. Optimization of cooling of gas turbine blades with channels filled with porous material. *International Journal of Thermal Sciences*. 2019, 136: 370-378.
49. PAWLAK W. I. BALICKI W. Influence of an inequality of gas thermal field at the jet engine turbine inlet on to the speed of transient processes—the results of experiments with real engine. *Journal of KONES Internal Combustion Engines*. 2003, 10: 3-4.
50. Szczeciński S. Balicki W., Głowacki P. Uszkodzenia silników turbinowych wywołane zderzeniami z ptakami. *Przegląd Sił Powietrznych*. 2009;2:15-21.

51. WITEK L. et al. Fracture analysis of a turbine casing. *Engineering Failure Analysis*. 2011, 18.3: 914-923.
52. Nascimento R. G. i Viana F. A. Cumulative Damage Modeling with Recurrent Neural Networks. *AIAA Journal*. 58.12: 5459-5471., 2020.
53. Nyulászi L. et al. Fault detection and isolation of an aircraft turbojet engine using a multi-sensor network and multiple model approach. *Acta Polytechnica Hungarica*. 15.2: 189-209., 2018.
54. De Giorgi M. G. i Quarta M. Hybrid multigene genetic programming-artificial neural networks approach for dynamic performance prediction of an aeroengine. *Aerospace Science and Technology*,. 103: 105902, 2020.
55. Stycunów A. i Manerowski J. Modelowanie faz zniżania i lądowania samolotu Boeing 767-300ER przy użyciu sztucznych sieci neuronowych. *Prace Naukowe Politechniki Warszawskiej. Transport*. 2014, Tom z. 102.
56. Sęp J. i Kozik P. Systemowe planowanie remontów silników lotniczych z wykorzystaniem sztucznych sieci neuronowych.
57. Mikrut Z. Tworzenie reprezentacji obszarów zdjęć lotniczych za pomocą sieci neuronowych klasy PCNN. *Automatyka*. 2007, Tom 11.
58. Mikrut S. i Mikrut Z. Sieci neuronowe w procesach automatycznej korelacji zdjęć lotniczych. *Archiwum Fotogrametrii, Kartografii i Teledetekcji*,. 2007, Tom 17b.
59. Czechowicz A. i Mirkut Z. Wykorzystanie sieci Kohonena do selekcji podobrazów na potrzeby dopasowania zdjęć lotniczych. *Automatyka*. 2007, Tom 11.
60. Vogt R. i inni. Sterowanie samonaprowadzających pocisków mózdzierzowych z wykorzystaniem sterownika opartego na sztucznych sieciach neuronowych. *Problemy Techniki Uzbrojenia*. 2005, Tomy 34, z. 94.
61. Nowak D., Tomczyk A. i Rogalski T. Analiza jakości sterowania bezzałogowym statkiem powietrznym w przypadku zastosowania redundancji analitycznej pomiarów. *Autobusy : technika, eksploatacja, systemy transportowe*. 2016, Tom 17, 12, strony 1248-1253.
62. Intelligent Flight Control System. NASA Facts. National Aeronautics and Space Administration. 2002.
63. Waszczyszyn Z. i Ziemiański L. Neural networks in mechanics of structures and materials—new results and prospects of applications. *Computers & Structures*. 2001, Tom 79, strony 2261-2276.
64. Pabisek E. Systemy hybrydowe integrujące MES i SSN w analizie wybranych problemów mechaniki konstrukcji i materiałów. Kraków : Wydawnictwo Politechniki Krakowskiej, 2008.
65. Potrzeszcz-Sut B. Analiza hybrydowa MES/SSN sprężysto-plastycznej konstrukcji kratowej poddanej obciążeniu cyklicznemu. *STRUCTURE AND ENVIRONMENT. ARCHITECTURE, CIVIL ENGINEERING, ENVIRONMENTAL ENGINEERING AND ENERGY*. 2014, Tom 6, 4, strony 12-16.
66. Pabisek E. Identification of an equivalent model for granular soils by FEM/NMM/p-EMP hybrid system. *Computer Assisted Mechanics and Engineering Sciences*. 18, 2011.

67. Pierret S. i Van den Braembussche R. A. TURBOMACHINERY BLADE DESIGN USING A NAVIER-STOKES SOLVER AND ARTIFICIAL NEURAL NETWORK. *Journal of Turbomachinery*. 1999, 121 (2).
68. Głuch J. i Butterweck A. Zastosowanie sztucznych sieci neuronowych do diagnostyki cieplno-przepływowej wieńców turbin parowych. *MECHANIK*. 2014, Tom XVIII, 7.
69. Grzymowska A. Model neuronowy jako alternatywa dla numerycznego modelu okółodźwiękowego przepływu pary przez palisadę turbinową. *MECHANIK*. 2014, Tom XVIII, 7.
70. Grzymkowska A. i Szewczuk N. Modelowanie przepływu pary przez okółodźwiękowe wieńce turbinowe z użyciem sztucznych sieci neuronowych. *MECHANIK*. 2016, 7.
71. Pierret S., Coelho R. F. i Kato H. Multidisciplinary and multiple operating points shape optimization of three-dimensional compressor blades. *Struct Multidisc Optim*. 2007, strony 61–70.
72. Dornberger R., Büche D. i Stoll P. Multidisciplinary optimization in turbomachinery design. *European Congress on Computational Methods in Applied Sciences and Engineering. ECCOMAS 2000* 2000.
73. Lee S. Y., i Kim K. Y. Design optimization of axial flow compressor blades with three-dimensional Navier-Stokes solver. *KSME International Journal*. 2000, strony Vol. 14, No.9, pp. 1005-1012.
74. Panchenko Y. i inni. Preliminary multi-disciplinary optimization in turbomachinery design. *PRATT AND WHITNEY CANADA CORP LONGUEUIL (QUEBEC)*. 2003.
75. Jyoshna D. i Krishnaiah D. Optimization Shape of Compressor Disc. *International Journal of Innovative Research in Science*. Vol. 6, Issue 7 2017.
76. Sai Prashanth R. C. i Yogananda A. Weight Optimisation and Burst Margin of Aero Engine Compressor Disc. *International Journal for Research in Applied Science & Engineering*. Volume 5, Issue VI 2017, strony 1105-1113.
77. K. Spodniak M. Semrád K. Draganová. Turbine Blade Temperature Field Prediction Using the. *Applied Sciences*. 11.6, 2021.
78. Keshtegar B. et al. Multi-extremum-modified response basis model for nonlinear response prediction of dynamic turbine blisk. *Engineering with Computers*. 1-12, 2021.
79. NASA. Intelligent Flight Control System. *National Aeronautics and Space Administration*. 2002.
80. Kontsevich A. G. Accounting for time factor in identification of gas turbine engine model using artificial neural network. *Aerospace technics and technology*. 2006, 10(36), strony 144-151.
81. Zhernakov S. V. Identification of gas turbine engines characteristics on neural network technology. *Information technologies*. 2010, 3, strony 39-47.
82. Zhernakovs. S. V. Trend analysis of aircraft gas turbine engine parameters based on neural network technology. *Vestnik UGATU*. 2011, Tom 15, 4(44), strony 25-32.
83. Cole W. E. *Neural Network Modeling of a Power Generation Gas Turbine*. 1997.

84. Asgari H. Modelling, Simulation and Control of Gas Turbines Using Artificial Neural Networks. University of Canterbury Christchurch : Nowa Zelandia, 2014.
85. Asgari H. i Chen X. Gas Turbines Modeling, Simulation, and Control: Using Artificial Neural Networks. CRC Press, Taylor & Francis Group. 2016, XXX.
86. Lee S. M., Roh T. S. i Choi D. W. Defect diagnostics of SUAV gas turbine engine using hybrid SVM-artificial neural network method. Journal of Mechanical Science and Technology. 23, 2009, strony 559-568.
87. Moghaddam J. Javadi, Farahani M.H. i Amanifard N. . A neural network-based sliding-mode control for rotating stall and surge in axial compressors. Applied Soft Computing 11. s. 1036–1043 2011.
88. Stephen O.T. Ogaji Riti Singh. Advanced engine diagnostics using artificial neural networks. Applied Soft Computing 3. 2003, strony 259–271.
89. Joly R.B. i inni. Gas-turbine diagnostics using artificial neural-networks for a high bypass ratio military turbofan engine. Applied Energy, 78(4). 2004, strony 397–418.
90. Achillas C., Tzetzis D. i Raimondo M. O. Alternative production strategies based on the comparison of additive and traditional manufacturing technologies. Int. J. Production Economics 178. 2016.
91. Hadroug N. et al. Faults detection in gas turbine using hybrid adaptive network based fuzzy inference systems. Diagnostyka, 17 (4). s. 3-17 2016.
92. Liu X. et al. PSO-BP neural network-based strain prediction of wind turbine blades. Materials. 12.12: 1889., 2019.
93. al. Liu Z. et. GA-BP neural network-based strain prediction in full-scale static testing of wind turbine blades. Energies. 12.6: 1026., 2019.
94. Sessarego M. et al. Design optimization of a curved wind turbine blade using neural networks and an aero-elastic vortex method under turbulent inflow. Renewable Energy. 146: 1524-1535., 2020.
95. Albanesi A. et al. A metamodel-based optimization approach to reduce the weight of composite laminated wind turbine blades. Composite Structures. 194: 345-356., 2018.
96. Zhang C. Y. et al. Creep-Based Reliability Evaluation of Turbine Blade-Tip Clearance with Novel Neural Network Regression. Materials. 12.21: 3552, 2019.
97. Zhang C. et al. Reliability-based low fatigue life analysis of turbine blisk with generalized regression extreme neural network method. Materials. 12.9: 1545., 2019.
98. Song L. K., Bai G. C. i Fei C. W. Dynamic surrogate modeling approach for probabilistic creep-fatigue life evaluation of turbine disks. Aerospace Science and Technology. 95: 105439., 2019.
99. Holland J. H. et al. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press. 1992.
100. Gwiazda T. D. Algorytmy genetyczne, kompendium. Vol. I, Operator krzyżowania dla problemów numerycznych,. Warszawa : PWN, 2007.
101. —. Algorytmy genetyczne, kompendium. Vol. II, Operator mutacji dla problemów numerycznych. brak miejsca : PWN, Warszawa., 2007.

102. Osowski S. Sieci neuronowe do przetwarzania informacji. Warszawa : Oficyna Wydawnicza Politechniki Warszawskiej, 2000.
103. McCulloch W. i Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*. 1943, Tom 5, strony 115–133.
104. Karlik B. i V. Olgac A. Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*. 2011, 1.4: 111-122.
105. Linnainmaa S. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's Thesis (in Finnish), Univ. Helsinki. 1970.
106. —. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*. 1976, 16, strony 146–160.
107. Riedmiller M. i Braun H. Rprop-a fast adaptive learning algorithm. *Proc. of ISCIS VII*, Universitat. 1992.
108. Charalambous Ch. Conjugate gradient algorithm for efficient training of artificial neural networks. *IEE Proceedings G (Circuits, Devices and Systems)*. 139.3: 301-310., 1992.
109. Møller M. F. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*. 1993, 6, strony 525-533.
110. MATLAB R2017A.
111. Łagosz M. i Szczeciński S. Konstrukcja Silników Lotniczych. Wybrane zagadnienia wytrzymałości i dynamiki konstrukcji. Warszawa : Wojskowa Akademia Techniczna, 1985.
112. Nowotarski I. Obliczenia statyczne i dynamiczne turbinowych silników lotniczych metodą elementów skończonych. Warszawa : Wydawnictwa Naukowe Instytutu Lotnictwa, 2001.
113. Hagan M.T., H.B. Demuth, and M.H. Beale. *Neural Network Design*. Boston, MA: PWS Publishing. 1996.
114. Braun M. i Riedmiller H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *Proceedings of the IEEE International Conference on Neural Networks*,. 1993, strony 586–591.
115. Wojtaszek M., Śleboda, T., Czulak, A., Weber, G., & Hufenbach, W. A. WOJTASZEK, M., et al. Quasi-static and dynamic tensile properties of Ti-6Al-4V alloy. *Archives of Metallurgy and Materials*. 58, 2013.
116. Lim S. S., Lee H. J. i Song S. H. Flow Stress of Ti-6Al-4V during Hot Deformation: Decision Tree Modeling. *Metals*. 10.6: 739, 2020.

SPIS TABEL

Tabela 1.1. Podział faz lotu podczas zniżania i lądowania (55)	13
Tabela 1.2. Zakres danych dla poszczególnych serii (84)	28
Tabela 1.3. Porównanie wyników diagnozy sprężarki dla różnych algorytmów, gdzie: GPA – <i>Gas Path Analysis</i> (analiza linii prądu), ANN – <i>Artificial Neural Network</i> (SSN), RMS – błąd średniokwadratowy (88).	30
Tabela 2.1. Zakresy danych wejściowych do uczenia SSN	52
Tabela 2.2. Dane użyte do sprawdzenia sieci	53
Tabela 2.3. Zakres danych wejściowych do programu MES liczącego naprężenia .	55
Tabela 2.4. Znormalizowane długości elementów	56
Tabela 3.1. Efektywność sieci w zależności od ilości warstw ukrytych.....	63
Tabela 3.2 RMSE, MAPE i R^2 dla ostatniej iteracji algorytmu.....	67
Tabela 4.1. Wymiary gabarytowe i geometryczne warunków brzegowych dla optymalizowanych stopni optymalizowanego wirnika, gdzie E – moduł Younga, ρ – gęstość, ν – liczba Poissona, ml – masa łopatk, rsm – promień środka masy części zamkowej, $rsml$ – promień środka masy łopatk, V – objętość części zamkowej. Pozostałe oznaczenia zgodne z Rys. 3.4.....	71
Tabela 6.1. Wartości parametrów wymiarowych tarcz, wyliczane w trzech cyklach optymalizacji, dla określonych stopni sprężarki. Pogrubiono najlepsze rozwiązania. ...	94
Tabela 6.2. Zestawienie objętości optymalizowanych elementów konstrukcyjnych	97
Tabela 6.3. Wartość średnia δ [%] i odchylenie standardowe σ [%] błędu względnego dopasowania oraz współczynnik determinacji R^2 naprężeń wyznaczonych przez algorytm SSN i program ANSYS δ [%]	104

SPIS RYSUNKÓW I WYKRESÓW

Rys. 0.1. Rodzaje odwzorowań realizowane przez perceptron (19), gdzie XOR – alternatywa rozłączna	11
Rys. 1.1. Porównanie przebiegu prędkości przyrządowej i wygenerowanej przez SSN podczas lądowania samolotu (55).....	13
Rys. 1.2. Ogólna struktura algorytmu hybrydowego przedstawionego w pracy (67)	17
Rys. 1.3. Schemat danych wejściowych do uczenia SSN wraz z zaznaczoną pętlą sprzężenia zwrotnego błędu średniokwadratowego; (a) schemat turbiny gazowej, (b) schemat uczenia Sztucznej Sieci Neuronowej.....	17
Rys. 1.4. Struktura trójwarstwowej sieci neuronowej z wsteczną propagacją błędu do wyznaczania parametrów turbiny gazowej (38)	18
Rys. 1.5. Wyniki optymalizacji płaskiej palisady turbiny; a) geometria łopatk turbiny, b) rozkład liczby Macha na powierzchni łopatk turbiny (67).....	18
Rys. 1.6 Rozkład liczby Macha na promieniu zewnętrznym (a), promieniu średnim (b), promieniu wewnętrznym (c) kanału sprężarki (71)	19
Rys. 1.7. Rozkład ciśnienia w kanale przepływowym turbiny: a) obliczenia za pomocą SSN, b) obliczenia CFD, c) wartości błędu (68)	20
Rys. 1.8. Algorytm optymalizujący łopatkę 3D turbiny (72)	21
Rys. 1.9. Charakterystyka sprężarki i wentylatora wraz z linią pracy (41)	21
Rys. 1.10. Struktura modelu w LabView wraz ze schematem jego integracji z układem FADEC (42)	23
Rys. 1.11. Metoda identyfikacji modelu przez SSN w oparciu o rzeczywiste dane (42).....	23
Rys. 1.12. Odpowiedź modelu silnika na skokową zmianę położenia dźwigni sterowania silnikiem w funkcji czasu, na poziomie morza; a) zredukowana prędkość obrotowa, b) ciąg, c) zapas pracy statecznej, d) temperatura przed turbiną, e) masowe natężenie przepływu paliwa (43)	25
Rys. 1.13. Schemat blokowy hybrydowej metody wektorów nośnych i Sztucznych Sieci Neuronowych.....	26
Rys. 1.14. Schemat blokowy algorytmu określającego optymalną strukturę SSN identyfikującej stan turbiny gazowej (84)	27
Rys. 1.15. Błąd średniokwadratowy modelu Simulink i NARX dla wybranych wyników wszystkich symulacji (84), gdzie M1-M4 – serie danych opisane zgodnie z Tabelą 1.2, N – prędkość obrotowa, PRc – spręż sprężarki, T02 – temperatura za sprężarką, T04 – temperatura za turbiną.....	28
Rys. 1.16. Prędkość obrotowa w funkcji czasu. Odpowiedzi trzech różnych kontrolerów silnika odrzutowego (84), gdzie: <i>Rotational Speed</i> – prędkość obrotowa [obr/min], <i>Time</i> – czas [s]	29

Rys. 1.17. Zależność przepływu informacji między modułami modelu silnika odrzutowego (44).....	31
Rys. 1.18. Wynik testowania przeszkolonej sieci neuronowej dla jednego z silników (44).....	31
Rys. 1.19. Struktura dynamicznej sieci neuronowej do wykrywania usterek sprężarki niskiego ciśnienia dwuwirnikowego silnika odrzutowego (44)	32
Rys. 1.20. Porównanie jakości dopasowania modeli statystycznych z wykorzystaniem danych nieużywanych w czasie przygotowywania algorytmów na przykładzie predykcji kosztu jednostkowego (90)	33
Rys. 2.1. Krzyżowanie i mutacje	36
Rys. 2.2. Komórka nerwowa (102)	37
Rys. 2.3. Model komórki nerwowej według McCullocha-Pittsa.....	38
Rys. 2.4. Dwuteowa belka wysięgnikowa obciążona siłą skupioną, gdzie: F – wektor siły, l – całkowita długość belki, x – współrzędne przekroju obliczeniowego wzdłuż osi belki, H – wysokość dźwigara, h – wysokość ścianki dźwigara, B – szerokość pasa dźwigara, b – szerokość pasa bez ścianki, X – oś odciętych.....	46
Rys. 2.5. Schemat Sztucznej Sieci Neuronowej (110).....	47
Rys. 2.6. Naprężenia obliczone za pomocą Sztucznej Sieci Neuronowej (σ_{SSN}) i analitycznie (σ_{ANAL}). Na osi pomocniczej zaznaczono zakres możliwych wartości błędu względnego (δ)	47
Rys. 2.7. Przebieg procesu uczenia Sztucznej Sieci Neuronowej.....	48
Rys. 2.8. Błąd średniokwadratowy uczenia Sztucznej Sieci Neuronowej.....	49
Rys. 2.9. Tarcza prosta; a) przypadek rzeczywisty, b) przypadek uproszczony do obliczeń, gdzie: n – prędkość obrotowa, ω – prędkość kątowna, R_0 – promień otworu centralnego, R – promień zewnętrzny tarczy, σ_w – ciągnięcie wieńcowe.....	50
Rys. 2.10. Schemat Sztucznej Sieci Neuronowej do obliczeń tarczy o stałej grubości	51
Rys. 2.11. Ustawienia procesu uczenia sieci (110).....	52
Rys. 2.12. Naprężenia (σ) i błąd względny (δ) popełniony przez sieć na promieniach bieżących (r) – dla przykładu 1 (Tabela 2.2).	53
Rys. 2.13. Naprężenia (σ) i błąd względny (δ) popełniony przez sieć promieni bieżących (r) – dla przykładu 2 (Tabela 2.2).	53
Rys. 2.14. Model tarczy profilowanej z przedstawionym zagęszczeniem siatki obliczeniowej, gdzie: ω – prędkość kątowna, R_0 – Promień otworu centralnego, R – promień zewnętrzny tarczy, σ_w – ciągnięcie wieńcowe, r_i – promieniowa współrzędna i -tego węzła obliczeniowego, h_i - grubość i -tego elementu.....	55
Rys. 2.15. Schemat blokowy zastosowanego algorytmu	57
Rys. 2.16. Rozkład naprężeń w tarczy profilowanej wyznaczony za pomocą MES (σ_{MES}) SSN (σ_{SSN}) wraz z błędem względnym (δ).....	58
Rys. 3.1. Schemat blokowy algorytmu do optymalizacji tarczy sprężarki	60

Rys. 3.2. Parametry wejściowo-wyjściowe SSN ze współczynnikiem szkolenia, testowania i walidacji,.....	61
Rys. 3.3. Schemat SSN do obliczania naprężeń, gdzie: Input 24 – dane wejściowe; Hidden Layer 1, 2, 3, 4 – warstwy ukryte, po 50 neuronów każda; Output Layer – pojedynczy neuron warstwy wyjściowej z sieci; Output 1 – parametr wyjściowy z sieci	64
Rys. 3.4. Tarcza stopnia sprężarki osiowej turbinowego silnika odrzutowego: obiekt rzeczywisty (a) oraz model wzorcowy tarczy z zaznaczonym obszarem optymalizowanym (b)	65
Rys. 3.5. Kształt zoptymalizowanej tarczy i odpowiadający jej rozkład naprężeń zredukowanych, gdzie x – profil tarczy po optymalizacji; h - profil tarczy przed optymalizacją; δ – błąd względny naprężeń; σ_{MES} – naprężenia wyznaczone za pomocą MES; σ_{SSN} – naprężenia wyznaczone za pomocą SSN	66
Rys. 3.6. Zestawienie profilu tarczy wyjściowej (kolor szary) i zoptymalizowanej (kolor pomarańczowy). Powiększono obszar największych zmian.....	66
Rys. 3.7. Porównanie czasu pracy pojedynczej iteracji algorytmu wzorcowego, opartego wyłącznie na MES i algorytmu wykorzystującego SSN	68
Rys. 4.1. Wirnik silnika AŁ-21F3.....	69
Rys. 4.2. Model optymalizowanych stopni sprężarki silnika AŁ-21F3.....	70
Rys. 4.3. Model parametryczny tarczy z oznaczeniem głównych wymiarów. Kolorami zaznaczono ścieżki, wzdłuż których wyznaczano naprężenia do optymalizacji	72
Rys. 4.4. Optymalizowana tarcza sprężarki: a) obiekt rzeczywisty, b) model CAD	73
Rys. 4.5. Zdyskredytowany model tarczy z zaznaczonymi warunkami brzegowymi, gdzie A i C – strefa warunku symetrii, B – strefa ciągnięcia wieńcowego	73
Rys. 4.6. Wirnik do optymalizacji zespołu sprężarkowego	74
Rys. 4.7. Rozkład naprężeń wzdłuż pierwszej ścieżki.....	75
Rys. 4.8. Model tarczy o stałej grubości przygotowany do walidacji programu	76
Rys. 4.9. Walidacja kodu napisanego w języku APDL	76
Rys. 4.10. Model tarczy z oznaczonymi kolejnymi obszarami.....	77
Rys. 4.11. Oznaczenia punktów niezbędnych do wyznaczenia objętości bryły powstałej przez obrót wokół osi OX.....	78
Rys. 5.1. Maksymalny i średni błąd względny dla wybranych struktur sieci neuronowych i algorytmów uczących po pierwszym cyklu uczenia.....	82
Rys. 5.2. Maksymalny i średni błąd względny dla wybranych struktur sieci neuronowych i algorytmów uczących po drugim cyklu uczenia.....	82
Rys. 5.3. Maksymalny i średni błąd względny dla wybranych struktur sieci neuronowych i algorytmów uczących po trzecim cyklu uczenia	83
Rys. 5.4. Fronty Pareto dla wszystkich sieci uczonych za pomocą algorytmu GDM	85

Rys. 5.5. Fronty Pareto w skali logarytmicznej dla wszystkich sieci uczonych za pomocą algorytmu GDM. Zbliżenie na największą dokładność	85
Rys. 5.6. Fronty Pareto dla wszystkich sieci uczonych za pomocą algorytmu SCG	86
Rys. 5.7. Fronty Pareto dla wszystkich sieci uczonych za pomocą algorytmu SCG. Zbliżenie na największą dokładność.....	86
Rys. 5.8. Fronty Pareto w skali logarytmicznej dla wszystkich sieci uczonych za pomocą algorytmu SCG.....	86
Rys. 5.9. Fronty Pareto w skali logarytmicznej dla dwóch najlepiej dopasowanych sieci uczonych za pomocą algorytmu SCG	86
Rys. 5.10. Front Pareto dla wszystkich badanych sieci uczonych za pomocą algorytmu RPROP	87
Rys. 5.11. Front Pareto dla wszystkich badanych sieci uczonych za pomocą algorytmu RPROP. Zbliżenie na największą dokładność	87
Rys. 5.12. Front Pareto w skali logarytmicznej dla wszystkich badanych sieci uczonych algorytmem RPROP	87
Rys. 5.13. Front Pareto w skali logarytmicznej dla dwóch najlepiej dopasowanych sieci uczonych za pomocą algorytmu RPROP	87
Rys. 5.14. Front Pareto w zależności od ilości neuronów w warstwie ukrytej z zaznaczonymi liniami trendu. Sieć uczona metodą SCG	89
Rys. 5.15. Front Pareto w zależności od ilości neuronów w warstwie ukrytej z zaznaczonymi liniami trendu. Sieć uczona metodą RPROP.....	89
Rys. 5.16. Dopasowanie danych uczących dla algorytmu a) SCG (NN 20-2-1) i b) RPROP (NN 20-3-1), gdzie dane <i>input</i> to naprężenia wyliczone przez APDL, a <i>output</i> to dane przewidziane przez SSN.....	90
Rys. 5.17. Porównanie dopasowania danych wykorzystanych do uczenia i sprawdzających dla algorytmu SCG (NN 20-2-1), gdzie dane <i>input</i> to naprężenia wyliczone przez APDL, a <i>output</i> to dane przewidziane przez SSN.....	91
Rys. 5.18. Porównanie dopasowania danych wykorzystanych do uczenia i sprawdzających dla algorytmu RPROP (NN 20-3-1), gdzie dane <i>input</i> to naprężenia wyliczone przez APDL, a <i>output</i> to dane przewidziane przez SSN.....	91
Rys. 6.1. Model wirnika sprężarki silnika AŁ21-F3.....	93
Rys. 6.2. Porównanie wirnika wzorcowego (a) z wirnikiem po optymalizacji (b); (c) obszary największych zmian geometrii przed optymalizacją i po niej.....	96
Rys. 6.3. Naprężenia Hubera-von Misesa wirnika po optymalizacji.....	98
Rys. 6.4. Naprężenia Hubera-von Misesa wirnika po optymalizacji. Odształcenia przeskalowane 100-krotnie celem ich uwydatnienia	98
Rys. 6.5. Składowa promieniowa przemieszczeń wirnika. Odształcenia przeskalowano 100-krotnie.....	98
Rys. 6.6. Dopasowanie naprężeń zredukowanych Hubera-von Misesa-Huckiego ($\sigma_H - M - H$) wyznaczone przez wykorzystywany algorytm i wyznaczone w programie ANSYS workbench (σ_{ANSYS}), dla wszystkich optymalizowanych stopni	99
Rys. 6.7. Dopasowanie naprężeń zredukowanych dla 3 stopnia.....	100

Rys. 6.8. Dopasowanie naprężeń zredukowanych dla 4 stopnia.....	100
Rys. 6.9. Dopasowanie naprężeń zredukowanych dla 5 stopnia.....	100
Rys. 6.10. Dopasowanie naprężeń zredukowanych dla 6 stopnia.....	101
Rys. 6.11. Dopasowanie naprężeń zredukowanych dla 7 stopnia.....	101
Rys. 6.12. Dopasowanie naprężeń zredukowanych dla 9 stopnia.....	101
Rys. 6.13. Dopasowanie naprężeń zredukowanych dla 10 stopnia.....	102
Rys. 6.14. Dopasowanie naprężeń zredukowanych dla 11 stopnia.....	102
Rys. 6.15. Dopasowanie naprężeń zredukowanych dla 12 stopnia.....	102
Rys. 6.16. Dopasowanie naprężeń zredukowanych dla 13 stopnia.....	103